



Sundin, Lovisa (2022) Graphical scaffolding for the learning of data wrangling APIs. PhD thesis.

<https://theses.gla.ac.uk/83122/>

Copyright and moral rights for this work are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This work cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

Enlighten: Theses

<https://theses.gla.ac.uk/>
research-enlighten@glasgow.ac.uk

Graphical Scaffolding for the Learning of Data Wrangling APIs

Lovisa Sundin

Submitted in fulfilment of the requirements for the
Degree of Doctor of Philosophy

School of Computing Science
College of Science and Engineering
University of Glasgow



University
of Glasgow

October 2021

Abstract

In order for students across the sciences to avail themselves of modern data streams, they must first know how to wrangle data: how to reshape ill-organised, tabular data into another format, and how to do this programmatically, in languages such as Python and R. Despite the cross-departmental demand and the ubiquity of data wrangling in analytical workflows, the research on how to optimise the instruction of it has been minimal. Although data wrangling as a programming domain presents distinctive challenges - characterised by on-the-fly syntax lookup and code example integration - it also presents opportunities. One such opportunity is how tabular data structures are easily visualised. To leverage the inherent visualisability of data wrangling, this dissertation evaluates three types of graphics that could be employed as scaffolding for novices: subgoal graphics, thumbnail graphics, and parameter graphics. Using a specially built e-learning platform, this dissertation documents a multi-institutional, randomised, and controlled experiment that investigates the pedagogical effects of these. Our results indicate that the graphics are well-received, that subgoal graphics boost the completion rate, and that thumbnail graphics improve navigability within a command menu. We also obtained several non-significant results, and indications that parameter graphics are counter-productive. We will discuss these findings in the context of general scaffolding dilemmas, and how they fit into a wider research programme on data wrangling instruction.

Contents

Abstract	i
Acknowledgements	xvii
Declaration	xviii
1 Introduction	1
1.1 Motivation	2
1.1.1 Why is data wrangling important to research?	3
1.1.2 Why is data wrangling important to teach?	3
1.1.3 Why is data wrangling <i>education</i> important to research?	4
1.1.4 Why are graphics as a scaffolding technique interesting to research?	4
1.2 Thesis statement and research questions	5
1.3 Approach	6
1.4 Thesis structure	7
2 Defining the scope	9
2.1 Defining data wrangling	9
2.1.1 What data wrangling includes	9
2.1.2 What data wrangling is not	11
2.2 Learner characteristics and contexts	12
2.2.1 Majors and non-majors	12
2.2.2 Curricular context	13
2.2.3 End-user programmers	13
2.3 Data wrangling technologies	14
2.3.1 Relational databases & SQL	14
2.3.2 Spreadsheet software	14
2.3.3 Menu-driven software	15
2.3.4 Automated data wrangling	15
2.4 Programming-based software	15
2.4.1 Language characteristics	16

2.4.2	R	17
2.4.3	Python	18
2.4.4	Summary	19
3	Literature review	20
3.1	Conceptual barriers	21
3.1.1	Challenges with dataframes	22
3.1.2	Challenges with split-apply-combine	24
3.1.3	Challenges with vectors and matrices	25
3.1.4	Section summary	25
3.2	Programming-related barriers	26
3.2.1	Problem comprehension	26
3.2.2	Plan composition	29
3.2.3	API lookup	30
3.2.4	Example adaptation	33
3.2.5	Debugging	36
3.2.6	Section summary	38
3.3	General learning theories	39
3.3.1	Cognitive load theory	39
3.3.2	Scaffolding	41
3.3.3	Graphical scaffolding	44
3.3.4	Section summary	46
3.4	Graphics in data wrangling	46
3.4.1	Program visualisation	47
3.4.2	Query visualisation	47
3.4.3	Explanatory diagrams	48
3.4.4	Visual programming languages	49
3.4.5	Node-link diagrams	49
3.4.6	Dynamic previews	49
3.4.7	Cheat sheets	50
3.4.8	Section summary	52
3.5	Scaffolding techniques	53
3.5.1	Plan composition	53
3.5.2	API lookup	56
3.5.3	Example adaptation	59
3.5.4	Section summary	61
3.6	Conclusion	61

4	Method	62
4.1	Breadth versus depth	62
4.2	Observational versus experimental designs	63
4.3	Randomised controlled trials	64
4.3.1	Specificity of the intervention	64
4.3.2	The choice between rigour and generalisability	65
4.3.3	The logistics of separating conditions	65
4.3.4	Designing a fair control condition	66
4.3.5	Attention as a mediating variable	67
4.3.6	The lack of educational measurements	67
4.3.7	Ethical implications	68
4.3.8	Recruitment of volunteers	68
4.3.9	Process metrics and product metrics	69
4.3.10	The challenge of statistical power	70
4.4	Conclusion	71
5	Parameter graphics	72
5.1	Background	72
5.1.1	The split-attention effect	73
5.1.2	Icon-based API cheat sheets	74
5.1.3	Parameter graphics in data wrangling	75
5.1.4	Multilingual cheat sheets	76
5.2	Research questions	77
5.3	Tutorial design	78
5.3.1	Tutorial	78
5.3.2	Cheat sheets	80
5.3.3	Exercises	80
5.4	Study 1	82
5.4.1	Method	82
5.4.2	Results	84
5.5	Study 2	85
5.5.1	Method	86
5.5.2	Results	88
5.6	Discussion	90
5.6.1	Future implications	91
5.6.2	Methodological considerations	91

6	Subgoal and thumbnail graphics materials	93
6.1	Menu design for API lookup	93
6.1.1	Ontology design	94
6.1.2	Thumbnail graphics	97
6.1.3	Walkthrough of ontology	99
6.2	Subgoals for plan composition	104
6.2.1	Subgoal labels	105
6.2.2	Subgoal graphics	106
6.3	Chapter summary	106
7	Design of Slice N Dice	109
7.1	Overall structure	110
7.1.1	Spring 2020 pilot	111
7.1.2	Autumn 2020 pilot version	112
7.1.3	Final version	113
7.2	Recurring design considerations	114
7.2.1	The problem of low stakes	114
7.2.2	The assistance dilemma	115
7.2.3	Exercise design	116
7.3	Part 1 design	116
7.3.1	Operation cards	117
7.3.2	Data wrangling exercises	119
7.4	Part 2 design	120
7.4.1	Programming exercises	121
7.4.2	Documentation	123
7.4.3	Hints	123
7.5	Part 3 design	123
7.5.1	Subgoals	124
7.5.2	Time limits	125
7.5.3	Feedback	126
7.5.4	Hints	127
7.5.5	Un scaffolded exercises	127
7.6	Further data collection	128
7.6.1	Demographic survey	128
7.6.2	Pretest	129
7.6.3	Evaluation survey	130
7.7	Chapter summary	130

8	Slice N Dice pilot studies	132
8.1	Qualitative usability study	133
8.1.1	Method	134
8.1.2	Part 1 results	136
8.1.3	Part 2 results	143
8.1.4	Part 3 results	147
8.1.5	Conclusion	151
8.2	Quantitative pilot study	151
8.2.1	Method	151
8.2.2	Part 1 results	153
8.2.3	Part 2/3 results	158
8.3	Chapter summary	162
9	Slice N Dice validation studies	163
9.1	Subgoal graphic validation study	164
9.1.1	Method	164
9.1.2	Results	166
9.1.3	Discussion	168
9.2	Thumbnail graphic validation study	168
9.2.1	Method	168
9.2.2	Results	172
9.2.3	Discussion	177
9.3	Summary	178
10	Slice N Dice results	179
10.1	Method	179
10.1.1	Design	179
10.1.2	Procedure	180
10.1.3	Participants	182
10.1.4	Analytical approach	184
10.2	Part 1	187
10.2.1	Number of exercises completed	187
10.2.2	Inactivity events	187
10.2.3	Operation card reading times	189
10.2.4	Tooltip events	190
10.2.5	Time on task	191
10.2.6	Number of attempts	194
10.2.7	Evaluation	196
10.3	Part 3	200

10.3.1	Number of exercises completed	200
10.3.2	Timeout events	202
10.3.3	Tab events	203
10.3.4	Time on task	205
10.3.5	Tooltip events	207
10.3.6	Menu click events	208
10.3.7	Syntax errors	210
10.3.8	Semantic errors	211
10.3.9	Hint usage	213
10.3.10	Unscaffolded exercise performance	215
10.3.11	Evaluation data	216
10.4	Summary	220
11	Slice N Dice secondary analyses	222
11.1	Volunteer bias	222
11.2	Bug analysis	223
11.3	Correlational analysis	225
11.3.1	Process metrics	225
11.3.2	The association between SLICE N DICE parts	227
11.3.3	The influence of experience on performance	228
11.3.4	The influence of experience on the effect of graphics	231
12	Discussion	234
12.1	Summary of findings	235
12.1.1	RQ1: The effect of subgoal graphics	235
12.1.2	RQ2: The effect of thumbnail graphics	235
12.1.3	RQ3: The effect of parameter graphics	236
12.2	Reflections	236
12.2.1	Subgoal graphics	236
12.2.2	Thumbnail graphics	237
12.2.3	Parameter graphics	238
12.3	Implications	239
12.3.1	Should instructors use subgoal graphics?	239
12.3.2	Should instructors use thumbnail graphics?	239
12.3.3	Should instructors use parameter graphics?	239
12.4	Limitations	240
12.4.1	Tutorial improvements	240
12.4.2	Possible interface improvements	241
12.4.3	Possible data collection improvements	241

12.5 Future work	241
12.6 Finale	242
Appendix A Parameter graphic tutorial	244
Appendix B Multilingual cheat sheets	247
Appendix C Slice N Dice taxonomy	250
Appendix D Slice N Dice exercises	252
Appendix E Subgoal validation survey	271
Appendix F Thumbnail validation survey	276

List of Tables

8.1 Table describing the 8 participants involved in the usability study. Every participant was recorded using SLICE N DICE for 3-4h. Minimal < Basic < Intermediate. 134

10.1 The dependent variables analysed in terms of group comparisons. G and \neg G represent the central tendencies of the graphical and non-graphical control condition, respectively. The hypothesis column thus indicates which direction we expect to see in the effect, i.e. whether the graphical condition is larger or smaller than the control condition. Some variables, labelled -, are not compared based on subgoal graphic condition. Variables in **bold** are performance metrics. This table does not include self-reported variables. 181

10.2 The number of exercises completed depending on condition. W = Wilcoxon rank sum test. 202

10.3 Results. The *hypothesis* columns indicates whether graphics were expected to reduce or increase the value of the metric. *TG vs. \neg TG* and *SG vs. \neg SG* indicate the direction of the group difference, where - indicates a lack of noteworthy group difference. *PS* indicates effect sizes, given in probability of superiority [1]. A dash (-) in PS indicates that no inference test was pursued on account of descriptive statistics. Asterisk indicates significance at $\alpha=.05$ (Part 3 tooltip events were significant for permutation tests, but not Wilcoxon rank sum test). None was significant at the Bonferroni-adjusted level ($\alpha=.0016$). Bold metrics indicate distal performance variables. 221

List of Figures

1.1	Examples of tabular graphics	2
1.2	The research is focused on three aspects of the data wrangling workflow, each of which is targeted by a graphical scaffolding feature.	5
2.1	The relative search volumes of data wrangling-related Google queries, smoothed. <i>Data wrangling</i> is the lightest blue, on the rise in the bottom-right corner. . . .	10
2.2	The computational data life cycle adapted from Wickham and Grolemund [2, p.ix]	10
3.1	Illustration of the tidy data format and 3 common violations of it.	23
3.2	Data wrangling as a process consists of several distinct tasks that convert a problem statement into executable code. <i>API lookup</i> includes mental recall. <i>Example adaptation</i> and <i>debugging</i> should be interpreted as probable rather than logically necessary activities.	26
3.3	Word problems require the data analyst to draw upon domain knowledge in order to interpret what the problem asks for.	28
3.4	Diagram summarising Kelleher & Ichinki’s <i>Collection and Organisation of Information for Learning</i> (COIL) model [3,4].	31
3.5	A Pandas documentation entry, featuring a syntax summary, a short prose description, and a parameter list. Data wrangling functions tend to be highly configurable.	34
3.6	Code examples from official documentation sources	35
3.7	The programmer needs to map elements in the documentation’s code example to elements in the problem context.	36
3.8	Diagram by Baumer [5] illustrating a filter and pivot operation	48
3.9	Taipalus graphical notation for relational database queries [6].	50
3.10	QUERYVIZ visualises the meaning of a relational query using a novel notation [7].	50
3.11	Kandel et al.’s WRANGLER uses colour cues to help their users understand the semantics of proposed operations [8, p.3366]. Seen are deletion of rows and a table pivot from long to wide format.	51

3.12	In Zhang and Guo’s DS.JS, clicking on a function triggers a preview that summarises the operation. These images show filtering and grouping, respectively. [9, p.697]	51
3.13	Snippet from RStudio’s cheat sheet for the data wrangling library dplyr [10]	52
3.14	A demonstration of how subgoal labels could be augmented with subgoal graphics, for the example from Section 3.2.2.	55
3.15	Interface of Kandel et al. WRANGLER system [8]	57
4.1	CS education research generally faces a trade-off of whether to contribute to theory advancement or design progress. This dissertation is mainly focused on refining and evaluating a particular design feature (<i>Design goals only</i>). Diagram adapted from Nelson and Ko [11, p.34].	63
5.1	Examples of educational resources where icons replace purely symbolic notation.	75
5.2	The parameter graphics provide placeholders for passing arguments and other information that configure an API command.	76
5.3	Excerpts from the tutorial booklet explaining each operation in turn and introducing parameter graphics along the way.	79
5.4	The two first exercises in the Python sheet. The second contains a real participant response.	81
5.5	Study 1 used a single-group design in which participants were first given the tutorial, then the cheat sheet and exercise sets for each language, in counterbalanced order.	82
5.6	Although some participants had basic experience with procedural Python, most students had no experience in SQL or R, according to their own ratings (1=“No experience at all”, 5=“I can do complex commands in it”).	83
5.7	Plots relating to the total time on task in Study 1.	85
5.8	Plots relating to the accuracy distributions of Study 1.	86
5.9	In Study 2, people were given one of two cheat sheet styles. Shown are excerpts from the SQL cheat sheet. The tutorial booklet was similarly produced in two variants.	87
5.10	The control group and experimental group had a mostly similar distribution of experience, though the control group appeared more experienced with R (1=“No experience at all”, 5=“I can do complex commands in it”).	88
5.11	Differences in total score distribution between the experimental group (with parameter graphics) and the control group (without such graphics).	89
5.12	Differences in time on task score distribution between the experimental group (with parameter graphics) and the control group (without such graphics).	90
6.1	An example of an ambiguous thumbnail.	99

6.2	The <i>Create</i> category.	100
6.3	The <i>Access</i> category.	101
6.4	The <i>Calculate</i> category.	103
6.5	The <i>Combine</i> category.	104
6.6	The <i>Restructure</i> category.	105
6.7	Examples of two exercises, complete with subgoal labels and subgoal graphics, which are only visible for the SG condition.	107
7.1	SLICE N DICE incorporates two experimental factors: one that manipulates the provision of subgoal graphics (SG, ¬SG) and thumbnail graphics (TG, ¬TG). ¬ indicates absence of graphics. This produces 2x2 distinct conditions.	109
7.2	The system was implemented as a web application, using standard web technologies such as MongoDB, Express, Angular and Node (a MEAN stack). DataCamp Light (DCL) was an important external dependency that provides the IDE front-end widget and back-end code execution.	111
7.3	To balance pedagogical and methodological considerations, the application went through multiple iterations.	113
7.4	The final structure of SLICE N DICE. Part 2 has been split into two parts and 3 data wrangling programming exercises are unscaffolded - without subgoals for the first 10 minutes.	114
7.5	The landing page reveals the effort to make SLICE N DICE intrinsically rewarding and attractive, to increase the likelihood of a participant making an effort despite the low stakes.	115
7.6	The sidebar menu and a Part 1 operation card under the two conditions.	118
7.7	The thumbnail condition also influences the design of the tooltip that appears when the user hovers their mouseover the thumbnail for longer than 1 second. The contents of the tooltip are the same as the operation cards in Part 1.	119
7.8	Screenshot from Part 1 in the TG/SG condition, complete with both thumbnail and subgoal graphics. The ¬SG condition would only have the labels.	120
7.9	After the autumn pilot study, a hint was added to Part 1 to curb excessively frustration.	121
7.10	The interface of Part 2 has three main areas: a sidebar housing the menu and documentation, an exercise description at the top, and a DCL coding widget at the lower half. The menu is labelled <i>Taxonomy</i>	122
7.11	In Part 3, the sidebar has three tabs that the user can move between. The first panel - which is opened by default - displays the subgoal labels along with the graphics and hints. The second panel (labelled <i>Taxonomy</i>) displays the menu, which in turn hyperlinks to the third panel, which will contain the documentation entry.	124

7.12	Each subgoal corresponds to one, but occasionally more, data wrangling operations. Comments in the coding panel guides the user toward decomposing their solution	125
7.13	Examples of the feedback messages. The first two messages are produced by DCL submission test library while the success message is hard-coded by me. It is intended to reinforce a concept learnt from the exercise and motivate the learner to proceed.	126
7.14	In Part 3, every subgoal is associated with three hints, which are accessed at the user's discretion. The first hint provides the relevant location in the menu, the second hint provides further guidance on how to adapt the documentation example, and the final hint provides the executable code. The layout here does not reflect the actual interface: hints are made available in sequence and take up the same area.	128
7.15	After the demographic survey, participants were given a short pretest to gauge their programming experience more precisely.	129
8.1	During the usability study, the correct answer was immediately given once an operation had been given to each subgoal and the solution submitted.	136
8.2	In the usability study, Part 2 concepts were introduced through explanatory cards rather than as a documentation entry in the sidebar.	144
8.3	During Part 2 in the usability study, a hint and solution were available. These were removed afterwards. Both the hint strip and solution tab were built-in features of DCL.	145
8.4	Note the customised error message and hint, but lack of hints underneath each subgoal	147
8.5	The self-rated prior experience in Python and R (1= <i>Not at all</i> , 2= <i>A little bit</i> , 3= <i>Beginner's level</i> , 4= <i>Intermediate</i> , 5= <i>Advanced</i>).	154
8.6	Results relating to operation card reading times in Part 1.	155
8.7	Results relating to time on task in Part 1.	155
8.8	Results relating to correctness scores in Part 1.	156
8.9	Results from five items in the evaluation survey, rated 1 (<i>Not at all</i>) to 5 (<i>Very much</i>).	157
8.10	Each horizontal bar represents the persistence of a participant, thus showing the attrition of participants over time. The vertical line shows when basic programming exercises turn into multi-command data wrangling exercises and is inclusive.	159
8.11	Results relating to time on task in Part 2/3.	160
8.12	Results relating to process metrics during Part 2/3.	160
8.13	Results from five items in the evaluation survey, rated 1 (<i>Not at all</i>) to 5 (<i>Very much</i>). Since Part 2 and 3 were merged into one, this survey refers to both.	161

9.1	A snippet from the background section. Note the minimalist style of the graphics, carefully designed to avoid priming the participant.	165
9.2	The first subgoal graphic in the first exercise in the survey.	166
9.3	The relative proportions of 0-3 ratings for each subgoal exercise.	167
9.4	The four different variants presented in the variant exploration question.	171
9.5	The two types of problems contained in the survey.	171
9.6	The prior experience of survey participants.	172
9.7	The correct operation-graphic mappings for the five matching problems. The order of options was random and operations verbally formulated to be as self-explanatory as possible.	173
9.8	The graphics that were most commonly misinterpreted.	174
9.9	The performance in interpretation problems, grouped by graphical type. <i>With</i> means with data, <i>Without</i> means without data.	175
9.10	The response distributions on a Likert scale of 1 (<i>Not helpful at all</i>) to 5 (<i>Very helpful</i>) asking them how clear they thought the graphics were, with and without data in them.	176
9.11	Which of 4 variants that the participant prefers in a textbook explanation and thumbnail graphic.	176
10.1	Although only 68% of Part 1 participants completed it in full, the ratio between experimental conditions did not appear to change markedly upon completion.	188
10.2	Tab changes and timeout events were only recorded for 30 participants, which is too few to base inferences on.	189
10.3	The TG group reads all cards 2 minutes faster on average, but this is not significant.	190
10.4	The TG group uses the tooltip only half as often as the control group, an effect that is significant.	191
10.5	The difference in time between the two thumbnail conditions is negligible.	192
10.6	The difference in time between the two subgoal conditions is negligible.	193
10.7	Data relating to a potential interaction in Part 1 time on task.	193
10.8	The \neg TG group commits 33% more errors, but it is not significant.	194
10.9	The \neg SG group commits 7 more errors, but it is not significant.	195
10.10	Data relating to a potential interaction in Part 1 number of attempts.	195
10.11	Subjective ratings of how helpful participants found various scaffolding elements of Part 1.	196
10.12	Evaluation survey items grouped by subgoal condition.	198
10.13	Evaluation survey items grouped by thumbnail condition.	198
10.14	Each vertical bar represents the number of people completing that exercise.	200
10.15	Thumbnail-related group differences in tendency to complete Part 3.	201

10.16	The SG condition is over-represented among participants who finish all 18 exercises.	202
10.17	The TG is slightly more inactive, but not significantly or meaningfully so.	203
10.18	There are no evident group difference, although the \neg SG condition appears more dispersed.	204
10.19	The TG is slightly lower in its tab changes, but not significantly or meaningfully so.	204
10.20	The SG is does slightly fewer tab changes, but not significantly or meaningfully so.	205
10.21	The TG is slightly faster, but not significantly or meaningfully so.	206
10.22	The SG is faster by 3 minutes, but not significantly or meaningfully so.	206
10.23	Data relating to a potential interaction in Part 3 total time on task.	207
10.24	Tooltip usage appears less common in the TG group, and this is significant for a permutation test, but not a Wilcoxon rank sum test.	208
10.25	Why did so few participants click on the menu in order to access syntax information? The left scatterplot, which shows hint usage in relation to menu clicks, suggests that they relied on hints instead. The scatterplot to the right shows it in relation to tab changes, for example reflecting googling of external resources.	209
10.26	The thumbnail condition clicks on the menu less often than the control condition, but only negligibly so.	209
10.27	The thumbnail condition commits fewer syntax errors, but not meaningfully or significantly so.	210
10.28	The thumbnail condition commits fewer syntax errors, but not meaningfully or significantly so.	211
10.29	The thumbnail condition commits <i>more</i> semantic errors, but this is not significant.	212
10.30	The graphical condition commits fewer semantic errors, but not significantly so.	212
10.31	Data relating to a potential interaction in Part 3 total semantic errors.	213
10.32	The thumbnail condition uses hints slightly more often, but this is not significant.	214
10.33	The subgoal condition uses hints <i>more</i> often than the control, but this is not significant.	214
10.34	The subgoal condition uses hints <i>more</i> often than the control, but this is not significant.	215
10.35	Subjective ratings of how helpful participants found various scaffolding elements of Part 3.	216
10.36	Evaluation survey items grouped by thumbnail condition.	218
10.37	Evaluation survey items grouped by subgoal condition.	218
11.1	Experience-related differences among people who began and completed Part 1	223
11.2	The degree programmes that participants were enrolled in.	223

- 11.3 The relative proportions of error types, among all Part 3 exceptions raised by errors programming in Python. 225
- 11.4 The crossed out coefficients indicate non-significant results ($\alpha=.05$) for Bonferroni-adjusted p -values. n depends on the variable, since entries of 0 are excluded. . . 227
- 11.5 Associations between parts in time on task, with lines of best fit obtained through linear regression. 228
- 11.6 Associations between parts in the number of semantic errors, with lines of best fit obtained through linear regression. 229
- 11.7 Boxplots showing the relationship between experience and performance, as measured by three different experience measures (their degree major, their pretest score, their self-rated knowledge) and four performance metrics (the number of syntax errors, the number of hints, the number of semantic errors, and the time on task, all in Part 3). 230
- 11.8 Box plots showing group differences associated with subgoal graphic condition. The box plots indicate medians and inter-quartile ranges. Each graph has been truncated along the x-axis, omitting between 1 and 6 outliers in order to display differences more clearly. 232
- 11.9 Box plots showing group differences associated with thumbnail graphic condition. The box plots indicate medians and inter-quartile ranges. Each graph has been truncated along the x-axis, omitting between 1 and 6 outliers in order to display differences more clearly. 233

Acknowledgements

A PhD is a strange and solitary way for a person to spend 4 years. It takes a stable, supportive village for it to be achievable even in the best of times, and I can imagine a hundred ways in which the Universe could have intervened to derail it. Therefore, I am above all grateful for the comfort, health and stability that the Universe afforded me and my family during these past 4 years, which the pandemic has reminded me never to take for granted.

I would like to thank Professor Quintin Cutts, my supervisor, for the opportunity he gave me in taking me on as his PhD-student, for the trust he gave me in allowing me to try out new research directions, and for his infectious enthusiasm that always kept my morale high. I would similarly like to thank my secondary supervisor, Dr Jeremy Singer, for the opportunities he gave me and the positivity he brought to every meeting.

I would like to thank my CCSE colleagues, for the laughter and commiserations we shared, especially Jack Parkinson, Ethel Tshukudu, Dr Matthew Barr, Dr Joseph Maguire, Derek Somerville, Elizabeth Cole, Peter Donaldson and Dr Steve Draper.

The research in this dissertation would not have been possible without the help of Dr Nouri Sakr, Dr Juho Leinonen, Dr Eric Yao, Dr Syed Waqar Nabi, Dr Junaid Akhtar, and Professor Briana Morrison.

Thank you to all the colleagues who buoyed my spirits during lockdown: Dr David Maxwell, Dr William Pettersson, Jessica Ryan, and Tom Wallis.

My secret weapon during this entire journey has been Xavier. Thank you for everything.

Declaration

With the exception of chapters 1, 2 and 3, which contain background material, all work in this thesis was carried out by the author unless otherwise explicitly stated. This includes the conception of all studies, the software development, and the data analysis.

Parts of the work documented in Chapter 5, 8 and 10 have previously been published or accepted for publication:

L. Sundin, N. Sakri, J. Leinonen, and Q. Cutts. “Facilitating API Lookup For Novices Learning Data Wrangling Using Thumbnail Graphics,” in *Foundations of Data Science (FoDS)*, 2021.

L. Sundin, N. Sakri, J. Leinonen, S. Aly, and Q. Cutts. “Visual recipes for slicing and dicing data: teaching data wrangling using subgoal graphics,” in *21st Koli Calling International Conference on Computing Education Research*, pp. 1-10, 2021.

L. Sundin and Q. Cutts. “Introducing Data Wrangling using Graphical Subgoals-Findings from an e-Learning Study,” in *Proceedings of the Eighth ACM Conference on Learning@ Scale*, pp. 267-270, 2021.

L. Sundin and Q. Cutts. “Is it feasible to teach query programming in three different languages in a single session? A study on a pattern-oriented tutorial and cheat sheets,” in *Proceedings of the 1st UK Ireland Computing Education Research Conference (UKICER)*, pp. 1-7, 2019.

Also published by the author and relevant to the topic, but not included in the dissertation:

Z. Wang, **L. Sundin**, D. Murray-Rust, and B. Bach. “Cheat sheets for data visualization techniques,” in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1-13, 2020.

Chapter 1

Introduction

Few research fields have remained unaffected by the volumes of Big Data generated in the past two decades. From geneticists studying micro-arrays, to meteorologists processing sensor data, and linguists analysing Twitter feeds - it is up to incoming scientists to find creative ways of merging and harnessing new data streams. There is one major bottleneck, however: these streams are often poorly or inconsistently structured. If university-level students of biology, meteorology and linguistics are to benefit from the size of modern data sets, they not only need to know how to analyse it, but also how to *wrangle* the data into a format fit for analysis. And they need to learn data wrangling sooner rather than later, most flexibly using programming, in an already crowded curriculum.

For the pedagogical community, this presents a formidable challenge: how can we optimally teach students programmatic data wrangling where dedicated time resources are scarce and prior programming experience minimal? More realistically, can we design techniques and tutorials that measurably increase the efficiency with which novices learn programmatic data wrangling? Just as importantly, can we improve the subjective experience of learning data wrangling, to make it more enjoyable and motivating?

That challenge forms the overarching research objective of the work presented in this dissertation. Because programmatic data wrangling represents relatively untrodden ground in terms of prior research, the present dissertation takes a breadth-first approach. It reviews research on an array of potential learner barriers inherent to the data wrangling process, before eventually homing in on three critical stages: plan composition, syntax lookup, and code example adaptation. For each stage, it proposes ideas for how they could be scaffolded, before evaluating them experimentally.

This search for scaffolding devices is guided by one key observation: data wrangling operations, which tend to involve manipulations of tabular data structures, lend themselves readily to **tabular graphics**. Tabular graphics are graphics that, with some degree of abstraction, convey the shape of a data structure before and after a data operation is applied to it, and highlights the change that took place. Examples of such graphics are shown in Figure 1.1.

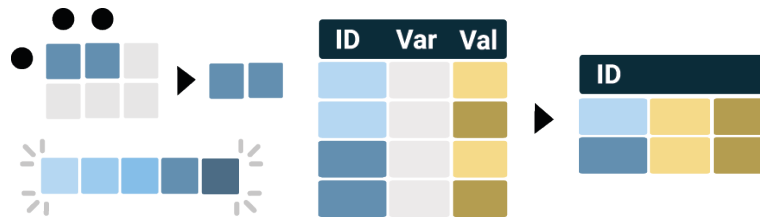


Figure 1.1: Examples of tabular graphics

The visualisability of data wrangling operations hands instructional designers an important lever to take advantage of. This could take the form of at least three possible interventions: **subgoal graphics** that visualise the necessary steps in a solution, **thumbnail graphics** that visualise the behaviour of API (*application programming interface*) commands, and **parameter graphics** that visualise the argument that should be passed as an actual parameter to a command. All three are explored and evaluated within this dissertation.

1.1 Motivation

In 2001, Purdue statistician Cleveland [12] proposed an action plan for expanding the foundations of statistics to also include computing, effectively reorganising statistics departments into a new field that he declared would be called *data science* [12, p.21]. Since then, commentators have questioned what exactly makes data science so different from classical statistics [13, 14], and whether it is a matter of re-branding or reflective of a more qualitative shift in practices.

The most noticeable change in practices is the increase in data *volume*. Where data historically were scarce and painstakingly acquired, in 2018 a DOMO report on data growth estimated that, in 2020, “1.7MB of data will be created every second for every person on earth” [15]. Another, less frequently discussed feature of data science lies in the *nature* of those data-generating mechanisms. Where previously data were often the result of carefully planned experiments, modern data streams tend to be by-products of business processes [13]: event logs accumulating on website back-ends, or social media corpora growing as an inadvertent side-effect of digital interactions. As a result, data analyses today are often ad hoc afterthoughts, which in turn means that the data schema may not be appropriately organised, or otherwise fit for consumption by statistical software.

This disorganisation of data implies that data wrangling is becoming an increasingly important skill to master for scientists and knowledge workers of all kinds. By “data wrangling” we mean the manipulation of array structures such as matrices and dataframes, usually in preparation for further analysis (we will refine this definition in Section 2.1). It also implies that, for a sociology major to make use of a crime data API in order to produce plots in Python, they will also have to learn how to wrangle the data. As one data science workshop instructor was quoted saying: “Most people I see have to learn to code in an absolute panic for their thesis” [16, p.5].

The educational research community must address this demand.

Even so, data wrangling is not an obvious choice for a research focus, especially considering the magnitude of other challenges faced by students, such as their eventual data analysis [17]. Why focus on the analysis' more menial, less glamorous precursor? And why focus specifically on the role graphics could play in facilitating it?

1.1.1 Why is data wrangling important to research?

Data wrangling is time-consuming. Among business intelligence professionals, the disorganisation of raw data formats and the “data janitor work” it incurs are recognised as major productivity bottlenecks [18]. Respondents in a 2016 survey of data scientists reported spending 60% of their time on data pre-processing tasks [19]. Another frequently cited estimate (but whose provenance is unknown) places this figure at 80% [20]. Regardless of its true percentage, data wrangling is generally perceived as a time sink that detracts from the more interesting activities downstream. It is therefore a matter of economic productivity to find ways of making the data wrangling process more efficient.

1.1.2 Why is data wrangling important to teach?

Currently, data wrangling is often ignored in introductory data science modules. For learners embarking on their first few research projects, this time sink is likely all the more frustrating, and risks discouraging learners from pursuing data science altogether [21]. Despite such risks, data wrangling is often ignored in introductory statistics courses [22, 23], which tend to provide pre-cleaned data [24]. While understandable and often justified when the focus is on statistical content, sanitised data present a misleading picture of actual research data [23], and may limit the ambition with which students can pursue their own projects using self-discovered data sets. Additionally, failing to teach data wrangling explicitly may increase data error rates in their research output [22].

Not all available data support the assertion that data wrangling is neglected. For example, Hardin et al. [25] surveyed seven data science syllabi from primarily the U.S., and found that all covered “data cleaning”. It is uncertain the extent to which this survey represents wider trends.

For many people, data wrangling is their first foray into programming. Data science skills are increasingly expected among early-career researchers regardless of their discipline, but many of them are tasked with learning it without formal training in computer science (CS) [16]. This presents a challenge, but also an opportunity, as data wrangling could be an entry-point into scientific computing. Array manipulation is the bread and butter of most scientific computing and a facility with transforming vectors, matrices, and dataframes is useful not only for *pre*-processing, but also for implementing the algorithms that constitute the eventual processing.

1.1.3 Why is data wrangling *education* important to research?

The number of potential beneficiaries is growing fast. The class of prospective learners far exceeds those majoring in CS and statistics. Enrolment statistics indicate that an increasing number of non-majors enrol in CS courses [26], often as a degree requirement [27], and survey data suggest their main goal is to apply programming to their own subject [28]. These non-majors may be undergraduates, from biology to linguistics, who neither expected to program as part of their degree, nor have an intrinsic motivation to do so [16]. These instructional challenges are explored further in Section 2.2. Although some findings from programming education research are likely to be transferable to non-majors learning data wrangling, those tightly coupled to conventional CS1 curricula (i.e. procedural or object-oriented programming) may be less applicable.

Data wrangling education is under-researched. 10 years ago, Kandel et al. referred to data wrangling as the “elephant in the room of data analysis”, playing on the fact that the labour invested in it usually goes unspoken. They also presented a list of data wrangling research challenges they hoped the community would accept [21, p.286]. None of these challenges, however, involved pedagogy. In the years since, major strides have been made in terms of API design [29], the development of analytics tools to support advanced data preparation (e.g. [30]), and end-to-end software for automated data wrangling (e.g. [31]). Meanwhile, the *pedagogical* literature on data wrangling remains fragmented, sparse, and mostly anecdotal. There is, in other words, a clear gap in the literature [32, 33].

Data wrangling is theoretically interesting. Data wrangling has multiple computational and conceptual properties that set it apart from the domains most programming education research is focused on. Computationally, it is vectorised, API-reliant, highly parameterised, and generally functional (see Section 2.4). Conceptually, it is tabular and consequently inherently visualisable in a way that for example imperative programming struggles to match. Exploring the pedagogical implications of this may carry insights over into other programming domains with shared commonalities, such as web development (also API-reliant) and relational database management (also tabular).

1.1.4 Why are graphics as a scaffolding technique interesting to research?

Focusing on tabular graphics may seem like an oddly specific aspect of a domain that so clearly needs to be investigated comprehensively. However, an insistence on evaluating concrete and reusable research products may increase the chances of actual adoption in other faculties. This is particularly important, since surveys suggest that adoption of CS educational innovations is often disappointingly low [34, 35]. Related to this concern, tabular graphics benefit from the

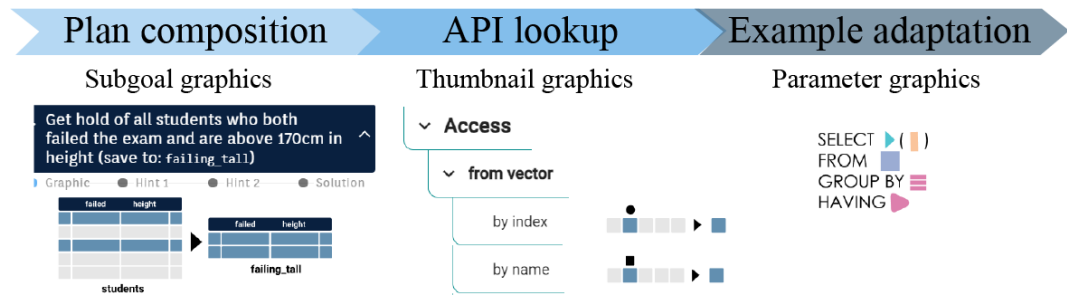


Figure 1.2: The research is focused on three aspects of the data wrangling workflow, each of which is targeted by a graphical scaffolding feature.

following:

Tabular graphics are low-cost. Tabular graphics could be easily drawn on a whiteboard, sketched on paper, and reused indefinitely. While creating polished vector graphics takes time, their production could in theory be automated. Given this low cost, if graphics produce at least a measurable improvement, they present a cost-effective intervention.

They have straightforward implications for IDE design. The interventions that are most likely to affect programming performance are those inherent to the programming workflow itself. The interface of the IDE (*integrated development environment*) is one such place where scaffolding features could be added unobtrusively. As will be demonstrated in this dissertation, graphics could be incorporated into documentation menus as thumbnails and code examples, and be used in more explicitly pedagogical features such as subgoal panels. In theory, open-source IDEs like Visual Studio, Jupyter notebooks and RStudio could be augmented with educational plugins that utilise tabular graphics.

1.2 Thesis statement and research questions

As hinted at in the introduction, this dissertation analyses the data wrangling workflow in terms of three distinct stages: plan composition, API lookup, and example adaptation. Three kinds of graphics have been developed to address each of these stages: subgoal graphics, thumbnail graphics, and parameter graphics, respectively. This framework is illustrated in Figure 1.2. Ultimately, the research goal is to evaluate whether the graphics improve data wrangling learning among novices, which is why our thesis statement is formulated as follows:

Subgoal graphics, thumbnail graphics, and parameter graphics facilitate the learning of programmatic data wrangling.

This thesis statement can be expanded into the following main research questions:

RQ1. What is the effect of subgoal graphics on the learning of data wrangling?

RQ2. What is the effect of thumbnail graphics on the learning of data wrangling?

RQ3. What is the effect of parameter graphics on the learning of data wrangling?

Each research question can in turn be expanded much further when we consider the full complexity of what “learning” and “effects” really mean. With regards to learning: the graphics will be evaluated by incorporating them into (primarily online) data wrangling tutorials, and from the participants’ behaviours we will measure a broad array of dependent variables that either reflect or mediate learning outcomes. Some of these are performance-related (e.g. the number of incorrect attempts and time on task), while others will be more indirect process proxies for learning efficiency (e.g. the number of API lookups). Still other metrics will be subjective in nature (e.g. self-reported motivation). In asking about “the effect on learning”, we mean the broader data pattern across these variables. For individual studies, these research questions will therefore be addressed through study-specific sub-questions and metric-specific hypotheses.

Implicit in the thesis statement is a claim of causation: we wish to ascertain that the provision of graphical aids *causes* positive learning outcomes. Any such claim is a tall order, especially in a complex domain like pedagogy, where learner preferences, prior experience, and other contextual factors all conceivably contribute to the observed effect. We are wise to moderate the premise. For example, the population of learners we will investigate are university students or end-user programmers with minimal experience in programming and/or data wrangling, who are motivated to learn data wrangling. The set of circumstances in which we will investigate graphics are mostly highly structured e-learning interventions, completed without human supervision.

1.3 Approach

The applicability of graphical aids to data wrangling could have been approached from a number of angles, each with its own set of methodological trade-offs. The chosen approach will be explained in greater depth in Chapter 4, but broadly it observes the following principles:

Breadth-first: Given how the current research does not fit snugly within a preexisting research programme, it seems appropriate to sketch out a few landmarks in the design space, instead of rendering a small corner of it in full detail, even though doing so trades off empirical reliability for theoretical range.

As mentioned, three different graphical interventions will be evaluated. These scaffolding features could be viewed as representing *conceptual replications* of a core thesis that tabular

graphics are helpful. Replications are considered conceptual when the manipulations and measurements involved are different. Their purpose is to establish the range, robustness and boundary conditions of a theoretical framework [36]. Conceptual replications can be contrasted with the more depth-first approach of *exact replications*, whose purpose is to eradicate false positives of a very specific phenomenon [36].

Experimental: Most of the undertaken research follows a *randomised controlled trial* (RCT) design that compares a graphical condition with a non-graphical condition. In principle, RCT designs isolate causal influences so that the effect size of graphics can be estimated without confounds. However, to give a more nuanced account of how graphics are used and perceived, subgoal graphics and thumbnail graphics will additionally be the subject of qualitative studies.

Reductionist: Educational RCTs often compare interventions that vary along so many dimensions that it becomes impossible to attribute observed effects to any particular “active ingredient” [37]. In order to make the causal attribution straightforward, the experimental manipulation is simple (the presence or absence of graphics), even though such a reductionist approach risks making the true effect smaller and harder to detect.

Opportunistic: Unless data collection is embedded within an existing course, educational data sets are high-priced commodities, as recruitment of participants has a price. It therefore makes sense for a researcher to minimise data waste and subject the data set to secondary analyses, even when they are not directly pertinent to the thesis statement. Any incidental findings judged to be of interest to the wider community will therefore be documented.

1.4 Thesis structure

Data wrangling education has yet to coalesce as a field, so to give its boundaries more definition, Chapter 2 provides a review of the terminology, academic contexts, and technologies involved. This is followed by a literature review in Chapter 3, which collates empirical evidence regarding the barriers novice data wranglers may face, how those barriers can be addressed, and the role graphical scaffolding could play in overcoming them. Following the literature review, Chapter 4 serves to explicate the methodological assumptions and priorities that underpinned the research.

Having thus described the challenge of learning data wrangling, and the *a priori* reasons for believing that graphics could help mitigate it, the dissertation proceeds with documenting a series of studies. The first of these, described in Chapter 5, is a pen-and-paper experiment that explores the effect of parameter graphics (i.e. RQ3) in the context of data wrangling *cheat sheets*. This study inspired a move towards larger-scale, e-learning based research, which is

why the remainder of the dissertation documents the work related to a large capstone study that simultaneously investigates the effect of both subgoal graphics and thumbnail graphics.

This capstone study was conducted using a specially built platform called SLICE N DICE. We dedicate Chapter 6 to an explanation of the scaffolding materials involved in the study, especially the design considerations behind the subgoal graphics and thumbnail graphics. We then describe the process and design justifications behind SLICE N DICE in Chapter 7. Following this, we present two small validation studies meant as a sanity check regarding the quality and interpretability of the graphics. We present the results from two pilot studies - one qualitative, one quantitative - in Chapter 8, before ultimately presenting the results from SLICE N DICE as they relate to research questions RQ1 and RQ2. This is then followed by Chapter 11, which contains more exploratory analyses of the SLICE N DICE, with less immediate pertinence to the research questions.

Finally, in Chapter 12, we zoom out to discuss the findings, implications, and limitations of the results, as well as how they fit into the continued research agenda of data wrangling education.

Chapter 2

Defining the scope

In this chapter, we will define the scope of the present research by refining our working definition of *data wrangling*, characterising the learner population, and justifying our choice of programming technologies to focus on.

2.1 Defining data wrangling

For a systematic field of research to evolve, a precondition is that a relative consensus exists on what terminology to use. The literature on what we will refer to as “data wrangling” is still splintered, with a proliferation of near-synonyms like *data munging*, *data preparation*, *data transformation*, *data manipulation*, *data cleaning*, *data pre-processing*, and *data management*. With such a wide selection of terms, why settle for *wrangling*, a word that literally means “to herd livestock”?

The term *data wrangling* appears to have been coined by Kimball [38] in 2008, who described the need to “lasso the data and get it under our control” and defined it as the steps between the operational source and the business intelligence interface. Kandel et al. [8] later adopted this term in 2011, naming their data transformation software WRANGLER. Soon after, McKinney [39], the original developer of Pandas (a key Python data wrangling library) followed suit, as did Wickham and Grolemund [2], developers of key data wrangling packages in R. In the years since, at least 8 books have been published that bear this term in their title. Although Google trend data suggest that *data wrangling* is not as widespread compared with more general terms like *data management* or *data cleaning* (see Figure 2.1), it is clear that *data wrangling* has momentum and is applied fairly consistently by stakeholders in data science.

2.1.1 What data wrangling includes

What the aforementioned terms have in common is their approximate location in the *data life cycle*. The workflow of a data analyst is often schematised as a cycle, starting with data acqui-

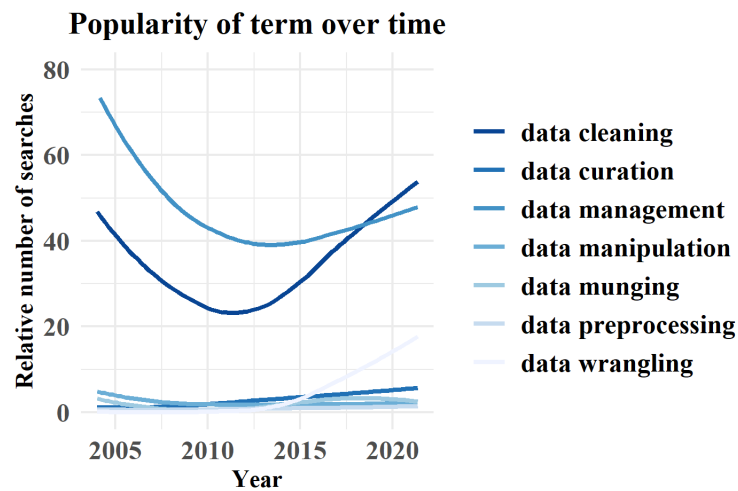


Figure 2.1: The relative search volumes of data wrangling-related Google queries, smoothed. *Data wrangling* is the lightest blue, on the rise in the bottom-right corner.

sition, pre-analysis preparations, then exploratory analysis, visualisation, inferential analyses, model training, before culminating in a decision. Multiple variants exist of this cycle, for example Wild and Pfannkuch’s [40] and Franklin et al.’s [41], but it is telling that neither of these conceptions contain data pre-processing, since at their time of their writing, the Big Data revolution was only in its infancy.

A more modern alternative is therefore Wickham and Grolemund’s life cycle [2, p.ix], which is focused on computational activities, ignoring stages like data acquisition and decision-making (see Figure 2.2). It describes *tidying* as the process of matching “the semantics of the data set with the way it is stored” and *transforming* as involving filtering, aggregating, and the derivation of new variables, among other things. *Data wrangling*, according to them, is the combination of these two.

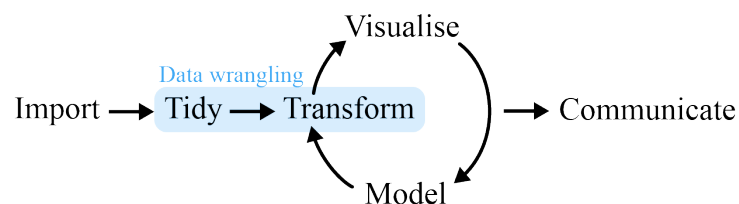


Figure 2.2: The computational data life cycle adapted from Wickham and Grolemund [2, p.ix]

Because Wickham and Grolemund (as well as Kandel et al. [8] and McKinney [39]) are mainly concerned with tabular data (e.g. vectors, matrices, dataframes), we will formulate the working definition as follows:

Data wrangling is the process of tidying and transforming tabular data

We will use this term mostly interchangeably with the other close synonyms, but are aware of some of their semantic baggage. As Horton et al. [42] have noted, *data manipulation* connotes

deliberate misuse of data, while *data munging* connotes destructive changes. *Data reformatting* overlooks that certain wrangling operations involve non-structural changes, such as the recoding of values, while *data transformation* can imply mathematical functions like rank or log transformations.

Sub-types of data wrangling

Data wrangling as a domain can be subdivided further. For example, Rattenbury et al. [43] distinguish between *intra-record structuring* and *inter-record structuring*. The former manipulate rows or columns on an individual basis (e.g. reordering columns, creating new columns by extracting values, or combining multiple columns into one). The latter manipulate multiple rows or columns at a time (e.g. filtering rows, aggregating rows, pivoting rows). We will be mainly, but not exclusively, concerned with the latter.

Another example is Morcos et al. distinction between *syntactic transformations*, which refer to format conversions that only require input and a closed formula (e.g. date format conversion), and *semantic transformations*, which depend on external reference information, such as currency conversion rates [44]. We concern ourselves exclusively with the former, but are aware that, in practice, semantic transformations feature prominently in data wrangling activities [45].

Other skills involved

In refining the definition of *data wrangling*, we wish to carve out a domain that is relatively distinctive in terms of the cognitive processes they draw upon. Programmatic array manipulation is one such relatively cohesive domain. As a skill, it involves what Baumer has called “the ability to think structurally about data and how to manipulate it” [25, p. 345] and a facility with turning real-world objectives into tabular data operations.

Practical intermediate-level data wrangling, meanwhile, is a highly heterogeneous skill: it requires a thorough understanding of different file formats (e.g. CSV, XML, JSON), date format conventions (e.g. POSIXct, POSIXlt), regular expressions, low-level data types (e.g. floating-point precision), and high-level data objects (e.g. the structure of Pandas `DataFrame`). Memorising these conventions is a cognitively very different task from the process of manipulating arrays. Since the former constitute declarative knowledge more than a procedural skill, we will focus on the latter and mostly set aside the issue of teaching technical, declarative facts.

2.1.2 What data wrangling is not

A complementary approach to defining data wrangling is to exclude activities that feel qualitatively different, and consign them to their own domain. Though real-world usages have a tendency to blur, we can differentiate data wrangling from the following:

Data cleaning: This tends to involve diagnosing errors (e.g. due to data entry, metric conversion), logic and consistency checks, outlier detection, and handling of missing data [22]. In general, these draw upon quite sophisticated statistical concepts, as well as a thorough understanding of the data acquisition context and problem domain. It is therefore far broader than we want data wrangling to be.

Data pre-processing: This term can similarly refer to data quality assurance [22], but is also used to refer to parts of the modelling process, such as feature selection and feature engineering (e.g. [46]). While these involve data wrangling, they also depend on data mining content knowledge, which this thesis is not concerned with.

Data management: This is probably the most comprehensive term, and is usually deployed in enterprise settings. Beyond data processing, it also covers governance, protection, and long-term preservation of data assets [47], which we will not concern ourselves with.

2.2 Learner characteristics and contexts

Ideally, the choice of scaffolding approach should take into consideration who the learners are, what motivations and experience they bring, and the contexts in which their training will happen. In data wrangling, such characterisations are complicated by the range of prospective learners. An interview study among data science teachers identified heterogeneity of student backgrounds, and their varying levels of programming skills and motivation, as a key instructional challenge [16]. Nevertheless, we may be able to make some general statements based on the academic context a learner is in: whether they are a data science major or non-major, and whether they learn it inside or outside of their formal education.

2.2.1 Majors and non-majors

Characteristics of data wranglers likely depend on their degree choice, since degree major constitutes a filter where self-selection biases come into play. Students whose major is not in CS, statistics, or related disciplines, are presumably less likely to have anticipated programming as being part of their degree, and therefore may perceive it as less relevant to their career. Interviews with data science teachers suggest that students are generally motivated to solve concrete data analytics problems, and consequently see programming as a tool rather than as an intrinsically enjoyable activity [16]. Not all psychologists or economists will have to program, for example: a psychology class contains both future therapists and future artificial intelligence researchers; an economics class both bank tellers and econometricians. Since programming modules are often a curricular requirement for majors [27], non-CS students may feel a lack of autonomy in doing them. Add to this the well-documented prevalence of programming anxiety

among non-majors [48, 49] and it becomes understandable that many instructors until recently have favoured point-and-click software over programming.

Compare this with majors of statistics or CS who presumably anticipated that their degree would involve programming, and at the very least self-selected on the basis of their confidence to learn it. They are also likely to be highly motivated [50] - either intrinsically, or by the promise of postgraduate employment opportunities - to learn programming. While exceptions to this exist, as evidenced by the drop-out rates in CS [51], these presumed characteristics imply that teachers can have a relatively light touch in their instructional approach for majors.

2.2.2 Curricular context

Data science is today taught in a variety of curricular configurations. Its Venn diagram quality (combining programming with statistics) leads to what Finzer has called the “data science dilemma” [52]: it could either be integrated within every STEM subject’s own department, or taught as a standalone, university-wide course, such as a CS1 module. The popularity of the latter option [27, 53] has led some instructors to re-design CS1 courses to be more centred around data analysis [54].

A full account of the advantages and disadvantages of either configuration is beyond the scope of this dissertation (see e.g. [54]), but it is worth comparing a typical, entry-level CS1 course with that of an introductory data science module. A CS1 course typically focuses on imperative programming, including control and iteration structures, with minimal reliance on libraries beyond the basic run-time packages. By contrast, a typical data science module begins with introducing rich libraries, and focuses on composing functional pipelines rather than algorithms [54]. Moreover, CS1 courses tend to begin with artificial, toy-like data sets, while data scientists often face authentic data sets [54].

The dilemma of how to meet the demand for data science skills among non-majors runs parallel to that of designing the curriculum of dedicated data science degrees. Much of current data science education research is focused on such degrees, which have grown rapidly in number over the last decade [55, 56], and whose entry-level course could serve as an introduction for majors and non-majors alike [57].

2.2.3 End-user programmers

Another common population of learners in programming-related research is the notion of an *end-user programmer*. It usually denotes someone who has not received professional training or who does not consider programming to be their primary occupation [58]. Nardi defines them as developers who program in order to support their own profession, as opposed to developing code for other end-users [59]. This may be STEM postdocs who attend local Software Carpentry workshops [16], mid-career professionals pivoting into Python, or self-taught *citizen data*

scientists [60]. If the number of online course offerings is any indication - from MOOCs hosted at Udemy or Coursera, to specialised platforms like DataCamp and DataQuest - the number of end-user data wranglers has grown dramatically in recent years.

2.3 Data wrangling technologies

While the term *data wrangling* is only barely in its teens [38], the actual practice of formatting tables has been around for much longer, and has been practised using a range of technologies. Charting their history, advantages and drawbacks gives us a better understanding of why courses are opting for teaching data wrangling programmatically, when other, more novice-friendly alternatives exist.

2.3.1 Relational databases & SQL

An important precursor to modern data wrangling APIs is Codd's relational model from 1969 [61], which proposed representing data in terms of tables (*relations*) and provided a set of operators for deriving new tables from input tables. Codd's model soon inspired SQL, a query language for relational database management systems. SQL's **SELECT** statements and support for table and schema modifications (e.g. insertions or deletions of tuples or columns) can all be said to constitute data wrangling, and are the most widely taught features of the language [62].

SQL remains a mainstay of data professionals - as of 2019, it was the second most popular data scientific software in US job advertisements [63]. Many data wrangling tasks can be accomplished within SQL (modern SQL implementations even support operations like pivoting), and SQL is often used to subset the data until it is small enough to be stored as a local file.

2.3.2 Spreadsheet software

Spreadsheet software been around since the late 1970s [64] and is the by far most widespread data wrangling technology, since it is included in basic software packages like Microsoft Office and G Suite. A 2015 O'Reilly survey found that more than 60% of data scientists use it routinely (via [63]). Spreadsheet interfaces allow data tables to be directly manipulated, with immediate feedback and little-to-no syntax. Owing to this live feedback loop, the barrier of entry to spreadsheets is low [65], making it a popular choice for introductory data science modules [66–68]. The liveness comes at a cost, however. Since the original data is not a persistent object (any manipulation modifies the original data) the workflow is not reproducible or easily audited [69], leading to a high risk of error [70].

2.3.3 Menu-driven software

While spreadsheets require the user to work within a grid that stores both data input and output, another group of software separates the two, and disallows direct manipulation. Like with spreadsheets, however, the primary mode of interaction is via menus, through which the user can access pre-defined functions and dialog windows to configure the functions further. Examples of such *menu-driven software* include statistical packages like SPSS and SAS, and special-purpose data wrangling systems like TRIFACTA [71].

SPSS is especially dominant, as the most popular data analytics software in social science courses [72,73] and academic research [63]. However, since 2010 SPSS' popularity in academia has been in decline [63], which, given the lack of serious menu-based competitors, seems to reflect a drift towards programming-based alternatives.

2.3.4 Automated data wrangling

For as long as the user retains fine-grained control over it, data wrangling is likely to remain time-consuming [74]. Within the last 10 years, this has spurred the development of automatic end-to-end data wrangling tools. These lines of research generally involve *programming by example*, where the program is synthesised based on user-provided input-output examples [75]. So far, these have primarily been successful for string and number processing (such as changing phone number format) [76–78] and data extraction [79], but also for table layout transformations [80–82]. One such feature, called FLASHFILL, has been integrated into Microsoft Excel [83].

2.4 Programming-based software

Given the power and promise of existing and future data wrangling technologies, why should students be taught to do it programmatically? The most versatile and expressive way of specifying a data wrangling procedure will always be to script it, using languages like Python or R. Scripting affords reproducibility [69] and, assuming the programming language in question is popular enough, the ability to leverage libraries authored by others. Programming languages also tend to be free to use (an important exception being Matlab) and far more attractive on the job market: a 2019 analysis of US job advertisements found Python to be the most popular data scientific software, followed by SQL, Java, and R [63]. While these were all mentioned more than 10000 times, SPSS was mentioned less than 3000 times. Moreover, compared with two years prior, Python more than doubled while R grew by 50% [63].

University departments are beginning to take note. While there is a general lack of high-quality data regarding the trends in programming-based versus menu-driven tools in introductory university courses, a 2019 survey of psychology courses in Canada reports that 19% of intermediate courses, and 40% of graduate courses already use programming [73]. A com-

parable study of psychology courses 20 years earlier found that the software used was almost completely menu-driven [72]. Combined with the numerous papers calling for a transition into programming for non-CS majors (e.g. [84–92]), we may very well be approaching a watershed moment in the technology an introductory statistics course is expected to teach.

2.4.1 Language characteristics

In this dissertation, we have chosen to focus on R and Python, since they are the most popular data scientific programming languages today [63], though other popular languages do exist (e.g. Matlab, Julia). The data scientific APIs of these languages have a number of distinctive features:

Interpreted: The languages tend to be *scripting languages*, which means that they (usually) are interpreted at run-time rather than compiled. This permits dynamic typing and interactive shell programming, in which the user can write a command and immediately execute it. This is known as a *read–eval–print loop* (REPL), and allows a data wrangler to inspect the result of a data transformation interactively.

Vectorised: *Vectorisation* refers to the process of operating on an array as a whole, instead of using iteration structures. This method allows the programmer to effectively parallelise the computation. This stands in contrast to the imperative programming of most CS1 syllabi [93], which instead uses loops.

Incidentally, whether a student learns loops or vectorised computations probably influences their ease of learning the other: Nolan and Temple report that loop-first students in their course tended to ignore vectorised solutions [25] while many of Patitsas’ [94] vector-first students “expressed a distaste” for loops, being already accustomed to the convenience of array manipulations.

Functional: While the languages themselves are not purely functional, the key data science libraries tend to rely on a wide set of pure functions that accept a data structure as argument (either implicitly or explicitly) and return a new one. They also tend to include higher-order functions [95]. Pieces of functionality that for a CS1 procedural programmer would be solved by an iteration-based pattern would for a data wrangler be solved using special-purpose, vectorised functions. For example, “Does an array contain a 0?” could be solved using a loop, but a data wrangler is more likely to use a function like `any()`.

Chainable syntax : Key libraries tend to support ways of chaining together functions involving the same object into a pipeline. This could work by way of method cascading or specialised pipeline operators, and makes functions structurally easy to combine, without the need for intermediate variables.

Highly parameterised : Because data scientific functions are often highly abstract, they need to be parameterised [96]. This leads to a complex function interface, with numerous and often opaque parameters, that may be either necessary or optional depending on other parameters. For example, the R function `reshape()`, for pivoting dataframes, takes 12 arguments, while the Pandas equivalent `pivot_table()` accepts 10. Understanding parameters often requires a low-level understanding of the data structures. For example, to flatten a NumPy array using `flatten()`, they could configure the `order` parameter as either `'C'` or `'F'`, representing C-style (row-major) or Fortran-style (column-major) order, respectively - explanations that likely are of scant help to novices.

Heavy package reliance: Although R has vector structures in its basic run-time library, in general a data analyst is likely to involve external APIs whose exact syntax they need to look up on the fly. This is captured in an observation by Patitsas, who introduced a data science API in her CS1 course before conditionals and loops: “from early on, students got in the habit of looking up built-in functions rather than reinventing functions from scratch” [94, p.333]. This implies that a data wrangler needs to use a much larger set of primitives. Memorising this constitutes either a significant upfront investment of effort, which is why details tend to be looked up on an as-necessary basis.

Of course, a programmer *could* get by through a smaller set of functions. APIs tend to have a long tail of rarely used (and possibly under-utilised) convenience functions, with a few functions representing as much as 90% of all method calls [97]. Indeed, the creator of the data wrangling package `dplyr` has claimed that its key *verbs* (see Section 2.4.2) allow the user to solve 95% of all data manipulation problems [98, p.203].

2.4.2 R

R is an open source programming language for scientific computing, created in 1991 by Gentleman and Ihaka [99] at the University of Auckland and today maintained by the R Core Team. Within academia, the popularity of R has increased steadily for the past 20 years, though its growth is beginning to plateau [63]. It has a very active user-developer community, and a wealth of well-vetted, user-contributed packages. Although R is normally used through scripting in the open source IDE RStudio, point-and-click interfaces are also available [100], as are computational notebooks [101]. The standard packages of R come with a large number of statistical functions and data structures, among them `vector`, `matrix`, which are both homogeneous arrays, and `data.frame`, which stores mixed-type data in equal-length columns.

Tidyverse

One collection of packages, called Tidyverse [29], has had particularly notable influence in popularising R. The packages in this collection share a set of design principles, inspired by the

UNIX command line [102], which stipulates that functions should return the same kind of data structure as they accept, allowing data to be piped from one function to the next [102]. Functions generally serve a well-defined class of problems, are highly configurable, and usually has a name in the form of a verb (e.g. `select`, `mutate`). The core data wrangling libraries are `dplyr` [103] and `tidyr` [104], which together form what they call a “grammar of data manipulation”. An example of its syntax is shown below, in which two columns (A,B) are selected from a dataframe and grouped by A, the B column is averaged, and the resultant mean values filtered based on whether they exceed 3.

```
df %>%  
  select (A, B) %>%  
  group_by (A) %>%  
  summarise (mean=mean (B) ) %>%  
  filter (mean>3)
```

R learnability

Numerous educators - primarily in psychology and biology - have published calls for transitioning from spreadsheets and SPSS into R [89–92]. Some have even called for it to be introduced in high schools [105]. Experience reports from introductory statistics classes that use R suggest it can be done with high levels of student satisfaction, despite the steeper learning curve [106].

A small but growing literature exists for empirically evaluating R’s learnability. For example, Stemock, et al. split STEM majors into an R and SPSS group and found that the R group consistently earned higher grades in statistics assessments, though not at a significant level [107]. Counsell and Cribbie [108] surveyed psychology undergraduate and postgraduate attitudes toward R and found that students, on average, viewed the language positively, but that students who were low in motivation found it too difficult. Another study directly compared students’ interpreter errors when using Base R with using Tidyverse, but found no significant differences in relative frequencies [109].

2.4.3 Python

Python is an open-source, general-purpose programming language designed by Van Rossum in the early 1990s [110]. The language supports multiple paradigms, but is generally considered object-oriented at its core [110]. For data wrangling purposes, the most common libraries are NumPy and Pandas, which are frequently used as part of a computational notebook, such as Jupyter [111].

NumPy: (NUMerical PYthon) is the foundational array programming API in Python. It emerged in 2005, based on an earlier array-based Python package [112]. The library centres

on a data structure known as the NumPy array, often referred to as an `ndarray` (as in *n*-dimensional). These arrays are technically tensors, in that they can support any number of dimensions, and are homogeneously typed.

Pandas: (PANel DAta) is a library for handling heterogeneous datasets through a structure called `DataFrame`, which is explicitly inspired by its R counterpart [113]. It also provides a one-dimensional structure called `Series`, which unlike `ndarray` objects allow string indices. An example code snippet, equivalent to then one seen in R, is shown below:

```
df[['A', 'B']] \
  .groupby('A')['B'] \
  .agg(mean=('B', 'mean')) \
  .query("mean>3")
```

Python learnability

Although many calls for teaching Python in introductory statistics or research methods modules have been published (e.g. [84–88]) little research has been conducted on the Python data science stack. Patitsas [94] published a report detailing her experience teaching a NumPy-first course for non-majors where NumPy was introduced before conditionals and loops. No formal data collection was reported, but Patitsas observed that, assuming the students were familiar with linear algebra, NumPy in some ways behaved more intuitively than basic Python. For example, `+` adds two arrays together in NumPy, but concatenates two lists in basic Python.

2.4.4 Summary

- We use *data wrangling* to refer to the process of tidying and transforming tabular data. For the sake of cohesion, we will exclude statistical analysis, data visualisation and more declarative knowledge (e.g. file formats) from this local definition.
- Data science modules today are often open to both majors (in data science or CS) and non-majors, and the two groups are likely to differ in the motivation and anxiety they feel towards learning programmatic data wrangling. Another important learner population is end-user programmers outside or within academia.
- Data wrangling has its origins in SQL, which remains popular, alongside spreadsheet software and menu-driven software like SPSS. Programmatic data wrangling is increasing in popularity, however, and associated APIs have a number of characteristics that are pedagogically relevant: they tend to be interpreted, vectorised, functional, chainable, and highly parameterised. R and Python are the most popular programming languages for data science, and are the ones we will focus on.

Chapter 3

Literature review

The broader question that this dissertation seeks to answer is whether tabular graphics can facilitate novices' introduction to data wrangling. The question can be resolved into two parts: the real-world problem we seek to address (i.e. novices' introduction to data wrangling) and the potential solution we seek to explore and evaluate (i.e. tabular graphics).

To evaluate graphics as a solution, our first step must be understanding the real-world problem, by reviewing literature on the *conceptual* and *programming-related* barriers faced by data wrangling novices (these are reviewed in Section 3.1 and 3.2, respectively). “Programming barriers” is an expansive topic at immediate risk of scope creep, but we will limit our discussion to barriers that are characteristic of (if not necessarily unique to) data wrangling, and organise our discussion in terms of five programming activities: *problem comprehension*, *plan composition*, *API lookup*, *example adaptation* and *debugging*.

Following this, in Section 3.3, we will review literature pertaining to solutions. It will begin by outlining general learning theories on cognitive load and scaffolding, which provide frameworks for predicting which factors exacerbate learners' cognitive effort, which design principles could reduce it, as well as the numerous trade-offs that come into play in instructional design. These frameworks will then be brought to bear on a review of how graphics could serve a scaffolding role. We focus on graphics not because they form the only strategy worth investigating, but because they - as will be argued - present particularly *low-hanging fruit*.

These two strands of literature - data wrangling barriers and general scaffolding theory - will then intersect in Section 3.4, where we catalogue previous examples of how graphics have played a scaffolding role in data wrangling workflows. Eventually, in Section 3.5, we will revisit the programming barriers of Section 3.2 and offer concrete proposals for how three stages - plan composition, API lookup and example adaptation - could be graphically scaffolded. Finally, these proposals home in on three intervention ideas - subgoal graphics, thumbnail graphics, and parameter graphics - which form the focal points of our research.

The current state of the field

Over the course of our review, we discovered that the literature on data wrangling pedagogy is small but nascent. Only a small sliver of data wrangling research is experimental - most of it takes the form of experience reports. The most relevant work has been published in the last four years (e.g. [24, 109, 114]). This is partially explained by the youth of the technologies themselves: the key data wrangling APIs (i.e. NumPy, Pandas, Tidyverse) came about after 2005 and became dominant much later. Another factor is the lack of dedicated venues. Although the number of journals and conferences interested in data science education is on the rise, many of them focus on data science majors, similar to how CS education journals focus on CS majors, and how a programming educator in biology may prefer to publish in a biology education journal. Due to this degree-aligned publishing landscape, novice-centred data wrangling pedagogy as a field has yet to unify its terminology and priorities into a coherent programme. Since no single search string is yet capable of efficiently capturing it, our review method has mainly relied on backward snowballing (i.e. tracing citations backwards) and recommendation engines.

To provide a more coherent picture, we will also review findings from neighbouring domains on the assumption that they will generalise. For example, the CS education community, though mostly focused on imperative programming [93], has a rich empirical and theoretical literature on how novices learn programming, the barriers they encounter when practising it, and which scaffolding interventions could help learners overcome them. The Human-Computer Interaction (HCI) literature is meanwhile dense with innovations aimed at boosting the productivity of end-user programmers working with complex APIs. The learner populations that these two domains are concerned with - CS students and experienced end-user programmers - do not perfectly overlap with novice data wranglers. However, the core information-processing barriers they are confronted with, and the solutions for surmounting them, are likely to be at least analogous to each other.

3.1 Conceptual barriers

If data wrangling is defined as the manipulation of tabular data structures, then a good conceptual grasp of such structures is an important prerequisite for it. According to Thayer et al.'s theory of *robust API knowledge* [115], effective API usage requires an understanding of *domain concepts*, i.e. the entities an API attempts to model. Understanding such concepts is believed to help programmers imagine what is possible in the API, and provides a basis for determining which API commands may be relevant to their goals [115, 8:6]. We will therefore begin by reviewing conceptual aspects of data wrangling that may pose difficulties for the learner, independently of the programming-related barriers.

In the case of data science, the main API entities are vectors, matrices, and dataframes, which in turn are abstractions of real-world entities. Understanding these data structures is not a

trivial matter. Even though two-dimensional tables are pervasive in everyday life, and have been used since ancient Babylonia [116], their structural principles are often left implicit. As Konold et al. have noted, research into the comprehension of data tables has been sparse, as “tables are so ubiquitous we tend to take them for granted” [117, p. 192]. In the sections that follow, we explicate why and how exactly that learners may struggle conceptually with manipulating tables.

3.1.1 Challenges with dataframes

Many scientific disciplines are primarily concerned with collections of heterogeneous data, in which each sample is measured along multiple variables. These datasets tend to be represented through dataframes (`DataFrame` in Pandas, `data.frame` in R). To standardise the way data is formatted within such structures, Wickham introduced the notion of *tidy data*, a format where each column represents one variable (attribute), each row one unit of observation (e.g. participant, trial, case), and where each value has its own cell¹ [119]. Tidy data could be understood as the desired output of most data wrangling processes, precisely because modern data science APIs have evolved to expect them. However, real-life tables - such as tables of contents, nutritional content labels, or even most spreadsheets - are meant for human consumption, not API consumption, and the informality in the day-to-day usage of the word “table” means that these principles may not be instinctively obvious.

Real-life tables are often in violation of tidy principles. For example: columns may represent conjoined attributes. If a column called *Name* stores first names concatenated with last names, you cannot easily sort by last name unless you first separate the two. Word processors and spreadsheets allow you to split and merge cells to create hierarchical tables, but this would violate the “one value per cell” principle. In day-to-day calculations, as well as many display tables, we store sums of each column at the margin, for example in a bottom row. That would be in violation of “each row represents one case” principle. Illustrations of these violations are shown in Figure 3.1.

Multiple studies suggest that these tabular conventions do not come naturally to us, and require some degree of explicit training. Falbel and Hancock [120] found in a data entry task that pre-teen students struggled to create a table in the expected format. The correct data table would have listed a set of humans and put their gender in a *Gender* column, but the children instead placed girls and boys in separate columns. The authors proposed that students are more intuitively drawn to *set-based* structures that explicitly split up discernible groups, such as a hierarchy, as opposed to a *property-based* structure, like a table.

Another study, by Konold et al. [117], asked a group of middle school, high school, college students and teachers to create “an organised record” (without specifying format) of the infor-

¹A more general and formal formulation of this principle states that each row should represent a unique relational mapping between values of independent variables to values of dependent variables [118, p.148].

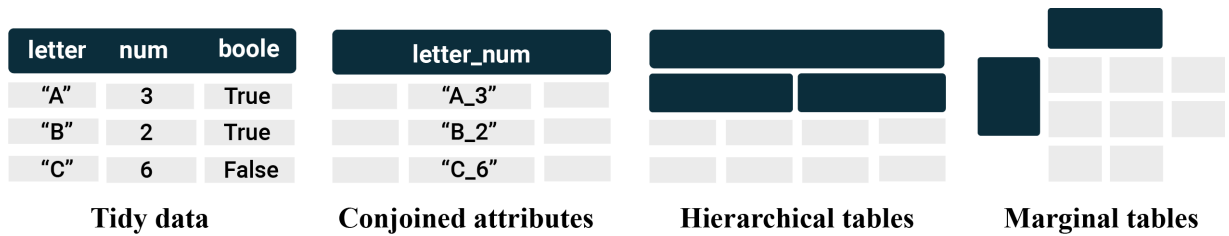


Figure 3.1: Illustration of the tidy data format and 3 common violations of it.

mation displayed within an image. They found that only 26% represented the data as a single flat table, while 58% created multiple flat tables, splitting the total data set based on a grouping variable. 14% - all by adults - created hierarchical tables. Overall, the likelihood of choosing a table increased with educational level. This suggests that, even despite extensive schooling, hierarchical data representations may come more naturally to us than tidy data.

Common table formats

For a sense of just how unintuitive flat tables can be, consider the notion of a *case*, *unit of observation* or *unit of analysis*. Konold et al. defines this as “the physical record of one repetition of a repeatable observational process” [117, p.194], but beyond the more archetypal examples of cases - individuals, physical objects, die throws - many units are either superordinate or subordinate to the level of abstraction we normally think in terms of. For example, when mice from different breeds have their weight measured weekly, there is an implicit three-level structure to that data: breed, mouse, and week. Out of these, mouse is likely to be the most natural unit of analysis, since it is what cognitive psychologists refer to as a *basic-level category*: it has high levels of within-category similarity and low levels of between-category similarity, which makes it perceptually more salient [121, 122]. If the data is then presented on a “one week per row” format, this may throw learners off, though research admittedly is scant. At least one case study, featuring a 7th-grade student, found that the student struggled with comprehending a graph in which points represented group-level attributes like maximum values [123].

Wide data. Time series data, or repeated measures data, are usually represented using a *wide format*, where each column indicates the measured value (e.g. height) at a unique time point, and each row refers to the system undergoing change over time (e.g. a child). This allows the user to conveniently follow the progression of values. From a processing point of view, however, it is problematic: if you wish to *aggregate* all measurements, it would require you to select all the relevant columns.

Long data. An alternative format is known as *long format*. It effectively lowers the level of observation from object to time point. The time point is *pivoted* from being a column name, to the value of a *time point* column, and a child is no longer represented through a single row,

but through a group of rows sharing the same value in a child identifier column (e.g. *Child*). Other examples of where long format may be counter-intuitive include *shapefiles*, where each row may represent an edge between two vertices and groups of rows together define a polygon, and network datasets that similarly represent edges as rows.

Nested data. Since the primary data exchange format of the modern web is JSON, many system developers choose databases that store data as JSON documents rather than relational tables. JSON is therefore an increasingly common data format that novice data scientists should know about. JSON is explicitly hierarchical, with arrays and objects (sets of attributes) nested inside other arrays/objects. Unlike CSV files, this means that data cannot be read in a way that immediately adheres to the tidy data condition of one value per cell. Instead dataframes are usually nested inside other dataframes, and need to be *unnested* into separate objects.

3.1.2 Challenges with split-apply-combine

Multi-level data, meaning data featuring categorical variables, are frequently subjected to aggregation operations that reduce many values into one. Tables can hold both raw data (*case data tables*) and aggregates (*summary tables*), and indeed one of the most common patterns in data wrangling involves transforming the former into the latter. This pattern is often known as *split-apply-combine* [124] and consists of three steps:

1. **Split** the rows into groups based on a categorical variable
2. **Apply** an aggregate function to one or multiple columns for each group of rows
3. **Combine** the aggregates into a summary table

This, when combined with table joining and row filtering, is also the general form of SQL **SELECT** queries, which the set of data wrangling-related transformations could be considered a rich superset of. For students who encounter data wrangling as part of an introductory statistics course, SQL-query equivalents in R or Python are likely to make up a large portion of their activities.

SQL operations on their own may not be *conceptually* that complex. Children as young as 5 years old have been observed to apply data wrangling operations like grouping, sorting, or row filtering in unplugged activities [125], and school children aged 11-13 have been observed to spontaneously use them in order to enhance data visualisations [126]. However, when represented through textual syntax, as they are in SQL, query writing is prone to numerous semantic errors [127]. Split-apply-combine operations can be far from trivial: they can involve multiple grouping variables, multiple aggregation functions, and implicit groupings that the learner may not realise. A recent bug analysis study of SQL queries by Taipalus found that semantic errors

involving aggregations tended to be associated with a lack of grouping, which the author attributes to a lack of understanding when grouping is necessary [128]. Similar results were found by Reisner [129] and Ahadi et al. [130].

3.1.3 Challenges with vectors and matrices

Although a social scientist's research is likely to be dominated by dataframes, since they mainly deal with scalar measures, they will inevitably also deal with vectors, if only because dataframe columns are usually made up of vectors. For budding neuroscientists, physicists or geographers, matrices are likely just as important a data structure to be familiar with, primarily as a way of storing inherently two-dimensional data (e.g. brain imaging, geographic data). While data wrangling should be considered separate from linear algebra (and therefore does not cover algebraic concepts like matrix multiplication), even a novice data wrangler is sure to come across matrices, for example as square matrices that store summary statistics for different variable combinations (e.g. correlation and co-variance matrices).

For something so ubiquitous in science and statistics, remarkably little attention has been given to students' conception of what vectors and matrices are and what they are used for. Worral has suggested that students tend to think of matrices as "nothing more than abstract rows and columns" [131, p.46], rather than as a representation of something in the real world. Moreover, the operations for dealing with vectors and matrices are generally different from those of a dataframe, in how they require a facility with indexing (dataframes are generally accessed through Boolean conditions). Data wrangling APIs tend to support a variety of different methods for indexing data, including numerical index, alphanumeric names, conditions or explicit Boolean masks. Murphy and Williams [132] suggest that students tend to mistake the index of elements with their value, and in their exam data found that their students struggled with combining array operations into coherent solutions.

3.1.4 Section summary

- The conventions of how data tables are formatted are not intuitively obvious, especially when it comes to grouping variables, multi-level data, long versus wide data, and raw versus aggregated data. These concepts and conventions need to be explicitly taught and not taken for granted.
- Although data wrangling operations may be simple enough for a child to understand, identifying relevant operations and combining them into a typical split-apply-combine pattern is non-trivial, as shown by semantic error studies from learners of SQL.
- Novice comprehension of vectors and matrices has received little prior research attention.

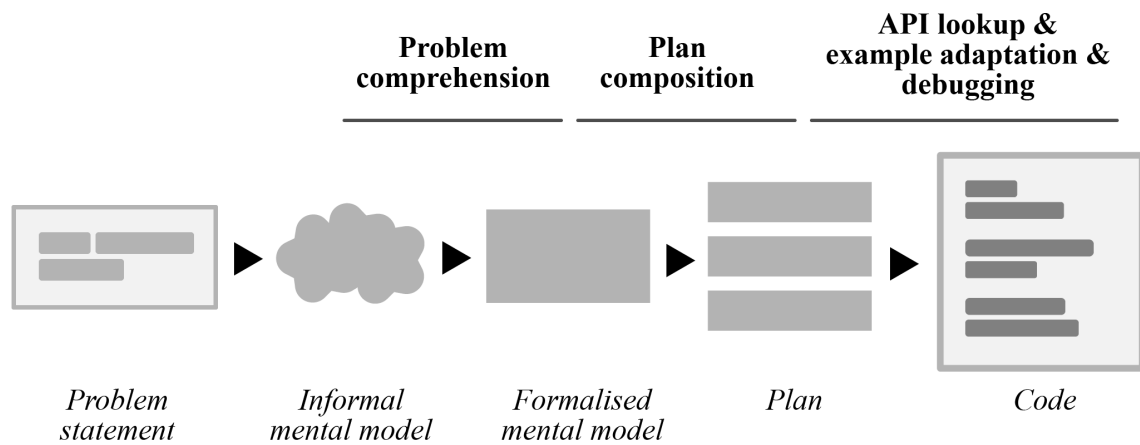


Figure 3.2: Data wrangling as a process consists of several distinct tasks that convert a problem statement into executable code. *API lookup* includes mental recall. *Example adaptation* and *debugging* should be interpreted as probable rather than logically necessary activities.

3.2 Programming-related barriers

Since our goal is to facilitate the *process* of data wrangling, our first step will be to tease apart the various sub-tasks and intermediate representations that this process involves. Each such task could be conceived of as a potential barrier, or pain point, and therefore a locus for potential scaffolding interventions.

The process model we will use mainly serves an organisational role, of unifying facts into a coherent framework. In other words, it is not a falsifiable assertion, but merely an analysis of the logical steps that data wrangling typically involves. According to this model, shown in Figure 3.2, the programmer begins with *comprehending* the problem by translating an informal problem description into a more formal mental representation. This representation is then used to *plan* a solution in terms of a sequence of data operations. For each such step, the programmer *looks up* the API command for implementing it, by either searching through external sources or a mental cache to find a code example and/or documentation, which the programmer subsequently *adapts* to integrate the command into their code solution, a process that likely involves *debugging*. As noted in Section 3.2.2, the process is realistically not strictly top-down, but also involves bottom-up, code-driven elements. Each step of the model will be analysed in terms of the barriers it risks introducing, as suggested by earlier literature.

3.2.1 Problem comprehension

The first step of a data wrangling problem is, as with any problem, to comprehend it. What does a data wrangling problem look like? A data wrangling task ultimately involves mapping an input data format to an output data format. In its barest form, a data wrangling problem simply supplies an initial schema and an expected schema (e.g. the schema expected by a

bar chart library function), requiring the learner to compare the pre-wrangling state with the post-wrangling state and then reverse-engineer their own problem statement (e.g. “separate the *Name* column into *Forename* and *Surname*”). This may constitute the most authentic problem format, since in actual practice, problem statements rarely come served on a platter, but are rather something the data analyst need to formulate themselves. However, in educational settings, it is usually convenient to specify a particular format implicitly, through some kind of natural language prompt, effectively turning it into a *word problem*.

Suppose the word problem were the following and that the data set stores income on a citizen level (see Figure 3.3):

How much richer (in median terms) is the wealthiest region compared with the poorest region?

Whether this constitutes a data wrangling problem is debatable, since the difference in income is a concrete, meaningful answer rather than a reformatted data set, but since the solution involves manipulations of tabular data, it offers a convenient stand-in for a more realistic reformatting problem. Regardless, what does it *mean* to “comprehend” the problem in this context?

Without an accurate interpretation of the problem prompt, the search for solutions may be too vague or misdirected to succeed, and once a candidate solution is found, its correctness cannot be verified. Multiple studies from the CS education literature suggest that problem misinterpretation is a common source of semantic errors and that students may be reluctant to reconsider their interpretation of a problem unless prompted to do so [133, 134]. The parsing of word problems, and the presentational aspects that influence it, have been extensively studied in mathematics [135], but less so in programming.

According to Nathan et al.’s theory of word problem comprehension in algebra, comprehension involves establishing a correspondence between the formal system (a *quantitative* mental model) and the learner’s informal understanding of the problem domain (their *qualitative* model) [135, p.330]. For example, “wealthiest” becomes formalised as “has the highest median income” and “how much richer” becomes formalised as “calculate the difference”, transforming the problem into the following:

What is the difference between the highest regional income median and the lowest?

Ultimately, this formalisation needs to connect with the schema of the provided data set. This is a key point of Reisner’s 1977 model of database query writing [129], which states that the user begins with looking for clues in the problem sentence for translating natural language words into data schema labels, for example mapping the regional-level “income” to the citizen-level `income` column of the dataset (see Figure 3.3).

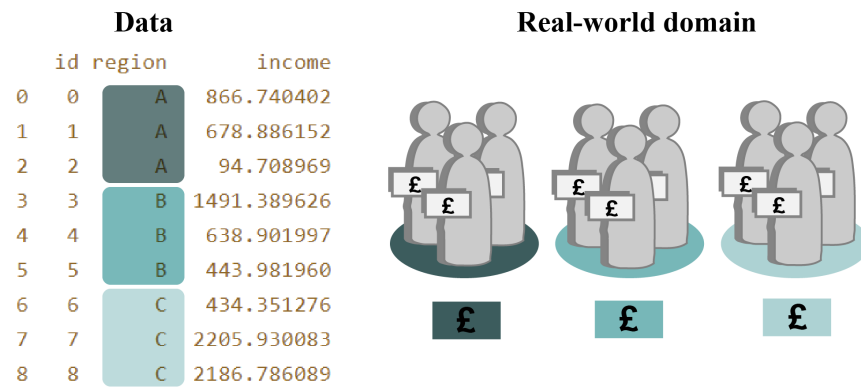


Figure 3.3: Word problems require the data analyst to draw upon domain knowledge in order to interpret what the problem asks for.

Domain knowledge barriers

The comprehension of a word problem generally draws upon informal real-world knowledge, which could be as basic as “region-level income is determined by the income of its citizens”, but could likewise involve terms or quantities that the programmer does not know. In data wrangling, this is a common occurrence: Morcos et al. define *semantic transformations* as data transformations that depend on external reference information [44], which could be everything from currency conversion rates to the exact formula for a metric.

Even absent such unknowns, simply the addition of a detailed domain description could interfere with problem-solving. Evidence for such a *problem description effect* is inconclusive however. Bouvier, et al. conducted a multi-institutional study that compared performance in a context-rich versus context-free version of the same programming problem, but did not detect a statistical difference [136], a result that was later replicated in a follow-up study [137]. Craig et al. compared performances on five different problems involving an already familiar context [138], with inconclusive results. Leinonen et al. [139] conducted a similarly comparative study but with more mathematically oriented problems. They found that contextual versions were easier, but generally produced longer completion time.

Incongruity barriers

Independently of the amount of context provided, linguistic aspects of the problem sentence, including both its structure and content, can influence the facility with which the problem is solved. This has been studied in the context of database query writing. Reisner’s model predicts that if the phrasing is lexically or structurally incongruous with the actual schema (e.g. asks for *surname* when the database column is called *family name*, or *birthday* when the column is called *dob*), then this will impede the problem-solving [129]. We find recent evidence for this in a 2021 study by Miedema et al. [127], which reported that students of SQL “struggled

significantly” when the data demand did not easily match the database schema.

Reisner’s own data also indicated a reliance on superficial clues in comprehending the conditions that the results should satisfy. For example, the error rate increased in exercises whose formulation mapped awkwardly onto programming syntax (e.g. “all pugs and poodles” would formally become `breed='pug' OR breed='poodle'`) [129]. Conversely, if the phrasing contains explicit clues of the syntactic roles of various elements, the error rate decreased. For example, if the problem asks for “the average weight of dogs, grouped by breed” that is more structurally and lexically analogous to `SELECT AVG (weight) FROM Dogs GROUP BY breed` than “each breed’s mean mass”.

3.2.2 Plan composition

Starting with the cognitive revolution of the 1960s [140], problem-solving has been frequently conceptualised as the progressive *decomposition* of a problem statement into subproblems, or equivalently, as a breadth-first process of plan composition, from a high-level plan to lower-level steps. The terminal nodes of this expansion would be steps sufficiently atomic that the user would feel confident a corresponding API primitive exists. In data wrangling the primitives in question would mostly be functions, combined in their logical order. In the example from before, such an plan could be:

1. Split rows based on region
2. Calculate the median income of each region
3. Take the largest and smallest median
4. Subtract the smallest from the largest

There is nothing absolute about this particular plan - it is but a single point in a design space of possible plans, some of which may employ a more pseudocode-esque language closer to API terminology (e.g. "Group records by region" instead of "Split rows"), assume a different set of API commands (e.g. if no `max()` function exists, they may have to sort the medians first) or use a different granularity (e.g. someone comfortable with split-apply-combine may collapse steps 1 and 2 into a single step). What matters is that the plan constrains the later search for corresponding commands, as part of a means-end analysis.

When a set of problems share high-level similarities in their composition, these can be abstracted into reusable problem-solving plans, also known as *templates* [141], or *schemata* [142, 143] (not to be confused with the data schema). These plans can be retrieved from the programmer’s memory (when abstracted from past problems or taught explicitly) or created if no appropriate plan exists. According to Soloway and Ehrlich [144], expertise can be characterised as having a large and well-organised mental library of such abstract, context-independent plans.

The model of programming as breadth-first, step-wise refinement is an idealisation. In practice, plan composition draws upon both top-down (memory-driven) and bottom-up (code-driven) processes. For example, implementation details could serve as memory cues, and both semantic or syntactic errors could reveal the chosen plan to be incorrect, leading to the plan being modified. If no plan exists, learners may search for a core computation, implement it, and then expand that solution bottom-up [145]. Indeed, moving back-and-forth between the plan- and code-level could help novices avoid committing prematurely to a plan and encourage them to reconsider it. This was suggested by a recent think-aloud study by Castro and Fisler [146], which found that CS1 students who used a more cyclical approach tended to outperform participants with a strict top-down or bottom-up approach.

Plan retrieval barriers

Plan retrieval is associated with several barriers. As pointed out by Glaser et al. [147], students can choose the wrong plan, or the plan could be too rigid, such that programmers fail to adapt it. There have been attempts to quantify the extent to which plan-related mistakes are to blame for bugs. In an early example by Spohrer and Soloway [148, 149], plan-related bugs were found to significantly outnumber bugs relating to misunderstanding of concepts, however given its focus on imperative programming, it is difficult to generalise to data wrangling. A more recent analysis of more than 160,000 SQL snapshots found that 40% were semantically incorrect (compared with 54% of syntactically incorrect snapshots) [130], suggesting that planning may be less of a barrier than the actual programming in data wrangling, or at least in its query-related subset of operations.

3.2.3 API lookup

Once a plan has been composed, it remains for the programmer to translate its steps into executable code. Norman has referred to this mapping problem - of “compiling” a high-level intent into low-level commands - as the *Gulf of Execution* [150]. Fangohr has argued that, from an educator perspective, bridging this gulf is less interesting than plan composition, since it is “in principle, an algorithmic procedure” [151, p.1212]. While that may be true for an experienced programmer, for a novice end-programmer, that very same gulf can appear much wider [152].

One complication that has already been discussed in Section 2.4.1 is that data wrangling APIs tend to be rich in specialised functions and therefore present a far wider menu of primitives than a CS1 course that limits itself to built-in language constructs. Although one could use the indexing operator to access the diagonal elements of a matrix, it is far more robust to rely on a convenience function like `diag()`, but it is also an onerous task to walk through and memorise all such commands.

Because of the number of operations, storage of API command syntax is likely *externalised*,



Figure 3.4: Diagram summarising Kelleher & Ichinki’s *Collection and Organisation of Information for Learning* (COIL) model [3, 4].

meaning that programmers do not memorise the syntax, because the reliability and low cost of retrieval render memorisation unnecessary. Because it is externalised, selection of API commands is likely preceded by an exploration or *search* for candidate commands, whether through a search engine, within-page hyperlinks, structured API representations like menus, manual search within page results, or other external aids (e.g. cheat sheets, lecture slideshows). We also note the widespread presence of *inherent documentation* (discussed in Section 3.5.3), such as tab completion or tooltips, that somewhat blur the distinction between internal and external API lookup, as the learner may never actually leave the IDE.

Recently, Kelleher and Ichinki proposed a model for describing the usage of external resources, called the *Collection and Organisation of Information for Learning* (COIL) model [3, 4], which segments the selection process into three stages. In the first stage, *information collection*, the programmer searches for relevant information and continually judges its relevance. In the second stage, *information organisation*, they store and organise this information to prepare for later use, for example by grouping tabs, copy-pasting, or simply keeping the resource open for cross-referencing. In the final stage, *solution testing*, they integrate the new information into their solution and debug it. The COIL model, summarised in Figure 3.4, is especially interesting for its emphasis on the strategies used to avoid burdening the working memory with API syntax (e.g. by copy-pasting code), in addition to externalising the long-term memory.

The most important form of external memory, and the one underpinning the most qualitative change in programmer habits for the past two decades, has been the Internet. Even for an expert programmer, foraging social media, forums, code repositories, and search engines for instructive code snippets is a natural part of the workflow. This kind of interleaved programming, which repeatedly or even predominantly consists of finding and integrating code snippets, is sometimes called *opportunistic programming* [153]. It tends to be used when robustness or performance can be sacrificed in favour of greater development speed, and the behaviour has mostly been studied among web developers and designers [154, 155], for whom the priority is to finish a prototype quickly. While we have not found any research specifically focused on the retrieval strategies used by data scientists, we posit that a novice-to-intermediate end-user programmer working on a data wrangling script is similarly likely to rely on opportunistic strategies.

Just how important is API lookup to modern programming workflows? Studies with experienced web developers suggest that they spend between a fifth [153] and half [156] of the time searching the web. Highly experienced programmers have confessed in interviews to repeatedly copy-pasting the same snippet of code “hundreds of times” [153]. One interview study quoted a

student as saying that CS “can feel like a more advanced googling degree” [157, p.219].

Research on novices’ information retrieval behaviours has received little attention in CS educational communities, however end-user programmers’ learning of unfamiliar APIs has been documented elsewhere. For example, Ko et al. [158] observed that end-user programmers often struggled with determining which API command to use, in what the authors referred to as a *selection barrier*. In one interview study with CS students on their web usage, conducted by Ben-David and Ma’ayan, students reported that the Internet could feel overwhelming, like shooting “in the dark” [157, p.222], and that they experienced cognitive overload. One study that tracked CS students’ Internet usage during a lab exam found that, while high-performing students frequently visited StackOverflow, low-performing students instead tended to revisit their old solutions [159], plausibly to avoid feeling overwhelmed by new information. We will therefore review potential causes as to why API lookup can be so overwhelming.

Query formulation barriers

While browsing through search engine results is intrinsically complex, the feelings of overload are probably compounded by novices’ inexperience in formulating queries and overall lack of a mental model on how an API works. Unfamiliar terminology could pose one source of difficulties, since formulating a query requires knowledge of appropriate keywords to yield relevant results. The target API’s terminology may be far removed from natural languages, and could be misaligned with other APIs that the programmer may be familiar with. There are plenty of such misalignment examples in data wrangling, where `select ()` is not intuitively about columns (as opposed to rows) while in other contexts is known as *projection*. APIs have local terminology - for example, what is known as *indexing* in Python is often called *subsetting* in R, and *fancy indexing* in Python does not map onto a neat term in R. *Pivoting* is sometimes called *reshaping*, *spreading*, or *gathering* depending on context; to *aggregate* is variously called to *summarise* or *reduce*.

Evidence for the jargon barrier can be found in at least two observational studies. In a think-aloud study by Ko and Riche, CS students were asked to assess software requirements involving unknown networking APIs, and several of the participants verbalised their struggle to find right keywords [160]. Participants who already had a conceptual understanding of the domain (i.e. domain concepts [115]), formed more effective queries, apparently because they judged the relevance of web resources more efficiently.

In the other study, by Wang et al. [152], researchers studied novices’ use of a searchable example gallery while completing an open-ended project in a block-based programming environment. Their data suggested that for almost 40% of the queries, the student never opened the code example. Qualitative analyses found that many of these dead-ends were due to a misalignment between how students articulated something, and how the gallery described it. They moreover found that failure to find the correct entry appeared to have a discouraging effect,

making students less likely to use examples later on.

A related barrier may be brought on by difficulties with refining their query when it is unsuccessful. In the previously mentioned interview study by Ben-David and Ma'ayan [157], students further along in their programming journey described having developed strategies for managing information overload, such as gradually broadening their search after trying to write as specific a query as possible. It is plausible that a lack of such strategies among novices contribute to them feeling overwhelmed. In one study with web developers with varying degrees of expertise, novices were found to rarely conduct query refinements or to follow up on the results [155].

API usability barriers

API designers face many trade-offs, and libraries vary in their inherent usability and complexity. This affects how novice-friendly the documentation appears, and how easily the novice can comprehend a candidate API command's run-time behaviour. To illustrate the anatomy of a documentation entry, we show an example of the Pandas `groupby` function in Figure 3.5. An entry typically consists of a function signature showing all the parameters with their default values, followed by a brief prose description. This is followed by a parameter list that explains each parameter and specifies the expected value types and what the return value type is. As mentioned in Section 2.4.1, the parameters are often numerous and interdependent. The `groupby` function, for example, has no fewer than 9 parameters, and can accept its main argument - the column(s) to group by - in four different ways. This high degree of parameterisation has been hypothesised by Olney and Fleming to be a key barrier for novice data scientists [96]. We find evidence for this in formative interviews conducted by Zhang and Guo [9], which suggest that novices tend to find production-grade data science APIs opaque and difficult to understand.

3.2.4 Example adaptation

Having identified an API operation and source as relevant, for example Pandas' `groupby`, the programmer needs to integrate that information into their partial solution. This requires *storing* and *retrieving* the information to/from external memory, for example by using copy-paste or leaving browser tabs open (what the COIL model would call *information organisation* [3, 4]). Ultimately, the code example must be *adapted* - the variable names substituted, the parameters properly configured - so as to serve its role within the programmer's own solution code.

Although the programming education literature has a rich and multi-decade long tradition of studying how novices make use of worked examples, this literature mainly deals with sample solutions that feature programming patterns of a procedural or imperative nature [161]. With high-level APIs, chances are that such patterns have already been encapsulated into their own convenience function. It is therefore important to distinguish between worked examples that aim to illustrate an abstract plan, and code examples that aim to illustrate the behaviour of a function

pandas.DataFrame.groupby ¶

`DataFrame.groupby`(*by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=<object object>, observed=False, dropna=True*) [\[source\]](#)

Group DataFrame using a mapper or by a Series of columns.

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

Parameters: **by** : *mapping, function, label, or list of labels*

Used to determine the groups for the groupby. If **by** is a function, it's called on each value of the object's index. If a dict or Series is passed, the Series or dict

Figure 3.5: A Pandas documentation entry, featuring a syntax summary, a short prose description, and a parameter list. Data wrangling functions tend to be highly configurable.

or API usage pattern. For opportunistic programming workflows to be efficient, the programmer needs to be able to quickly interpret a code example, estimate its relevance, and utilise it within their solution, a process open to several different impediments.

Example comprehension barriers

Because of the highly configurable nature of data science functions, the number of examples in a documentation entry can easily exceed a dozen. Difficulties in understanding these code examples would also make it difficult to utilise them. In terms of Ko et al. end-user programmer barrier scheme [158], complex examples can produce *use barriers* and *coordination barriers*, referring to a lack of understanding in how to use and combine the selected commands, respectively. Another scheme, by Wang et al. [152], calls the same phenomenon *understanding barriers*.

Little research has been published that rigorously analyses novices' issues with API code examples, but a first step is surveying the design and formatting choices made in popular APIs. For example, to be reproducible and easily copy-pasted, the code example typically includes the creation of toy data structures or the importing of external data sets, alongside other necessary boilerplate code such as the setting of random seeds [162]. Sometimes, as with Pandas (Figure 3.6a), the input data structure is explicitly printed out and displayed as it would look in an interactive shell. Other times, as with Tidyverse (Figure 3.6b), it is left to the user to either run the code or infer the dataset's structure from its construction. In some places, as with Pandas, the data structure is usually of the toy variety, and as minimal and self-explanatory as possible, with consistently used *metasyntactic variable names* such as `df` that signal their roles as place-

```

>>> df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
...                               'Parrot', 'Parrot'],
...                   'Max Speed': [380., 370., 24., 26.]})
>>> df
  Animal  Max Speed
0  Falcon    380.0
1  Falcon    370.0
2  Parrot     24.0
3  Parrot     26.0
>>> df.groupby(['Animal']).mean()
      Max Speed
Animal
Falcon    375.0
Parrot     25.0

```

```

by_cyl <- mtcars %>% group_by(cyl)

# grouping doesn't change how the data looks (apart from
# how it's grouped):
by_cyl
#> # A tibble: 32 x 11
#> # Groups:   cyl [3]
#>   mpg   cyl  disp    hp  drat    wt   qsec    vs
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1   21     6   160   110   3.9   2.62  16.5     0
#> 2   21     6   160   110   3.9   2.88  17.0     0

```

(a) Pandas official documentation

(b) Tidyverse official documentation

Figure 3.6: Code examples from official documentation sources

holders [163]. Meanwhile, Tidyverse’s official documentation illustrates functions using built-in datasets, some with 14 columns, which can be cumbersome to visually parse [49].

Another aspect of official API documentation is that entries tend to focus on one API element at a time [3], which makes it harder to integrate them. To pre-empt coordination barriers, a reference source may also provide *API usage patterns* [115]: more extensive code snippets that demonstrate how an element works in the context of other API commands. The entry in Figure 3.6a already does this to some extent, by showing how it works in combination with an aggregation function (`mean()`).

If an API usage pattern contains too many unfamiliar elements, this could backfire: Wang et al. [152] found that the likelihood of a novice integrating an example, while completing an open-ended code project, systematically decreased as the number of unfamiliar API elements increased in the example. In a data science setting, this echoes interview data from Zhang and Guo [9], who found that novices struggled with understanding chains of functional API calls, and had to resort to manually inspecting the intermediate states. The same authors [9] surveyed a small sample of textbooks, finding that most code examples feature long method chains within a single line.

Example integration barriers

Both the exploration and exploitation of API elements require the programmer to mentally, and then actually, map elements in the example code with that of their problem context. This becomes a matter of accurately configuring the parameters and substituting the variable names. Failing to establish the correct correspondences could lead to what Wang, et al. have called a *mapping barrier*, while failure to accurately adapt the example would be a *modification barrier* [152].

For example, in our running example of comparing the richest and poorest region (see Section 3.2), the programmer must use Pandas’ `groupby` to split the rows based on region. The API documentation example, however, involves a dataframe storing a set of animals and their

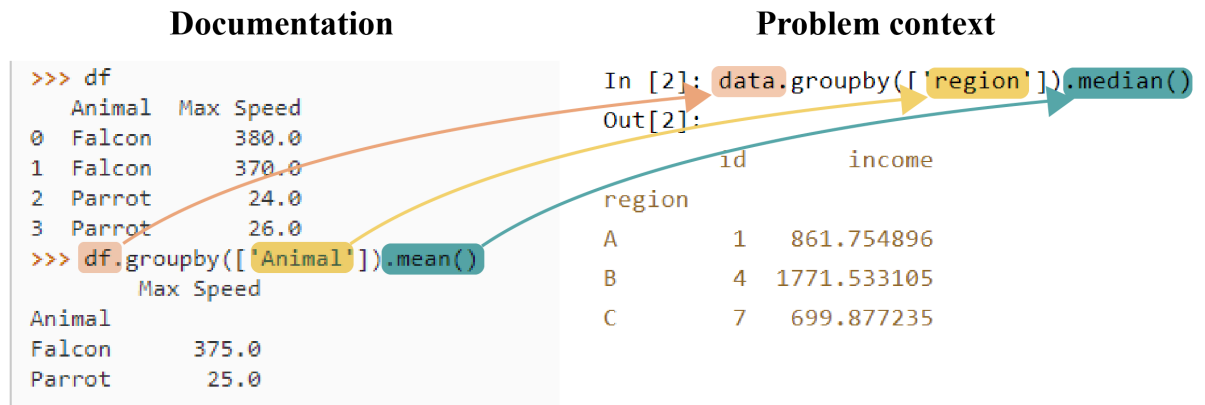


Figure 3.7: The programmer needs to map elements in the documentation’s code example to elements in the problem context.

maximum speed. To leverage the example, the programmer must identify the *Animal* column as playing a similar role as the *region* column, then substitute `'Animal'` for `'region'`, and so forth, as illustrated in Figure 3.7.

In the same example we also notice that subtle differences between the code example and problem contexts can reveal themselves: the code example only has one numeric column, but the problem dataset has two. `median()` will by default calculate the median of both, leading to an `id` column in the summary table. This exposes a vulnerability of copy-pasting. As captured by Thayer et al.’s notion of *robust API knowledge* [115], the more brittle the underlying understanding of the API is, the less flexibility they will have in adapting the code example. In this case, they will need to understand that the columns that should be aggregated need to be selected after the grouping (i.e. `groupby('region').income`).

Research by Ichinco et al. [164] suggests that, although mapping success matters, it is no guarantee of successful integration. They asked school children to modify a set of programs based on an on-screen example, in a block-based environment, and to also indicate the correspondences using pen and paper. The authors found that the correlation between task success and mapping success was low, which they interpreted as indicating that example adaptation needs to be plan-driven in order to be successful.

3.2.5 Debugging

Once the data wrangler has a solution, they run it and evaluate the results. Even assuming that they have composed an appropriate set of steps and, for each step, identified the correct API command, chances are that syntactic wrinkles remain to be ironed out. This may require IDE-focused debugging activities, or a renewed search for external resources, such as looking up an error message online. Errors can be broadly classified based on whether they execute (i.e. are syntactically valid) or not [165]. Unlike syntactic errors, *semantic* errors cannot be detected through interpreter error messages. As has been frequently noted, error messages tend

to be written to support proficient programmers and can elicit strong feelings of anxiety and frustration in novices [166]. One large-scale study on SQL students found that 51% of them abandoned a question when they were unable to resolve syntax errors [167]. What are the causes behind bugs generally, and data wrangling bugs specifically?

Information barriers

Information barriers refers to the inability to probe the *internal* value of a variable or behaviour of a program [158], such as verifying that the problem indeed was solved, or at the very least, that the program behaved as intended. This can be seen as a matter of bridging what Norman has termed the *Gulf of Evaluation* [150].

In spreadsheets, this issue is addressed through a live data view that reactively updates in response to value changes. Some standard IDEs incorporate similar tools for data wrangling, most notably RStudio. Others require that the programmers either mentally simulate an operation, explicitly print it, or use debugger tools. Since both R and Python are interpreted, programmers can use the read-evaluate-print loop (REPL) interface to test single-line commands. In notebook environments, they are moreover able to execute individual chunks, which theoretically could be leveraged to inspect intermediate output. However, as Rode and Ringel [49] have noted, R console output can be visually intimidating due to its closely clustered and often misaligned output. In a study of theirs, novices in R reported significantly higher anxiety about parsing R output compared with the more neatly formatted SPSS output, although this levelled out towards the end of a semester.

Overall, it is not known how instinctively this workflow, of implementing and inspecting, comes to novices. In a review study on debugging by McCauley et al. [168], the emerging finding was that novices suffer from a less coherent and hierarchical mental representation of the program's purpose, which prevented them from chunking the program in a way that helps them diagnose probable causes of errors. As a result, novices tend to take a depth-first approach, focusing on locating the error without regarding the program as a whole. An eye-tracking study featuring Java programmers found that more experienced programmers attended more to the source code than novices, and also switched between code, output, and a program visualisation more frequently [169], suggesting that experts use a more interrogative approach.

Understanding barriers

According to Ko et al.'s barrier scheme [158], *understanding barriers* refer to confusion about a program's *external* behaviour, such as run-time errors. These barriers are often studied by analysing relative frequencies of bug classes and theorising about their underlying causes, and several have been conducted for SQL [128, 167, 170]. Although most bug analyses are too different in their exercise sets and schemes to be aggregated, one emerging pattern is that syntactic bugs are more common than semantic bugs: Ahadi et al. [167] found that 54% of code snapshots

resulted in a syntactic error, and that five PostgreSQL error codes accounted for more than 53% of all syntax errors. Besides typos, these included references to undefined columns and grouping errors. Another consistent finding is that semantic bugs are more likely in complex programs. Smelcer [170] found that the likelihood of omitting a join clause rose with query complexity, while Ahadi et al. found that the most semantically incorrect concepts were, in order, self-joins, correlated sub-queries, **GROUP BY . . . HAVING**, simple subqueries and natural joins [130], all of which are conceptually non-trivial and likely cognitively taxing.

To the best of our knowledge, only two in-depth bug analyses that has been published for data wrangling. Rafalski, et al. [109] invited students to solve a collection of R tasks and found that the students struggled with interpreting R's error messages. Their bug analysis found that the most common syntax error was misnaming or misplacement of variable names, followed by extra characters. Other common errors were mistyping function names and missing arguments. Overall, they interpreted this to imply that naming conventions pose a significant barrier to novices in R. The authors also compared CS majors and non-majors in their error rates, and found that CS majors produced significantly (albeit marginally) fewer errors, suggesting that prior, imperative programming experience provides some benefit.

The second study, by Yarygina [171], collected error logs from three non-major students enrolled in an introductory data science course in R. The students varied in their academic and mathematical background, but the author found no meaningful differences in error proportions - for all of them, around half of all errors related to mistypings (this includes incorrectly spelled arguments or functions) while around 15% was made up of content errors (e.g. missing arguments).

3.2.6 Section summary

Plan retrieval barriers: Novices are less likely to have well-organised plans to draw from in approaching a problem.

Query formulation barriers: Novices may lack the keywords and API domain concepts necessary to formulate productive queries. They are also more likely to be overwhelmed by search engine results.

API usability barriers: Data wrangling APIs are highly parameterised, which add to their intrinsic complexity and can make syntax documentation difficult to parse.

Example comprehension barriers: Code examples are often formatted in ways that make them hard to comprehend, for example including boilerplate code and large sample data sets.

Example integration barriers: A programmer may fail to successfully modify an example, for example doing the correct variable substitutions. This could be due to a lack of robust API knowledge, or a lack of a coherent plan.

Information barriers: Novices are less likely to have a hierarchical program comprehension, and are therefore less efficient at debugging. The lack of a live view of a data structure requires the programmer to repeatedly inspect the state through a console, but the console’s bare, textual format can be anxiety-inducing.

Understanding barriers: Syntactic bugs are more common than semantic bugs in data wrangling. Analyses of R bug suggest that most syntax errors are attributable to mistypings.

3.3 General learning theories

The process of learning data wrangling draws upon general cognitive activities that the learning science community has already developed sturdy, conceptual frameworks for. While not the principal subject of investigation, any work in instructional design finds a natural point of departure in *cognitive load theory* (CLT). The concept, first introduced by Sweller in 1988 [172], is today among the most referenced theories in educational psychology at large [173] and CS education specifically [174]. Following a review of foundational concepts, we will explore how they relate directly to the design of data wrangling-related instruction.

3.3.1 Cognitive load theory

CLT rests on the well-established fact that human working memory is severely limited in its capacity and that cognition incurs a “load” that could exceed this capacity. However, by abstracting information into higher-order cognitive structures called schemata (also known as *plans, templates, chunks*), cognitive activities could instead be offloaded to the virtually unlimited long-term memory. This long-term storage is what constitutes learning, automation, and ultimately expertise [175].

In its original form [176], the theory holds that learning materials could incur two different types of load, which add up to the total cognitive load. They are defined in terms of their effect on the learning outcome:

Intrinsic This is inherent to the learning task itself, and completely determined by the interactivity of its elements (i.e. complexity). Note, however, that what counts as an element is learner-dependent: an expert would chunk a problem into more abstract entities, which would lower the element interactivity. Intrinsic load is thus conditioned on the learner’s expertise.

Extraneous This is defined as cognitive demands that are unnecessary to - and therefore detracting from - the task. A typical example of extraneous load is the effort associated with information retrieval and cross-referencing. If information sources that need to be men-

tally integrated are located far apart, for example a caption far removed from the figure it is meant to explain, then the extraneous load would increase.

It is worth noting that these definitions hinge crucially on what is considered *relevant*, and in turn, the definition of the task itself. For example, many instructors advocate introducing programming via block-based languages. The argument for block-based languages often states that the spelling out of textual syntax correctly is a skill that is irrelevant, or at least secondary, to the development of program composition skills [177, 178]. For that reason, we should seek to eliminate the incidence of syntax errors by making composition block-based. However, the assumption that the skill of typing syntactically valid code has less priority than composition is difficult to empirically determine. Similarly, one could argue that API lookup skills are secondary to programming skills in data wrangling, and that instructional settings should inform the students of all necessary syntax rather than have them look it up themselves. On the other hand, one could argue that the ability to independently find syntax information and comprehend less-than-ideal API resources is an equally essential skill. In short: what counts as “intrinsic” and “extraneous” complexity depends on which skill an instructor chooses to prioritise.

An important insight of CLT is that the goal of instructional design becomes one of removing irrelevant steps, elements, and inefficiencies from the target skill, since the intrinsic load is considered fixed [175]. Usually instructional guidelines, such as “Reduce the cognitive demands of cross-referencing”, come with caveats and boundary conditions, however. For example, if you cram too much information into the same display to avoid the costs of cross-referencing (also known as the *spatial contiguity principle* [179]), this would soon come up against other principles, such as the gestalt principle of using white space to group related information together (also known as the *segmenting principle* [179]). The practical implications of CLT are therefore far from strict prescriptions, but unavoidably a matter of trading off one source of extraneous load for another, such as locating sweet spots between too much and too little white space.

Germane cognitive load

The idea of instruction as aimed at minimising extraneous load changed somewhat with Sweller’s introduction of *germane load* in 1998 [180]. The observation prompting this was that, although a task’s inherent complexity could be considered fixed, not all of that complexity is directly pertinent to the development of cognitive schemata. Cognitive demands are considered *germane* when they are relevant to the *abstraction* of schemata that would generalise to novel and variable problems. The addition of germane load to CLT could thus be seen as defining a new category of relevance: relevance to schemata acquisition.

Because schema acquisition is precisely what most learning tasks hope to achieve, it could be considered a desirable form of cognitive load that should be increased rather than minimised, at least until it exceeds the working memory capacity. The concept has served a role in generating techniques for imparting a more abstract understanding that succeeds in a more variable

task domain [181]. Examples of germane load-maximising interventions tend to involve some form of explicitly guided abstraction, for example worked examples, subgoal labels, and self-explanations.

Criticisms of CLT

Although CLT provides a useful framework for instructional design, and helps discipline researchers into making explicit what they deem relevant, the question of whether or not it should be regarded as a scientific theory is controversial. One frequently raised criticism is that CLT is unfalsifiable, and therefore not a true theory in Popper's sense of the word [182]. For example, both the discovery that reducing white space improves learning outcomes and its negation could be seen as supporting CLT, depending on whether you test the spatial contiguity principle or segmenting principle. Another reason is that measurements of cognitive load - ranging from physiological, to subjective, and performance-based - all assume the theory's own assumptions, namely that extraneous load inhibits learning and is caused by poor instructional design [183]. As many have noted, defining extraneous load as the *cause* of poor instructional design, and subsequently arguing that it is *caused* by poor instructional design, amounts to a circular restatement [183].

Although CLT may not be a theory in a Popperian sense, Gerjets et al. [183] have argued that CLT should still be considered a valid theory in a structuralist sense [184]. By this they mean that CLT can be used to generate testable empirical predictions when considered in combination with more specific theoretical assumptions. As an example they mention how CLT predicts that approaches that emphasise abstract aspects of worked examples will lead to improved performance in unseen problems. However, this is arguably no less of a circular restatement: if the goal is to solve a variable set of problems, the fact that focusing learners' attention on invariant aspects of problems (i.e. abstraction) serves that goal well should not be surprising.

Although this short review of CLT is far from exhaustive or conclusive, it will have implications for the rest of the dissertation. We will make occasional mention of cognitive load to denote "mental strain", and view it as an important goal to identify and remove cognitive demands that are secondary to our stated learning objective, namely of enabling novices to write executable data wrangling scripts. However, we will be wary of arguing that the reason an intervention may or may not work is due to a reduction of extraneous load, since we are uncertain of whether that has explanatory value.

3.3.2 Scaffolding

CLT is intimately tied to the notion of scaffolding. *Scaffolding* refers to an intervention with two properties: it enables a learner to accomplish a task that they could not succeed at unassisted, and it facilitates continued success for when the scaffolding is eventually removed [185]. The

term is generally credited to Wood et al. [186], though the underlying ideas back much further [187]. According to the original formulation of Wood, et al. [186, p.98], scaffolding is not a purely cognitive process. Besides simplifying the task, highlighting critical features and explicating the problem-solving process, a scaffolding tutor should also serve a mood-regulating and motivational role. For example, they should elicit the learner's interest, reduce their frustration, and help them to maintain their focus [186].

Scaffolding is inherently good, in the sense that it is *defined* as something that is helpful - something that enables someone to do something they otherwise would not be able to, initially assisted, but eventually unassisted. It is possible to help a novice programmer solve a problem by simply giving them a solution, but that would hardly empower them to solve the next problem unaided. We must therefore distinguish between idealised scaffolding and *purported* scaffolding. This distinction could help resolve some of the tensions within the learning sciences, where different theories are in apparent conflict on whether scaffolding is necessary or counter-productive.

The scaffolding debate

Consider the minimal guidance stance, which draws upon Piaget's *constructivist* theory of learning, according to which learning happens through active exploration, interaction, and play. In the context of programming, this idea takes its most influential form in Papert's *constructionism*, which emphasises the need for letting the learner create their own knowledge organisation and artefacts [188]. It therefore advocates problem-based or project-based learning, where students are given an outlet for their own exploration. As of 2014, constructivism/constructionism was the most commonly used theoretical framework in CS education [174]. It has been argued by for example Hermans and Smit [189] that the community of programming instructors probably over-represents learners who themselves enjoy discovery-based learning and that, as a result, many instructors' approach is "implicitly constructionist" [189, p. 88].

Now consider the alternative stance, associated with Kirschner and Sweller [190,191], which emphasises explicit guidance. This idea is closely tied with that of germane cognitive load: if the goal is X, why not simply tell them how to do X, rather than wait for them to arrive there themselves through exploration? A common example is Biederman and Shiffrar's 1987 chick-sexing study [192], in which novices' chick-sexing ability was greatly accelerated when taught explicitly what sex characteristics to look for.

If the goal of educational research is to investigate whether *purported* scaffolding amounts to *idealised* scaffolding - to investigate *when, what, and for whom* an educational intervention is necessary and sufficient for inducing a specific competency - then much of the controversy relating to minimal versus explicit guidance is resolved. While this dissertation is focused on evaluating purported scaffolding techniques, it is not interested in making a stronger case that purported scaffolding techniques are always desirable, or that they cannot be counter-productive.

The scaffolding debate in data wrangling education

Where an instructor falls on the constructivism–explicit guidance continuum is likely to influence their instructional design choices. Within the data wrangling sphere, we find echoes of the same debate on a number of important design dimensions, such as language choice, data set choice, documentation choice, and exercise type.

Language Many data science instructors have chosen small toy languages or sub-languages instead of production-grade APIs. For example, Zhang and Guo [9] used a pedagogical API known as `datascience.py` for their work, the block-based TIDYBLOCKS [193] uses a Tidyverse-like syntax, while the creators of BLOCKPY [114] (another block-based data science environment) opted for their own custom API. Such intermediate technologies incur an overhead that may very well be justified, assuming they attract more students than they put off, and make learning of the production-grade API faster enough to compensate for the time spent on the didactic API. An alternative scaffolding approach would be to manipulate the *exposure* to API elements than the API itself, or otherwise address the API lookup barriers of Section 3.2.3.

Data set The data sets used in data wrangling instruction could be rich in contextual details and annotations, or self-explanatory and devoid of context. Domain knowledge is arguably extraneous to data wrangling as defined in this dissertation, and may interfere with the problem-solving, as argued by the previously discussed problem description effect [138]. On the other hand, authentic data sets are probably integral to developing a robust data literacy, critical thinking, and an appreciation for statistical variability [194]. Moreover, there are studies to suggest that authentic data sets, if pre-cleaned, could serve a motivational role. The CORGIS project by Bart et al. has collected pre-cleaned datasets relevant to all major subjects [195, 196]². In a recent paper, the CORGIS organisers reported survey results that indicated that most participants found the data very interesting and useful [197]. The trade-off between simplicity and motivation could be approached through *faded scaffolding*, where data complexity and authenticity is increased incrementally [194].

Documentation When solving an exercise, should information about the relevant API commands be provided for convenience, or should the student look them up themselves? As Robertson has argued [198], it could be considered a core learning objective that students should be able to search through information sources on their own. These information sources could be authentic, for example an official API documentation or plain search engine, or they could be carefully formatted and organised to be usable by novices. Which documentation source is preferred depends on whether program composition or online syntax retrieval skills is having higher priority in the instructional sequence.

²Collection Of Really Great and Interesting dataSets, think.cs.vt.edu/corgis

Exercise type A related dilemma is that of whether to provide closed-response exercises or to let students conduct their own open-ended data wrangling projects. At least one interview study on non-CS majors found that project-based assessments were perceived as more fair, authentic, and more conducive of personal pride [199]. One could also argue that a fluency with real-world problems is a top priority, and that it therefore should be introduced from the beginning. On the other hand, as noted by Guzdial, “If you’re spending time on the context, you’re not spending time on the content” [200, p.5].

3.3.3 Graphical scaffolding

Having thus outlined the general scaffolding theories, we will turn our focus towards graphical scaffolding. Graphical aids are frequently employed within instruction to scaffold schema acquisition, but data wrangling is unusually amenable to them. Vectors, matrices and dataframes are all two-dimensional³. Tabular data structures are therefore *inherently visualisable* - an attribute that a learner or instructional designer could choose to exploit, whether in the form of mental imagery, student-generated graphics, or external representations provided by instructional materials or IDE features.

The benefits of graphics

The source of graphics’ scaffolding potential relates to the architecture of the working memory. According to Paivio’s *dual coding theory* [201], the working memory has two distinct information-processing pathways - one verbal and one visual. If a learner is overloaded with linguistic information, they still have a visual channel available at their disposal, effectively increasing their cognitive capacity. The central prediction made by dual coding theory is therefore that *all* learners (assuming they have functioning eyesight) can process and retain more information if visual information is presented alongside linguistic information [202].

Visual media also have inherent advantages that textual media lack. Graphical design can draw upon a number of more or less universally understandable variables for encoding abstract relations. Whereas textual accounts require the reader to construct a mental model from a purely arbitrary, linear string of symbols, graphics can represent non-linear relationships and complex correspondences through their own visual structure. They can represent containment via nesting, change through a sequence of snapshots, and motion through arrows. Another key affordance is colour: as observed by Tufte [203], colour can be used to represent quantity (by gradation) and category (by hue). Colour can also be used to highlight information by being selectively applied. Similarly, gestalt principles such as proximity and similarity could be used to organise information and make visual search more efficient [204, 205]. Finally, graphics can make use of arbitrary diagrammatic notations or textual annotations, allowing them to expand their

³Technically, dataframes could be tensors and exceed two dimensions.

expressivity even further.

Multiple representations

Graphical representations may also work synergistically with other kinds of representation, by promoting more relational and abstract processing. This is argued by Ainsworth in her DeFT framework on multiple external representations [206]. According to this framework, external representations (whether iconic or symbolic) serve different functions. They can provide concrete information to be simply read off (e.g. a photo), they can reduce cognitive effort by facilitating search (e.g. a family tree), and they can influence problem-solving by constraining the range of inferences (e.g. a construction manual) [206]. When multiple representations are presented together, the nature of their relationship could also vary in purpose. They could complement each other or constrain the interpretation of each other [206]. Furthermore, the very existence of multiple representations could encourage the learner to integrate them into a deeper understanding, for example comparing the formulaic and graphical representation of a mathematical function.

Scaffolding in graphic design

The learning scenarios in which graphical aids have been used and studied range from anatomical illustrations to mechanical diagrams, so we should exercise caution in estimating how directly applicable their findings are to data wrangling. It almost certainly makes no sense to treat graphics as its own cohesive category that is either uniformly good or uniformly bad. Well-designed graphics certainly are, while poorly designed graphics most definitely are not.

Nevertheless, some graphical design principles are robust enough to be broadly applicable to tabular data graphics. The literature and debate on these principles present something of a scaffolding debate in miniature. A graphic could be authentically messy and disorganised, or it could be meticulously designed to highlight the most relevant content and relationships. The former would appeal to a constructionist, while the latter would appeal to an explicit guidance advocate. The former prevents learners from depending on well-designed graphics as a crutch, while the latter guides the learner more directly towards the target schema. In a 2017 review by Mayer [179], the following multimedia principles are listed as especially robust and effective⁴:

Coherence principle: Learning is improved if learning-irrelevant details are removed, including decorative elements.

Signalling principle: Learning is improved if task-relevant elements are highlighted through arrows, colour or spotlights.

⁴We have constrained our review to static graphics, for which reason other principles relating to animations or interactivity have not been included.

Segmenting principle: Segmented, self-paced lessons improve learning compared with continuous lessons. This can be regarded as a form of aided chunking [207].

Spatial contiguity principle: Words should be placed close to the corresponding graphical element and distances for cross-referencing should be reduced.

Each principle has associated boundary conditions [179]. For example, segmentation is preferred until a lesson becomes so disjointed it begins disrupting the learner's mental connections and focus. A schematic diagram without any decorative details is preferred until it becomes so boring that the reader loses interest and abandons it. Highlights can be effective - but only up to a point. Without a massive, corporation-scaled A/B-testing apparatus to experimentally optimise graphics, or a diagram-generating neural network, graphical design remains more of an art than a science. But once created, the task of evaluating graphics becomes one of determining for *what* purpose exactly, in *what* context, and for *whom*.

3.3.4 Section summary

- Cognitive load theory is a central framework in education. It classifies cognitive demands as either relevant (*intrinsic*) or irrelevant (*extraneous*) to a particular learning objective. The cognitive demands conducive to more abstract schemata development is called *germane*.
- Scaffolding refers to an intervention that enables a person to do something single-handedly that they otherwise would not be able to do. In practice, scaffolding typically involves explicit guidance towards the target schema.
- Scaffolding resides on a continuum, with explicit guidance on one end, and constructionist approaches on the other. Many dimensions of data wrangling instruction (e.g. the language taught, the data sets used) could be seen as instances of this continuum.
- Graphics present a way of presenting the target schema more directly, and can encourage more germane processing. Research into multimedia design has identified several principles that reliably increase the germane processing.

3.4 Graphics in data wrangling

The insight that tabular data structures are visualisable - and that graphics depicting them could serve a function within data wrangling workflows - is not a novel one. What follows here is a review of the ways in which tabular graphics have been employed previously.

3.4.1 Program visualisation

Program visualisation systems - software that visualises the run-time behaviour of programs - present one relatively well-explored, visualisation-related area within programming education. Generally these visualise low-level behaviours such as the changing state of the stack or memory heap, in order to explain concepts like local variables, pointers, control flow, and function calls. Such systems are therefore less relevant to high-level functional programming like data wrangling. A 2013 review of program visualisation by Sorva et al. [208], as well as a 2016 review by Hidalgo-Cespedes et al. [209], exclusively concern themselves with imperative and object-oriented programming, probably because functional array manipulation is considered to be too abstract to belong to the category.

Another category is that of *algorithm visualisation* (AV). This has been an active area of research since the 1980s, but it appears to have converged on a particular class of algorithms common to object-oriented CS courses, such as sorting, graph traversal, tree manipulation and dynamic programming algorithms. That was the case for Hundhausen's influential AV meta-review from 2002, and is still adhered to in AV studies from the past few years (e.g. [210]). These algorithms have in common that they are general-purpose without any particular data context in mind and that they do not involve relational operations. Thus viewed, both program and algorithm visualisation appear to be of limited relevance to data wrangling.

3.4.2 Query visualisation

Multiple program visualisation systems exist for visualising SQL queries. Most of these systems are research prototypes deployed in individual courses, and few of them have been empirically evaluated. According to a recent systematic review of SQL education [62], only two visualisation-related studies included scientific data.

eSQL [211]: The first query visualisation software is probably the eSQL system from 1997. It shows a step-by-step display of how the result of a query is evaluated, intended for small databases of educational use. A query like **SELECT . . . WHERE . . . GROUP BY . . .** will return an intermediate data table highlighting the rows satisfying the **WHERE** condition, and another intermediate table highlighting the column selected from that subset, with blank rows separating each group. The system was not empirically evaluated.

SAVI [212]: Like eSQL, it visualises a query through a step-by-step display in the same tabular format, but unlike eSQL, it is browser-based and incorporates reversible animations to transition between intermediate states, in addition to simple colour highlights and explanatory messages. No pedagogical evaluation was reported.

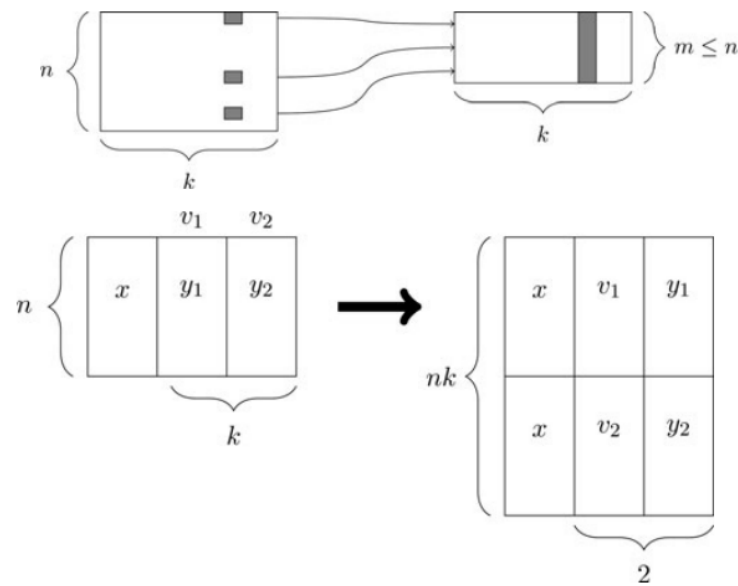


Figure 3.8: Diagram by Baumer [5] illustrating a filter and pivot operation

Dietric et al. [213]: This unnamed system introduced interactive animations that explain database concepts (e.g. decomposition, normalisation, primary and foreign keys) as well as how queries work. Visually, the design is similar to eSQL and SAVI, in that the data is visible, and highlighting is used to indicate correspondences. A single-group study found that their assessment scores increased significantly when using it.

SQL in Steps [214]: This is an online environment that allows users to develop queries incrementally, step by step. The user goes through a series of tabs corresponding to the possible operators and specifies arguments via a menu. A live view of the result is visible, although it does not incorporate any icons or multimedia principles. No pedagogical evaluation was reported.

viSQLizer [215]: A master’s thesis project that visualises query processing in a similar way to SAVI, but with more selective animations that make them quicker to run. A small user study compared it with another online tutorial lacking animations, and found that the results were similar.

3.4.3 Explanatory diagrams

Graphics serve a central explanatory role in many textbooks and online resources that seek to explain the semantics of various data operations. Baumer [5] reported his experience teaching a small data science course for liberal arts students with some programming experience. In his course, 3 weeks were dedicated to data wrangling in R and SQL. He asked the students “to think about a physical representation of what these operations do” and provided them with diagrams like the ones shown in Figure 3.8, though no formal feedback was reported.

3.4.4 Visual programming languages

Another strand of research concerns *visual programming languages* (VPLs), which refer to languages that are *specified* using a graphical interface, as opposed to mere visualisations of syntactic code. This includes visual query languages (e.g. [216]) and visual array manipulation languages (e.g. [217, 218]). Most research in this area dates back to the early 1990s, and most VPLs involve idiosyncratic visual grammars that are more symbolic than depictive, and therefore require considerable practice before making sense. Though interesting in their own right, we will not explore these further, since our goal is to teach syntax-based programming.

3.4.5 Node-link diagrams

Node-link diagrams could be used to depict the structure of inter-related datasets, as with entity-relationship diagrams. They can also be used to indicate dynamical relationships, such as the transition between computational steps, for example depicting the data flow from one function to the next [219]. *Dataflow programming languages* let users configure a digraph whose nodes represent processing steps and whose links represent data flow. Modern, GUI-driven data mining platforms for academic or professional use generally adhere to the visual dataflow paradigm. These include RAPIDMINER [220], WEKA [221], ORANGE [222], KNIME [223] and RED-R [224]. It is worth noting that, in these systems, the nodes themselves are generally black boxes, though sometimes decorated with icons (e.g. [224]). As such, they may be well-suited for orchestrating complex analytical pipelines, but less so for pre-processing tasks. No pedagogical studies of these have been published.

Another flow-based innovation are the various visual formalisms that have been created to facilitate SQL query writing and comprehension. For example, Taipalus has introduced a notation for planning SQL queries [6] that captures the logic of the query, by representing it as a graph where nodes are tables, edges are joins and specific columns or expressions appear as properties (see Figure 3.9). Another formalism is used in the QUERYVIZ system [7] (see Figure 3.10), which succinctly shows the attributes and relations relevant to a particular query, and various line types and bounding box elements to represent SQL constructs. Neither system appears to have been evaluated.

3.4.6 Dynamic previews

Kandel et al.'s WRANGLER system (which TRIFACTA is based on, mentioned in Section 2.3.3) recommends data wrangling operations to the user via a menu. Each operation can be *previewed* as a ghosted overlay within the data panel itself. This preview employs colour cues in various ways to convey operation semantics. For example, as seen in Figure 3.11, deletion will highlight the rows-to-be-deleted in red, and a table pivot will use colour to show correspondences between the pre- and post-transformation states. Their user study observed that “users relied almost

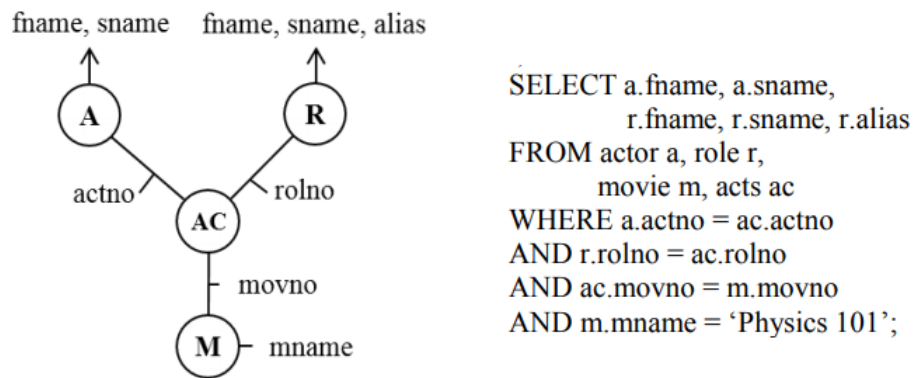


Figure 3.9: Taipalus graphical notation for relational database queries [6].

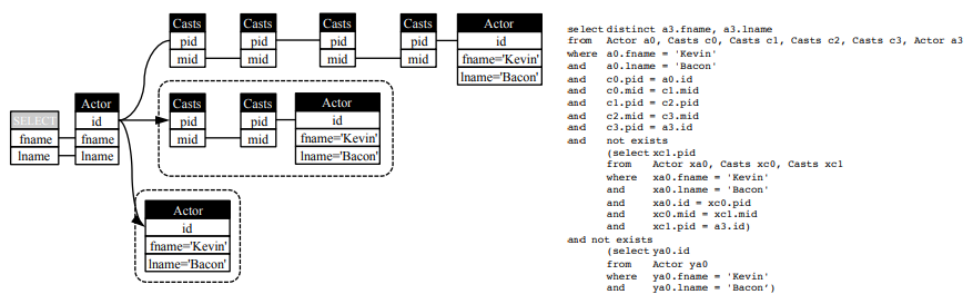


Figure 3.10: QUERYVIZ visualises the meaning of a relational query using a novel notation [7].

entirely on the previews” with one participant noting they “just look at the picture” [8, p.3371]. This feature was in fact rated as significantly more useful than the direct manipulation interface.

Another preview functionality is provided by Zhang and Guo’s DS.JS system [9], a plugin for manipulating data embedded within websites. DS.JS allows the user to click on a function within the code, which will trigger a visual preview of the transformation accomplished by it. The preview utilises cues such as highlighting to indicate selection, spatial separation to indicate grouping, and arrows to signify transition (see Figure 3.12). In their user study, featuring learners with prior data analysis experience, it was reported that participants used the previews extensively while acquainting themselves with the API. Among their suggestions, participants suggested an ability to “freeze-frame” intermediate tables to have them persist after clicking on an operation, suggesting a desire for more static visualisations.

3.4.7 Cheat sheets

RStudio hosts a popular collection of almost 60 cheat sheets that describe key R libraries and APIs [10]. Most of them incorporate graphical thumbnails and mnemonics to illustrate operations and core mental models, alongside simple code examples, see Figure 3.13. On their GitHub page⁵, they outline a set of visual design principles for contributors to conform to. This

⁵<https://github.com/rstudio/cheatsheets>

Figure 3.11: Kandel et al.’s WRANGLER uses colour cues to help their users understand the semantics of proposed operations [8, p.3366]. Seen are deletion of rows and a table pivot from long to wide format.

Figure 3.12: In Zhang and Guo’s DS.JS, clicking on a function triggers a preview that summarises the operation. These images show filtering and grouping, respectively. [9, p.697]

includes the use of a single highlight colour to be used throughout, a second colour for differentiating groupings, clear groupings using boxes and background colours, and plenty of white space. As they write, the cheat sheets are not meant as documentation, but as a quick reference for just-in-time learning:

“A cheat sheet is more like a well-organized computer menu bar that leads you to a command than like a manual that documents each command.”

The project is under a creative commons license and has, as of June 2021, 56 unique contributors, 900+ forks and more than 2900 GitHub stars. While the cheat sheets have not been subject to any formal evaluations that we are aware of, these metrics indicate an organically grown popularity in the R community and beyond: Pandas has since released their own version of a data wrangling cheat sheet ⁶, while DataCamp has created cheat sheets for NumPy ⁷. Entire communities exist that are dedicated to programming cheat sheets ⁸ but they do not necessarily incorporate thumbnail graphics to the same extent.

⁶https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf
⁷https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf
⁸<http://www.cheat-sheets.org/> and <https://opensource.com/downloads/cheat-sheets>

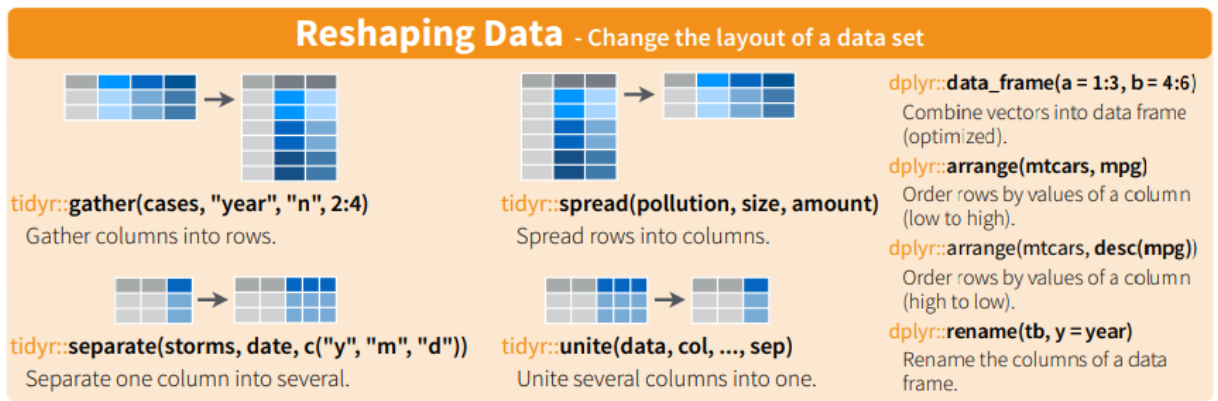


Figure 3.13: Snippet from RStudio's cheat sheet for the data wrangling library dplyr [10]

How are cheat sheets used in real life? A recent interview study with professional programmers suggests that programmers tend to use cheat sheets when switching between languages [225]. Pedagogical research into programming-related cheat sheets has mostly investigated them as student-generated crib sheets to bring into exams. Hsiao and Lopez [226] found, in the context of an object-oriented course, that these sheets tended to be organised and be more likely to contain code examples than other kinds of syntax description. The average cheat sheet only devoted 8% of the space to graphics but 40% of students systematically used highlighting to emphasise content. In the exam, the level of organisation predicted performance in declarative exercises, but not in exercises requiring procedural skills. De Raadt similarly permitted cheat sheets in an introductory programming course and found that students who included abstract representations (including diagrams) tended to perform better [227], while Hamouda and Shaffer [228], in a similar context, found that cheat sheet level of organisation was predictive only for comprehension, but not application. Of course, these observational studies do not reveal whether the sheet design led to performance improvements, or if those who created diagrammatic and well-structured sheets were already high-performing students.

3.4.8 Section summary

- The literature on programming and algorithm visualisation systems is mostly limited to imperative and object-oriented programming of general-purpose algorithms. Several query visualisation tools exist, but have usually not been empirically evaluated.
- Several dataflow programming technologies exist in which the user creates a data processing pipeline by directly manipulating a node-link interface. Furthermore, diagrammatic node-link notations exist for supporting query writing and comprehension. Again, these have not been pedagogically evaluated.
- Dynamic previews, illustrating how a data transformation will operate, have shown to be helpful and popular in two usability studies.

- Cheat sheets, which typically feature thumbnail graphics, are a popular resource in data science communities, but have not received much scholarly attention.

3.5 Scaffolding techniques

In this section, we will review instructional design innovations that focus on three stages of the programming process - plan composition, API lookup, and example adaptation - since these are the most amenable to graphical scaffolding. For each stage, we will discuss the role graphics could serve in augmenting various scaffolding techniques, and propose a specific graphical intervention. These proposals feed directly into our thesis statement, which states that subgoal graphics, thumbnail graphics, and parameter graphics facilitate programmatic data wrangling.

3.5.1 Plan composition

Plan composition and program design have frequently been scaffolded by guiding learners towards the type of top-down, breadth-first programming approach that research ascribes to experts. Pedagogical approaches on this theme have a long history, but have mainly been confined to object-oriented software engineering and imperative programming.

Pattern-oriented instruction

Within imperative programming, top-down approaches have often assumed the form of *pattern-oriented instruction*, which focuses on teaching transferable program design patterns explicitly. The approach is most closely associated with the composition of algorithms that combine loop-related patterns through nesting or interleaving (e.g. variants of loop with sentinel, loop with flag) [144, 229–231].

Such patterns are often practised through *worked examples*, a signature pedagogy in programming education [161]. Patterns could also be available for browsing within the IDE through pattern libraries, as with Ebrahimi’s VPCL (Visual Plan Construct Language) [232] and Guzdial et al.’s GPCEDITOR [185]

Examples of pattern-oriented SQL instruction do exist. Al-Shuaily [233] developed a set of patterns (e.g. dynamic filtering, subquery, self-join) where each pattern was described in terms of its purpose, when it is applicable, and a set of query examples. An evaluation study suggested that the pattern-oriented group had higher post-test scores compared with a control group given standard lecture notes.

Overall, we would expect pattern-oriented approaches to have limited applicability in data wrangling, since high-level patterns have mostly already been encapsulated by a function, (e.g. `map()`, `any()`, `all()`). Because the functions are in general highly optimised, they can be combined using simple pipeline sequences, without nestings or interleavings. The main

challenge in data wrangling would instead be identifying the sequence of steps necessary to accomplish the task.

Subgoal labels

Another well-documented scaffolding technique in CS education is *subgoal labelling*, which refers to a way of augmenting worked examples by segmenting them into meaningful sets of steps and annotating each segment with a short description. It can thus be seen as a form of chunking. By making the structure more salient and the problem decomposition more explicit, subgoal labels are believed to assist with schema formation [234].

In multiple studies by Margulieux et al. [235–237], subgoal labels were used to scaffold the learning of block-based app development. In that context, subgoal labels were used to group interface instructions (e.g. “1. Click on X, 2. Click on Y”) into more meaningful steps (e.g. “Create component”). Across these studies, subgoal labels have been associated with higher instruction completion rate and faster subgoal completion [234–236] though not all studies have replicated the result [238].

In another manifestation of the scaffolding dilemma, some indications exist that self-generated labels outperform instructor-generated labels [234], especially if encouraged through label placeholders [239]. However, this has been partially contradicted by another study by Morrison et al. [240]. That study involved loop patterns instead of app development, and subgoal labels were instead of the type “Initialise variables” and “Determine loop conditions”. The experimental group received subgoal labels, while a control group was given placeholders for writing their own, self-generated labels, but no main effect was detected. In another study, this time using Parsons puzzles, those *given* labels performed higher [241]. In yet another study from 1998, self-generated subgoal labels were incorporated as a feature of Guzdial et al.’s GPCEDITOR system [185] where they were visualised as a tree in a separate panel. In their evaluation study, they noticed that higher-ability students tended to skip past the subgoal-labelling step [185]. Overall this data pattern suggests that conclusions of whether instructor-provided or self-generated subgoal labels are more useful are premature, and likely to depend on prior ability, domain, and subgoal design.

It is worth noting that this subgoal-related research has mainly focused on labelling worked examples presented before or simultaneous with the actual exercise. The provision of subgoal labels *without* an accompanying solution - meaning, when there is no worked problem to base the solution on, only subgoal labels - has not been explored. This label-only scenario may be more applicable to API-reliant forms of programming like data wrangling, in which the learner routinely seeks external information of how to implement a subgoal.

1. Split rows based on region



2. Calculate the median income of each region



3. Take the largest and smallest median



4. Subtract the smallest from the largest



Figure 3.14: A demonstration of how subgoal labels could be augmented with subgoal graphics, for the example from Section 3.2.2.

Subgoal graphics

In the programming domains that previous subgoal label research has been focused on - block-based app development [234], imperative loop patterns [240], and object-oriented Java programming [242] - subgoals have not necessarily implied a visible change to a data structure. A subgoal label like “Determine loop conditions” is difficult to visualise, but a data wrangling label like “Filter the rows” produces an intermediate table that is straightforward to visualise and highlight.

If we revisit the example running through Section 3.2, on the difference in median income between the richest and poorest region, then we could use the proposed plan of Section 3.2.2 as subgoal labels. Moreover, we could augment those labels with subgoal graphics (see Figure 3.14) that visualise the state before and after a subgoal is accomplished, with colour used to emphasise the structural change it necessitates.

Such a feature would be consistent with the previous review of graphical scaffolding. When viewed through Ainsworth’s DeFT framework [206], a subgoal graphic can facilitate search for information relevant to their understanding of the solution and help constrain the range of inferences of what a subgoal label implies (e.g. how a subgoal modifies a data structure) . Viewed thus, subgoal graphics serve a complementary role to subgoal labels. Moreover, in providing a complementary representation, Ainsworth’s framework predicts that it can promote deeper, more schema-conducive processing [206].

Subgoal graphics are also consistent with the previously reviewed multimedia principles [179] (see Section 3.3.3). Consistent with the segmenting principle, subgoal graphics segment the solution. If placed immediately below subgoal labels, they observe the spatial contiguity principle, and if they use colour to highlight subgoal-relevant changes, they observe the signalling principle. If exact data values are omitted, they also adhere to coherence principle, by ignoring subgoal-irrelevant information.

We find no direct precedents for this idea in the review of data wrangling graphics in Section 3.4. However, it is evident that SQL query visualisation tools like SAVI [212] and QUERYVIZ [7] serve similar roles of visualising the logic of a solution, and theoretically could be used to scaffold plan composition. It is also clear that the dynamic previews in WRANGLER [8] and DS.JS [9] serve a similar purpose, albeit in that case of visualising a *potentially* promising operation, not the solution. Neither instance features subgoal labels.

What potential impact would subgoal graphics have within the API-reliant workflow of a data wrangler? When subgoal labels are formulated at an abstract level far removed from the data structure, such as “Find each region’s median income” instead of “Split rows...”, then subgoal graphics would constrain the translation of that label into data operations (i.e. a split-apply-combine operation). Beyond plan composition, this could help learners formulate productive queries when they are unfamiliar with the API’s preferred terminology, perhaps leading them to search for “Split rows based on column”. Subgoal graphics may therefore help learners with the last compilation step in converting the problem statement into executable code. Finally, it allows them to verify the soundness of intermediate results.

3.5.2 API lookup

We noted in Section 3.2.3 that implementing a program often requires looking up the syntax of relevant operations on-the-fly, which is hampered by a lack of awareness of API-specific keywords, difficulties with formulating and refining queries, and with deciding whether retrieved documentation resources are relevant or not. Tools aimed at mitigating these hurdles are generally focused on reducing the need for text-based search, by either passively or proactively displaying which operations are available and relevant. Moreover, to facilitate relevance judgements, the tools generally attempt to provide short summaries of how each operation works.

	Year	Property_crime_rate
0	Reported crime in Alabama	
1		
2	2004	4029.3
3	2005	3900
4	2006	3937
5	2007	3974.9
6	2008	4081.9
7		
8	Reported crime in Alaska	
9		
10	2004	3370.9
11	2005	3615
12	2006	3582

Figure 3.15: Interface of Kandel et al. WRANGLER system [8]

Mixed-initiative interfaces

Within data wrangling, one previously researched approach is *mixed-initiative interfaces*. These are interfaces that proactively recommend data operations, often ranking them based on estimated relevance [243], as with the WRANGLER [8] and DS.JS [9] systems (see Section 3.4.6). Usually these relevance estimates are based on community data-derived associations between API commands and certain code blocks. Such a ranked list could be integrated into IDEs to appear as a menu, within tooltips or through automatic code-completion.

The WRANGLER system, shown in Figure 3.15, has been evaluated twice, both times through small studies featuring professional programmers. The first was a within-group study that compared performance in Excel with a non-proactive version of WRANGLER that only recommends operations based on user interactions (e.g. manual row selections) [8]. It found that WRANGLER led to faster completion time, but that the efficacy of the mixed-initiative interface was ultimately bounded by participants' conceptual understanding of the operation and its relevance to their problem. Otherwise, the authors notices, users risked getting stuck in a cul-de-sac: upon choosing an incorrect operation, they would then be recommended further inappropriate operations. In the second study, the same non-proactive version was compared with a proactive version that suggested operations based on static analysis [243]. The authors found that participants often ignored proactive suggestions and only used them as a last resort, perhaps due to insufficient visibility or a general distrust of popups.

Outside of data wrangling, the HCI field has experimented with IDE plugins that automatically search for applicable code examples and retrieve them in an easily auto-integrated format. One such system, BLUEPRINT by Brandt et al. [244], was evaluated in a user study with professional programmers, and found that programmers performed better than the control group.

There are also some examples of recommendation engines for novices. Ichinko et al. developed a system called the THE EXAMPLE GURU that, based on static analysis of the current code state, suggests context-relevant API commands [245, 246]. The API in question was the

Looking Glass API for 3D animations, which is programmed using a block-based editor. They compared the use of the THE EXAMPLE GURU with a control condition that simply featured an embedded documentation panel, and found that novice programmers used more than twice as many novel API commands with the Guru. Importantly, the project was an open-ended project, and did not have a clear end state the way a data wrangling task usually has.

Menu-based solutions

In less open-ended environments - as an educational workbench with a well-defined syllabus is likely to be - the set of operations may be small enough for a static, menu-based solution to be feasible. This approach tends to be used in block-based programming interfaces. Block interfaces typically feature a *palette* - a static toolbox, usually displayed in a left-side pane, that is designed to make the available commands easier to browse through. Often these are organised categorically or hierarchically [177], and this structure could help scaffold the learner's own mental organisation of the domain. Moreover, Weintrop and Wilensky have noted [177], browsable categories form an easily navigable "memory cache" to inspire solutions and enhance discoverability.

The cheat sheets reviewed in Section 3.4.7 could be viewed as analogue versions of such a palette - indeed, RStudio's GitHub page characterise them as such [10]. Following their finding that most of their students' R interpreter errors were due to misnaming of variable names, Rafalski et al. [109] suggest providing cheat sheets of syntax and naming conventions as a teaching tool. Therefore, although menu-based solutions have received little attention outside of block-based contexts, there are reasons for believing that they could be an effective way of providing documentation. This assumes that the programming domain is closed-ended, as with an educational syllabus, since more open-ended projects would require access to search engines.

Thumbnail graphics

For a static menu to be navigable, its overall structural organisation and the purpose of each individual command needs to be transparent. In block-based programming, the shape, colour and names of blocks help signal their function. More generally, *thumbnail graphics*, graphical summaries, are a common aid for facilitating the user's evaluation of the contents' relevance [247]. We already saw them in the context of documentation cheat sheets, in which they were used to succinctly illustrate API commands' behaviour. The dynamic previews of WRANGLER [8] and DS.JS [9] can also be seen as instances of such thumbnails, and their user studies revealed that those previews were extensively relied upon. While they have never been the subject of their own dedicated experiment, there are reasons for believing that thumbnail graphics could meaningfully enhance the navigability of a command menu.

3.5.3 Example adaptation

In Section 3.2.4 we noted that the eventual step of code implementation can be hampered by opaque API documentation, a failure to understand code examples, and difficulties in making the necessary adaptations to code examples in order to integrate them into the solution. Scaffolding aimed at lowering these barriers would presumably guide the example adaptation either directly, by providing a partial example-to-problem mapping, or indirectly, by improving the visual presentation of code examples.

Inherent documentation

Modern APIs already come equipped with extensive *inherent documentation* [69] that integrates documentation into the IDE, via intelligent code-completion and tooltips that show a function's syntax signature when a function is clicked upon within the code. This prevents the user from having to continually cross-reference with an external documentation source, which in turn is likely to facilitate example integration. The efficacy of these is ultimately bounded by the clarity of the API: if the parameter names are unclear, and if the documentation is verbose, they would be of limited utility. Even if the future makes auto-parameterisation of functions an IDE standard (as with the SNIPMATCH plugin [248]), the user needs to be able to verify the soundness of its suggestions.

Fill-in-the-blank exercises

In online data wrangling platforms like `DATA CAMP` (`datacamp.com`) and `DATAQUEST` (`dataquest.io`), a typical course consists of slideshows that present example code snippets of a particular API command, and then asks of the learner to invoke that command on another data set. In `DATA CAMP`, the exercises are often scaffolded by providing a partial solution with fill-in-the-blank fields, leaving a few function names or arguments blank. Often more and more steps are omitted over time, to help students transition into independent problem-solving, as with faded worked examples [249]. Fill-in-the-blank exercises thus serve to constrain the range of possible errors that a participant can make in their configuration of an API command. Such exercises can be automatically generated [250], and computational notebooks like Jupyter are well-suited for delivering them [251].

Theoretically, this could be scaffolded even further, by narrowing down the possible arguments into a multiple-choice question. Block-based editors provide their own version of this: the creators of `BLOCKPY` argued that, since block primitives can be configured through drop-down menus and togglers, they are especially well-suited to the heavily parameterised design of data science APIs [96].

Whether fill-in-the-blank exercises are preferable ultimately depends on the value attached to keeping the programming experience authentic or not. Encountering an empty code editor is

likely to arouse anxiety in a complete novice, but this could be theoretically mitigated by other means, for example by gradually increasing the complexity of the problem, and through the provision of subgoal labels.

Code example design

Examples can be made easier to comprehend through certain design considerations. For example, syntax highlighting is a near-ubiquitous feature in code editors and syntax documentation alike, and serve to visually separate language keywords from variable identifiers and arguments.

There are also other design dimensions worth considering. A study analysed the characteristics of popular answers on StackOverflow and found that, compared with low-scoring answers, popular answers tended to be concise and to omit less relevant code [252], consistent with the coherence principle of multimedia theory [179]. Examples also tended to highlight the most critical elements (observing the signalling principle) and to divide the code into smaller steps (the segmenting principle). Another finding was that popular answers tended to use the same context as the question, effectively saving the poster the effort of having to do the example adaptations.

Others have experimented with visual formatting dimensions. Ichinco and Kelleher [253] asked children what they noticed in a set of block-based example code snippets, which varied in their style of visual emphasis: using a drop shadow, arrows, or outlines combined with fading. They found that students were most successful when critical code snippets were given a drop shadow. Another study by the same authors [254] tested the ease with which programmers recalled elements from block-based versus textual code examples. While the code examples in blocks used bounded boxes to visually separate keywords from arguments, the textual code only used syntax highlighting. The study found no strong differences associated with format.

Providing a prose explanation of how the code example works is not necessarily beneficial to the example integration. In yet another study, Ichinco et al. [164] explored three ways of annotating examples within block-based environments. This included providing a brief summary of code behaviour, line-specific annotations, and highlighting of critical code elements. The study found that the three types are better than a complete absence of annotations, but that the three styles did not significantly differ in their associated performance levels. Another study, by Thayer et al. [115], manipulated the extent to which participants received explanatory annotations while completing tasks in unknown JavaScript APIs. They obtained mixed results, where annotations improved progress only in some tasks, possibly due to annotation quality.

Parameter graphics

Example adaptation requires the learner to mentally map the example context to their problem context. This presumably would be easier if the learner had a clear expectation of the parameters defining a particular command, and if the representation of these parameters remained consistent across APIs. For example, metasyntactic variables like `foo` and `bar` are used across

languages [163] and, by the same token, variable names like `df` for dataframes and `col` for a column identifier could theoretically be standardised for data wrangling APIs. Alternatively, the role of a variable could be signalled within the documentation through a graphic that visualises the parameter (e.g. the dataframe or column). Graphics could thus serve as placeholders to signal to the user “what goes where”, reducing the need to peruse lengthy documentation entries. Parameter graphics have not been previously explored in the literature.

3.5.4 Section summary

- Plan composition in data wrangling problems is highly amenable to subgoal labelling, since the plans are usually simple sequences of steps. Subgoal labels could theoretically be combined with subgoal graphics that visualise each step.
- In educational settings, API lookup can be scaffolded through menu-based solutions, which could be either static or powered by a recommendation engine. Thumbnail graphics could theoretically make the menu entries easier to comprehend.
- Example adaptation could be scaffolded by providing partial example-problem mappings, as with fill-in-the-blank exercises. The visual appearance of examples, such as segmenting and highlighting, could also benefit the process. Consistent parameter names or parameter graphics that depict the parameter could help the user comprehend which value should be passed where.

3.6 Conclusion

Data wrangling involves a complex, iterative workflow with three core stages: plan composition, API lookup, and example adaptation. Each of these involve cognitive demands that vary in their relevance to the task and to schema acquisition. Various graphical scaffolding strategies could be used to increase the salience of more schema-relevant information. For example, plan composition could be scaffolded through subgoal labels and subgoal graphics. API lookup can be scaffolded through menu-based solutions that incorporate thumbnail graphics. Example adaptation can be scaffolded through documentation sources that utilise multimedia design principles, and potentially also parameter graphics that illustrate the argument that should be passed to a particular parameter. However, whether subgoal graphics, thumbnail graphics and parameter graphics have added pedagogical value, beyond that provided by simple text, has not been previously researched, and forms the research subject of the current dissertation. Hence, **our thesis statement is that subgoal graphics, thumbnail graphics, and parameter graphics facilitate the learning of programmatic data wrangling.**

Chapter 4

Method

A research programme is shaped by a number of considerations. Some of these concern value judgements regarding which methodological properties should take priority over others. Others concern philosophical commitments regarding what abstract concepts like *population*, *effect* and *measurement* actually mean in a particular context. Still others involve practical realities, such as the cost of recruitment, logistics and other resource constraints. This chapter aims to make these background assumptions explicit and justify the chosen research method, which is a series of small exploratory studies culminating in a larger, more summative capstone study.

4.1 Breadth versus depth

One of the most consequential dilemmas a researcher is presented with is the *exploration-exploitation trade-off*: whether to allocate limited resources in a deep or broad fashion. A depth-first research programme would focus on providing a maximally reliable answer to a single question, which would involve an extensive validation process and repeated replications. By contrast, a breadth-first approach would spread the focus more widely, answering multiple questions, but sacrificing their answers' reliability in the process.

Nelson and Ko offer a useful framework for describing how CS educational researchers can choose to allocate their resources [11]. In Figure 4.1, three different allocation patterns are shown. If the goal is to advance a learning theoretical explanation, then an educational intervention only needs to be adequate, not a perfectly polished “high-fidelity” intervention, and most resources are instead spent on validating measures and conducting a high-powered experiment (the top pattern). However, if the goal is to optimise learning in a specific domain such as data wrangling, it makes sense to focus on broad design explorations, sampling as many parts of the design space as possible (the bottom pattern), without ambitions to test a general learning theory.

This dissertation is concerned with evaluating the impact of a particular pedagogical intervention, namely graphics describing data wrangling operations. Such graphics inhabit a design

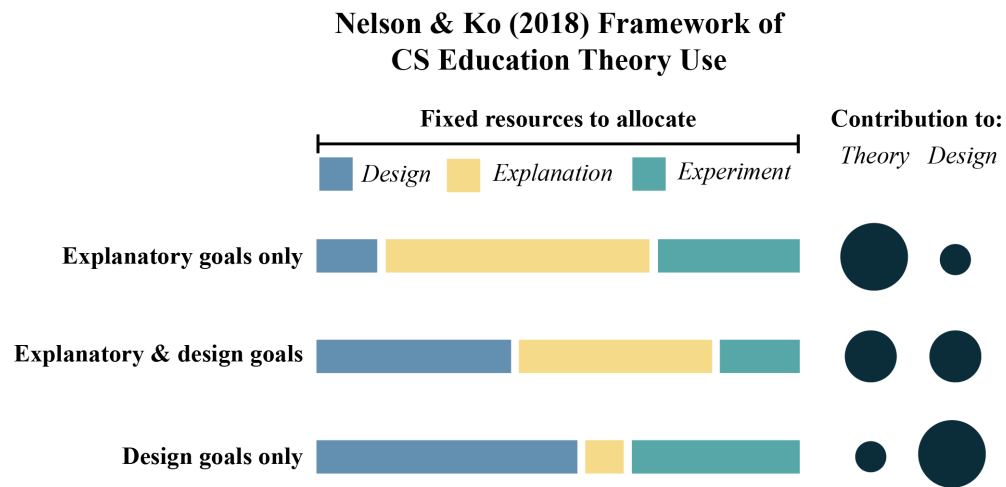


Figure 4.1: CS education research generally faces a trade-off of whether to contribute to theory advancement or design progress. This dissertation is mainly focused on refining and evaluating a particular design feature (*Design goals only*). Diagram adapted from Nelson and Ko [11, p.34].

space, defined by dimensions like colour palette, whether or not data are included, whether highlights are used, and so forth. These graphics will be embedded in an instructional design context that is likewise subject number to a long sequence of design choices. Although digital platforms routinely make such fine-grained design choices on an empirical basis using A/B testing, that basis is often made up of 10,000s of roaming online visitors. Without such a vast participant pool, it is down to exploratory studies and the experimenter’s own intuitions to develop the materials.

In this case, this means that the intervention (the graphics) and the accompanying apparatus (a data wrangling tutorial) will be developed with relatively little rigour, based on the literature, small pilots, and to some extent personal judgement. However, once this exploratory design process has converged on a particular design, the task becomes one of creating generalisable, reliable knowledge regarding the pedagogical effects of data wrangling graphics. After a main design variant of the graphics and apparatus has been chosen, even though it represents a virtual sliver of the design space, it will therefore be submitted to a large summative trial. Most of the remaining chapter will discuss considerations regarding this larger capstone study involving SLICE N DICE, our e-learning platform for data wrangling instruction.

4.2 Observational versus experimental designs

This dissertation is interested in making causal claims: do tabular graphics cause educationally relevant effects? Such causal questions could be answered with varying degrees of rigour. By *rigour* we mean the design’s ability to eliminate competing, equally plausible candidate explanations for any observed effect.

One approach we could have chosen would have been to make use of naturally occurring data, as part of an observational design. For example, the graphical aids could have been made

available for students to use, and the number of times they were accessed online could have been correlated with the observed performance in a data wrangling assignment. Such a design would be unobtrusive, naturalistic, and cheap to administer. However, it is likely that those seeking graphical aids are more motivated or curious than those who do not. Observational educational studies are likely to overestimate effects [255], even when potential confounds have been controlled for [256].

This is why the capstone SLICE N DICE study will follow an experimental design. It is worth remembering, however, that causation is not the only aspect that matters when evaluating an intervention: whether students voluntarily seek out graphical aids, and whether they are organically popular, remain important predictors for actual usage and adoption. For these questions, which we will not cover, observational data do have an important role to serve.

4.3 Randomised controlled trials

In designing educational experiments, an implicit analogy is often drawn between the evaluation of an education intervention, and that of a therapeutic intervention. The latter are typically conducted using a *randomised controlled trial* (RCT) design. Generally considered a “gold standard” in medical research, an RCT effectively simulates a counterfactual: using random assignment, systematic differences in baseline characteristics are removed between groups. Any group differences that result are therefore attributable to the treatment, making RCTs an ideal instrument for causal inference.

RCTs are rare in CS education research, however: according to a 2016 review, they made up only 3-8% of recently published research [257]. Several structural reasons could help account for this: educational RCTs are resource-intensive to conduct, and face obstacles that clinical RCTs do not have. In this section we will identify such issues and specify how they will be addressed.

4.3.1 Specificity of the intervention

In a clinical trial the intervention is generally reproducible - little will vary from one vaccine injection to the next - but the administration of an educational intervention is rarely codified to that degree [37, 258]. This is especially the case when the intervention is human-delivered, as teachers could interpret the intervention differently [259]. To ensure the intervention’s integrity, the most scalable solution would be to computerise it completely, which is what we will do.

It is important to note that such a specificity in the intervention also implies a specificity in the claim. In using a completely fixed intervention, we are not testing the “effect of graphics”, but rather the effect of a specific set of graphics in a very specific task. Theoretically, this lack of generality could be ameliorated by loosening up the design specifications, or by seeking replications by others [258, 260]. For example, we could have recruited a group of teachers to create

their own graphics and evaluated those, to gauge whether the description of subgoal graphics and thumbnail graphics are reproducible enough. However, such a range is not necessarily a priority, since the graphics themselves could be reused. In our case, we content ourselves with knowing that the materials will be made available, and remain cautious of extrapolating any claims beyond this particular set of graphics.

4.3.2 The choice between rigour and generalisability

The degree of control in the intervention has a close analogue in the degree of control over the circumstances in which participants engage with it. Within clinical trial design, *efficacy trials* take place under laboratory conditions that are kept as consistent as possible, to maximise the statistical power and rigour. *Effectiveness trials*, meanwhile, strive for field-based settings representative of the target environment in which learning is likely to occur, to ensure that the effect remains meaningful across variable contexts. The trade-off between rigour and generalisability mirrors that between depth and breadth: the more robust you want a finding to be, the noisier conditions you will have to test the intervention in and, muddling the causal account.

We have already established the intention to computerise the study, but computerised studies vary in their control. An ideal efficacy trial would have taken place in a laboratory setting, where the student sits in an isolated booth, instructed to leave the smartphone in the pocket and to concentrate on the task at hand. A more naturalistic study would be more like a MOOC, where the participants choose their location and schedule their practice as they please.

How robust do we want our documented effect of data wrangling graphics to be? Both efficacy and effectiveness trials face up against recruitment as a limiting factor. In principle, a highly controlled efficacy trial would have been the more desirable option, for reasons that have to do with the need for statistical power (see Section 4.3.10). However, such a strict experimental protocol would have been difficult to recruit for: laboratory conditions, even where remote, incur a degree of discomfort, making it unlikely for volunteers to sign up to it without compensation, which for multi-hour tutorials quickly become prohibitively expensive. They would also be inappropriate during COVID-19, and not scaleable.

Therefore, we settled for an online at-home study, which arguably hits a sweet spot that represents a realistic learning scenario in which many end-user programmers learn data wrangling today, while also being relatively shielded from extraneous factors like collaboration among participants, or excessive tutor assistance.

4.3.3 The logistics of separating conditions

The decision to computerise the study and let students complete it at home helps address another issue: that of randomly allocating and then blinding participants to their assignment [261]. Even for computer-mediated interventions, if embedded within a physical classroom, there is a risk

of students peeking on each others' monitors and thus inferring the experimental manipulation. While a clinical trial participant is unlikely to understand the details of their treatment, university students may have sophisticated understandings of pedagogy themselves, and therefore second-guess the manipulation. Solutions to this generally require physical separation, such as letting students complete the experiment from home, in which case the only risk to participant blinding would be if they share their screens with each other.

4.3.4 Designing a fair control condition

An RCT implies a control group, which in medicine is straightforward enough: the patient receives a biochemically inert placebo, or the currently mainstream treatment benchmark. In education, however, there is rarely an "inert" condition available, or a standard practice default. This makes the idea of *comparative effectiveness* - of optimising education through head-to-head comparisons among alternatives - difficult to implement.

It is probably for this reason that CS education research is dominated by single group designs, with neither a control group nor baseline [257]. The issue with such designs is that, as stated by Cook, they effectively ask "If you teach them, will they learn?", which in itself is not informative, as teaching tends to beget at least some learning [262]. Equally, if an educational study has a control group that is not given *any* intervention, it just confirms that learning is possible [258]. To constitute a fair comparison, the control group must be given materials that appear to be just as pedagogically potent, and what Larkin and Simon have called *informationally equivalent* [204]: the control group cannot be denied information that is necessary, or an obvious shortcut, for succeeding in the task.

A related issue is the need for effect interpretability [258]. If an intervention and its control condition vary along more than one dimension, we cannot know which dimension we should attribute the effect to. If one object is blue and square, and the other is red and round, we do not know whether the preference for the latter is due to its shape or colour. If we gave 50% of the participants a carefully designed tutorial that incorporated graphics, and gave the other 50% an excerpt from a standard textbook, we do not know whether the superiority of one is due to its use of graphics or other characteristics.

What would an appropriate control condition be for data wrangling graphics? The most straightforward control condition would be textual subgoal labels and menu descriptions, and for the treatment condition provide subgoal graphics and thumbnail graphics *in place of* text, by way of a 2x2 factorial design. This would allow us to state that one kind of representation is superior to the other, assuming that they were kept reasonably informationally equivalent.

Providing text is not incompatible with providing graphics, however. On the contrary, Ainsworth's theory of multiple representations [206] suggests that text and graphics could act in synergy by complementing each other. It would therefore be more informative to let the experimental conditions provide graphics *in addition to* (not *in place of*) textual descriptions. This

means that performance improvements can be directly attributed to the addition of graphics, and allows us to potentially measure their “added value”.

The second choice - which is the one used in SLICE N DICE¹ - is not immune to criticism. One could argue that providing two complementary representations biases the comparison, by giving the experimental group access to *more* information. However, the provision of an additional representation is not a guaranteed bonus: it could be both redundant and distracting. Another drawback is that the subtlety of the experimental manipulation can make any true effect difficult to pick up. This drawback, which is a serious one, will be addressed in Section 4.3.10.

4.3.5 Attention as a mediating variable

In clinical trials, it can generally be ascertained whether the treatment was properly received by the patient. By contrast, a student’s engagement with data wrangling graphics is largely covert [37]. By itself, a simple RCT design cannot disentangle whether an intervention’s null effect was due to inherent defects, or whether the learner simply ignored it. To do so, one would have to measure the participants’ attention to the intervention, since attention mediates its effect.

Conversely, if there *is* a measurable effect, but attention levels are higher in the treatment condition, that would suggest that it is its motivational properties, rather than cognitive properties, that is the “active ingredient”. One could also argue that the distinction between “failure to teach” and “failure to motivate” is inconsequential, because motivation is a necessary prerequisite for learning. However, if the graphics merely serve to motivate, educators could search for cheaper and more targeted ways to motivate, such as verbal encouragement.

A computer-based apparatus allows us to capture engagement metrics such as cursor movements and keyboard presses. Short of brain imaging and eye tracking, digital instrumentation presents the most scaleable way of measuring whether data wrangling graphics are actually attended to, which is the approach we have gone for.

4.3.6 The lack of educational measurements

In a clinical trial, the dependent variable could be hospitalisation rate or blood pressure. In education, the dependent variable is generally a complex and vague knowledge construct, operationalised through an arbitrary set of exercises. Even a quantitative metric, such as time on task² or the proportion of correct responses, is only interpretable in the context of that specific set.

Because of the arbitrariness inherent to educational measurements, and the lack of obvious controls in educational research, there is a high risk of using *treatment-inherent measures* [263]. For example, if one group is taught data wrangling using graphics, while the control group is not,

¹With one exception: in SLICE N DICE’s operation cards, the descriptions will be *either* graphical *or* use a syntax-like representation.

²We use *time on task* to denote completion time, to distinguish it from time off task such as time of inactivity.

then a treatment-inherent measure would be one that asks participants to depict their solution graphically, or to identify a graphically depicted solution. This creates a circularity: if the test involves graphics, then the most effective training will almost certainly also involve graphics. As a result, experimenter-made measures are associated with inflated effect sizes [264].

One way of mitigating this in theory is to use standardised instruments or concept inventories as shared benchmarks. Unfortunately, the scarcity of such instruments in CS education severely restricts which research can be done [265]. Validating an instrument is a long, iterative process that requires a sample size of several hundreds [266]. Once validated, it is at immediate risk of saturation (e.g. being made accessible online [267]) and Goodhart's Law (i.e. becoming a target that the educator overfits the intervention to). For the purpose of evaluating graphics, it is simply not cost-effective.

The way we have addressed this issue is by measuring performance through an actual programming task, for example using time on task, completion rate, and number of incorrect attempts. These are still intrinsically meaningless (in the sense that they only make sense in terms of the specific exercise set), but are at least neutral with respect to graphics.

4.3.7 Ethical implications

In an RCT, a clinician may assign some people to a potentially detrimental treatment, or withhold a superior treatment from them. For this reason, RCTs should ostensibly only be conducted when there is genuine uncertainty regarding the therapies' relative merits (so-called *equipoise* [268]). However, because directional hypotheses tend to be what justifies a study in the first place, genuine *equipoise* is rare [269].

At first sight, there are few ethical issues attached to educational RCTs - certainly nothing comparable to those faced by medical researchers. Educational interventions are generally non-invasive, and impart at least *some* benefit - just more or less of it. Nevertheless, course performances contribute to grade averages, and grade averages matter to career success. To mitigate any such inequities, the British Educational Research Association (BERA) recommends delivering the intervention to the control group after the study has been concluded [270].

In the current case, we do not have true *equipoise* - based on the literature review and subjective beliefs, we have stated directional hypotheses that subgoal and thumbnail graphics improve a variety of metrics. Consistent with BERA's recommendations, we will offer participants a post-experiment remedial resource that includes graphics, so that the control group also can avail themselves of them.

4.3.8 Recruitment of volunteers

To ensure the availability of participants, the most efficient way of recruiting would be to embed SLICE N DICE into the delivery of a course, perhaps even measuring performance via the mid-

term or final exams, thus guaranteeing both recruitment and retention at no added cost. But when the design is an RCT, and when the task is cognitively taxing or of no direct relevance to their course performance, this becomes harder to justify. Furthermore, it depends on the sustained and extensive involvement of a lecturer in co-designing and organising the tests and materials. A stressed academic may have little incentive to do unless education is their primary research field.

This leaves us with unpaid volunteers and a non-credit bearing opt-in mechanism. The dependence on unpaid volunteers presents its own threat to validity, however, as the more intrinsically motivated students would be more likely to volunteer for it - the very students who could probably learn the topic successfully on their own anyway. Attrition may similarly be non-random. Such volunteer biases may be partially mitigated through post-hoc analyses, but will unavoidably haunt any volunteer-dependent RCT. Furthermore, the lack of direct, academic stakes could also affect data quality, as participants may not be motivated to do their best.

Another issue with relying on volunteers is that, to obtain high retention rates, a minimally viable research prototype probably would not be enough: SLICE N DICE would have to be as sleek, attractive, and intrinsically rewarding as possible. Relying on volunteers to sign up also means the researcher has to divert significant resources towards marketing.

Despite the numerous drawbacks and validity threats associated with volunteer-based education research, it is the only affordable solution for a mid- to large-scale RCT. The diversion of resources into making the intervention attractive could be touted as a benefit: it makes the materials more directly reusable, since it does not require instructors to create their own polished implementation of it.

4.3.9 Process metrics and product metrics

In a typical educational RCT, there is a distinct training stage, in which the intervention is manipulated and where mediating variables such as engagement and other interactions may be recorded. This is then followed by a test stage, where the scaffolding and feedback mechanisms are removed, and which is designed to test more durable learning achievement. Metrics of the former kind are sometimes called *process data*, while the latter kind is called *product data*. In their review of CS educational measurements, Margulieux et al. [271] called for researchers to include both, which computerised instrumentation readily permits.

In our study, we were wary of a design that separated training from testing. The issue mostly arose from the reliance on volunteers: asking of participants to complete tests without any feedback would require them to commit time to something they did not learn from immediately, and may trigger frustration or test anxiety, increasing the risk that they drop out. Although we did introduce one test-like condition in which scaffolding was temporarily removed for a certain time limit, mostly we were interested in making the learning process itself more efficient. We argue that, if data wrangling graphics improve the efficiency with which participants look up

syntax and infer which operations to use, then that fluency would translate to shorter time on task and fewer errors when aggregated across the entire exercise set.

Expensive as independent samples are to acquire, we also decided to collect a wide array of subjective metrics from each participant. This is not pure data greed: it is possible that the behavioural, cognitive and attitudinal effects of an intervention vary in their magnitude or direction. However, since it is difficult to make confident, well-supported predictions for all such variables, the study also becomes more exploratory than strictly confirmatory.

4.3.10 The challenge of statistical power

Looming large among the discussion of RCTs is the statistical concept of *power*, i.e. the probability of an experiment detecting a true effect. The issue of under-powered studies has dominated meta-scientific commentary for more than 30 years [272]. Much of this discussion has revolved around sample size. In psychology, the median cell size in 2011 was 24 [273]. A CS education review of 2013-2017 papers, conducted by Margulieux et al. [271], found that quantitative studies had a mean size of 401 and a median of 100 participants (probably on account of quasi-experimental classroom interventions). Because small samples have higher standard errors, they have a higher false negative rate *and* a higher false positive rate, meaning that the observed effect is more likely to be a fluke [274].

The sample size necessary for a study depends on *a priori* estimates of the effect size. However, unless a study forms part of a long series of replications (which are very rare in CS education [275]), there is usually little to anchor this estimate in. Furthermore, effect sizes are highly context and procedure-dependent [255]: perhaps counter-intuitively, rigorous RCT designs tend to show *smaller* effect sizes [264]. A 2019 meta-analysis of large-scale ($n > 1000$) educational RCTs suggests that average effect size is around 0.06 *SDs* [276], which, if used for sample size calculations, would mean that 20,000 participants would be necessary [276]. Even for regular “small” effect sizes, the required sample size (for a power of 0.8) is around 800.

One reason for the disappointing effect sizes of large RCTs is that multi-institutional studies are inevitably nested into smaller classes such that sample heterogeneity risks offsetting any gains in precision conferred by the larger sample size [37]. As a result, according to Norman [37], interventions “will likely be lost in a sea of unexplained variance” and are often not worthwhile. As a solution to this catch-22, Norman proposes focusing instead on reductionist efficacy trials with homogeneous samples [37] that are then replicated across many labs [277]. However, this brings us back to the expenses and logistical challenges of recruiting for efficacy trials, leaving us between a rock and a hard place.

The work undertaken in this dissertation lacks a clear precedent to base a priori sample size calculations on. Given its randomised, experimental, and reductionist nature, we should expect the potential effect to be small. Though we aimed to recruit widely to achieve a sample size of at least 200, take-up within each individual cohort it was advertised to was not large enough for

Norman's vision of a homogeneous sample [37] to be feasible. Instead we recruited small numbers from multiple institutions, and relaxed homogeneity restrictions. To boost power, our main strategy will instead be to collect interaction data from the experiment, since short-term variables and process variables tend to be more responsive to interventions than post-test achievement variables [255, 276]. Ultimately, however, we must face up to hard, practical constraints and accept that even an educational RCT with hundreds of participants may fall shy of desirable power benchmarks, and that pedagogical effects are realistically smaller than the existing publication record may suggest [255].

4.4 Conclusion

From our review of methodological considerations, and from our experience in conducting the parameter graphic study of Chapter 5, a research programme emerges that begins with an exploratory, iterative refinement process, and ends with a large RCT. This RCT will be administered through an e-learning platform, called SLICE N DICE, since e-learning permits easy randomisation, reduces contamination, and ensures reproducibility. To economise with participant requirements, it will manipulate the provision of subgoal graphics and thumbnail graphics as independent factors of the same study. Numerous dependent variables will be measured, to capture multiple aspects of data wrangling graphics' impact, and because proximal process variables are more statistically powerful than product variables.

Theoretically such a tutorial could be integrated into an existing course. That has many benefits (participant supply among them) but it could also be open to roaming online visitors. To appeal to as broad an audience as possible, and fit into as many course contexts as possible, it would need to be short and generally applicable (i.e. not tailored to a particular subject). One could think of such an e-learning tutorial as a "capsule intervention", to be slotted into a variety of contexts.

A capsule intervention would incur no extra effort on the lecturer's behalf (indeed, it would potentially save them effort) or any risks of interfering with a pre-existing curriculum. It could be embedded in a tutorial, as preparatory reading before the course starts, or simply as homework. Depending on local ethical guidance, it could provide formative feedback or simply exist as an optional activity. This allows the study to be more easily launched in multiple institutions, and thus adds generalisability, albeit at the cost of statistical power.

An open capsule tutorial like SLICE N DICE would not be without weaknesses. Like any low-stakes voluntary activity, it would have to be carefully designed to be intrinsically enjoyable. It may have to support multiple programming languages in order to have wide enough appeal, and volunteer bias is likely to be an issue.

Chapter 5

Parameter graphics

This chapter documents two studies undertaken within the first third of the doctoral programme, which resulted in a 2019 publication [278]. Both studies involve a paper-based experiment in which volunteers completed example adaptation exercises in three different languages (SQL, R, and Python), aided by a short tutorial and a set of syntax cheat sheets. The cheat sheets incorporated **parameter graphics**: small icon graphics representing the type of value that should be passed to a particular placeholder slot within the API command syntax. Study 1 presents a single-group study that trials the use of parameter graphics, while in Study 2, a random sample of the participants were given a control condition in which these parameter graphics were removed, thus forming an RCT. The main goal was to measure whether the parameter graphics improved example adaptation performance, to address **RQ3**¹.

Like later research, this project centres API lookup and code example adaptation as skills that can be explicitly taught, and presents a graphical scaffolding technique to facilitate it. Unlike subgoal or thumbnail graphics, however, parameter graphics do not visualise a data structure being wrangled. Rather, they visualise something more abstract: information needed to configure the data operation. As such, they are not directly continuous with later research, but the studies nonetheless played a formative role in shaping the remainder of the research programme, by laying bare the logistical challenges of “unplugged” programming research, which will be discussed towards the end of the chapter.

5.1 Background

The act of looking up API syntax requires continually comparing API documentation and example code with a particular problem context. This requires parsing code examples to understand what each identifier represents. Suppose you have searched for “How to average each group”, with the intention of aggregating the prices of various car brands, and the results yield the fol-

¹While the order of the research questions reflect the programming workflow, the dissertation is chiefly organised in chronological order.

lowing sample snippet:

```
animals.groupby('species')['height'].mean()
```

Adapting this example to your current problem context involves analogising, a process that Gentner called *structure-mapping* [279], in which *species* and *car brands* are brought into alignment as corresponding attributes, and as instances of the more abstract placeholder *column to group by*.

The challenge of parsing a code example could be characterised as one of *cross-referencing*: it requires spatially separated, mutually analogous sources of information to be continually compared. It is spatially separated, because a description of the API syntax may be either completely absent, embedded in surrounding documentation paragraphs, or put in lengthy parameter description lists. Similarly, your own problem context may be in a separate IDE window and not easily juxtaposed with the code example. The process of figuring out “what goes where” in a function signature consequently involves cross-referencing between the code example, code example explanations, and problem context. If cross-referencing is cumbersome, it presents one of the most straightforward sources of extraneous cognitive load, as already discussed in Section 3.3.1. When multiple pieces of information need to be integrated, the distance between them has no intrinsic value, except possibly training the working memory. Reducing this burden would be especially pertinent to programming novices whose working memory is more likely to be already near capacity.

5.1.1 The split-attention effect

Within multimedia design theory, the phenomenon of cross-referencing as a source of task-irrelevant cognitive load is referred to as the *split-attention effect*, or equivalently, the *spatial contiguity effect*. It is most typically exemplified by the case of a picture and explanatory caption, for which the effect predicts that spatial adjacency leads to better performance in tasks that depend on the working memory [280]. Experimentally, the evidence for this effect is robust: a recent meta-analysis by Shroeder and Cenkci found that integrated designs perform better in a variety of different domains [281]. The reason for this is that spatial separation requires the learner to repeatedly search for related information, a search that exacts a cognitive cost and further requires them to store information in working memory for longer [282]. The cost incurred by spatial distances appears to cause learners to avoid engaging in integrative activities, as indicated by eye-tracking studies [280], and consequently leads to measurably worse learning outcomes [281]. While this research has not covered programming education specifically, it provides plausible grounds for believing that spatially scattered API documentation hampers the learning thereof, as well as programming productivity in general.

Solutions to the split-attention effect

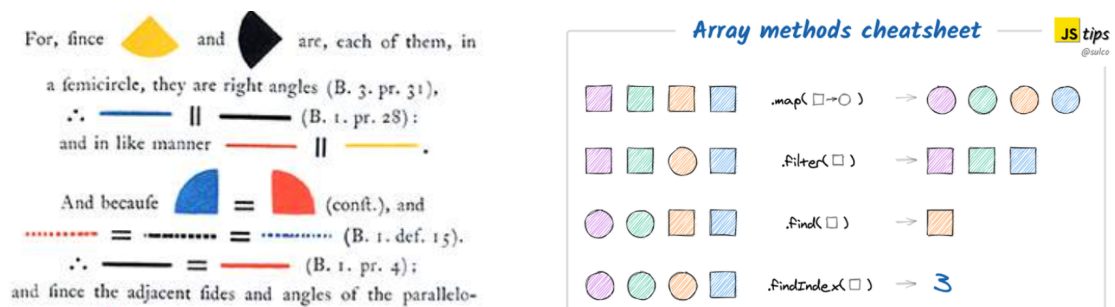
The split-attention effect has been addressed throughout distant and recent history via an assortment of typographical and digital innovations that facilitate the search. Consider the footnote, a late 16th-century device for avoiding the clutter of marginal or interlinear annotations [283]. Traditionally, footnotes linked references to their referents by way of arbitrary symbols (e.g. * or †) or numbered superscripts. Another technique common in technical diagrams are pointer lines or arrows connecting graphical components with corresponding explanations, which eye-tracking analysis has suggested leads to more integrative eye movements [284]. Another technique is to highlight corresponding element with the same colour, which is associated with higher scores on multiple-choice post-tests [285] and retention tests [286].

On paper media, these solutions would all have to be static, which means that graphical features meant to connect elements add clutter that potentially could distract the user. With hypermedia, more adaptive and action-dependent solutions become possible. For example, connecting lines or colour codes may appear only when the user hovers over an element. Another solution is to use *tooltips*: text boxes that pop up by the cursor upon hovering to help resolve the referent.

More specific to the API lookup case, we discussed in Section 3.5.3 how code example adaptation could be facilitated by various design conventions for argument placeholders. One such convention is *metasyntactic variables* [163], where identifiers are replaced with standardised names like `s` (for a `Pandas Series` object) and `df` (for a `DataFrame` object). Another such convention is to standardise the data set, so that variables have context-specific names, but from a familiar context, the most popular probably being Fisher’s iris data set [287], which is often used in the Tidyverse documentation. A third convention is to use self-documenting identifiers, such as `dataframe`. Each of these solutions has its own advantages and drawbacks. Metasyntactic variables are succinct but tend to require at least some upfront learning to make sense. Standardised data sets likewise require upfront learning of its schema, and are mostly useful for reproducible code snippets (since they do not require structures to be built from scratch). Self-documenting identifiers do not require learning, but risk being verbose and - unless syntax highlighting is employed and learned in advance - may be difficult to distinguish from keywords and syntactic elements.

5.1.2 Icon-based API cheat sheets

One potential way to achieve both succinctness and semantic clarity is to use icons. Iconic representations as stand-ins for textual syntax have previously been used in geometry. Consider the 1847 edition of *Euclid’s Elements* by the Irish mathematician Oliver Byrne [288]. The edition is remarkable in how it replaces all letters representing geometric elements with colourful graphics of the same shape and colour as them, shown in Figure 5.1a. Byrne claimed his method meant



(a) Excerpt from Byrne's edition of *Euclid's elements* [288, p.131]

(b) Sułkowski's JavaScript API cheat sheet on array methods [290].

Figure 5.1: Examples of educational resources where icons replace purely symbolic notation.

that the book could now be “acquired in less than one third the time usually employed, and the retention by the memory is much more permanent” [288, p.ix], an intuition that although empirically untested foreshadows the split-attention effect. Byrne's work has been given pedagogical attention before [289], but the efficacy thereof has never been addressed through an experimental design.

Without apparent connection to Byrne, similarly styled resources have recently gained popularity in JavaScript online communities, in the form of API cheat sheets. For example, in 2020 software engineer Tomek Sułkowski authored a widely shared ² tweet with a cheat sheet for array methods, where colour was used to identify the positional identity of an array element, and shape represented an arbitrary second characteristic [290]. The cheat sheet, seen in Figure 5.1b, has since spawned various adaptations. As a metric, social media popularity is sensitive to several confounds, but it does provide a meaningful signal that a broad class of learners find the resource intuitively helpful and valuable, and the likelihood of adoption is high. Could an equivalent resource be developed for data wrangling?

5.1.3 Parameter graphics in data wrangling


In the domain of data wrangling, a closer analogue to Byrne's graphics would be graphics representing the arguments used to configure an API command. We may refer to such representations as parameter graphics. For example, if we limit ourselves to operations involved in the typical split-apply-combine pattern or SQL **SELECT** query, we may identify the following pieces of information as candidates for parameter graphics (“parameters” in a loose sense, since not all are explicit function parameters):


Columns to join by: if we wish to perform an inner join on two tables, using SQL's **INNER JOIN**, then this requires the user to specify which columns to join by. It could be represented as two matching columns (**||**).


²As of August 30th 2021, it has received more than 9400 likes and been shared more than 3200 times [290].


SELECT		()	SELECT	AVG(Age)
FROM				FROM	Table
GROUP BY				GROUP BY	Class
HAVING				HAVING	AVG(Age) > 25;


Figure 5.2: The parameter graphics provide placeholders for passing arguments and other information that configure an API command.

Condition to filter rows by: a row filtering command like SQL’s **WHERE** . . . requires the user to specify a condition, which could be represented graphically as a row ().

Columns to select: a column selection like SQL’s **SELECT** . . . requires the user to specify the labels of a set of columns, which could be represented graphically as a column ().

Columns to group by: a grouping like SQL’s **GROUP BY** . . . requires the user to specify the labels of a set of columns to split rows by, which could be represented graphically as three rows, representing groups ().

Function for aggregating: an aggregation like SQL’s **AVG** (. . .) requires the user to specify the function to use when aggregating rows. It could be represented as a triangle, evoking a many-to-one operation ().

Condition to filter aggregates by: after an aggregation a user may wish to filter the resultant aggregates using SQL’s **HAVING**, which requires the user to specify the condition. It could be represented as a rounded triangle, similarly evoking a many-to-one operation ().

To be understood, these graphics would still require some degree of upfront learning, but they can be captured by short examples, such as that shown in Figure 5.2. Having learnt them, it is furthermore possible that they would facilitate query planning by encouraging the participant to parse elements in the problem context in terms of them. To use the example in Figure 5.2, upon reading an exercise prompt like “What is the average age of all classes whose average age exceeds 25?”, the learner may tentatively try out which parameter graphic “average age exceeds 25” or “classes” correspond to. This could be compared to the technique used by Qian [291], who described explicitly training students to consider keywords like “who are...” as a cue for a **WHERE** clause, “return the...” as a cue for the **SELECT** columns, and “per/each/every” as a cue for **GROUP BY**. The learner may thus be sensitised to a particular schema.

5.1.4 Multilingual cheat sheets

By making salient the various inputs a particular operation requires, on a level of abstraction higher than that of a specific programming language, parameter graphics may also be well-positioned to serve as keys to a Rosetta stone for writing scripts in multiple data scientific lan-

guages. This reflects the fast-paced reality of modern data science workplaces where practitioners are expected to be programming polyglots. The structural similarities between the three languages have been taken advantage of before. For example, Baumer [5] and Broatch et al. [24] have reported teaching SQL and R in conjunction, and found that students mostly enjoyed it. In the studies described later in this chapter, cheat sheets will be created for SQL, Python (Pandas) and R (dplyr). This also creates an incidental within-subjects variable, which we could opportunistically explore in terms of whether certain languages are more amenable to the cheat sheet treatment than others.

As discussed in Section 2.4, SQL, Pandas and dplyr all have strong commonalities due to their shared roots in Codd's relational algebra [119, 292]. As a result, they all offer convenient support for the above-mentioned data operations. However, they also have syntactic characteristics that set them apart. SQL is distinctly declarative in how it specifies *what* output is wanted instead of *how* to achieve it, and combines commands through nesting rather than sequencing. This feature could mean that commands are harder to combine. Compare this with R's dplyr, which is usually used with a pipe operator, allowing functions to be chained together into a sequence (e.g. `df %>% group_by(A)`). Python Pandas too allows methods to be chained, but using the dot operator (e.g. `df.groupby('A')`). It is plausible that the functional and segmented nature of R and Python makes them more appropriate for cheat sheets, since a cheat sheet can only fit a finite number of code examples, and with a chained rather than nested syntax, it may be easier to generalise how to combine operations.

5.2 Research questions

On the topic of parameter graphics, and their role as a key for multilingual code examples, there are several research questions we would like to ask:

RQ3a. Using parameter graphics, could a beginner in data science implement queries in three unfamiliar languages in less than 2 hours?

For a sense of how rapidly beginners can become versed in parameter graphics and use them to author short data wrangling scripts, some benchmark is required. One option is to choose an educationally relevant time cut-off. 2 hours represents a common tutorial duration length, and if participants could demonstrably author relational queries in 3 unknown languages within that time frame, the activity would seem well-suited as a tutorial assignment.

RQ3b. Do learners score worse in SQL compared with the other languages?

Does SQL's nested syntax make parameter graphics and cheat sheets harder to utilise? We hypothesise that performance scores will be lower with SQL than with Pandas and dplyr.

RQ3c. Will a learner's accuracy increase if they are trained using parameter graphics?

If the effect of parameter graphics is experimentally isolated, does it provide measurable value beyond what a graphics-free set of materials would? We hypothesise that total performance scores will be higher in a sample that uses parameter graphics.

RQ3d. Will a learner's time on task decrease if they are trained using parameter graphics?

All else being equal, we want the time taken to accomplish a task to be short. If parameter graphics reduce the time taken to see correspondences among code examples and problem context, and similarly reduce the time it takes to look up the meaning of code example parameters, then we expect time on task to be shorter for participants given such graphics.

5.3 Tutorial design

To probe the effect of parameter graphics, they would have to be embedded within an instructional context. A tutorial was designed with the goal of teaching beginners how to author queries using relational operations in three languages. This tutorial was accompanied by a set of cheat sheets and exercises for each language (R dplyr, Python Pandas, and SQL). All of these tutorial components formed part of the intervention, and so the dependent measure was the performance in the exercises rather than a separate post-test.

5.3.1 Tutorial

The tutorial was designed as a six-page paper booklet. It was paper-based in order to allow annotation and reduce distractions. Since many participants would be complete beginners, there was a risk of them being stalled by compiler errors or becoming stuck in a monkeys-and-typewriters situation. There was therefore no code execution or direct feedback available.

The tutorial assumed no prior knowledge of relational operations, and therefore began with explaining the concept of a table, and all the six operations mentioned above. Each operation was explained using a graphic featuring sample input and output, with colour highlights to draw attention to selections and correspondences. Along the way, the tutorial introduced the parameter graphics and the entities they represent (e.g. the conditions to filter by, the columns to join by). To scaffold the participant's parsing of an exercise (see Section 5.3.3), six worked examples were provided that explicitly identified which entities in the problem context corresponded to the parameter graphics. Finally, there were six practice exercises where participants could try out identifying the entities themselves. Two pages from the tutorial are shown in Figure 5.3 and the full version is in Appendix A.

Querying tables:

The following are the main operations used for queries. Each has an associated "component".

Selecting rows

Sometimes we wish to select certain rows based on a criterion. For example, we only want the cases whose Date is later than 2000-01-01.

Let's symbolise this **criterion** as a green row: Other possible criteria are:

- Director is "James Cameron"
- Name is not "Mamma Mia!"
- Date is in January

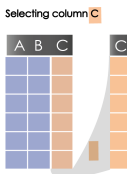


Selecting columns

We may only be interested in one or several attributes (columns). For example, we may ultimately only be interested in the Directors.

Let's symbolise this **list of columns** as an orange column: Other possible lists are:

- Date
- All
- Date, Name



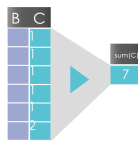
Aggregating columns

We often want to "aggregate" one or several columns. This means finding a single-value summary for it. For example, we may want to find the number (count) of all directors.

Let's symbolise this aggregation function as a cyan rectangle: . Other functions are:

- Sum
- Average (mean)
- Max

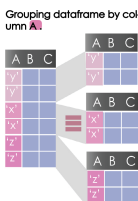
Summing column C:



Grouping rows before aggregation

Whenever we aggregate a column, we often wish to have a separate summary value for each separate group of data. For example, suppose the Movie table had a column "Genre". Then we may want a count of all movies in the thriller-genre, in the romcom-genre and so on.

Let's symbolise the column we "group by" as a red rectangle: .



Piecing the operations together:

In order to process a query, we need to translate a natural English sentence into these types of query components. For example, a typical query would be:

"For all cases where B is above 4, what is the average C for all cases, grouped by A?"

Which rows are wanted? Which columns? Does it have to be aggregated? Grouped before aggregated? We can start annotating it...

"For all cases where **B is above 4**, what is the **average C** for all cases, **grouped by A**?"

We only seem concerned with rows where $B > 4$

This is the aggregation function required

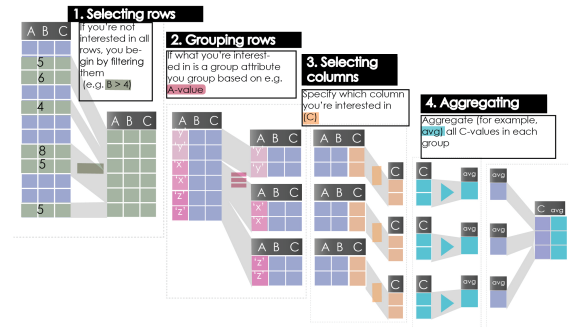
C is the column we eventually want the average

A must be a categorical attribute that splits all rows into groups

We can summarise what each types of component corresponds to:

- Columns C
- Row criteria $B > 4$
- Agg function Average
- Group by A

Then, have a look at how these components are used for a sequence of operations that start with the original table and returns the information that the query asks for.



Don't care too much about ordering. Our focus is on the ability to translate English into our list of components.

(a) Page 4 in the tutorial booklet.

(b) Page 5 in the tutorial booklet.

Figure 5.3: Excerpts from the tutorial booklet explaining each operation in turn and introducing parameter graphics along the way.

5.3.2 Cheat sheets

The tutorial teaches students an abstract schema on how relational operations can be chained together, and trains them to parse problem descriptions in terms of the parameter graphics. In order to turn these abstract solutions into API-specific syntax, it remains for the participant to consult a cheat sheet. These sheets were meant as a superficial lookup table: it does not explain how the language's paradigm or basic constructs work, nor does it discuss details in data structures and related API concepts. In terms of Thayer et al.'s concept of *robust API knowledge*, the understanding conferred would be necessarily brittle [115].

Each cheat sheet contains seven usage patterns, which may contain one or more operations and are meant to exemplify how operations in the language are combined. In order, it contains column selection, row filtering, row filtering *and* column selection, aggregation, split-apply-combine, split-apply-combine *and* aggregate filtering, and finally joining. For each entry, there is a thumbnail (the same explanatory graphic as in the tutorial sheet), a code example with parameter graphics, and a code example without them. For an example, see Figure 5.9a. The cheat sheets can also be found in appendix B.

5.3.3 Exercises

Three exercise sheets - one per language - were created containing six exercises each. As shown in Figure 5.4, the participant is given the schema of two tables as the problem context, which all six exercises will be about. Alongside each exercise there is a list of all parameter graphics, meant as cues to scaffold the participant's structure mapping, just as they had been doing in the tutorial.

The exercises get progressively more difficult: in each sheet, the first exercise always involved a simple selection or filtering, the second exercise a combination of these, the third a simple aggregation, the fourth a grouping and aggregation, the fifth two operations together with a join, and the sixth a join and filter aggregate (i.e. **HAVING** . . .). Each language had its own unique set of exercises. For example, Python exercises dealt with recipes while SQL exercises dealt with flights. The computational and linguistic complexity of exercises were controlled manually to ensure that they were equally difficult. This control procedure involved grouping operators based on complexity (e.g. column selection and row filtering were regarded as equal, join and grouping were regarded as equal) and equalising the number of instances per group across exercises.

Exercises were meant to be accompanied by cheat sheets, and are either isomorphic to one of the patterns provided on the cheat sheet, or they require the participant to attempt a combination of operations that is not featured explicitly in the cheat sheets.

Recipes

Name	Author	Nr_served	Type	Cost

Cookbooks

Bookname	Author	Cuisine	in_Paperback

Which recipes are of type dessert?

- Table ■
- Columns ■
- Row criteria ■
- Agg function ▶
- Group by ■
- Joining condition ||
- Agg-criteria ▶

What are the cuisines of Jamie Oliver's books?

- Table ■ *Cook books*
- Columns ■ *Cuisine*
- Row criteria ■ *Author = Jamie Oliver*
- Agg function ▶
- Group by ■
- Joining condition ||
- Agg-criteria ▶

Cookbooks [Author = Jamie Oliver]. Cuisine

Figure 5.4: The two first exercises in the Python sheet. The second contains a real participant response.

Marking scheme

A marking scheme was developed to assess the accuracy of participants' responses. This marking scheme emphasised the correct selection and ordering of operations, as well as the correct choice and location of parameter values. Crucially, the scheme ignored errors attributable to obvious typos, or to information they impossibly could have known, such that **AVG** in SQL corresponds to `mean` in R (any function name suggesting the computation of an average would have qualified). Similarly, the fact that the equality operator `=` in SQL is expressed as `==` in R was ignored (they were free to express their condition using pseudo-code, as long as it was comprehensible). The rationale behind these were that these errors that could easily have been corrected in response to feedback in an execution environment, or that were due to omission of information, given the scope of a 2-hour experiment and single A4 cheat sheet.

The scheme worked similar to edit distance in that 1 point was deducted for each instance of substitution, omission, or irrelevant insertion of an operation. Additionally, 1 point was deducted for the incorrect choice of parameter values (e.g. wrong filtering criteria). Half a point was deducted for a redundant operation that did not impact the result. The six questions were weighted 1,2,2,3,6 and 8 points, respectively. This means that the maximum score per language was 22, and the maximum possible score overall was 66. The minimum possible score was 0. The scoring was done by a single rater. When the same rater rated a sample of 10 exercise sheets 2 years later to gauge intra-rater reliability, scores were the same for 54 of the 60 exercises (90%), with unreliability mainly in the last exercise of the exercise sheets.

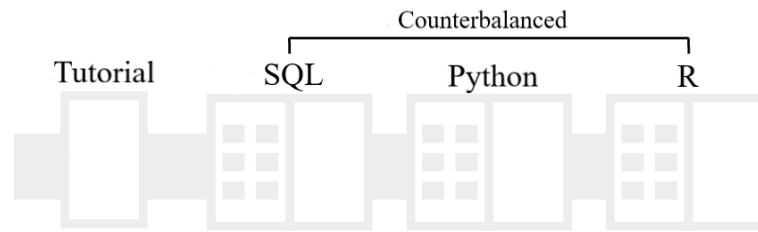


Figure 5.5: Study 1 used a single-group design in which participants were first given the tutorial, then the cheat sheet and exercise sets for each language, in counterbalanced order.

5.4 Study 1

The first study involved a sample of participants that consisted exclusively of beginners, as this would allow us to see whether example adaptation exercises would require prior programming experience in order to be feasible. All participants in this study were given materials with parameter graphics.

5.4.1 Method

Design

A single-group design was used with one independent within-subjects variable (language). Two dependent variables were measured: one product measure (the score) and one process measure (time on task). Although this single-group design allows of no comparison-based claims regarding parameter graphics' efficacy, we will reserve RCT designs for Study 2 and here use a within-subjects design to heighten power and reduce ethical concerns.

Procedure

Ethical approval was obtained by the local college's ethical review board. Participants signed up to a 2-hour time slot on a one-by-one basis and were welcomed into a quiet room on campus. They were asked to read through a consent form. To reduce evaluation apprehension, this form underscored that it was not an aptitude test, but a usability study for evaluating the materials. Following this, they were handed a short demographic survey and the tutorial booklet. They were instructed to read it at their own pace, but were also told that it was not recommended to take more than 40 minutes to read it. Upon finishing the tutorial, they had the opportunity to ask questions, and were subsequently handed the first exercise booklet with the accompanying cheat sheet. Languages were counter-balanced for each participant to prevent order effects (see Figure 5.5), but not counter-balanced with respect to exercises (e.g. Python was always paired with the same exercise set). Participants were instructed to "take their time, but if they ever got stuck, to skip past it".

Time on task was recorded by the experimenter, from the moment that participants were

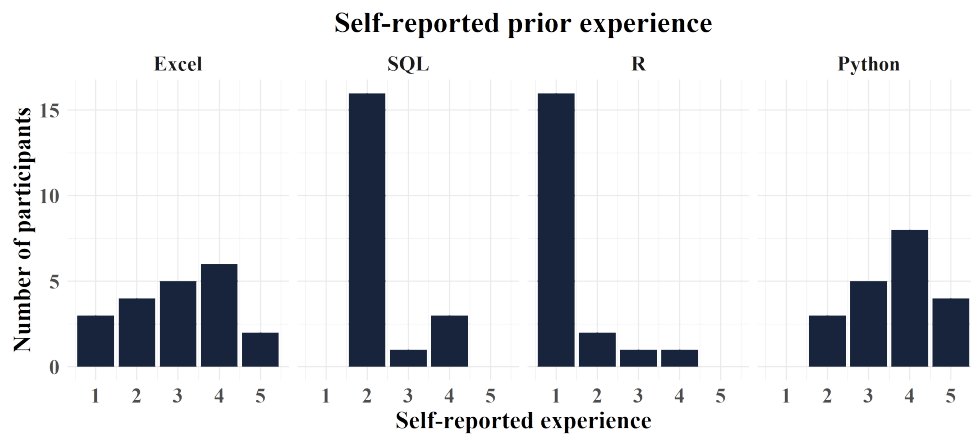


Figure 5.6: Although some participants had basic experience with procedural Python, most students had no experience in SQL or R, according to their own ratings (1=“No experience at all”, 5=“I can do complex commands in it”).

given the tutorial booklet until they completed the entire exercise set. We therefore did not record time per language, which would have allowed us to compare languages in how quickly their exercises were completed. Although we had left the schedule empty to allow participants to complete the task at their own pace, the next slot sometimes came directly afterwards, leading to occasionally overlapping sessions with two participants in the room simultaneously. Of course, we could not prevent participants from scheduling in appointments after the agreed-upon time slot, leading to a natural time frame within which completion was implicitly expected. Upon completion, the participant received £5 in compensation for their time.

Participants and recruitment

27 participants were recruited from a local CS0 introductory course in programming that just had begun covering basic Python (procedural, not data wrangling), as well as a third-year engineering class that covered elements of procedural C programming. Out of these, only 20 participants (15 women, 5 men, median age=19) remained for the analysis, as 5 withdrew mid-experiment for different reasons, 1 participant misunderstood the task, and for 1 participant time on task was not recorded. All participants were recruited on an extra-curricular basis; participation was not credit-bearing. Everyone had some previous exposure to programming, as confirmed via a questionnaire, but could nevertheless be characterised as beginners. Figure 5.6 shows the result of a survey Likert-scale item asking participants to rate their experience (1=“No experience at all”, 5=“I can do complex commands in it”) in 4 data-related technologies: respondents mostly lacked any experienced in R and SQL, but some reported intermediate Python experience. We assume this reflects the basic Python competence in the CS0 course.

5.4.2 Results

RQ3a. Using parameter graphics, could a beginner in data science implement queries in three unfamiliar languages in less than 2 hours?

RQ3a posed what amounted to a feasibility test: is it possible to complete the example adaptation exercises within 2 hours? We noted previously that the 2 hour threshold, though ultimately arbitrary, represents a common tutorial length, and means that the activity could be completed in a single tutorial session.

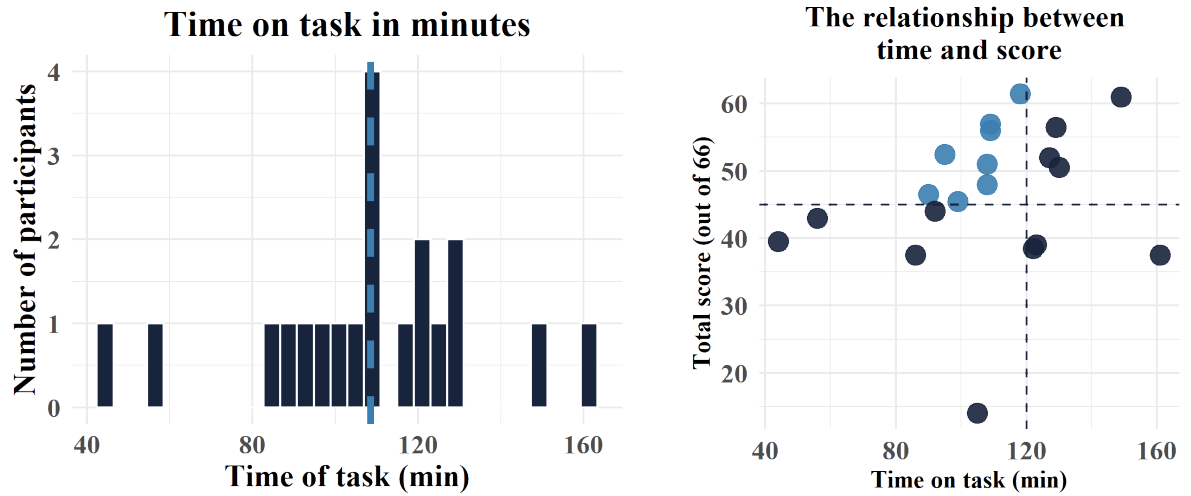
When looking at the distribution (Figure 5.7a), the overall time on task (reading the tutorial + completing the booklets) was, in median terms, 108 minutes (range=44-161min, IQR=30min). It appears that, 10 minutes before the scheduled slot was about to end, participants tended to hand in their materials. 70% (14 out of 20) handed in the cheat sheets within 120 minutes. While that ratio is impressive, time on task in our context indicates the time taken *until they handed it in* - not until all exercises were correctly completed. It is possible that they handed in a haphazardly completed booklet set, especially given that the payment was unconditioned on accuracy and stakes overall low. Therefore, an additional threshold needs to be set for the score. If we set this threshold to be 45, they scored on average 15 (out of 22, i.e. 68%) in each of the three sets. This intersection of criteria - visualised in Figure 5.7b - leaves 8 participants out of 20. Seen this way, only 40% successfully completed the task within 2 hours. If the limit is placed at 43, this becomes 50% - not insignificant for a group of participants with either minimal or zero prior knowledge in programming.

Discussion. Whether the observed completion ratio is satisfactory or not will, absent a control group, depend on the criteria we set. It is clearly possible for complete beginners to solve simple to moderately complex queries example adaptation problems in unfamiliar languages, and for at least half of the participants, it is feasible within 2 hours. As a threat to validity, the actual time spent was presumably influenced by external cues, such as their other scheduled commitments and participants arriving for the next slot.

RQ3b. Do learners score worse in SQL compared with the other languages?

To determine whether languages differed in the ease with which example adaptation was performed, we begin by plotting the three score distributions as box plots (Figure 5.8b). We had hypothesised that SQL would have *lower* scores, but it is visually clear that SQL scores are concentrated towards the upper bound (median=20, IQR=2.9), while scores in R (median=12.75, IQR=4.9) and Python (median=13.25, IQR=4.6) are scattered more widely.

To determine the effect of language, we sought to perform a one-way within-subject ANOVA. Following the removal of the left-extreme outlier seen in Figure 5.8a, normality was tested through a Shapiro-Wilks test, which found that, except for SQL, group distributions were not



(a) The distribution of time of task durations. Dashed line shows the median.

(b) The association between time and score. Light blue means a time below 120 and score above 45.

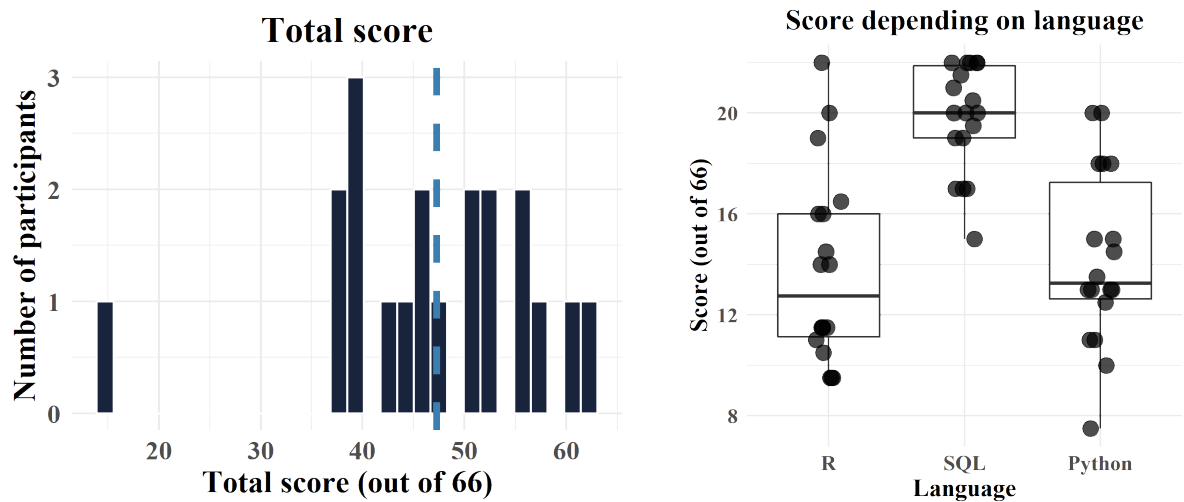
Figure 5.7: Plots relating to the total time on task in Study 1.

significantly different from the normal. Mauchly's Test of Sphericity indicated that the assumption of sphericity had not been violated ($W=.93$, $p=.52$). The result indicated that there was a significant language effect $F(2,36)=37.8$ ($p<.05$). We can therefore reject the null hypothesis that language does *not* influence accuracy scores. As an example of a post-hoc paired t -test, SQL has significantly higher scores than Python $t(18)=8.2$ ($p<.001$) 95% CI [4.2,7] with a mean difference of 5.6.

Discussion. Despite the fact that exercises had been carefully controlled for the intrinsic complexity between languages, the study found a significant main effect in language, with SQL ranking the highest in accuracy. This is contrary to expectation, which reasoned that participants would struggle with its nested syntax. The result could mean that the English-like nature of SQL syntax overrides any confusion relating to the nesting of statements, but alternative explanations are also viable, including that the R pipe-operator or Pandas method-chaining are confusing. It is also possible that the high score for SQL is a subtle selection effect: key operators were chosen based on the Venn overlap between the three languages, and given that SQL is the least expressive language, SQL is also more optimised for these operations.

5.5 Study 2

To address **RQ3c** and **RQ3d**, which sought to evaluate parameter graphics from a comparative effectiveness perspective, a second study was conducted that introduced a control condition. As a matter of pragmatism, it was necessary to recruit participants from more experienced classes,



(a) The distribution of total accuracy scores. Dashed line shows the median.

(b) Box plots with the distribution of scores for each language. Boxes indicate medians and IQR.

Figure 5.8: Plots relating to the accuracy distributions of Study 1.

since those students tended to be more intrinsically motivated and therefore more likely to sign up. Unlike Study 1, the participants here are likely to have had some prior data wrangling experience in R.

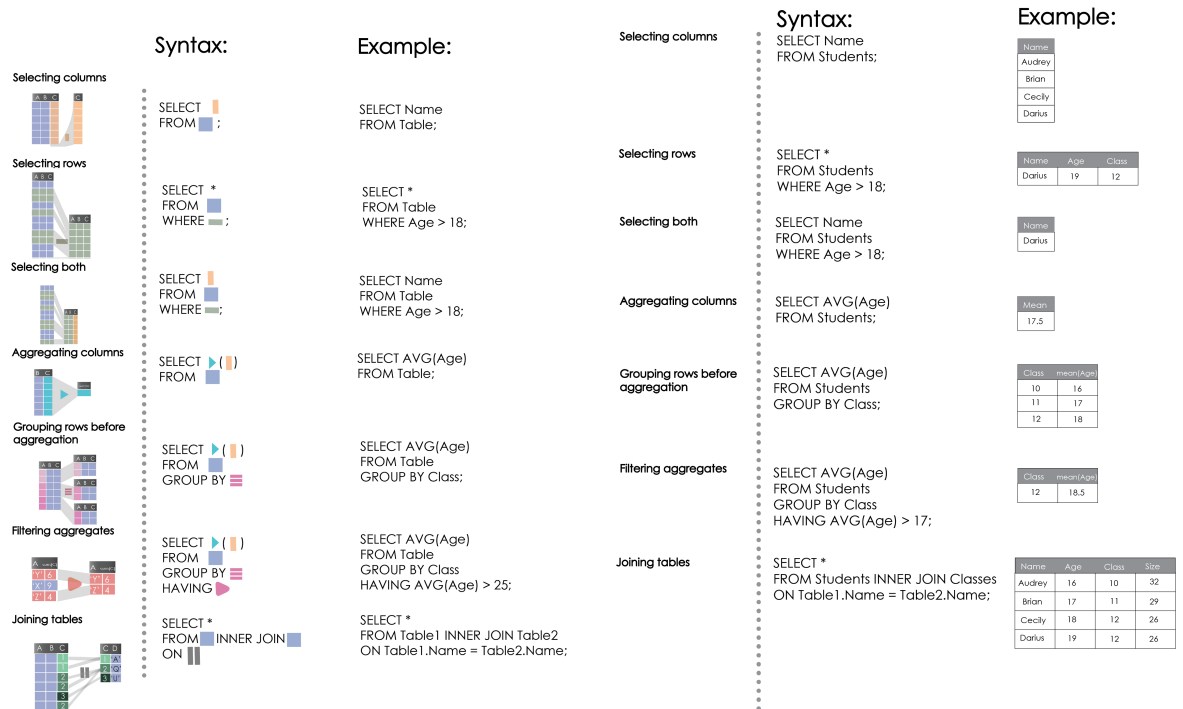
5.5.1 Method

Design

Study 2 follows an RCT design in which participants were randomly assigned to one of two conditions as a result of manual booklet shuffling. The first condition contained parameter graphics, and was identical to that of Study 1. Designing the control condition presented the age-old methodological dilemma, where too subtle a change makes the effect undetectable, and too large and multi-dimensional manipulations reduce informational equivalence of the conditions [204] and therefore the experiment's internal validity. The approach we settled on was to keep the difference small: the control booklet contained the same examples and practice exercises on how to recognise the required operations, but without mention of the icons (see Figures 5.9a and 5.9b). Again to assert informational equivalence, exercise sheets contained concrete sample input and output for each operation, instead of thumbnail graphics with parameter graphics.

Procedure

Procedurally, Study 2 was identical to Study 1, except that participants were also allocated to a condition based on the physical shuffling of experimental materials. There was one small difference: as participants mostly had previous R experience, R was consistently given as the



(a) The experimental condition contains parameter graphics as well as a code example and thumbnail graphics.

(b) The control contains a code example and post-state graphic for each operation.

Figure 5.9: In Study 2, people were given one of two cheat sheet styles. Shown are excerpts from the SQL cheat sheet. The tutorial booklet was similarly produced in two variants.

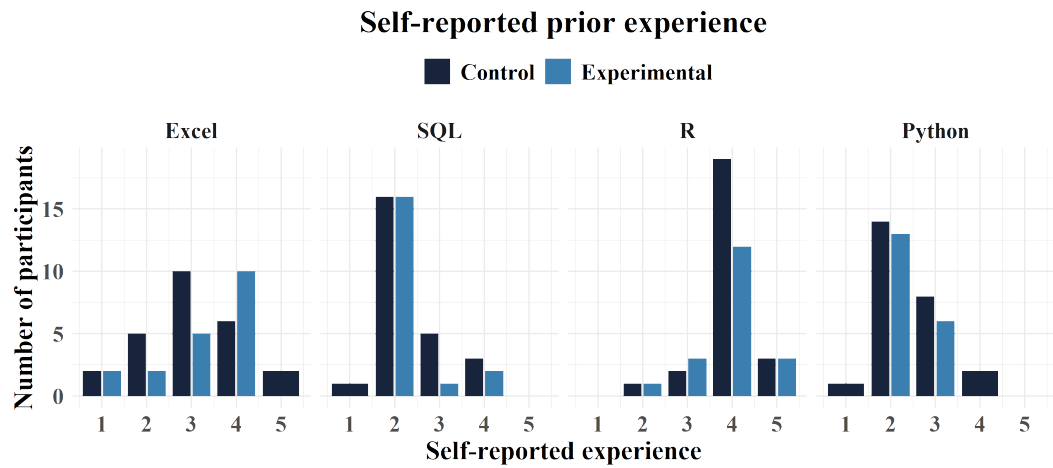


Figure 5.10: The control group and experimental group had a mostly similar distribution of experience, though the control group appeared more experienced with R (1=“No experience at all”, 5=“I can do complex commands in it”).

first exercise sheet, with only SQL and Python mutually counter-balanced.

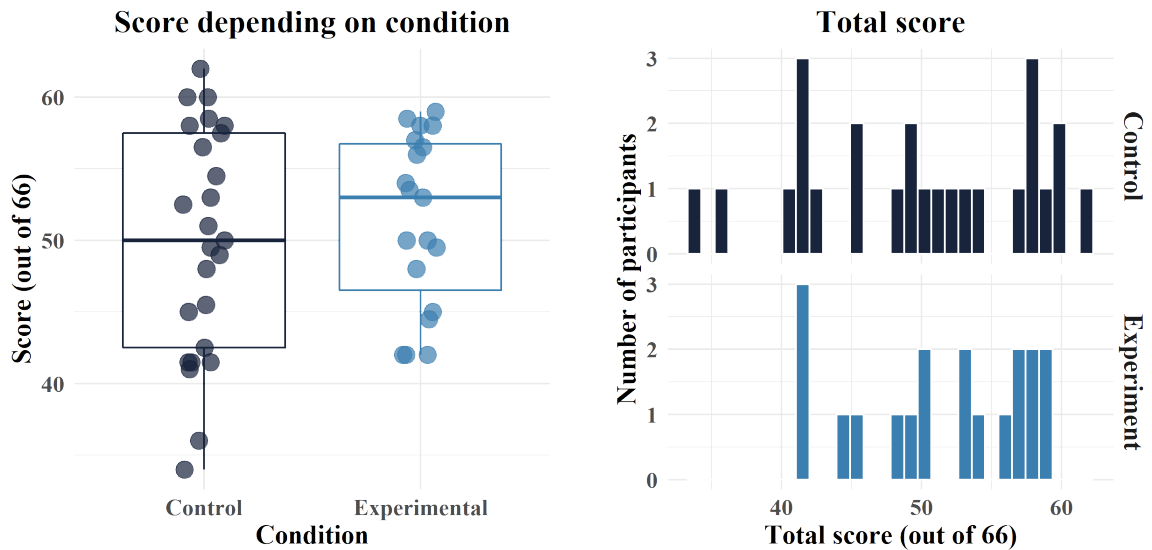
Participants and recruitment

44 participants were recruited through email advertisements at introductory programming courses offered at the local university, during the first three weeks of the semester. We focused recruitment on non-CS programming courses (including statistics, psychology, biology and data science) since the students were likely to have an intrinsic motivation to learn programming, and were representative of the target audience. By manually shuffling the experimental booklets, participants were randomly assigned to either the parameter graphic condition ($n=19$, 13 female) or the control condition ($n=25$, 16 female). The unbalanced cell sizes reflect the fact that we employed pseudo-random assignment, as opposed to alternating assignment, and that recruitment ceased once all local programming courses of relevance had been contacted. In Figure 5.10, showing responses to the same experience survey as in Study 1, we see as expected that participants report having R experience. The samples exhibit similar experience levels regarding Python and SQL, but the experimental group appears slightly more experienced in Excel, and less experienced in R. We are satisfied that baseline differences are unlikely to confound the results.

5.5.2 Results

RQ3c. Will a learner’s accuracy increase if they are trained using parameter graphics?

We had hypothesised that participants with parameter graphics would perform more accurately in the exercises. To compare the two groups in their performance, we began by plotting the distributions, shown in Figure 5.11. The control group appears to have a wider distribution



(a) Box plot showing the group differences.

(b) Histogram showing the group differences.

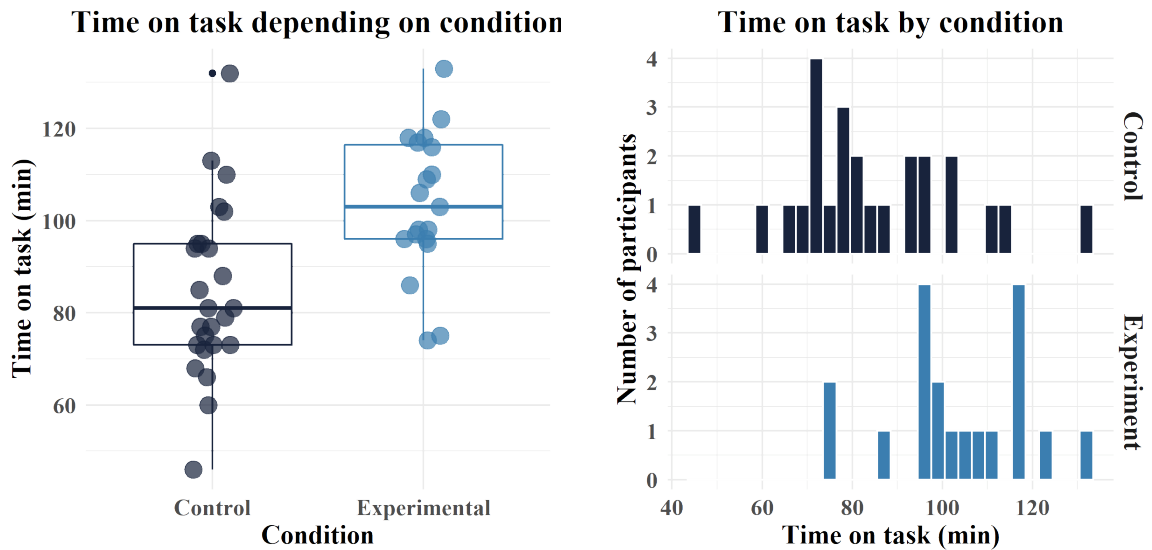
Figure 5.11: Differences in total score distribution between the experimental group (with parameter graphics) and the control group (without such graphics).

(median=50, IQR=15) compared with the experimental group (median=53, IQR=10.25). The difference in central tendency does not translate to a clear group separation in the density of scores, as neither distribution conforms visually to a normal distribution. As determined by a Wilcoxon rank sum test, the two distributions are not significantly different ($W=213$, $p=.57$). We therefore lack sufficient basis for rejecting the null hypothesis that parameter graphics do not affect scores.

Discussion. If the null hypothesis is true, one conceivable explanation is that the parameter graphics *added* cognitive load by introducing partially arbitrary graphical jargon. However, with so few (7) icons to memorise, it is not a satisfying one. Another possibility is that the slightly higher prior experience of the control group offset any pedagogical benefits of the experimental group. The null effect could also be a methodological artefact: it is possible that our choice of control condition erred on the side of being too subtle for the difference to be picked up, especially by an outcome measure that, besides being able to substitute variables, also depends on the ability to abstract problems and combine operations. It is possible that a more proximal process metric would have been more affected by the inclusion of parameter graphics.

RQ3d. Will a learner's time on task decrease if they are trained using parameter graphics?

Our prediction had been that parameter graphics would lead to shorter time on task. Upon inspecting the distributions of Figure 5.12, the data appear to show the opposite pattern, where the control group (median=81min, IQR=22) appears to concentrate around a shorter time duration than the group with parameter graphics (median=103min, IQR=20.5). Since Shapiro-Wilks



(a) Box plot showing the group differences.

(b) Histogram showing the group differences.

Figure 5.12: Differences in time on task score distribution between the experimental group (with parameter graphics) and the control group (without such graphics).

test suggested that both distributions are normal, a Welch's two-sample t -test was conducted, $t(41.56)=-3.7$ ($p<.001$) with a 95% confidence interval of $[-29.4, -8.7]$ and a Cohen's d of -1.1 (Hedge's g of -1.08). While mindful of the small sample size, we cautiously interpret this as evidence for an effect in opposite direction to the hypothesis, meaning that parameter graphics *increase* the time on task.

Discussion. Why did the data contradict the hypothesis? The time on task estimates of Study 2 are subject to the same caveats as those in Study 1. Absolute values are likely influenced by external factors such as scheduling, however those factors would operate equally on both groups due to the randomisation. Reading parameter graphics incurred a certain overhead cost in the time taken to learn it, and it is possible that, while using them, participants repeatedly sought to verify their interpretation, which cost additional time. It also required time to write out which problem context entity a particular icon corresponded to. Another explanation is that the experimental condition was longer because the cheat sheets contained more information: in addition to a code example, it featured thumbnail graphics as well as parameter graphics, while the control only contained example output.

5.6 Discussion

The study involved multiple pedagogical tools that are either under-explored or wholly unexplored in the literature. The results do not provide evidence favourable to the proposed intervention of parameter graphics since time on task was significantly higher than the control condition

(RQ3d), and the null hypothesis remains plausible in the case of performance scores (RQ3c). This does not constitute a complete cul-de-sac, since no hypothesis could ever be answered without replication when confidence intervals as wide as ours are involved. Other dependent variables that are not investigated in the study, such as motivational ones, could very well be sensitive to the graphics.

Another pedagogical tool in the study were the cheat sheets and example adaptation exercise sheets. Using these tools, our data suggest that complete beginners could adapt code examples in three unfamiliar languages within a single 2-hour session (RQ3a) and that it is especially effective in the case of SQL (RQ3b). While we do not have data on how the training afforded by these activities transfers to “real” programming, we do know that beginners will be able to write moderately complex queries in that time frame. We therefore propose integrating it as an introductory activity in a database course.

5.6.1 Future implications

Researchers interested in the potential of parameter graphics could expand upon the current studies by adding a post-test *without* parameter graphics for both groups. In such a post-test, cheat sheets would presumably remain, since their *raison d’être* is to remove the need for memorisation. Another augmentation in future research would be to collect attitudinal data, to get a sense of whether graphics are *perceived* as helpful, in addition to (or in spite of) their objectively measured efficacy.

In our line of research we will not investigate parameter graphics further. Consistent with our design-centred priority of breadth over depth, we will redirect our future attention to other types of graphics, namely subgoal graphics and thumbnail graphics. Another reason for this is doubts about parameter graphics’ extensibility: relational operations are but a small subset of data wrangling, and while parameter graphics may be relatively simple and interpretable for SQL, adapting it for other functions in Pandas or dplyr may make the number of icons far too large and ambiguous to justify memorising them.

5.6.2 Methodological considerations

The experience of running this study foregrounded a number of issues and challenges that led us to re-evaluate and adjust our methodological approach. Most of the issues ultimately relate to the intervention being paper-based.

While the pen-and-paper medium in theory has many benefits (e.g. low setup costs, no configuration and an absence of distractions for the participant), it does not permit easy collection of process metrics. Although time on task was measured, we could not easily record more fine-grained milestones like time per exercise. Besides time on task, many other behavioural data would have been useful to collect, and would have enhanced the cost-effectiveness of the study:

for example, the number of times the tutorial and cheat sheets were referenced, and whether participants actually engaged with the graphics and not just the code examples.

Paper-based studies do not support program execution. Informally, we noted that several participants expressed exasperation at not being able to test their solution and receive immediate feedback, and voiced disappointment at not learning “actual programming” instead. It is true that, given the lack of error message feedback, participants also derived less intrinsic value from the study, since they may proceed to repeat the same mistakes without corrections. If “programming” requires writing syntactically precise, typo-free code, then it is also true that the marking scheme (which did not penalise obvious typos) did not measure it, but rather a slightly more higher-level kind of problem-solving. As a matter of construct validity, it may therefore be more accurate to describe the study as measuring pseudo-code composition than programming.

In general, paper-based studies are labour-intensive and imprecise: even though we attempted to represent all possible mistakes and errors in the marking scheme, it was not completely unambiguous, reducing its intra-rater reliability. Since time collection was not automated, it was also error-prone: exactly how much measurement error it added cannot easily be established without multiple observers, and ultimately measured when participants *chose* to announce their completion. Furthermore, paper-based studies mostly require a physical co-presence of the experimenter and the participant, which does not scale well, and leads to the scheduling bias discussed in relation to **RQ3a** where participants adjust their response meticulousness to complete it within the scheduled slot. All of these issues can be addressed with computerised instrumentation, which was the approach chosen for subsequent studies.

Chapter 6

Subgoal and thumbnail graphics materials

As a design-focused research project, this dissertation revolves around a set of core materials, all of which will be incorporated into SLICE N DICE, the platform built for the capstone study. These fall into two categories: non-graphical scaffolding features that we have identified as relevant to data wrangling, and graphical scaffolding features that we posit would make those features even more effective. The non-graphical features include **subgoal labels**, which potentially could be augmented with **subgoal graphics**, and a **command menu**, which could be improved with **thumbnail graphics**. To fairly represent the context in which the graphics may ultimately be employed, and for the purpose of retaining participants, it was crucial that all of these materials were user-friendly. If the menu were too confusing, or the subgoal labels too opaque, then it is possible they would be so distracting that the graphics would end up being ignored altogether. Their design is informed by prior research literature, personal judgement, as well feedback from the two pilot studies described in further depth within Chapter 8. Some of that feedback will be previewed here to account for how the materials reached their final form.

This chapter discusses the design of both the non-graphical features and their graphical augmentations. Since the menu will be how participants look up data operations throughout SLICE N DICE, we will begin by describing its design, before describing the thumbnail graphics. Following this, we will describe the subgoal labels and graphics.

6.1 Menu design for API lookup

In our literature review and discussion of novices' barriers to API lookup, we observed how palettes in block-based programming environment provide a constrained and browsable way of exposing available API commands. While real-life projects and open-ended school projects may not respect such hard constraints on the command set, in closed-response instructional contexts constraints may help preventing the learner from feeling overloaded. A command menu also encourages educators to define the learning domain more formally: it implies that the class of problems that a student *should* be able to solve corresponds to the class of problems solvable

using a (reasonably complex) combination of the available commands. By formalising a domain, any associated resources can also be reused and shared more easily.

An explicit representation of a learning domain is sometimes called an *educational ontology* [293]. In computing, an *ontology* is defined by Swartout et al. as a “hierarchically structured set of terms for describing a domain” [294, p.138]. In practice, they range from simple, controlled vocabularies, to complex graphs containing relationships of many different kinds. An ontology is generally understood to be in a machine-understandable format, such as the Web Ontology Language (OWL), though in this case we will store it as a simple JSON file.

In our discussion of how we designed it, we will use the term *ontology* to refer to the abstract structure and contents of the learning domain. We will use *menu* to refer to a digital widget within SLICE N DICE that instantiates that ontology.

6.1.1 Ontology design

Several methodologies for engineering educational ontologies have been proposed [295, 296]. Sosnovsky and Gavrilova’s approach [293] is particularly relevant, as they applied theirs in the context of a C-programming ontology. They propose a five-step approach that begins with *glossary development* (elicitation of essential concepts) and *laddering* (the definition of abstraction levels), then assigns concepts to the right abstraction levels, by either breaking them down (*disintegration*) or grouping them (*categorisation*). Finally, the ontology is *refined*, for example by removing any residual redundancies, ambiguities or contradictions.

There are also more automated approaches available for deriving ontologies. For example, one could crawl through semi-structured data such as textbooks and API references [297–299] or through software repositories [300]. In our case, however, we had a number of constraints relating to the ontology’s role as an experimental instrument and educational resource, which an automated approach would be less useful for. We therefore largely followed the process recommended by Sosnovsky and Gavrilova. To elicit concepts, Sosnovsky and Gavrilova used an authoritative textbook [293]. Similarly, our glossary development used *Python for Data Analysis* [39] by Wes McKinney (the creator of Pandas) and *R Cookbook* [301] by Paul Teetor, two comprehensive and popular textbooks published by O’Reilly Media. We based the ontology’s contents on NumPy 1.18.0, Pandas 1.2.0 (both for Python 3) and Base R 4.0.3 ¹.

Design requirements

An ontology has a number of non-functional requirements [293]. Some of these relate to its contents, which should also be abstract enough to outlive any low-level changes in the API, and

¹Notably, we did not base it on Tidyverse, due to two reasons. Firstly, Base R is more analogous to NumPy/Pandas. The second reason related to recruitment-related concerns: several local courses that we sought to recruit participants from used Tidyverse, and by offering a complementary rather than completely overlapping resource, we could allay lecturers’ concerns about interfering with their pedagogy, and increased the chance that the participants would still be relatively naive to the API.

ideally apply to more languages than one. Because Python and R are both dominant languages in data science, it would be useful if the ontology were multilingual, allowing it to serve as a Rosetta stone for translating a command from one language into the other. This appears achievable, given how the two have co-evolved and share considerable similarities.

Another set of requirements relates to its size and shape. The ontology should be navigable and efficient in terms of the number of decisions that would have to be made before the correct entry is located. Given its role as a menu backbone, it therefore makes sense to structure it as a tree. This tree should be small enough to be learned within approximately 30 minutes, but large enough to challenge a participant with some previous background knowledge. If traversed, the ontology should not feel overly repetitive or too concerned with minutiae. It should also be symmetrical and well-balanced, with Sosnovsky and Gavrilova [293] mentioning design heuristics such as keeping the number of branches per node to Miller's number of 7 ± 2 .

Organisational layout

When the novice is in the process of imagining which type of operation would be necessary to accomplish a particular subgoal, we assume that this mental conception takes the form of informal wordings like “Attach the vector to the dataframe”. However, it would be unwieldy to include “Attach vector to dataframe” as its own, top-level entry, since that would result in a flat and impracticably wide ontology. Instead, the ontology would have multiple levels, presenting a sequence of choices that the participants would need to make in order to retrieve the operation. One possible solution would be to let the first choice be about the high-level objective, the second choice about the data structure, and the third choice about the particular operation:

High-level objective: What is the overall purpose of the operation that is being searched for?

Early in the design process, we settled on five such categories: *Create*, *Access*, *Calculate*, *Combine*, and *Restructure*. “Attach the vector to a dataframe” would fall into the *Combine* category, since it is combining a vector with a dataframe.

Data structure: We had early on decided to limit ourselves to vectors, matrices and dataframes.

However, “Attach vector to a dataframe” could conceivably fall into both the *vector* and *dataframe* category: a strict tree does not permit an easy way to accommodate this, except redundantly adding the entry to both subcategories. In the end, we stuck to the principle of placing operations involving different data structures under the most complex data structure. That way, the navigation path for “Attach vector to a dataframe” would begin *Combine > dataframes*.

Operation: The action verb used spontaneously by the user (e.g. “Attach”) may or may not be congruent with the terminology of the API: examples of more API-congruent verbs would be “bind” or “concatenate” (in R, the relevant function would be `cbind()` while

in Python it would be `concat()`). To label the leaf-nodes, we took care to choose words that were as jargon-free and unambiguous as possible. For example, instead of *concatenate*, we simply use *with vector*, such that the path becomes *Combine > dataframes > with vector*.

Note how this final path through the menu traces out the sentence. This was deliberate, and although most paths form awkward sentences at best, it turns the user's task into one of formalising their own formulation "Attach vector to dataframe" into the available *Combine > dataframes > with vector* operation.

The semantic relationship of a parent-child link in the ontology could be regarded as an implicit *is-a* relationship, in the sense that the *Combine > dataframes* element represents a class of "operations for combining dataframes", of which *with vector* is a member. As such, the ontology is technically a *taxonomy*.

One drawback with this structure is that it produced redundancies. For example, it required separate entries for element-wise arithmetic involving vectors, matrices and dataframes, even though all three operations are syntactically identical. However, there are other cases in which the syntax would vary depending on data structure (e.g. positional indexing looks different for dataframes and matrices in Python), so these redundancies were kept in the interest of consistency.

We concede that there cases in which a more syntax-based ontology would make more sense. For example, if the user has already committed to a function (e.g. `reshape()`), moves on, and then later wants to debug it, it would be useful if the user could access the documentation based on that name. As a menu, a more comprehensive solution may have been a *faceted* classification system in which the user creates browsable trees on the fly [302], by choosing which facet (i.e. categorical level) to organise the tree around: objective, data structure, operation or syntax. This could theoretically be augmented with a built-in search engine that, for phrases like "Access row", infers the intended syntax. Neither faceted nor search-based features were seriously explored in our line of research, however, since graphics - not ontology structure - were the primary object of investigation.

Different iterations

The above-mentioned structure describes the final version of the ontology, but earlier versions did not consistently use the three levels. For example, it did not consistently subdivide objective-level categories with data structure categories, and occasionally broke down third-level operation entries into even finer categories, usually based on axis (e.g. *bind > by row/by column* or *pivot > wide to long/long to wide*). Following our usability study (see Section 8.1), this taxonomic depth was found to confuse users. Empirical studies that have investigated the trade-off between depth and breadth in menu design suggest that, in order to reduce search times and error, they

should minimise the number of hierarchical levels, ideally not beyond two, with at most eight items per level [303, 304]. Together, this prompted the reorganisation of the new version, which adhered to the same objective-structure-operation organisation throughout. In order to avoid exceeding three levels, the finer operation categories were either coalesced so that they instead were explained *inside* the documentation entry, or given their own third-level entries (e.g. *pivot long to wide* and *pivot wide to long* became their own operations).

Ontology contents

Having thus explained the ontology's structure, it remains to explain the considerations factored into the selection of operations. We had constrained the ontology contents to vectors, matrices, and dataframes, since these are tabular and easy to visualise in two dimensions. The ontology therefore excludes tensors with more than two dimensions, or non-tabular data such as trees. Notably, the ontology refers to structures on a conceptual level - i.e. as vectors, matrices, and dataframes - since those terms are not tied to any particular API implementation. Occasionally, documentation entries would briefly describe other structures like `list` and `tuple`, since some functions require them as arguments.

As previously noted, we sought to make the ontology language-agnostic, such that it maps easily onto commands in both Python (NumPy, Pandas) and R (Base R). We therefore elicited important commands from two textbooks, placed them within the ontology schema outlined above, and discarded entries that lacked a clear analogue in either of the languages (note that we did not seek congruity with other languages, such as Julia or Matlab). For example, in R matrix rows and columns can be given alphanumeric names, but NumPy ndarrays' rows do not - hence operations involving names in matrices had to be discarded from the ontology.

This multilingual requirement sometimes created tensions. Consider the decision to categorise operations based on whether they concern vectors, matrices and dataframes. In R, the correspondence with the three data structures is straightforward, since the relevant object classes are called `vector`, `matrix` and `data.frame`. In Python APIs however, both vectors and matrices are implemented as `ndarray`. This sometimes led to duplication, where vector and matrix-related entries mapped onto near-identical code. Furthermore, vectors could also be implemented as Pandas `Series`, which is the data structure that makes up the column in `DataFrame` objects. For this reason, vector-related documentation entries had to contain information about both.

6.1.2 Thumbnail graphics

The ontology serves as a backbone for a command menu in SLICE N DICE, used for accessing syntax documentation. To facilitate visual scanning, and to minimise any remaining ambiguity regarding the behaviour of the command, each operation within the menu was also assigned a

thumbnail graphic. Each graphic displays a pre-operation and post-operation state. For the programmer to quickly judge an operation as relevant and differentiate it from other, behaviourally similar functions, thumbnails need to be designed with several considerations in mind:

Spatial constraints: Thumbnails need to be small, only about 1.5cm in height. For this reason, the data structures displayed within them must be necessarily small, rarely exceeding six elements in length, and do not include any actual data. Although a number of operations could effectively be visualised using arrows (for example, a zigzag arrow to indicate the folding of a vector), such details would not fit comfortably within the limited area, and therefore were generally not used.

Widely used conventions: The less learning overhead required to interpret a graphic, the better. For that reason, the graphics rely only minimally on arbitrary symbols not found elsewhere.

Colour: Colour, in particular, is leveraged to convey the command's behaviour. Occasionally it is used to indicate selection, but elsewhere colour is used to indicate correspondences and categorical groups. In still other cases, colour saturation is used to indicate the relative magnitude of a value. This reliance on colour unfortunately means that the graphics may be less accessible to colour-blind learners and not easily printed in black and white.

General visual design principles: In their own cheat sheets, RStudio provides 9 visual design principles. Among those principles the recommendation is to choose a single highlight colour and a secondary, sparingly used colour for the sake of differentiation [10]. These principles will be applied to our thumbnails as well.

Archetypal: In practice, operations could involve data structures of any shape. It would not be possible for a thumbnail to encompass all possible instances. Instead, an *archetypal* shape had to be chosen. For example, element-wise vector arithmetic is illustrated using two input data structures of length 4, and not using an edge case such as vectors of length 1.

Ambiguity: In deciding which shape and data structure to use in a thumbnail graphic, it is important to minimise the risk that the user under-generalises or over-generalises the intended meaning. To illustrate this, suppose you were looking for a way to filter rows based on a condition and see the thumbnail in Figure 6.1. It could mean “Return rows where C (or B) equals 7”, but it could also mean “Return the row named 'e'”, “Return the fourth row” or even more plausibly, “Return the last row”. Removal of such ambiguity requires that the graphics observe tacitly held assumptions about visual communication, for example the assumption that a graphic only includes relevant information and that any edge cases (which may cause under-generalisation) are not accidental.

	A	B	C
'n'	2	1	8
'a'	1	4	4
'm'	2	5	5
'e'	2	7	7

Figure 6.1: An example of an ambiguous thumbnail.

6.1.3 Walkthrough of ontology

What follows now is a tour of the ontology (and associated thumbnails) as it looked in its final shape, which is the version that was used in the SLICE N DICE project described in Chapter 7. The tour proceeds through the high-level categories, one by one, describing its contents, use cases, and thumbnail graphic design. The full version is available in Appendix C.

Create

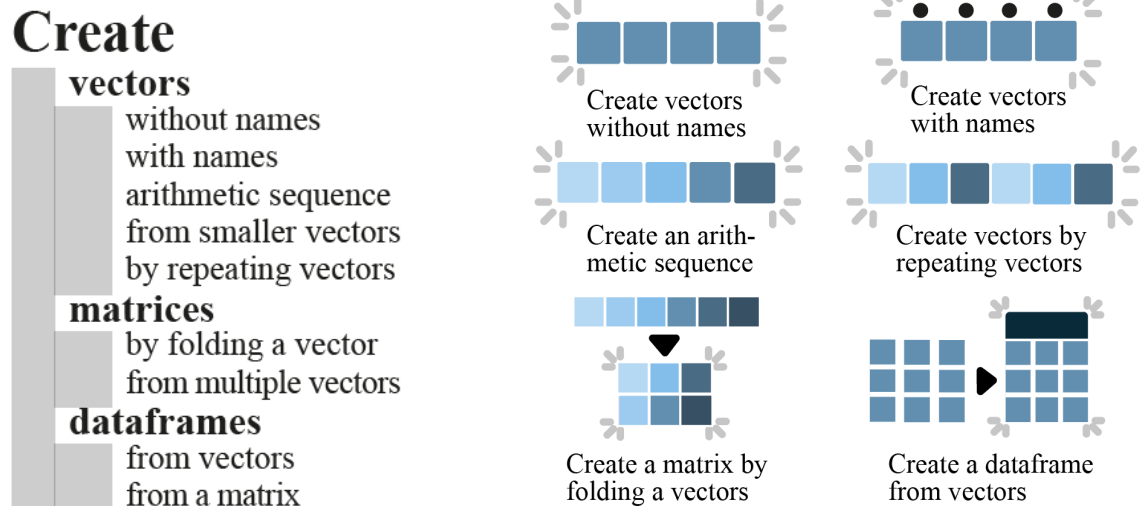
Structure. Introductions to APIs often begin by showing how to construct the fundamental objects involved. In the current case, that would mean the three data structures. In actual practice, data sets are unlikely to be created manually, but are rather read in from local files or database endpoints. However, on-the-fly data structure creation is common enough to justify learning it, especially for instructional purposes, since reproducible code examples in documentation tend to feature them. Other common cases are creating placeholder arrays for storing computational results, or generating an index vector, such as an arithmetic sequence for the purpose of accessing every third element. The *Create* category is represented in Figure 6.2.

Thumbnail graphics. The thumbnail graphics associated with the *Create* category are surrounded by sunbursts in the corners. These sunbursts were deemed necessary since some of the operations in this category create a structure from nothing, and therefore have no “pre-state”. Simply displaying a static vector would seem to suggest that no change had taken place.

Some of the vector-creating functions operations produce distinctive relationships among the values. For example, an arithmetic sequence is archetypally one with ascending values: to indicate this, the shade of vector cells increases at regular intervals. Similarly, a vector with repeating elements would display a sub-vector repeated twice, with shade used to indicate correspondences.

In some cases the operation creates a new structure from simpler data structures². In these

²These operations could also qualify for the *Combine* category. Though their current placement within *Create* adds a degree of inconsistency, it serves a purpose within SLICE N DICE for providing more possible operations in the first few exercises of Part 1.



(a) The *Create* category contains 9 data (b) A subset of the *Create* thumbnails. Sunbursts help indicate change when there is no clear pre-transformation state.

Figure 6.2: The *Create* category.

cases, shape is generally sufficient to convey the change. One exception is *Create a matrix by folding a vector*, since it requires shading to indicate correspondence between the value's position in the vector, and the value's position in the matrix. A subset of the thumbnails are shown in Figure 6.2b.

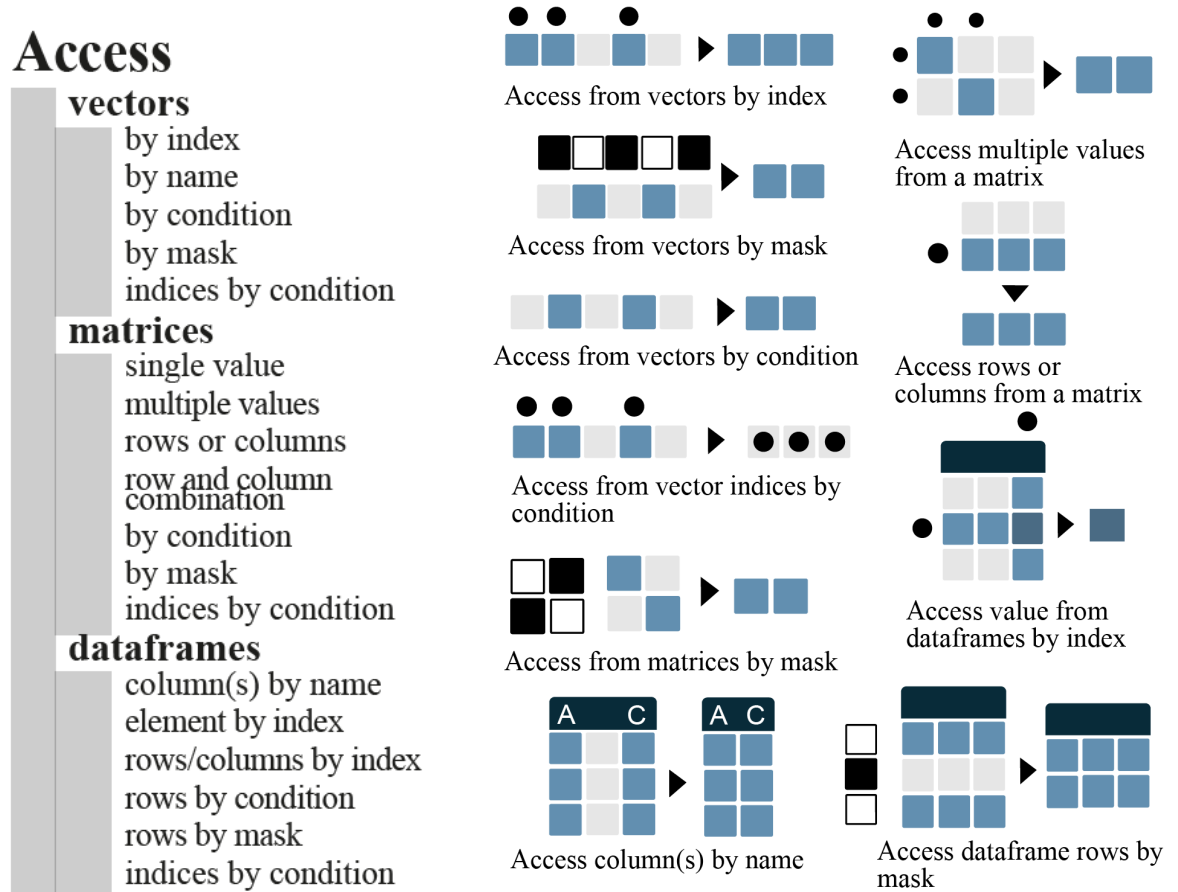
Access

Structure. Subsetting and indexing operations - what we call *accessing* here, to minimise unfamiliar jargon - forms a ubiquitous task in data wrangling. It includes tasks as simple as accessing the third element in a vector, to accessing elements based on any arbitrary number of conditions. In general terms, there are three types of accessing methods: *position-based* (i.e. using an index), *label-based* (i.e. using an alphanumeric label), and *Boolean indexing* (which could either be direct, via a Boolean mask, or indirect, via a logical condition). For a specific data structure, there is usually an idiosyncratic way of accessing values that reflects the data structure's purpose. For example, dataframe columns are usually accessed via their label and dataframe rows based on their value (via a condition). The ontology's contents (seen in Figure 6.3a) prioritises the most common use cases, but also makes a point out of distinguishing between frequently overlooked subtleties. For example, there is a difference between accessing a *combination* of rows and columns in a matrix (`mat[0:1, 0:1]` in NumPy) and a collection of individually specified elements (`mat[[[0, 1], [0, 1]]]`). Another example is the difference between accessing *values* based on a condition (`x[x==2]`) and accessing the *indices* where the condition passes (`np.where(x==2)`).

Access-related operations are not only used for accessing, but also for modifying (`x[0]=5`)

and incrementing values ($x[0]=x[0]+5$) in the structure. These were not regarded as distinct operations, but the documentation entries will make that distinction internally.

Thumbnails. Access-related operations have in common that only a subset of cells (usually) remains after they have been applied. As is already ubiquitous in spreadsheets and the graphical scaffolding examples surveyed in the literature review, one intuitive design choice is to highlight the selected subset. That convention has been used here too, but colour highlights are not sufficient to distinguish between Boolean, label-based and position-based indexing. Some additional graphical elements were therefore needed. For position-based indexing, a black circle was added. For label-based indexing, a black square was used (for dataframe columns, a column label was included). For Boolean indexing, a black-and-white mask was shown for masking operations, while for condition-based access, the values were highlighted without any other elements included, to emphasise that the subset was determined by the contents. A selection of the thumbnails are shown in Figure 6.3b.



(a) The *Access* category contains 18 (b) The *Access* thumbnails use colour highlighting to indicate data operations. selection and black circles to indicate index.

Figure 6.3: The *Access* category.

Calculate

Structure. Data wrangling-related calculations can be distinguished by their arity and cardinality. *Element-wise* operations are characterised by their one-to-one relationship, in how the first element of one vector is added to the first element of the other. Element-wise arithmetic could involve unary (`np.floor(x)`), binary (`np.add(x, y)`, i.e. $x+y$) or higher-arity functions, but in the ontology, *element-wise* formed its own third-level category. In both R and NumPy, arithmetic between an array structure and a scalar causes the scalar to be *broadcast* so as to effectively form an element-wise operation: to draw attention to this edge case, it was given its own category (*with single value*).

Another common calculation is defined by a many-to-one relationship, namely *aggregation*. The typical example is summing all elements in a vector to obtain a single scalar representing the sum. Aggregations could be applied to an entire data structure or to subsets of elements. In matrices they could also be done on a column-wise or row-wise basis, while in a dataframe they could additionally be done on a group-wise basis, where groups are defined by a categorical variable. In the ontology (Figure 6.4a), these types of aggregations have been given their own entries.

Other operations included the derivation of a new column in a dataframe, as well as pair-wise arithmetic, in which each element of one structure is combined with every element in another, to produce a matrix that stores the combination of each possible pair.

Thumbnails. As mentioned, we primarily distinguish calculations based on whether values are processed in parallel or aggregated to produce a single value. Shape is generally sufficient to represent this, but when the calculation involves subsets, colour is needed to distinguish those subsets. In the case of row- or column-wise aggregations, shade is used, since the columns tend to be of the same numeric kind. In the group-wise case, hue is additionally used to distinguish variables from one another. Group-wise aggregations present an especially interesting case, since it actually implies three stages (i.e. split, apply, and combine). The grouping generally produces an intermediate data structure in which rows are internally grouped (e.g. `DataFrameGroupBy` in Pandas). In the thumbnails, this intermediate state is included. The graphics are shown in Figure 6.4b.

Combine

Structure. Data structures can be combined in two main ways - either bound together along their vertical or horizontal axis (i.e. *concatenated*), or merged. Although several different joins exist (e.g. outer joins, left joins), we limited the ontology to inner joins. The structure is shown in Figure 6.5a.

Calculate

vectors

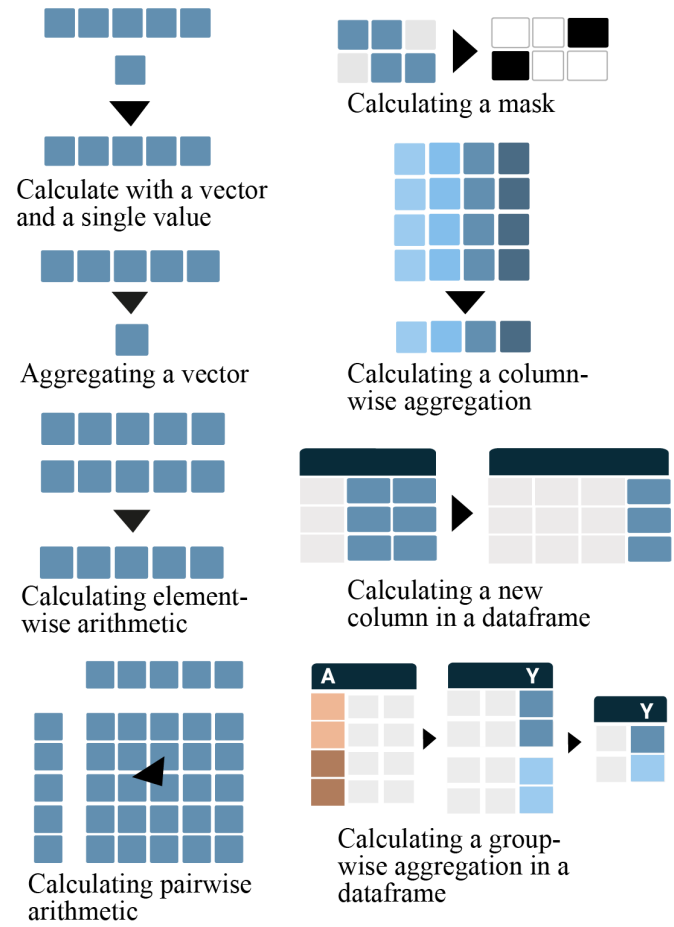
- with single value
- elementwise
- aggregation
- pairwise

matrices

- with single value
- elementwise
- aggregation
- mask
- rowwise aggregation
- columnwise aggregation

dataframes

- with single value
- elementwise
- calculating new column
- aggregation
- rowwise aggregation
- columnwise aggregation
- groupwise aggregation

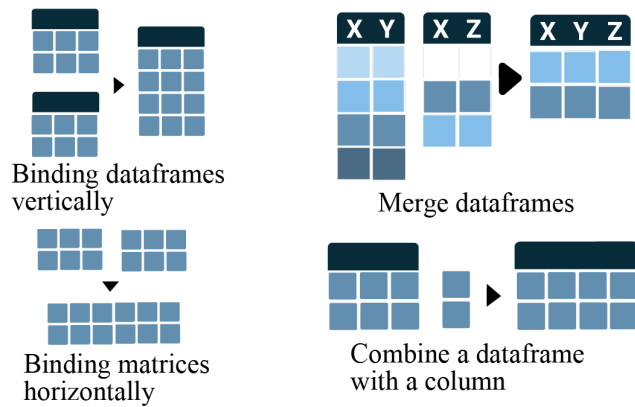
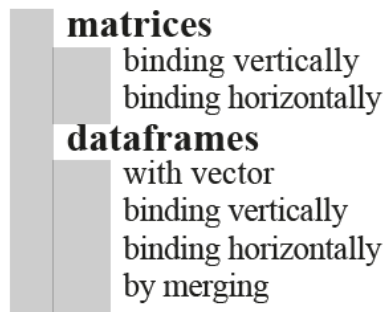


(a) The *Calculate* category contains 17 data operations.

(b) The *Calculate* graphics mainly rely on shape to convey its logic.

Figure 6.4: The *Calculate* category.

Combine



(a) The *Combine* category contains 6 data operations.

(b) The *Combine* graphics mainly rely on shape, except for merging.

Figure 6.5: The *Combine* category.

Thumbnails. For operations that involve binding, shape is sufficient to convey the operation's logic. Merging is more difficult, since it involves binding together observations that may be in different order, and the discarding of observations that do not appear in both structures. In the end, shade was used to identify rows. The graphics are shown in Figure 6.5b.

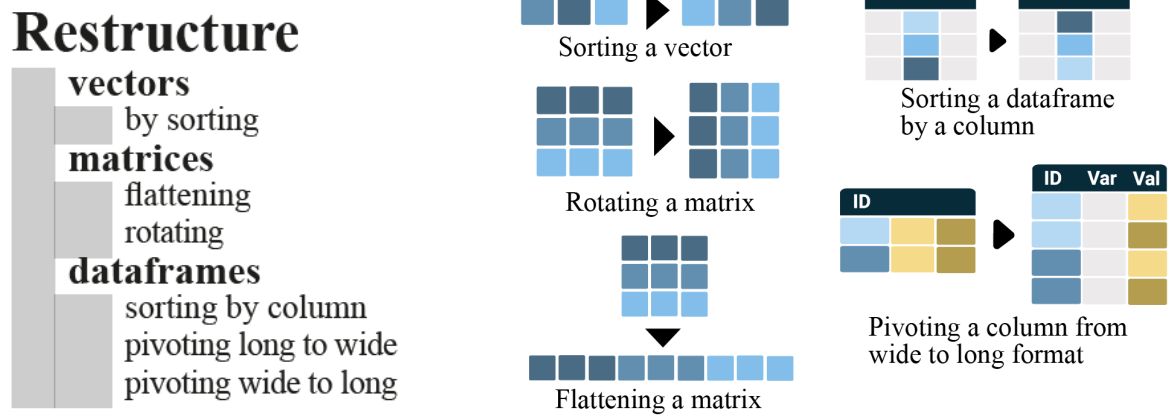
Restructure

Structure. *Restructuring* refers to any rearrangement of values within the data structure. This includes sorting vectors, transposing and flattening matrices, as well as pivoting dataframes between a long to a wide format. The category's structure is shown in Figure 6.6a.

Thumbnails. In conveying how an operation rearranges the values, the key lies in establishing a correspondence between an element in its prior position and the element in its subsequent position. This was again done using shading. However, as with the group-wise aggregations, since pivoting involves columns that carry distinct roles, the columns were also given distinct hues. Subsets of rows within those columns were then indicated by shading (Figure 6.6b).

6.2 Subgoals for plan composition

Subgoals are the second instructional feature that graphics are hypothesised to improve. They consist of two main components: subgoal labels and subgoal graphics. The subgoals are meant to provide a high-level plan that the learner could base their solution implementation around. They are therefore specially made for a particular exercise. This could be contrasted with Margulieux et al. [234], where subgoals are used to augment structurally isomorphic worked problems. One reason for applying subgoal labels to the exercises, as opposed to separate worked examples, is that worked examples would require the number of problems that a user is given

Figure 6.6: The *Restructure* category.

to be effectively doubled. Moreover, as discussed in the literature review, data wrangling as a programming domain is less pattern-governed and more functional than the domains covered by previous subgoal labelling research, which gives worked examples less of a reuse value.

6.2.1 Subgoal labels

After creating a set of data wrangling exercises (the design considerations of which are explained in Chapter 7), each exercise was decomposed into a series of subgoal labels. This decomposition process was governed by three main constraints:

Correspondence with a single operation: Each subgoal was mostly constructed so as to map onto one operation from the ontology. The reason for this was to help stylise the workflow so as to (hypothetically) require one ontology lookup cycle per subgoal, which would make any downstream analyses of the workflow easier to interpret. In early exercises, this constraint was occasionally a source of awkwardness, usually due to a mismatch between different APIs. For example, in NumPy the repetition of entire arrays and repetition of values are accomplished through separate functions (`np.tile` and `np.repeat`, respectively), which would imply separate subgoals. Meanwhile, in R the `rep` function can be configured to accomplish both at the same time. In R, the choice of having two subgoals may therefore appear contrived.

For later exercises, this constraint was loosened somewhat to reflect how the learner may chunk common usage patterns. For example, to aggregate a variable group-wise, the correct column would first need to be accessed. A subgoal would then correspond to both the column selection and the subsequent aggregation.

Phrased at the problem context level: Another consideration was the need to not “spoil” which

API command to use so as to effectively provide the solution. To accomplish this, we took care to express the subgoals in plain English and preferably in terms closer to the particular problem context. Exercises ranged in their realism - from context-free to highly contextualised - and this dictated the highest possible abstraction level. For example, subgoal labels in an artificial problem could be expressed in terms of *rows* and *columns* (Figure 6.7a) while more realistic problems have subgoals using problem context terms (e.g. *countries* and *heights*, see Figure 6.7b).

Independent of the graphics: Finally, in order for the control group (i.e. the group without graphics) to not be at a disadvantage, the labels had to be self-contained so as to not *depend* on information within the graphics. Otherwise, we would not be able to claim that the experimental group and a control group were informationally equivalent.

6.2.2 Subgoal graphics

In the experimental group, subgoal labels are accompanied by subgoal graphics. For each label, there would be a corresponding graphic that visualises the necessary data transformations. These graphics are meant to provide a structural intuition of the necessary change, by for example indicating whether vectors should be combined vertically or horizontally, and whether a particular calculation results in a single value (i.e. aggregation) or a structure with the same size as the input (i.e. element-wise). In contrast with thumbnail graphics, which are meant to be archetypal, the subgoal graphics are tailored to the particular problem and data set.

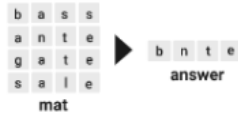
To avoid confusion, it was important to make the subgoal graphics stylistically consistent with the thumbnail graphics. Although subgoal graphics can be larger than thumbnail graphics, it would still be unwieldy to display real data within the structures. The subgoal graphics therefore show the pre- and post-operation states of a data structure primarily using highlighting and shapes. This is possible since all the data sets are small enough to fit within a graphic - with larger, real-life data sets, these shapes would have to be abstracted further.

In addition to showing the pre- and post-operation shapes, the graphics occasionally include annotations to specify *which* condition is applied (e.g. “Women who scored above 70%”) and *which* arithmetic operator is used (e.g. “Add”). This is so that the graphics do not depend on the subgoal labels in order to be interpreted. Again, these were expressed in natural language in order to not reveal fully which API command to use. Two data wrangling problems are shown in Figure 6.7, along with subgoal graphics and subgoal labels.

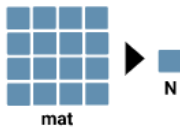
6.3 Chapter summary

- The SLICE N DICE system will involve two non-graphical scaffolding features: subgoal labels and a command menu. These will be augmented with subgoal graphics and thumb-

How would you access all the diagonal elements from a matrix `mat`, from the top-left to the bottom-right corner?



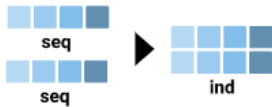
1. Get the number of rows in `mat` (save to: `N`)



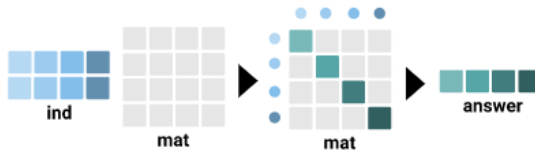
2. Use `N` to create a sequence that contains every row index, up until `N` (save to: `seq`)



3. Construct a matrix containing `seq` twice as rows, so as to indicate the row and column indices of the desired elements (save to: `ind`)



4. Use that matrix as index for `mat` (save to: `answer`)



(a) Exercise 6 in Part 3 is relatively artificial. It asks of participants to extract the diagonal without any convenience function such as `diag`.

Suppose you have a dataframe (called `women`) containing the number of kids that each woman has, along with their income and the countries they lived in.

In the countries with more than 2 kids per woman on average, what is the mean income?

ID	country	nr_kids	income
1	'C1'	3	200
2	'C1'	2	300
3	'C2'	0	100
4	'C2'	4	200
5	'C3'	5	50
6	'C3'	5	100

women

mean_income
250
75

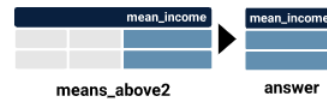
1. Categorise each country's average number of kids and average income (save to: `means` with column names 'mean_kids' and 'mean_income')



2. Get hold of the countries that have, on average, more than two kids (save to: `means_above2`)



3. Obtain those countries' income means as a single column dataframe (save to: `answer`)



(b) Exercise 13 in Part 3 is a more realistic task, similar to SQL.

Figure 6.7: Examples of two exercises, complete with subgoal labels and subgoal graphics, which are only visible for the SG condition.

nail graphics, respectively.

- The menu will be organised as a three-level ontology. The first level represents high-level objective, the second represents data structure, and the third level represents operations.
- Operations are language-neutral concepts that map onto analogous API commands in both Python (NumPy/Pandas) and Base R.
- Thumbnail graphics are small, archetypal depictions of an operation's behaviour. They do not include data and mainly rely on colour and shape.
- Subgoal labels are steps in the solution plan for a particular problem. They are expressed in as abstract and natural-language terms as possible, to avoid spoiling exactly which operation is meant to be used.
- Subgoal graphics are superficially similar to thumbnails, but are tailored to a specific problem. Just like thumbnail graphics, they do not display data, but unlike thumbnail graphics, their shapes reflect the shapes of the actual data structures. Moreover, the subgoal graphics may contain short annotations to communicate exactly which calculation is necessary.

Chapter 7

Design of Slice N Dice

Our methodological discussion of Chapter 4 concluded that, to address the research questions, computer-based instrumentation was necessary in both the delivery of the intervention, and the collection of behavioural data that result from it. Moreover, given how volunteer supply is such a limiting factor in human-subject experimental research, it would be more cost-effective if the system simultaneously evaluated the effect of both subgoal graphics (to address RQ1) and thumbnail graphics (RQ2), using 2x2 factorial between-subject design. The system would therefore exist in four different variants. The subgoal graphic condition will be referred to as **SG** (indicating that graphics were present) and \neg **SG** (indicating absence), while the thumbnail graphic condition is referred to as **TG** (presence) and \neg **TG** (absence). This is represented in Figure 7.1.

We conceived of the system as an accelerated tutorial to get programming novices up and running quickly with authoring short data wrangling scripts. Such a software artefact could take many potential forms, for example an IDE plugin or a standalone desktop application. Others have found that the installation and configuration of software often pose a major hurdle to novices' introduction to data science [9]. For this reason, and because it would be time-

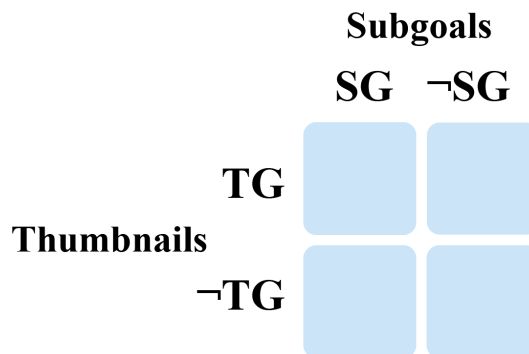


Figure 7.1: SLICE N DICE incorporates two experimental factors: one that manipulates the provision of subgoal graphics (**SG**, \neg **SG**) and thumbnail graphics (**TG**, \neg **TG**). \neg indicates absence of graphics. This produces 2x2 distinct conditions.

consuming to assist participants remotely with their software setup, the experimental platform was implemented as a web application. As such, a participant could simply access the website - called SLICE N DICE - via a URL within their browser ¹.

This chapter will detail the design and evolution of the data wrangling tutorial built into SLICE N DICE, as well as the user interface. Its structure evolved over the course of multiple pilot studies, described in Chapter 8. In this chapter, the platform will mostly be discussed component by component, with early iterations briefly described along the way.

Technology stack

The SLICE N DICE web application was developed by me using the MEAN stack (MongoDB as the database, Express as the web framework, Angular for the front-end and Node for the back-end). We sought to make the system self-contained, such that the user would not have to switch between multiple applications or tabs. To accomplish this, we surveyed various plugins that allow the user to author and execute R and Python scripts in the browser. While there are several in-browser Python implementations (e.g. Skulpt², Brython³) and at least one that supports NumPy and Pandas (Pyodide⁴), fewer options exist for R. Because of the anticipated need to attract participants with preferences for either language, it was deemed important to support both. The service we settled for, called DataCamp Light ⁵ (DCL), is an open-source plugin that lets the developer embed IDE widgets within the HTML, and have the code (both Python and R are supported) execute server-side. DCL was created by DataCamp, a data science MOOC company that handles the back-end execution of the programming sessions. The widget contains a scripting area and an interactive shell, and can moreover include assessment elements such as submission tests, hints, and solutions. It was therefore well-suited to the needs of SLICE N DICE, although a few modifications were made to the DCL source code that will be discussed later. The tech stack is summarised in Figure 7.2.

7.1 Overall structure

The desired learning outcome of SLICE N DICE was the ability to write short scripts for wrangling data programmatically. This target skill combines two distinct competencies: the ability to accurately identify and combine data wrangling operations, and the ability to look up API commands and accurately combine those commands into an executable script. We will refer to these as *non-programmatic data wrangling* and *programming*, respectively. We reasoned that

¹Assuming the browser is Chrome, FireFox or Edge, and has cookies and JavaScript enabled.

²<https://skulpt.org/>

³<https://brython.info/>

⁴<https://pyodide.org/>

⁵Available at <https://github.com/datacamp/datacamp-light> and protected by GNU Affero General Public License.

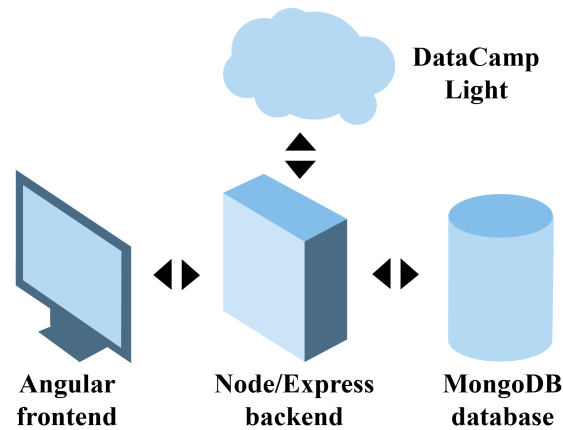


Figure 7.2: The system was implemented as a web application, using standard web technologies such as MongoDB, Express, Angular and Node (a MEAN stack). DataCamp Light (DCL) was an important external dependency that provides the IDE front-end widget and back-end code execution.

both skills had to be securely in place before the learner begins specialising in data wrangling API syntax, since no prior knowledge would be presumed. As a result, in all iterations, SLICE N DICE took on a segmented structure, with a preliminary block of training activities that the learner would have to go through before being exposed to data wrangling API syntax.

The platform is centred around a menu of data operations, which is detailed in Section 6.1. From the beginning, the idea was that SLICE N DICE would contain a foundational part that introduces the user to the menu’s contents by traversing it, operation-by-operation, while displaying a short, conceptual description of the current operation within an **operation card**. Different SLICE N DICE iterations have either interleaved programming concepts *within* these operation cards, or introduced programming *after* the learner has been familiarised with the menu contents on a non-programmatic level.

7.1.1 Spring 2020 pilot

In the first prototype to be tested by student volunteers, the app consisted of two main parts: a preparatory Part 1 and a main Part 2. Originally, this Part 1 was organised as a deck of operation cards that contained both a conceptual explanation *and* corresponding code syntax. After every five cards, a multiple-choice question would appear that asked them about the syntax of one of the recently presented operations. After inviting a handful of pilot participants to try out the app in person, it was clear that learners were frustrated at being confined to MCQ exercises about the syntax and not having the opportunity to program immediately. Moreover, the need to memorise syntax contradicted the menu’s role as an external memory for syntax.

In response to this feedback, Part 1 was re-designed to instead include 59 programming exercises, one following each operation card. Within such an exercise, the participant was shown the documentation entry and asked to apply the code example for a new context. The exercises

were technically optional - the learner could simply flick through them - and the correct solution was available. Part 2 meanwhile consisted of 8 data wrangling programming problems, and constituted the main part in which subgoal graphics would become available.

This version (represented in Figure 7.3a) was trialled during Spring 2020 using a blended delivery mode, where student volunteers from across the university signed up to attend in-person sessions in a computer laboratory. At this point, the ontology and the set of exercises was different than the final structure outlined in Section 6.1. Additionally, due to sample size concerns, subgoal and thumbnail graphics were manipulated as a within-subject variable: each exercise was randomly assigned to one of the 4 conditions.

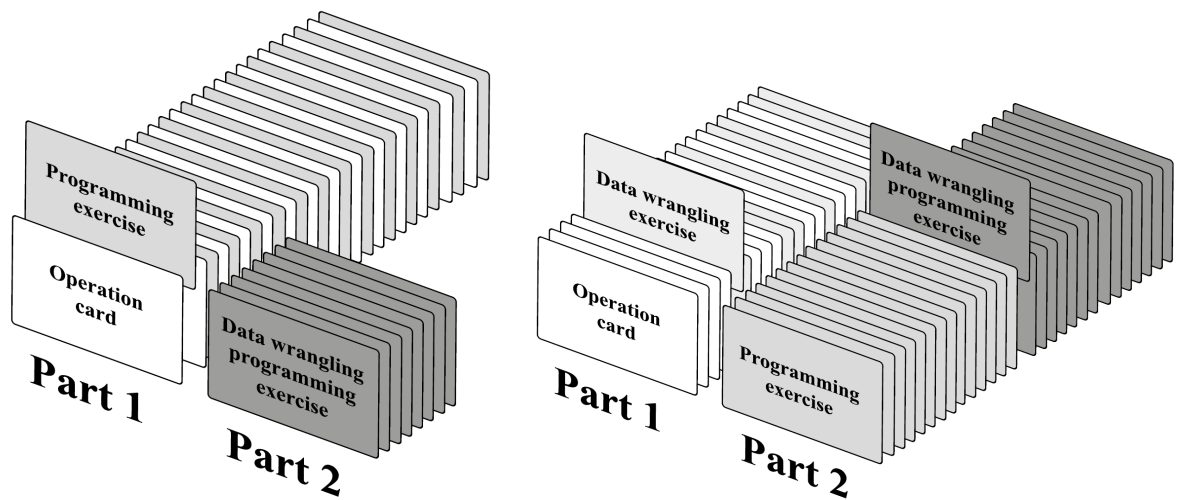
Although it was clear from informal observations during this pilot that this structure was motivationally effective - volunteers signed up with the aim to learn data scientific programming and immediately got exposed to it - it suffered from two important methodological flaws. The first was that, by interleaving the exposure to programming concepts within the introduction of data wrangling concepts, we would not be able to measure the comparative influence of conceptual versus programmatic skills on data wrangling performance. It therefore wasted an opportunity for collecting data relevant to theory-building.

Secondly, because Part 1 was meant as a preparatory part, it did not feature any experimental manipulation and therefore could not address the research questions. This became a problem when - probably due in part to Part 1's repetitive design - many students dropped out before even reaching Part 2. It is impossible to estimate how severe the attrition was, since COVID-19 caused this study to be abruptly discontinued, but out of 88 participants who began Part 1, only one participant completed Part 2. Consequently, this iteration yielded little valuable data and underscored the importance of designing the application with participant engagement and retention in mind. It was clear that the "preparatory" part had to be either shortened, made more engaging, or made to incorporate an experimental manipulation relevant to the research question, so that incomplete data would still be valuable.

7.1.2 Autumn 2020 pilot version

Following this feedback, the structure of the app was overhauled. Part 1 remained a deck of operation cards that the learner systematically traverses through. However, in order to shorten its duration, the 59 programming exercises were replaced by 10 **non-programmatic data wrangling tasks** scattered throughout the card deck. In these tasks, the user is presented with a data wrangling exercise that has already been broken down into a set of subgoals, complete with labels and (for the SG group) graphics. For each subgoal, the user is asked to identify and select the operation that is required to accomplish the subgoal. Since the introduction to programming was removed from Part 1, it was placed in Part 2, as a set of simple programming exercises that transition into more complex data wrangling programming exercises.

This design (shown in Figure 7.3) was used in a small usability study during Autumn 2020,



(a) The structure of a spring 2020 pilot.

(b) The structure of a autumn 2020 pilot.

Figure 7.3: To balance pedagogical and methodological considerations, the application went through multiple iterations.

which was focused on refining and vetting the exercises, as well as a more extensive, subsequent pilot study. Both are described in Chapter 8. At this point, confidence in our ability to recruit and retain participants had grown and the experimental conditions were turned into between-subjects conditions, in order to reduce any spillover effects. Furthermore, the ontology and the graphics were improved, along with smaller interface modifications described later in the chapter.

7.1.3 Final version

The final version remained largely the same as that of the autumn 2020 pilot study, but redistributed the data wrangling exercises of Part 2 into their own part. This allowed participants with prior programming experience (but without a data wrangling background) to skip exercises they may have found too easy. This allowed SLICE N DICE to appeal to a larger audience. Additionally, an explicit three-part structure - where non-programmatic data wrangling, introductory programming, and programmatic data wrangling all had their own dedicated part - reinforces the idea that SLICE N DICE trains and tests distinct competencies in isolation from each other.

One second design modification of note was the inclusion of **unscaffolded exercises** in Part 3. “Unscaffolded” means that subgoals (and any graphics and hints associated with them) were removed for a limited duration. This afforded us the ability to, as a dependent variable, see whether subgoal graphics would influence the ability to plan a solution single-handedly, when the “training wheels” are taken off. The menu and thumbnails remained present during these exercises. The final design is summarised in Figure 7.4. It contains 9 data wrangling exercises in Part 1, 10 programming exercises in Part 2, and 18 programmatic data wrangling exercises in Part 3, 3 of which are unscaffolded.

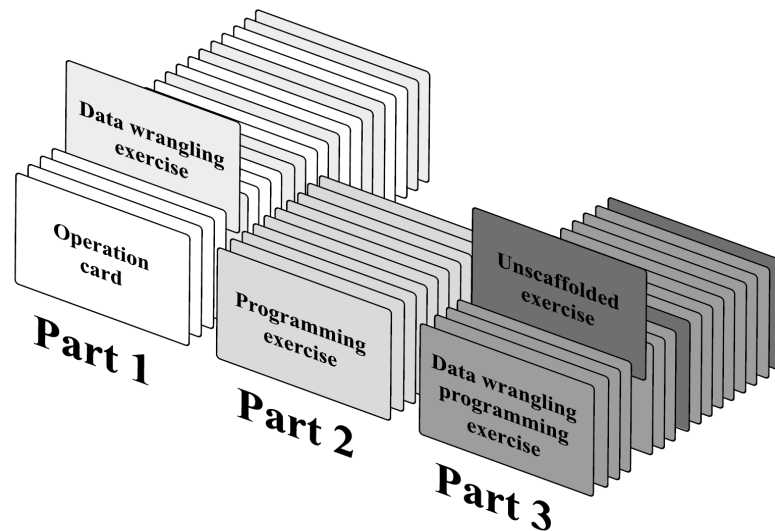


Figure 7.4: The final structure of SLICE N DICE. Part 2 has been split into two parts and 3 data wrangling programming exercises are unscaffolded - without subgoals for the first 10 minutes.

7.2 Recurring design considerations

Throughout the design of SLICE N DICE, a number of constraints and considerations loomed large, mirroring the trade-offs explored in Chapter 4. Since their implications spanned across all three parts, we will discuss them here, in advance of our more in-depth description of each part.

7.2.1 The problem of low stakes

Since participation in SLICE N DICE will not carry any academic credit or impact participants' grade, it is a low-stakes assessment. As a result, the learner may not be motivated enough to exert their best effort [305]. This, in turn, means that the recorded performance levels would under-represent the true ability of low-effort participants, and introduces a major confound where high-ability/low-effort learners may perform at the same level as low-ability/high-effort learners. How could participants be encouraged to exert more effort, despite the lack of stakes?

According to Eccles-Wigfield expectancy-value model of learning motivation, the effort a student chooses to invest depends on their own perceived competence, the perceived task difficulty and costs, how relevant it is to their future (*utility value*), how much performance matters (*attainment value*), and how enjoyable it is (*intrinsic value*) [306]. This model suggests several levers for increasing participant effort. For example, the platform should be high in its direct utility for the participants (not just for the experimenter) and marketed as such.

Although SLICE N DICE is principally an experimental vehicle, as opposed to an educational service, the line between experiment and service becomes blurry given that, to recruit an adequate number of participants, the study would have to be as intrinsically rewarding and attractive as possible. Considerable time and effort therefore went into honing the platform be-



Figure 7.5: The landing page reveals the effort to make SLICE N DICE intrinsically rewarding and attractive, to increase the likelihood of a participant making an effort despite the low stakes.

yond what the scientific objectives technically would require. For this reason, the result is at once a high-fidelity education product, designed to be directly useful, and a research prototype, designed to answer research questions. This is exemplified through the visual appearance of the platform (including its landing page, Figure 7.5), which mimics that of popular tech companies, as well as the marketing effort, which involved video advertisements and the creation of a bespoke data wrangling cheat sheet e-book to incentivise completion.

7.2.2 The assistance dilemma

Another impact of the issue of low stakes was a fear of ever putting a participant in a situation where they perceived a task to be too difficult (or themselves as too incompetent), causing them to drop out. To prevent this, the app had to include some measure of guidance if the student ever gets stuck. All three parts therefore include hints of different kinds.

The provision of hints opens up what Koedinger and Aleven have called the “assistance dilemma” between giving and withholding information [307, p.239]. It is a well-documented phenomenon that learners sometimes make unproductive use of hints, for example requesting help before even trying, or mindlessly clicking through them [308]. Unconditional hints therefore reduce the attainment value, since participants’ effort matters less to their ability to progress within the course, reducing overall learning gains. Methodologically, the provision of hints introduces a troublesome co-variate, since a patient (i.e. hint-averse) participant would take longer to solve an exercise than a less patient participant of similar ability. And yet, if excessive frustration is not averted, only a disciplined few would persist.

Several theoretical solutions exist to the dilemma. The simplest would be to explicitly instruct against excessive hint use. Another would be to gamify the application so as to penalise hint use by deducting points from an experience point score (a technique used by DATA CAMP).

A more sophisticated solution would condition the hints' availability based on the learner's performance level, meta-cognitive skills, or motivational state, the way a skilled human tutor can. This was far beyond the scope of the current research questions, however. Generally, hints will be provided unconditionally but with decreasing availability: in Part 1 and 2, they are presented automatically upon error as feedback, while in Part 3 they require the participant to actively request them, one subgoal at a time.

7.2.3 Exercise design

Although the task type changes from non-programmatic in Part 1 to programmatic in Part 3, the data wrangling exercises themselves remain consistent in style. Each exercise presents an image of the data structure in its initial state, as well as the data structure in its required state, and the exercise description is written in plain English, avoiding technical API terminology. The exercises required between 2 and 6 data operations in order to be solved, and were generally sequenced so that later exercises were more complex (involved more operations), contextualised, and linguistically incongruous with with solution than early ones. An example of an early, artificial exercise is “Given matrix `mat`, square every number and sort the sums of each row from large to small”. An example of a later, contextualised exercise is “Which country has the highest male average BMI?”. We thus employed a faded scaffolding technique. Only operations from the ontology would be necessary, so that SLICE N DICE was completely self-contained. We deliberately chose data sets that were small enough for a solution to be verified mentally, since Part 1 was non-programmatic. Finally, the exercises are well-structured, in the sense that they have a fixed answer. In both Part 1 and 3, test cases were embedded to check the correctness of their response. All exercises can be found in appendix D.

These exercises were drafted and refined in response to feedback from the usability study and the autumn pilot study. Any exercises deemed too simple, difficult, or confusing were removed or modified prior to the spring 2021 capstone study. Furthermore, a web form was embedded that allowed participants to directly report any lingering bugs or errors in the exercises. Although the exercises and accompanying problem-solving content (e.g. graphics, test cases, subgoals) were handcrafted, there is no obvious reason for why such exercises could not be auto-generated in the future.

7.3 Part 1 design

Part 1 consists of a deck of 65 operation cards, with 9 data wrangling exercises interspersed among them. The overall goal of Part 1 is to cultivate structural intuitions about how a data wrangling problem can be decomposed into a sequence of operations, as well as to familiarise the participant with the contents and structure of the menu, and with the style of the thumbnail graphics, subgoal labels and subgoal graphics. A participant should therefore arrive at Part 3

well-acquainted with the scaffolding features and with all the data operations. We will now discuss the design behind the operation cards and exercises.

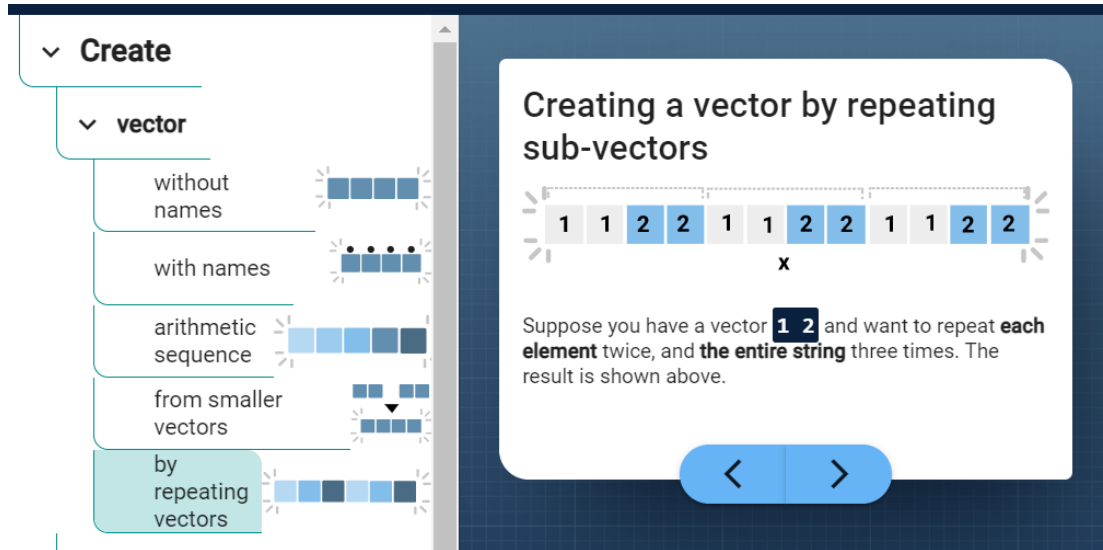
7.3.1 Operation cards

The interface in which operation cards are displayed shows the menu in a sidebar to the left, along with thumbnails in the TG condition. As the deck of cards is browsed, the ontology is virtually traversed from top to bottom. The menu is automatically opened to reveal the current operation, which additionally is highlighted. The card itself contains a short explanation of how the operation works, along with an example. Since we sought to clearly separate data wrangling from programming, this example is *not* a code example: rather, it simply displays the pre-operation and post-operation state of the data structure.

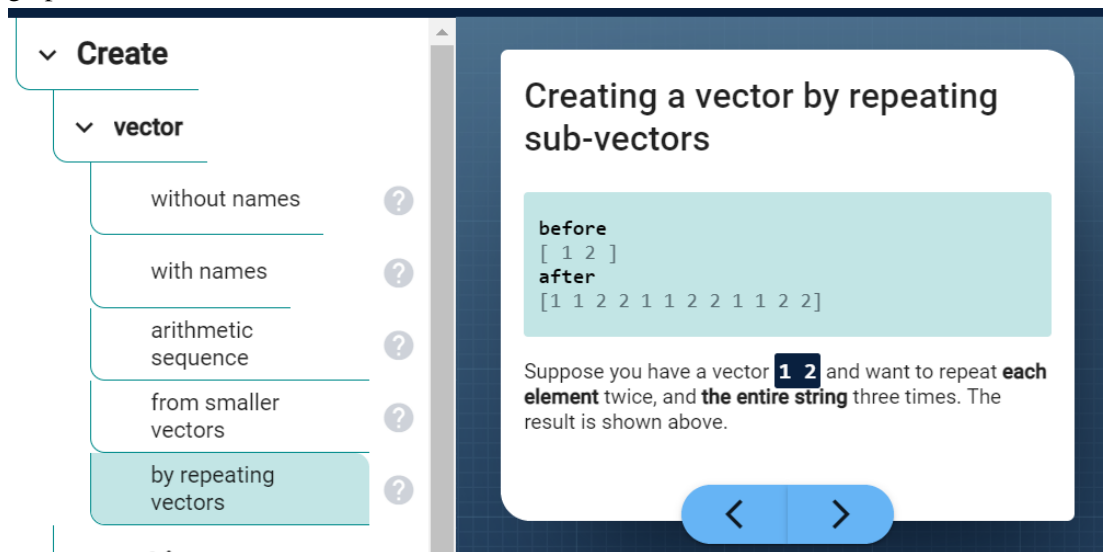
Initially, in the spring pilot study, this example was textual for every participant, with a shell-like representation that uses a style similar to that of standard shell output (e.g. `[4 5 6]`). However, because the operation card was synchronised with the menu, the operation card example also seemed like a good opportunity to reinforce the impact of thumbnail graphics, by representing the example graphically. We therefore added an additional detail to the TG condition, conjoined with the presence of thumbnail graphics, that we refer to as **operation card graphics** (see Figure 7.6a). These graphics were in the same style, and contained the same content as the shell-like representation given to \neg TG (Figure 7.6b). This therefore afforded us an opportunity to test whether operation cards were faster to read when the example was represented graphically.

Tooltips

Importantly, the goal with SLICE N DICE was not to measure working memory and memorisation. Indeed, the user was informed, via an interactive on-boarding tour, that they did not have to memorise anything. To make the operation card explanations available within the data wrangling tasks, we instead experimented with **tooltips**. In early iterations (up until the pilot study), the tooltip was a simple strip containing the operation's name (e.g. *Create a dataframe from a matrix*). This was not deemed to be informative enough, so in the final version the tooltip was fitted with the contents of the operation cards. As with the operation cards, this meant that the TG condition also had a graphical tooltip (Figure 7.7a) and that the \neg TG group had a shell-like representation (Figure 7.7b), with the text content being the same. This tooltip was triggered when the user moused over the thumbnail. Users were informed of the tooltip feature via a guided tour.

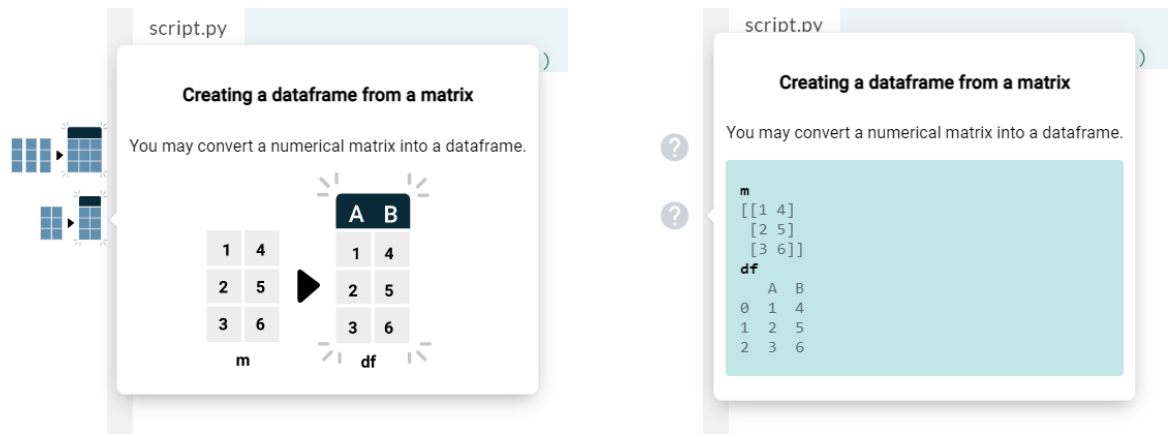


(a) The TG condition features thumbnail graphics and operation cards with similarly styled graphic with data.



(b) The \neg TG condition displays question marks instead of thumbnails and the operation card displays its example using a shell-like representation.

Figure 7.6: The sidebar menu and a Part 1 operation card under the two conditions.



(a) In the TG condition, the thumbnail is graphical and the tooltip contains a similar graphic, albeit populated with data. (b) In the \neg TG condition, the thumbnail is a simple question mark and the tooltip contains a shell-like representation.

Figure 7.7: The thumbnail condition also influences the design of the tooltip that appears when the user hovers their mouseover the thumbnail for longer than 1 second. The contents of the tooltip are the same as the operation cards in Part 1.

7.3.2 Data wrangling exercises

Each of the 9 exercises in Part 1 presents a data wrangling task that could be accomplished using a sequence of the operations covered by operation cards up until that point. The task could be described as an *operation selection task*. The problem has been decomposed already into a series of subgoals, with labels and graphics (for SG). These labels are, as noted in Section 6.2.1, expressed in natural language, with as little reference to formal data wrangling concepts as possible, and with care taken not to “spoil” which operation is required to implement it. Participants are asked to identify the corresponding operation from the menu (or rather, the subsection of the menu covered up until that point). As such, the task shares characteristics with many other types of programming tasks: the menu presents a series of options or distractors, similar to a multiple choice question or Parsons puzzle.

Superficially, the task looks like block-based programming: the operations in the menu are draggable and the user drags them into a drop-zone associated with each subgoal label. A screenshot is shown in Figure 7.8. There is no code execution triggered by this action, however: under the hood, the operation selection is simply checked against a pre-specified answer. Although early stage iterations experimented with making the selections executable, it was ultimately deemed unnecessary for the goals of Part 1, since it risked stalling the participants and would take considerable time to implement.

Hints

In the autumn pilot study, participants were required to make a selection for every subgoal in a task and then submit it before receiving any feedback and being allowed to proceed. Usability

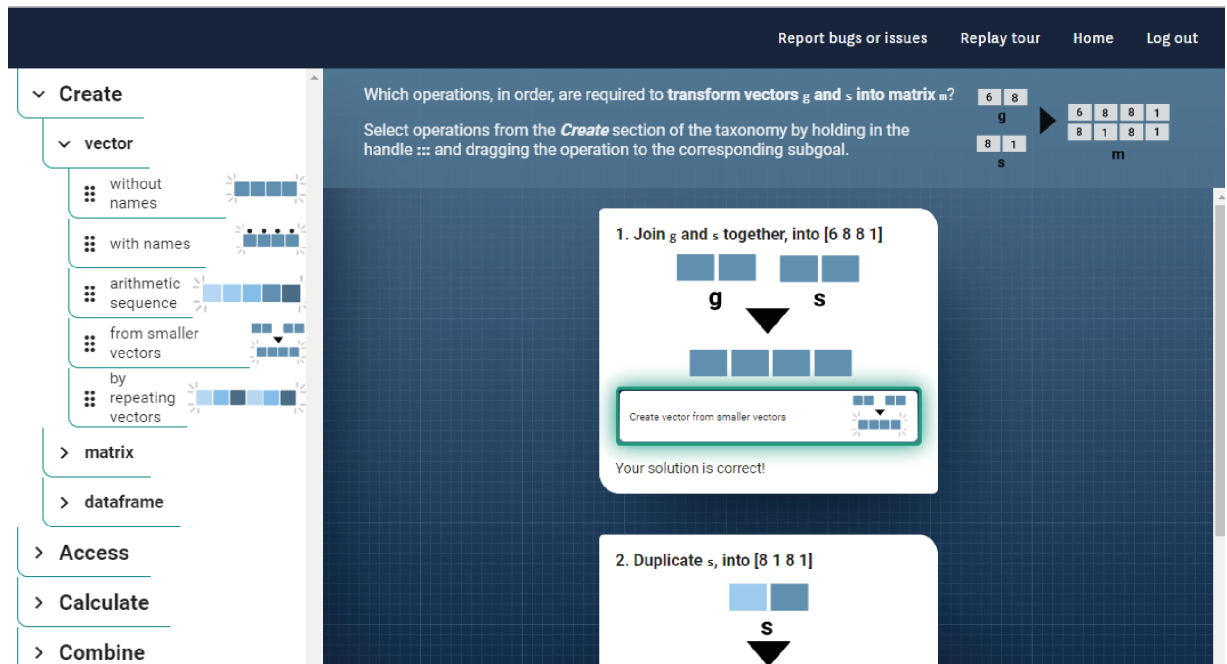


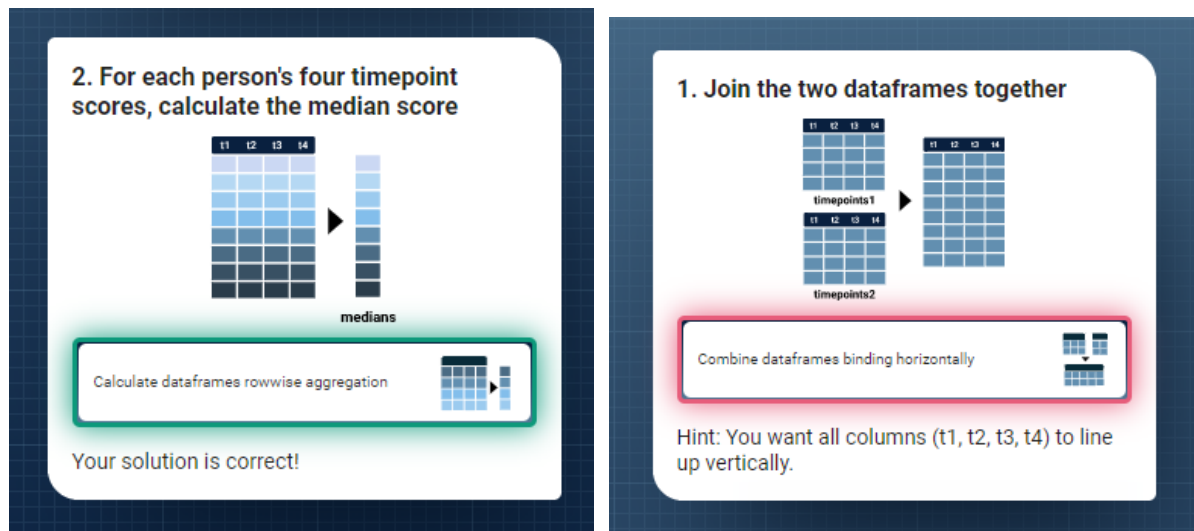
Figure 7.8: Screenshot from Part 1 in the TG/SG condition, complete with both thumbnail and subgoal graphics. The \neg SG condition would only have the labels.

study feedback suggested that, being stuck on a problem without any guidance was a significant source of irritation. As a result, modifications were made so that participants received feedback on a subgoal basis. Once an operation had been dropped in a drop-zone, the participant was immediately made aware whether it was correct (it would glow green) or incorrect (it would glow red). Moreover, if incorrect, a hard-coded hint would appear in order to constrain the operation search space. Although this risked making the task too easy, familiarising the participant without stalling them was considered a higher priority. Both scenarios are shown in Figure 7.9. Once all subgoals have been correctly solved in an exercise, the participant is allowed to proceed to the next exercise (or operation card).

7.4 Part 2 design

The objective of Part 2 was to give participants without any programming experience a foundational understanding of the programming workflow. This includes basic concepts such as variables, data types, and conditions, as well as a facility within the API lookup cycle of referencing syntax documentation, adapting code examples, and testing the solution.

The interface of Part 2 looks similar to Part 1, with a menu sidebar and an exercise description at the top. The main difference is that the sidebar now houses both the menu and the syntax documentation, in separate tabs. The menu's data operations are no longer draggable: instead, clicking on them triggers a corresponding syntax documentation entry to slide into the frame. The second important difference is that the subgoals are now gone and replaced with a DCL



(a) If correct, the drop-zone will immediately glow green.

(b) If incorrect, the drop-zone will glow red and a hint will immediately appear.

Figure 7.9: After the autumn pilot study, a hint was added to Part 1 to curb excessively frustration.

programming widget. The interface is shown in Figure 7.10. Other details include a progress bar and a button for re-loading the exercise. Once the solution has been correctly submitted, the user is allowed to progress to the next exercise.

7.4.1 Programming exercises

The programming exercises are deliberately simple, mostly one-liners, and expected to take less than 5 minutes each for a novice to solve. They generally steer clear of data wrangling concepts, and are focused on giving participants the practical minimum of knowledge necessary to begin solving programmatic data wrangling problems. These concepts are, in sequence:

1. **Variables:** The concept of variable assignment and evaluation is fundamental to programming. The first exercise asks of participants to assign values to variables and then make a calculation referencing those variables.
2. **Scripting and the interactive shell:** The programming widget contains both a scripting area and an interactive shell. The shell provides both a way of inspecting the current state of the data structures, and a place to experiment with API commands. The second exercise teaches the participant to utilise both.
3. **Scalar arithmetic:** The third exercise asks the learner to make a succession of simple arithmetic calculations while saving the results to intermediate variables.
4. **Data types:** The fourth exercise involves three data types: numbers, strings and Booleans (logicals), by simply asking participants to practise assigning such variables.

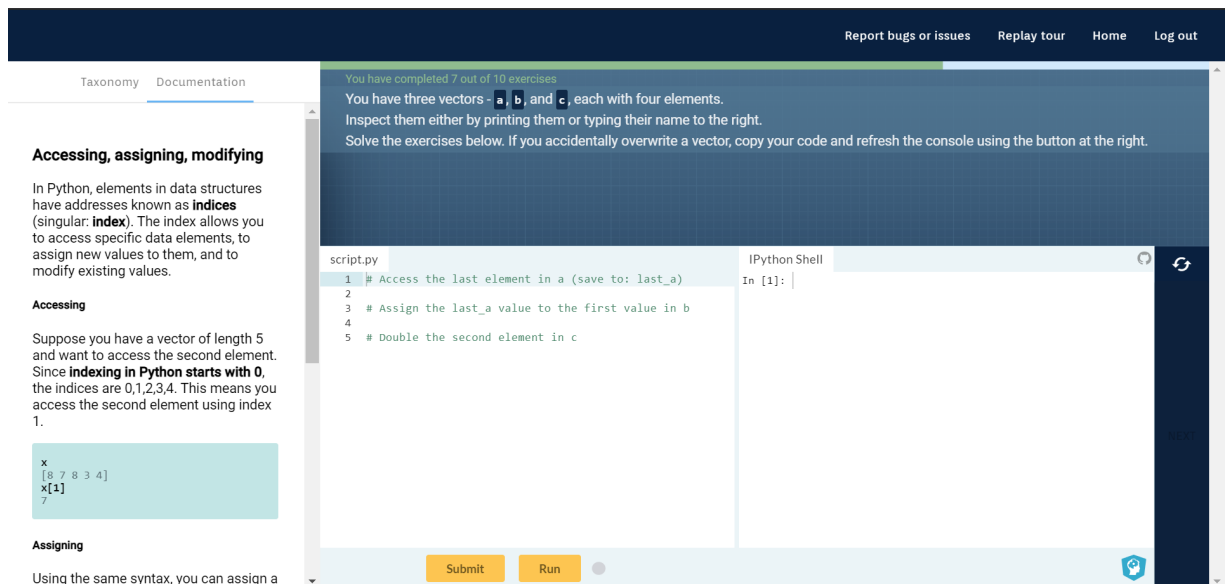


Figure 7.10: The interface of Part 2 has three main areas: a sidebar housing the menu and documentation, an exercise description at the top, and a DCL coding widget at the lower half. The menu is labelled *Taxonomy*.

5. **Logical conditions:** Participants were introduced to comparison operators and logical operators.
6. **Vectors:** Users learning Python are taught to create an `ndarray` while R-users create a `vector`. The structures are only superficially covered - the user is given no insight into their object attributes or methods, for example. The participant is also asked to perform element-wise arithmetic by looking up the entry in the menu on their own. They are told where in the menu to look: the intention is to reinforce the API lookup workflow.
7. **Functions:** The participant is introduced to the concept of a function, and then asked to look up and apply two functions from the menu to a pre-loaded vector. They are again told where in the menu to look.
8. **Accessing and modifying vectors:** Learners are introduced to indexing, and taught how indexing can be used to access values, modify them, and update (e.g. increment) them. They are tasked with practising all three.
9. **Matrices:** As with vectors, they are introduced to the 2D `ndarray/matrix` structures. They are again guided to look up two API commands from the menu and apply them to the matrix.
10. **Dataframes:** The learner is introduced to the `DataFrame/data.frame` structure, and again guided to look up API commands for the solution.

Upon loading an exercise, the sidebar is by default opened at a documentation entry that both explains the current concept and provides directly applicable code examples. When arriving at

the exercise on accessing and modifying vector elements, the sidebar looks as shown in Figure 7.10. These documentation entries were also accessible from the menu, where they feature as their own top-level category *The basics*.

Under the hood, the programming session is pre-loaded with any data structures required by the exercise. The participant has two ways of testing their solution: a *Run* button triggers the script to execute and output to display, and a *Submit* button that additionally triggers a suite of hard-coded submission test cases. The participant is only allowed to proceed if all the test cases are correct. As is discussed in Section 7.5.2, there is no explicit time limit, but the *Next* button is quietly enabled after 15 minutes.

7.4.2 Documentation

The documentation entries used in SLICE N DICE had been purpose-built to be as minimalist and novice-friendly as possible, including only relevant information and with unnecessary, technical details mostly stripped away. Each entry - examples of which are seen in Figures 7.10 and 7.11 - begins with a short prose paragraph describing the API command, its parameters, and any other background information that may be necessary. The documentation used semantic markup, boldening key information and making code syntax visually distinct.

One or more illustrative code examples are provided in each documentation entry. These examples are structured like interactive shell sessions, with commands in bold and output in grey. The examples are *not* reproducible, since that would require setup boilerplate code to be included, which would risk confusing a novice. The examples are deliberately context-free, with minimal toy datasets, and also follow widespread conventions in using metasyntactic variables like `df` for dataframes. Parameter graphics were thus not employed.

7.4.3 Hints

To address the risk of excessive frustration and consequent attrition, a hint in the form of an error message was automatically displayed at the bottom of the IDE if the solution failed to pass the test case upon first attempt. These messages are generated by the DCL package and simply inform the user if any expected variables are missing or incorrect, as shown in Figure 7.13. If correct, a hard-coded success message will appear reinforcing a concept they will have learned by then, for example: “Fantastic! It is easy to forget to add quotation signs around strings, or to remove them when using logical values. Click on *Next* to proceed.”

7.5 Part 3 design

The main part of SLICE N DICE is made up of 18 programmatic data wrangling exercises. The targeted skill is the ability to author short scripts to solve data wrangling problems. These

The screenshot shows the Slice N Dice interface with two subgoals and a taxonomy sidebar. The first subgoal is: "1. Get hold of all students who both failed the exam and are above 170cm in height (save to: failing_tall)". It includes a graphic showing a grid of 'students' with columns 'failed' and 'height', and a resulting 'failing_tall' grid. The second subgoal is: "2. Given those students, find the maximum days of absence (save to: answer)". It includes a graphic showing a grid of 'failing_tall' with columns 'absence' and 'height', and a resulting 'answer' grid. The taxonomy sidebar has three tabs: 'Subgoals', 'Taxonomy', and 'Documentation'. The 'Taxonomy' tab is active, showing a menu with 'The basics', 'Create', 'Access', 'from matrix', and 'Calculate'. The 'Access' section is expanded, showing 'from vector' with sub-items: 'by index', 'by name', 'by condition', 'by mask', and 'indices by condition'. The 'Documentation' panel is open, showing the title 'Accessing values from dataframe by masking' and a text block explaining that in Python, passing a list or ndarray of True/False values as index in a dataframe returns rows corresponding to True. A code block shows a dataframe 'df' with columns A, B, C and rows 0, 1, 2. A 'mask' array [True False True] is applied to 'df[mask]', resulting in a new dataframe with rows 0 and 2.

Figure 7.11: In Part 3, the sidebar has three tabs that the user can move between. The first panel - which is opened by default - displays the subgoal labels along with the graphics and hints. The second panel (labelled *Taxonomy*) displays the menu, which in turn hyperlinks to the third panel, which will contain the documentation entry.

problems require multiple data operations to be identified and sequenced correctly, and mapped onto API syntax. The exercises were ordered so as to increase in complexity, with the first 4 exercises limited to vectors, the next 6 also involving matrices, and the last 8 potentially involving all three data structures.

The interface of Part 3 - shown in Figure 7.11 - looks very similar to Part 2, with the important difference that the sidebar now contains an additional tab, labelled *Subgoals*, in which the subgoal labels (and graphics) are displayed. The menu, documentation, and coding area otherwise remain the same.

7.5.1 Subgoals

The subgoals are meant to help structure the plan composition, by verbally and/or visually cuing the type of data operation required. Each subgoal label recommends a variable name for storing the resultant value or data structure (e.g. “Save the result to `failing_tall`”) while the last subgoal always says “Save the result to `answer`”. This affords a way to verify intermediate results: the submission test cases check whether these intermediate variables exist and whether their values are correct. The coding area moreover contains comments to help structure the solution into corresponding steps, see Figure 7.12. The subgoals are displayed all at once, not revealed one by one, in order for the learner to form a holistic understanding of the program plan. However, subgoals that had been correctly solved (as detected by the submission test cases) will

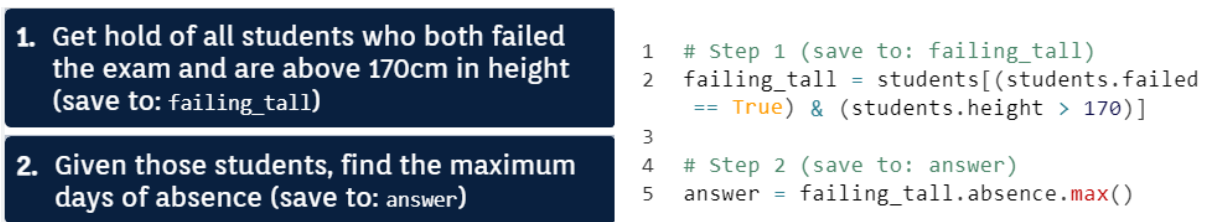


Figure 7.12: Each subgoal corresponds to one, but occasionally more, data wrangling operations. Comments in the coding panel guides the user toward decomposing their solution

turn green in order to indicate partial progress.

The subgoals tab is opened by default when an exercise is loaded, and the sidebar will automatically return to it after 1 min. This was meant to nudge the participants into engaging with the subgoals as much as possible. In the spring 2020 pilot study, the subgoals were in a separate panel that had to be actively triggered in order to be displayed. As a result, the feature was rarely used. Similarly, in the usability study (Chapter 8), the entire subgoal labels were written inside the coding area as comments. This led them to neglect the subgoal pane and consequently also the subgoal graphics, hence the more minimal inline comments.

There is a drawback to making the subgoals as visible as possible for the purpose of boosting the effect size. By being the default, there was no obvious way of measuring the extent to which participants voluntarily made use of it and consciously attended to it. We experimented with various methods that made the graphics visible but not *completely* visible, so that some action on the part of the user would be required to view the graphics clearly. In one version, the graphics were blurred and resolved only when hovered over; in another, the graphics were too small but magnified when moused over. Ultimately, these were deemed too obtrusive: the final version displays graphics somewhat too small to clearly resolve text, and clicking on a graphic causes it to enlarge within a modal window.

7.5.2 Time limits

One theoretical solution for enhancing student concentration is to impose an explicit time limit. This could take the form of simply informing them “You have 10 minutes to solve this exercise” or through a visible countdown clock, after which the learner either automatically progresses or is presented with the correct solution. Such a time limit could be generous or intentionally severe, so as to create a speed-accuracy trade-off. When Rafalski et al. [109] conducted an experiment featuring a data wrangling task, they imposed a generous 20 minute limit, since there is no point in having participants continue in perpetuity.

A time limit also comes with drawbacks. Because our tasks are organised to be progressively more difficult, if a student stalls on a simple exercise, letting them proceed on to a more complicated exercise after a timeout would only cause them to stall again. Moreover, if they were presented with the solution after the time limit expires, they may choose to wait passively

The user has forgotten a variable

Did you define the variable `failing_tall` without errors?

The user implemented the subgoal incorrectly

Did you correctly define the variable `failing_tall` ?
Expected something different.

The user has solved the exercise successfully.

Nice work! Note how it is possible to apply multiple conditions on rows.
Click on NEXT to proceed.

Figure 7.13: Examples of the feedback messages. The first two messages are produced by DCL submission test library while the success message is hard-coded by me. It is intended to reinforce a concept learnt from the exercise and motivate the learner to proceed.

until it appears. Time limits require participants to continually monitor the time remaining, introducing time management as a potential confound. If a participant fails to solve the exercise under a potentially ill-calibrated time limit, they may be discouraged from continuing, believing themselves to be far below average in aptitude. The experience of learning programming is already fraught with anxiety for many, and a time limit-induced test anxiety could impair this learning process even further [309, 310]. It could also discourage volunteers from signing up to the study, as only a dispositionally competitive learner may choose to undergo such stress without recompense.

Ultimately we settled on a compromise, in which the *Next* button initially is disabled and greyed out, but silently becomes functional after 15 minutes. More importantly, since the solution will be made available (see Section 7.5.4), there should be no risk of getting completely stalled.

7.5.3 Feedback

As previously described, both Part 2 and 3 include test cases used for verifying the learner's script when they click *Submit*. These test cases were authored using libraries provided by DCL. For each subgoal, the test cases test whether the result variable exists, whether it is correct, and whether the entire exercise has been solved correctly. This produces automated feedback messages, shown in Figure 7.13. The success messages were customised for each exercise to summarise what had been learnt from each exercise, in order to motivate the learner.

7.5.4 Hints

To address the assistance dilemma, the instructions ask of participants to “Please use as few hints as possible” and the provided hints need to be actively accessed one by one. By associating hints with specific subgoals, the hope was to deter participants from looking up the entire solution immediately, and instead only request a hint for the specific subgoal they struggled with. Similarly, we made use of *graded hints*, in which hints are arranged as a cascading sequence that gets progressively closer to the solution [311, 312]. This again added an intermediate layer that the user would have to click through before the solution is available. Each subgoal was associated with three hints, which adhered to the following pattern:

Hint 1 : targets difficulties in API lookup. It informs the user where in the menu they can find an operation relevant to the subgoal. It is formulated as (by way of example) “Have a look at *Create > vectors > by repeating vectors*” and also functioned as a hyperlink that, when clicked, opened up the relevant documentation entry. This feature was deemed relevant from a usability point of view, in that the actions it substituted for would have been a mechanical lookup anyway.

Hint 2 : targets difficulties in example adaptation. It goes into greater detail regarding the syntax, commenting on which function to use and which parameter values to pass, without ever providing the actual code.

Hint 3 : targets debugging difficulties, and simply contained the solution, i.e. the executable code corresponding to the subgoal. This was made available as a last resort, to prevent frustration. The code still had to be either transcribed or copy-pasted.

An example set of hints is shown in Figure 7.14.

7.5.5 Un scaffolded exercises

Another feature of Part 3 was the inclusion of 3 un scaffolded exercises. This was motivated by a desire to measure performance and planning ability once the training wheels were off, and subgoals (including their graphics and hints) could no longer be depended upon. Performance in un scaffolded exercises was regarded as a dependent variable (i.e. scaffolding was *not* regarded as an additional independent variable). The same exercises were un scaffolded for every participant, namely the last vector exercise (the 6th exercise), the last matrix exercise (the 10th) and the last dataframe exercise (the 18th).

The un scaffolded exercises are effectively post-tests interspersed among training exercises, but once again introduces a risk of frustration and attrition. A compromise was struck in which the subgoals were unavailable for a duration of 10 minutes, which was deemed sufficiently long for the exercise to conceivably be solved, and short enough for a participant at a complete loss to not drop out in frustration. After that, the subgoals and associated hints will appear.

1. Get hold of all students who both failed the exam and are above 170cm in height (save to: `failing_tall`)

None Hint 1 Hint 2 Solution

Have a look at **Access > from dataframe > rows by condition**.

None Hint 1 Hint 2 Solution

You have two conditions that you need to chain together using the AND-operator **&**. Don't forget the brackets!

None Hint 1 Hint 2 Solution

```
failing_tall =
students[(students.failed == True) &
(students.height > 170)]
```

2. Given those students, find the maximum days of absence (save to: `answer`)

None Hint 1 Hint 2 Solution

Have a look at **Access > from dataframe > column(s) by name** and **Calculate > vectors > aggregation**.

None Hint 1 Hint 2 Solution

You want to access **absence** from **failing_tall** and then run the **max()** function, which you can simply chain on to the **absence** column.

None Hint 1 Hint 2 Solution

```
answer = failing_tall.absence.max()
```

Figure 7.14: In Part 3, every subgoal is associated with three hints, which are accessed at the user's discretion. The first hint provides the relevant location in the menu, the second hint provides further guidance on how to adapt the documentation example, and the final hint provides the executable code. The layout here does not reflect the actual interface: hints are made available in sequence and take up the same area.

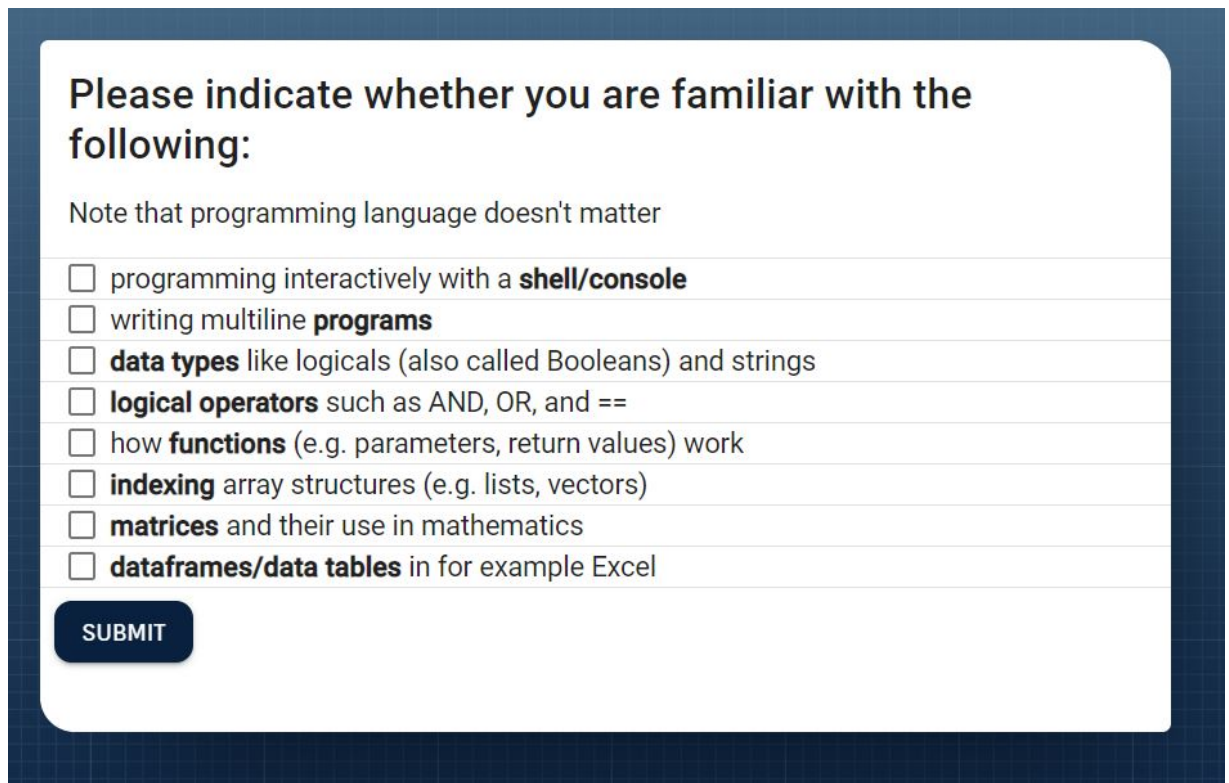
7.6 Further data collection

SLICE N DICE was equipped with rich logging capabilities of behavioural data, including the content and timestamp of code executions, clicks and mouse-overs in the menu and documentation, key presses in the coding area, and interactions with graphics. As mentioned in Chapter 1, we also sought subjective feedback, in order to determine if graphics made the data wrangling experience more enjoyable or motivating. Such data would need to be elicited explicitly via a survey, rather than through logging. Furthermore, we wanted to collect data about learner characteristics - such as degree programme, gender and prior experience - both to determine whether randomisation had balanced the conditions appropriately, and to explore them in a co-variate capacity. In this section, we describe the surveys embedded in SLICE N DICE.

7.6.1 Demographic survey

At the very beginning, after signing up, participants are given a demographic form asking them about their gender, which recruitment group they belong to, what their degree major is, what their reason for participating is, as well as the language they wish to complete SLICE N DICE in. This was to get a sense of where participants came from and what their motivations were.

There were also 5 Likert-scale questions. The first of these aimed to observe whether the



Please indicate whether you are familiar with the following:

Note that programming language doesn't matter

- programming interactively with a **shell/console**
- writing multiline **programs**
- data types** like logicals (also called Booleans) and strings
- logical operators** such as AND, OR, and ==
- how **functions** (e.g. parameters, return values) work
- indexing** array structures (e.g. lists, vectors)
- matrices** and their use in mathematics
- dataframes/data tables** in for example Excel

SUBMIT

Figure 7.15: After the demographic survey, participants were given a short pretest to gauge their programming experience more precisely.

participant was normally drawn to visual media in their learning. It asked participants “How often do you seek or create graphics when learning new material?” (*Very often, Sometimes, Rarely, Very rarely, Never*). The second question asked them about their English comprehension, in order to check whether particular subgroups could have under-performed due to a language barrier. Participants were asked “How often do you struggle with understanding English?” (*Very often, Sometimes, Rarely, Very rarely, Never*).

The last three questions asked participants to rate their experience in Excel, R, and Python NumPy/Pandas (*Not at all, A little bit, Beginner’s level, Intermediate, Advanced*). While we were under no delusions that a single, self-reported Likert-scale offered a reliable or precise measure of previous experience, front-loading the tutorial with an objective assessment could have deterred participants from continuing.

7.6.2 Pretest

To give another, more quantitative measure, the demographic survey was followed by a 8-item checkbox pretest in which participants indicated whether they were familiar with a context (mirroring the programming concepts of Part 2). This pretest is shown in Figure 7.15. If the participant scored above 5, they were advised that Part 2 was optional to them.

7.6.3 Evaluation survey

At the end of each part, participants were moreover given a short evaluation survey, asking them to rate their experience along several dimensions, and to raise any other ideas or feedback they had in an open-ended response item. The ratings were given on a scale of 1 (*Not at all*) to 5 (*Very much*) on the following dimensions:

- **Enjoyability:** How enjoyable did they find the part? Presumably, the degree of enjoyability predicts the likelihood of adoption.
- **Effort:** How effortful did they find it? Effort has been shown to influence test performance [305], and is usually measured by post-test self-report items [305]. A desirable outcome would see graphical conditions being perceived as *less* effortful.
- **Motivation:** How motivated were they while completing the part?
- **Concentration:** How concentrated were they? Although concentration and effort are closely linked, concentration arguably reflects a more deliberate form of exertion.
- **Subgoal helpfulness:** How helpful did they find the subgoals? This was asked to all participants.
- **Hint helpfulness:** How helpful were the hints? This was also asked to all participants.
- **Thumbnail graphics helpfulness:** How helpful were the thumbnail graphics? Only the TG group was asked about this. Although it carried the risk of demand characteristics, by being embedded among so many other items the risk was reduced of participants deducing the research hypothesis.
- **Subgoal graphics helpfulness:** How helpful were the subgoal graphics? This was only asked to the SG group.

There are several validity risks attendant to these measures. With effort, for example, students could attribute failure to a lack of effort [313] and social desirability bias may lead them to overestimate their effort. Moreover, effort may vary dynamically over the course of the tutorial, which a single scalar measure cannot capture [313]. To mitigate response fixedness, some items had a 5 signify a positive response, while elsewhere a 5 indicated a negative response.

7.7 Chapter summary

- SLICE N DICE is a special-built web application that contains three parts and has two fixed variables embedded: the provision of subgoal graphics and the provision of thumbnail graphics.

- Part 1 consists of a set of operation cards and 9 data wrangling exercises. The exercises are non-programmatic and require the participant to select operations from the menu that match each subgoal.
- Part 2 provides a foundation to programming, featuring 10 programming exercises that cover topics like variable assignment and data types. There are no subgoals associated with this part, but the learner is trained in the API lookup workflow.
- Part 3 contains 18 programmatic data wrangling exercises. Each has a set of subgoals and, for each subgoal, 3 hints. 3 exercises are unscaffolded (i.e. without subgoals for the first 10 minutes).

Chapter 8

Slice N Dice pilot studies

Rolling out SLICE N DICE into multiple classrooms and asking of people to dedicate their time to complete it would require assurances that both the front-end and back-end data collection worked correctly. For pedagogical and methodological reasons, the interface would need to be intuitive, the instructions unambiguous, and the exercises calibrated so as to be neither too difficult nor too easy. Furthermore, since much of the content of SLICE N DICE - including graphics, documentation, and exercises - were manually created, there were numerous opportunities for human errors to creep in. Finally, before large-scale recruitment began in earnest, we needed *some* reality check, however crude, that the graphics were subjectively useful in this context, and could not be outright dismissed as an intervention.

For these reasons, the present chapter describes two studies conducted during the autumn semester of 2020 involving early versions of SLICE N DICE (the main study, described in Chapter 10, took place in the spring of 2021)¹. The first study's chief goal was to elicit ideas of how the user experience and tutorial contents could be improved, and to gather qualitative reflections on the role graphics played in their problem-solving process. The second study was a quantitative pilot study meant to give us a glimpse of the data patterns and distributions to expect, although it was too small in sample size to carry much inferential weight. The pilot study was also instrumental in refining the data schema and deciding how to operationalise the dependent variables.

The chapter presents the studies in the order they were conducted: first the qualitative usability study, then the quantitative study. The outcomes of these studies have already been hinted at in Chapter 7, since they informed the final design. However, we believe there is value in situating this formative feedback in its proper context, and in reporting the results in greater detail.

¹As noted in Chapter 7, we also conducted another study in Spring 2020 that could be regarded as a pilot study. However, though influenced by this earlier study, SLICE N DICE was built from scratch. Moreover, due to study's COVID-19-related discontinuation, we will not describe it any further.

8.1 Qualitative usability study

We noted in our methodological discussion (Chapter 4) that, as our line of research is design-based as opposed to theory-focused, qualitative methods serve an essential role in providing quick, low-cost feedback to different design iterations. Since many of the variables we care about - motivation, enjoyment, confusion, and learning - are not directly observable (and at least the first three are fundamentally subjective), it makes sense to conduct such a design exploration using interviews. As a research method, user interviews are not without shortcomings, as responses are inevitably biased towards the easy-to-verbalise and socially desirable. However, when coupled with observational data, it can readily flag features of the app that are confusing, off-putting, malfunctioning, or unnoticed.

The objectives of the study range from the low-level to the high-level. At the lowest level, we want to detect bugs, typos, and errors in the tutorial contents. Although crucial to the app development and subsequently addressed, we will not remark upon such minutiae at length. At the middle level, we sought to remove or modify items that were too confusing or difficult. We will similarly not recount the evolution to individual exercises. Finally, at the high level, we sought feedback on how the overall task, interface design, and graphical features were perceived. The write-up of the usability study will be focused on such high-level concerns, and centred on the following research questions:

Are the explanations understandable? In Part 1, a card contains a prose explanation and example, the latter of which is represented either graphically or using a shell-like notation. Do people appreciate the card design, and do they prefer the graphical or textual version?

Is the task sufficiently clear? The operation selection task of Part 1 is a relatively unproven type of exercise, while the coding environment in Part 2 and 3 necessitate workflows quite unlike that of a standard IDE. To work effectively as an exercise and assessment, the role of the scripting and shell areas, the hint mechanism, and the documentation all have to be clear. Are the instructions and the interface clear enough for the participants to progress without issues?

Is the menu sufficiently navigable? Can participants navigate the menu efficiently to find operations in Part 1 and syntax in Part 2 and 3? To what extent do they rely on thumbnail graphics when doing so?

How are the subgoals and subgoal graphics perceived and utilised? Are the subgoal graphics perceived as helpful, indispensable, redundant or annoying?

Do participants appreciate the task? Part 1 was designed with many considerations in mind, among them its benefits to theory development and its role as an advance organiser. However,

Usability study participants

ID	Subject	Gender	Experience
P1	Engineering	Female	Basic experience in C, Matlab
P2	Engineering	Female	Basic experience in Matlab
P3	Engineering	Male	Minimal experience in R
P4	Physics	Female	Basic experience in Matlab
P5	Chemistry	Female	None
P6	Physics	Male	Basic experience in Matlab
P7	Biology	Female	None
P8	Engineering	Female	Intermediate experience in C, Matlab

Table 8.1: Table describing the 8 participants involved in the usability study. Every participant was recorded using SLICE N DICE for 3-4h. Minimal < Basic < Intermediate.

it is possible that participants view the part as blocking more productive activities by delaying their introduction to programming. Is this an issue we need to be worried about? Meanwhile, the programming exercises of Part 2/3 are hardly authentic: they involve toy data sets and contrived problem contexts. They also offer ample scaffolding features, some of them optional. Are participants bothered by the lack of realism, and is the availability of scaffolding appropriate or excessive?

8.1.1 Method

Participants

This study took place before the autumn semester had started, hence there were no undergraduate students available to recruit. Instead, we aimed our recruitment effort at non-CS PhD-students in STEM subjects, as these too are likely motivated to learn programming. The study was advertised through an email-list for STEM PhD-students. The only exclusion criterion we applied was that they were not experienced in Python or R, as it proved difficult to find STEM PhD-students without any programming background whatsoever. In the end, 10 participants were recruited. Because 2 participants withdrew mid-study for unknown reasons, we present the remaining 8 participants in Table 8.1. Out of these, 6 had some basic programming knowledge. 3 participants (P3, P5, and P7) chose to do it in R, while the rest chose Python.

Procedure

Participants were booked in for 4x1h sessions, separated by a week. They were compensated with a £10 Amazon voucher per hour. The sessions took place remotely using screen-share via Zoom. With participants' informed consent, sessions were recorded and transcripts auto-generated.

The first session began with me explaining the purpose of the app. The goal with the study was to “raise any confusions, bugs, errors, or ideas for how to improve it”. Participants were encouraged to voice out loud any feedback or ideas they had as they proceeded, using a think-aloud protocol. Before starting, they were asked about their previous experience in programming, what challenges they had perceived having in the past, and what subject they studied. Towards the end of Part 1, participants were asked whether they found the task helpful, which representation on the operation cards they preferred, whether they made use of the graphics, how navigable they found the menu to be, and whether they in real life would be tempted to skip past Part 1. At the end of Part 3 they were asked the same questions (except the last). The interview technique was semi-structured: although the same questions were repeated for participants, follow-up questions were also posed.

Since this was the first time SLICE N DICE was tested by people external to the project, we had little insight into how long it would take to complete the app. To fit the sessions within the target duration of 4h, the following protocol was established. Unless participants had intermediate previous experience (in which case they could skip it), the first session would be spent on Part 1. The second session would be spent on foundational programming (i.e. Part 2) and the third and fourth on programmatic data wrangling (Part 3).

To ensure that feedback was provided on as many exercises as possible, participants would receive verbal hints and guidance by me in debugging their solution. Since they received additional support, the circumstances are *not* fully representative of those in which SLICE N DICE eventually would be completed in. This downside was accepted due to time limitations and the need to vet exercises; the quantitative study featuring the vetted exercises would later contribute with more realistic time estimates.

As with the main study, participants were randomly allocated to a condition with subgoal graphics and thumbnail graphics either present or absent, independently of each other. However, two toggles were also added to the interface so that, mid-way through each part, participants got to experience the opposite condition. Each participant therefore experienced both the presence and absence of either type of graphic.

Analytical procedure

Low- and mid-level suggestions (e.g. typos, unclear wordings) were noted down during the session and generally addressed through modifications immediately afterwards. To collate utterances and behaviours relevant to the high-level research questions, each screen-share recording was re-watched by me, and pertinent material transcribed and included for analysis. The methodology can be characterised as a form of grounded theory, in search for emergent themes instead of hypothesis-testing.

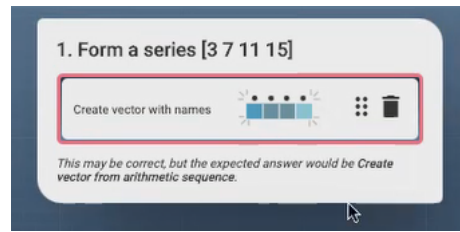


Figure 8.1: During the usability study, the correct answer was immediately given once an operation had been given to each subgoal and the solution submitted.

8.1.2 Part 1 results

The version of SLICE N DICE that the participants received was different from the final one presented in Chapter 7. It was also ever-changing, with changes suggested by one participant sometimes immediately implemented before the arrival of the next participant. In Part 1, the main differences were in the ontology underpinning the menu, which at the beginning had not yet received the regular three-tier structure (described in Chapter 6) that was introduced towards the end as it became clear the ontology was too nested. Another difference was tooltips, which initially did not exist at all, and also introduced towards the end. Initially, there was no feedback at all upon submission and participants were taken immediately to the next exercise, but eventually the correct answer was displayed upon submission, as shown in Figure 8.1. The exercise set was also slightly different, and larger: the 12 exercises of the usability study was eventually reduced to 9. Originally some subgoals mapped onto several operations, but this number was eventually constrained to 1 per subgoal.

Are the operation cards understandable?

The operation cards were generally read without obvious impediments. Occasionally, participants engaged in extensive backtracking to compare operations with each other. Three participants explicitly commented on the cards, with P7 noting that the examples are “easy to understand”, P6 saying that “these kind of short, sharp descriptions are quite good”, and P5 noting (emphasis added):

“There’s a lot of information to unpack, but it’s all obviously **written very clearly.**”

Among the operations, pivoting and group-wise aggregations appeared to be the operations to cause the slowest reading. P5, a beginner (and native English speaker), occasionally expressed confusion over terminology such as *element*, *value*, and *aggregation*, and suggested a lookup table to gather such words in one place. P2 similarly commented that many of the words, such as *matrix*, may be difficult to a beginner. On other occasions, the example was ambiguous, causing people to later under-generalise: for example, P3 wondered whether an *aggregation* referred specifically to addition, since the example featured addition. P1, meanwhile, suggested adding

more signposting within the deck, such as a card indicating when the user enters a new section, for example from vectors to matrices. Another issue arose from the attempt to disentangle and delay the introduction of programming concepts. Some of the operation cards required the mention of data types and indices, which provoked questions among the complete beginners.

Regarding the operation card graphics, three people spontaneously expressed liking them. P1, upon first seeing them, said they “like the little illustration, that’s good”, while P7, upon toggling to see it, stated “that’s much easier to visualise now”. P6 called them “definitely useful”. Moreover, when asked explicitly whether they preferred the graphical version or the textual one, most participants preferred the graphics:

“I think I preferred the graphics. The graphics made it seem like a bit less information. Yeah, definitely graphics.” (P5)

“I’d say the graphics because it was easier for me to visualise.” (P7)

Two stated that they wished to have access to both representations:

“I think I probably preferred the graphics, but it would have been **good to be able to toggle.**” (P6)

“I think I liked both because having the text sometimes you just don’t wanna read it but having the graphics you kind of want something to help you explain it so, a bit of both to be honest” (P3)

In addition to these general reflections, there was also highly specific feedback. For example, in an *Access* graphic where the post-state (i.e. the accessed elements) lacked the highlights of the pre-state, P5 was confused by which elements ultimately were accessed. In another instance, a graphic featured elements that had the same value as their index (e.g. the first element had value 1), causing confusion over whether it was the indices or values that were being accessed.

Discussion. The graphical versions of operation card examples seem to be subjectively preferred over the shell-like notation, and the bite-sized organisation of the card deck and succinct descriptions appear to be appreciated. As a result, the design of this part went mostly unchanged, but the texts and examples were examined anew to minimise jargon and ambiguity.

Is the task sufficiently clear?

Since it was unlikely that participants had encountered anything resembling Part 1’s operation selection task before, we were very interested in whether the instructions and interface were clear enough for them to proceed unassisted. Despite the interactive tour guiding participants through each necessary step and interface component, and one participant (P1) calling the instructions

“really clear”, it soon became evident that this was not sufficient. Once the tour had concluded, most participants were confused about how to proceed. P5 did not initially understand she was meant to drag the operations. P2 was similarly at a loss how to proceed at first, but with the two types of graphics activated, found the task straightforward, eventually saying “I think the interface made perfect sense”. Expanding on this point, she said:

“I think **initially it was difficult for me to understand what I was meant to do**, but that process got a lot more easier the more I did it.” (P2)

One participant, P3, struggled considerably with the interface and what he was meant to do. In part this may have been caused by how the drop-zone looked like a text input field (“Are you supposed to drag or type, which is it?”), by how the drag handle did not appear obviously draggable, and by a confusion of terminology. P3 was confused by the use of “operation” and thought it referred to the data structures. He also appeared to be overwhelmed by the menu, which he had almost fully expanded, and as a result began selecting operations haphazardly.

“If you hadn’t told me, **there was no way for me to know that it was drag and drop.**” (P3)

When asked about it, P3 did not remember the tour, and suggested that a text should be added to each exercise reminding them of the drag-and-drop. It is worth noting that he initially lacked subgoal graphics, which could have contributed to the confusion. At least some of this confusion appeared to be more diffuse:

“A lot of the technicalities, like... you said it’s meant for novices but to me, some of the analogies seem to be by or for someone who is familiar with R who would understand that, but **as a novice I didn’t understand what it actually meant.**” (P3)

Another recurring issue was that several participants (P1, P5) did not understand that they were only meant to choose operations among the categories previously covered. There were also several concrete suggestions. P1 commented that it was not clear when an exercise was correct, because the border went only subtly green. P5 suggested that it would be good to be able to revisit past exercises.

Discussion. The feedback prompted us to make the instructions explicit in the description of every single exercise, including what to do (“Select operations from the menu by dragging the :: handle”) and which categories to choose from. Furthermore, the tour was made replayable. Feedback was provided upon submission, and correct answers were given a more noticeable green glow, while red answers would glow red. The number of exercises was also reduced, from 12 to 9, removing the ones found to be too ambiguous.

Is the menu sufficiently navigable?

As noted, the ontology was restructured and pruned over the course of the usability study, because it soon became evident that the inconsistent and deep nestings of the early ontology iteration complicated navigation. Often it prevented participants from even seeing all operations available since they were reluctant to explore the deepest levels. Despite the need for revision, participants reported having a systematic workflow that consisted of determining the top-level category first, and then using the thumbnails and/or tooltips to select the specific operation. Determining top-level category mostly appears to have been straightforward:

“I would use it where I did not necessarily know under which subheading, but **I would know I would find it** in the *Calculate* section, and then I would look for the right subsection.” (P2)

“Whether it was *Access* or *Calculate*, **it was very easy to see where you’d have to be in it.**” (P5)

“I would go for the specific group first, and from there, vaguely scan over the words, but really then **look for then picture that was kind of in my mind.**” (P6)

Sometimes this strategy failed, due in part to incorrect assumptions about the menu’s structure. For example, in a calculation-related subgoal, P5 reasoned (correctly) that the input values initially would have to be accessed, and therefore looked under *Access*. Elsewhere, the operation imagined as necessary to fulfil a subgoal was too specific and the participant failed to generalise the search. For example, when one subgoal was stated as “Add the two vectors in reverse order”, two participants (P1, P6) went looking for an “add in reverse order” operation, even though the “Create a vector from smaller vectors” would work. The subgoal thus constrained their search incorrectly, but looking through the menu they eventually found the correct operation unaided.

Interestingly, later in Part 3, P1 commented that the dragging of relatively abstract operations also could have the effect of attaching a too narrow meaning to the operation. P1, when reading the subgoal “Calculate product of the array” said she was unsure what to do. After guided towards the aggregation, she said:

“I think, because previously aggregation was tied to things like addition and mean, that I didn’t really understood it could be used for product. /.../ Because previously, instead of typing out the function for mean and instead simply taking an aggregation block, **it hadn’t really dawned on me that... there would be different functions within the aggregation part** of the documentation.” (P1)

Recall that the menu’s appearance changed throughout, with tooltips initially being absent, then simply being the operation’s title, and eventually featuring the full operation card contents.

Regarding thumbnail graphics, there were some unprompted comments. For example, P7 called them “very helpful” and used them systematically to locate operations. P1 occasionally made strategic use of the thumbnails and surveyed the tooltip before selecting an operation, saying:

“I also **like having a copy of these [tooltip] illustrations** because it is sort of like... you have the title... and then I [hover] to see what is in there. It is really good.” (P1)

Among those who completed the study before tooltips were introduced, P2 often asked questions that a tooltip is likely to have answered, for example what an aggregation or pairwise calculation meant. P6 found the thumbnails to be too light against the white background and therefore difficult to read. Another aspect worth considering is that, in being overly nested, the menu may also have made the thumbnails less effective, since they were only visible if the super-ordinate node was expanded.

Discussion. By the time of the last participant, the menu had settled into a structure very similar to its final form. It is difficult to evaluate whether this made a qualitative difference to their navigational efficiency, and the design of the study afforded no way for the participant to compare them directly, but we are satisfied that its new regular form was an improvement. The perception of the thumbnails also seems broadly positive. It was impossible to observe directly when and whether participants glanced at thumbnails (since this glancing may not translate to a cursor movement), but it was clear that the tooltips were used strategically to clarify what an operation meant. We are therefore also satisfied that the tooltips were a substantial improvement. One pedagogical-methodological trade-off we remain unsure of is whether to keep the menu collapsible or not. On the one hand, a collapsible menu allows for a neater, less overwhelming appearance. On the other hand, a collapsed menu provides fewer opportunities for the thumbnails to be visible and thus effective. In the end, we kept it collapsible to prevent information overload in Part 1.

How are the subgoal graphics perceived?

Unprompted comments regarding the subgoal graphics were generally positive, with P5 saying that she “really enjoyed” them and P2 saying that “the images really help - they kind of make the words make more sense”. Several participants described the graphics as “important” to their process (P1, P7). P5 was concerned over the risk of over-scaffolding, saying that she “probably looked at the graphics a bit more than I would have liked to”. She appeared conflicted over whether the graphics were simply helpful or *too* helpful:

“It just feels... not like cheating because I’m really glad it’s there, but **obviously you could just look at the pictures**. Actually that helps because if you get stuck you can just look at the pictures and that **makes you realise why that’s the right answer**. So yeah, I like it.” (P5)

“It did make it easier, but it was good to start without them and have them either be optional, because **I think otherwise I would have relied on them completely**, and maybe that would have meant I would have relied on them without understanding as much.” (P5)

In the quotes above, P5 appears to have used the graphics productively as a cue for self-explanation, and also as a way to verify her tentative solution. P2 also reflected on the meta-cognitive self-regulation needed to use the graphics productively, and not depend on them:

“I think in the beginning I was relying on [the graphics] quite heavily, but the further I got through, **I was trying to find the thing I was looking for based on my understanding and then use the image to check whether I was right**. I would try to find the subgoal I was looking for, and then see if I was correct.” (P2)

To P7, the availability of graphics implied the option to engage or disengage with them:

“I thought **the images were helpful** because you can make it as easy or hard as you want.” (P7)

Discussion. The participants appear to be conscious of the assistance dilemma, and the potential for unproductive use of the graphics. It is true that, especially in the TG/SG condition, the subgoal graphics may be only simply compared with the thumbnail graphics, to locate the correct operation, and that reflecting on its relationship with the subgoal label and problem domain requires a degree of self-control. Such an effect could be revealed by an interaction analysis. Nevertheless, the subgoal graphics appear to be perceived as valuable, important, and enjoyable.

Do participants appreciate the task?

At the very end, participants were asked explicitly about the fact that Part 1 did not include any programming, whether it was frustrating to them, and whether they would have felt tempted to skip it. The opinion here appeared to be fairly split among those who saw it as relevant and those who did not. P6, who had some previous Matlab experience, reported finding the task “maybe a little frustrating”. In real life, he believed he would probably skip past it, saying that he would “rather just see it run and then find out why”. He furthermore explained:

“**It didn’t feel like I was programming, if you know what I mean?** It felt like I was hunting for things, I guess, but I am still thinking about the operations more pictorially rather than thinking about seeing it run.” (P6)

P2, another participant with previous Matlab background, similarly expressed an urge to program immediately, even though she thought that “it is quite helpful to learn what you are gonna be doing before you do it” and that she probably would have completed the Part 1 in real life anyway, since there were exercises in it. She explained:

“I think it helps to go and do it right away, because I think it helps to see output and then understand what is happening. I think probably reading before is more valuable but, typically for me, I would try and do it to understand what a specific instruction actually does.”(P2)

P5, who was a complete beginner, believed the exercise was useful, but that it initially was difficult to understand its relevance, since it did not fit the mental image of what programming entails:

“I think I would have been overwhelmed with straight away trying to code something. The only thing that would be useful is having something that said how this relates to programming. Especially for a complete beginner, you think of programming as someone typing code. It can be hard to see the connection. But it definitely makes it an easier introduction and less scary.”(P5)

Despite now believing in its relevance, she thinks she probably would have skipped past it:

“Yeah, I think [I was] quite tempted [to skip it]. I have done online courses before where in the beginning you have these text boxes, and you have to read and press *Next*. It’s easy then, **unless you’re really motivated, to just skip through them and get to the first question.**”(P5)

Several participants echoed P5’s impression that the non-programmatic introduction of Part 1 could help mitigate feelings of information overload, by building up a more robust foundation:

“When learning a new language, **you are sometimes bogged down with syntax,** and so to have an idea of the rules of the language without having to worry about those errors, especially when it comes to learning as you go... you may not know if the error is due to the syntax or if the operation you are trying to do isn’t carried out that way. **Having it begin this way (without programming) sort of removed that ambiguity.** It made it very clear which rules of the language I understood and which I didn’t.”(P1)

“I definitely learnt a lot of things that I didn’t know. /../ Even when **I thought I knew how I would go about answering things, I was proven wrong,** that I didn’t know what to do.” (P1)

“I think this was very helpful because I guess **you need to understand how some basic functions work before you do actual programming.** And this helps visualise the next steps.” (P7)

Interestingly, P3, the participant who previously had struggled with understanding the interface and started off frustrated by it, appeared to have warmed to the task. After completing it he said:

“Obviously you need to know the basics of it before you go into programming. In my opinion, it’s better to learn to cook [before attempting to] start cooking. **Some people would say ‘You learn as you go’ but if you know the basics of it, then it would give you a better understanding of what you are actually doing**, as opposed to someone saying ‘Just type this code.’. That way when you go into programming you have some slight clue of what you are actually doing. So when you create a dataframe you go ‘Oh, I just created a dataframe’ as opposed to if you just jumped straight to the code, and don’t know what a dataframe really is. So in my opinion, this approach is good.” (P3)

Discussion. The range of attitudes on the utility of a non-programmatic, concept-first part was expected, but it was interesting that the opinion did not neatly align with expertise: both complete beginners and prior programmers preferred code-first or concept-first structures. The instructional sequence does not have to be that strict, but is here motivated by the desire to separate conceptual skills from programmatic skills. The disagreement does suggest that, at the cost of some added variance in the data, SLICE N DICE should make Part 1 optional and technically allow participants to start with Part 2.

8.1.3 Part 2 results

As mentioned in Chapter 7, what we in the main version refer to as Part 2 (programming) and Part 3 (programmatic data wrangling) were in the usability study merged into a single Part 2. Part 2 was at this point nested into sections: clusters of exercises that were preceded by an explanatory card that introduced a programming concept. The first few sections concerned foundational programming concepts (e.g. variables, functions, data types) and we will use “Part 2” to refer to these, and “Part 3” to refer to the later sections, which involved data wrangling exercises.

For an example of these explanatory cards, see Figure 8.2, which shows the card for scripting. This card contained the same information as the corresponding entry in the sidebar menu, but it was left to the user to look it up. Unlike the main study, the relevant entry in the documentation was not directly opened by default.

At this point, the scaffolding in Part 2 was similar to the final version in how it contained a hint, however the hint was not embedded in the error message, but rather available through a *Hint* button. The hint simply informed participants of where in the menu they would find relevant information. Additionally, and in contrast with the final function, there was also a

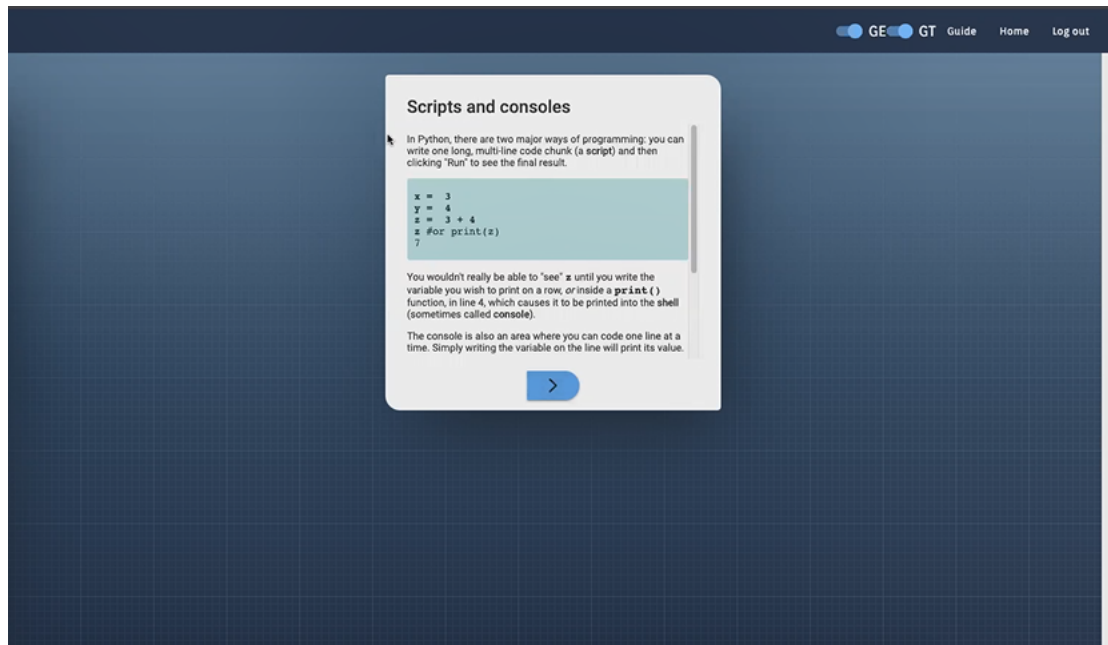


Figure 8.2: In the usability study, Part 2 concepts were introduced through explanatory cards rather than as a documentation entry in the sidebar.

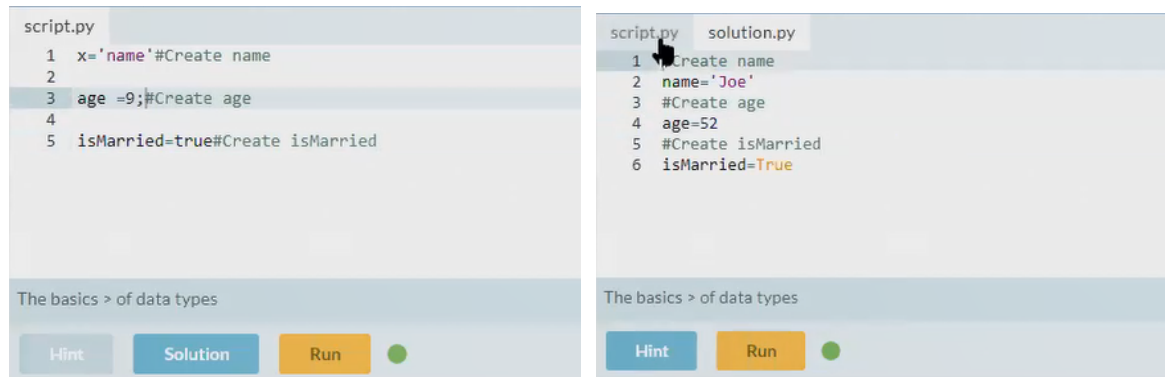
Solution button available that displayed a sample solution. Both of these features were part of the DCL plugin and are shown in Figure 8.3. For the purpose of the usability study, progress was not conditional - they could simply click *Next* if they felt satisfied they understood it.

Is the task sufficiently clear?

The explanatory cards were read without issues, except in instances where they referred to components in the programming interface (e.g. references to the scripting area or the console), P1 wished for them to be visible. The fact that explanatory cards removed them from the IDE context (see Figure 8.2) caused some irritation. Several participants did not initially realise they were meant to look at the menu at all, and were therefore completely stumped.

When in the coding interface, the interactive tour and the explicit instructions of the first few exercises appeared to have made the task straightforward enough. To some, like P1, it was not immediately not obvious that, via the menu, they still had access to the explanatory card preceding the exercise.

Programmers with experience, like P4, proceeded efficiently through the initial programming exercises. For complete beginners, the early exercises also involved near enough transfer to not stall them excessively. Even so, several recurring errors had to be addressed through explication in the documentation and less ambiguous exercises. For example, when asked to create a variable with a particular value, beginners sometimes confused the variable identifier with the value. Not knowing what to assume as fixed and what to assume as flexible in the documentation, beginners sometimes drew incorrect generalisations about syntactic rules. For



(a) The hint showed as a strip at the bottom of the script. (b) The solution was made available after clicking on the hint.

Figure 8.3: During Part 2 in the usability study, a hint and solution were available. These were removed afterwards. Both the hint strip and solution tab were built-in features of DCL.

example, two participants placed blank spaces within variable names, between function name and the subsequent brackets, and used keywords for variable names.

Participants were repeatedly confused at not seeing any output even though they had no print statements in their script (only assignments). It was also clear that beginners had to be explicitly guided through the workflow of saving intermediate results in variables, printing them to the console, inspecting its state and moving onto the next step.

The sequencing of concepts was at times awkward. For example, `np.array()` was introduced before explaining the concept of libraries and without explaining object-oriented concepts. Since the documentation was deliberately superficial and refrained from diving into low-level logic of either API, participants (P1, P5, P4, P7) were sometimes frustrated by things left unexplained. For example, the method for accessing the number of rows in a matrix `m` in R was simply given as `dim(m)[1]`, without explaining that `dim()` returns an integer vector in which the number of rows is the first element, which frustrated P7. Other questions concerned how method chaining really worked in Python and the rules for how functions could be nested.

Discussion. Many of the issues relating to participant confusion prompted small, incremental improvements to the documentation and how questions were formulated. In response to feedback, the *Next* button as moved to a more visible place, and the pre- and post-state of the sample data in each exercise was visualised in the exercise description, to help remove ambiguity. Later, after the pilot study but before the main study, the issue of participants not exploring the menu was addressed by making the relevant entry open up by default in the sidebar, replacing the explanatory card.

Regarding participants' frustration with shallow documentation, it is difficult to imagine ways of satisfying this curiosity without incorporating external resources such as the official documentation, or allowing participants to use Google as part of the workflow. Ultimately, the

simplicity of the documentation was a deliberate choice meant to prevent cognitive overload among complete beginners, but for high-curiosity learners, it is possible that could backfire.

Is the menu sufficiently navigable?

During Part 2, there was little menu exploration, which is not that surprising given that the exercises rarely required information not available in the explanatory card. Perhaps partly due to the time confines, it was common for participants to go straight to the hints to see where in the menu to find relevant material. P4, even with previous programming experience, often used hints “to check”. Therefore, most time appeared spent on code example adaptation, and not operation selection. These concerns notwithstanding, P7 found the navigation experience “smooth”. At one point, when thumbnails had been deactivated, P5 said that she preferred how it was before, “when one could see what [the operation] is doing”, presumably referring to the thumbnail graphics.

Discussion. The lack of exploration in the menu during Part 2 is not a big issue, since focus should be on learning to code and to adapt code examples, not on composing multi-operation scripts, which would require further navigation.

Do participants appreciate the task?

Perhaps due to its higher fidelity to real-world programming, participants were generally motivated throughout Part 2. P6, who had previously expressed reservations about Part 1, said that he believed “this would be the ideal way to learn”:

“I much preferred to this. You can kind of see written down examples and you can try to apply that to your problem and, if you get stuck, you have the hints and the solution. **Otherwise you can spend a lot of time messing around, trying to visualise it.** Last time I was doing a lot of visualising but this, I think, was a lot better.” (P6)

He especially appreciated being able to see the correct solution:

“Not only are you trying to extrapolate, you actually have the solution. Often there are many ways of doing it and this way you can see if you have over-complicated it.” (P6)

P4 also seemed to benefit from seeing a solution, and repeatedly toggling back and forth between the script and solution tab as soon as she seemed confused.

The screenshot shows the SLICE N DICE pilot study interface. On the left, there are three subgoals:

1. Categorise each country's average number of kids and average income (save to means)
2. Get hold of the countries that have, on average, more than two kids (save to means_above2)
3. Obtain those countries' income means (save to answer)

The main area contains a text prompt: "Suppose you have a dataframe (called `women`) containing the number of kids that each woman has, along with their income and the countries they lived in. In the countries with more than 2 kids per woman on average, what is the mean income?"

Below the prompt is a table with columns `ID`, `country`, `nr_kids`, and `income`:

ID	country	nr_kids	income
1	'C1'	3	200
2	'C1'	2	300
3	'C2'	0	100
4	'C2'	4	200
5	'C3'	5	50
6	'C3'	5	100

Below the table is a small table with columns `country`, `mean_kids`, and `mean_income`:

country	mean_kids	mean_income
C1	2.5	250
C2	2	75
C3	5	75

The R Console shows the following code and output:

```
script.R
1 means <- aggregate(list(income=women$income,
2 nr_kids=women$nr_kids), by=list(country
3 =women$country), mean)
4
5 #Categorise each country's average number of
6 kids and average income (save to: means)
7
8 #Get hold of the countries that have, on average
9 , more than two kids (save to: means_above2)
10
11 #Obtain those countries' income means (save to:
12 answer)
```

The R Console output shows several warning messages: "Warning message: argument is not numeric or logical: returning NA" and a parsing error: "Parsing error in script.R:1:69: unexpected ','".

A custom error message is displayed at the bottom: "The contents of the variable `means` aren't correct. Remember that you have want to group rows by `country` and average both `nr_kids` and `income`. Have a look at Calculate > dataframes > groupwise aggregation."

Figure 8.4: Note the customised error message and hint, but lack of hints underneath each subgoal

Discussion. Overall, despite its brevity, Part 2 does appear sufficient to instill in participants an iterative workflow of looking up documentation, adapting code examples, and debugging their solution, which forms its chief purpose.

8.1.4 Part 3 results

Part 3 was at this point similar to the final version in that the sidebar contained a tab where subgoal labels and graphics were displayed. Unlike the final version, however, there were no hints associated with the subgoals: instead, a hard-coded hint displayed as a strip in the IDE once the participant submitted an incorrect solution. The hint anticipated a common error and suggested where in the menu to look for information (see Figure 8.4). As shown in the same Figure, the subgoals were also available as in-line comments within the IDE. Finally, unscaffolded exercises had not yet been introduced into SLICE N DICE, but was rather a post-pilot addition.

Like with Part 2, most spontaneous think-aloud commentary during this part related closely to specific exercises, and it was mostly in the post-interview that reflections and suggestions were elicited.

Is the task sufficiently clear?

Despite the complexity of the interface, there were no direct complaints about it. P2 commented that “Once you kind of get used to it, it makes sense” and that the tutorial helps mitigate the complexity. She especially liked the multi-tab sidebar:

“I think it worked well having it there on the side so you can look at it and then you would have an idea of the task you were trying to do./.../ **It was quite good having**

it all in one place.” (P2)

Concrete suggestions for interface improvements included a progress bar, more space to type, and the ability to backtrack through past exercises. P5, a beginner, asked for “more examples or a bit of a slower pace”.

Discussion. Part 2 probably served participants well as a warm-up in how the interface largely remained the same. In light of the positive feedback, no major changes were subsequently made to the interface.

Is the menu sufficiently navigable?

Participants again engaged in little menu exploration and appeared to rely extensively on the hints in order to locate the relevant entry. The readiness with which they consulted hints could mean that they did not see the ability to retrieve information unaided as intrinsically useful, giving them little incentive to opt out of that scaffolding. It could also mean that they were genuinely overwhelmed, and the hints provided some optional relief from that, allowing them to focus on the coding.

“I’d be much happier with myself if I managed to figure out how the code works. As opposed to, if I’m stuck, **I’d have to scroll through all of this documentation** and I don’t actually know which one is right. And it would be my choice whether or not to use it. Maybe after a while I’d stop using them.” (P3)

P5 described feeling overwhelmed by the sheer number of operations, and the possibility that any of them might be relevant, saying that “It felt like quite a big jump from last week.” (i.e. Part 2). To what extent did she make use of the thumbnails? P5, when asked about it, reported still using the thumbnail graphics:

“Quite a lot, like, I probably wouldn’t know what *Access > from dataframe > rows or columns by index* would mean, but with the graphics you do. And I like that they get bigger when you hover over them.” (P5)

Even so, P5 still relied more on hints:

“I think they [i.e. graphics] are really good, and they show you what to do, but at least for me, having the hints is definitely helpful because... the taxonomy, it makes sense, and if I were familiar with it all, it would make sense to find everything but because I am not, **I still needed to know where to look in it for the right things.**”

P3 suggested giving participants a hint on what part of the documentation to look in (for example, which category) instead of precisely which entry.

Discussion. The reluctance to properly familiarise oneself with the menu and navigating it unaided is in some sense rational. The menu, while useful as a mental organisation of the domain, will not be present in their post-SLICE N DICE coding, and knowledge thereof was not tested as a goal in itself. However, the ability to mentally clarify what operation is needed to implement a subgoal *is* important, and excessive reliance on hints is therefore undesirable. The way we addressed this problem was to place the hints in the sidebar to ensure they see the subgoal graphic *before* using it. Moreover, hints are broken down by subgoal, such that the taxonomy locations are not presented all at once.

How are the subgoal graphics perceived?

The participants appeared to make consistent use of the subgoals written inside the IDE as green in-line comments:

“The words in green really helped. They were really useful when there was no hint. Before that I was just reading the hints.” (P8)

However, one surprising observation was how rarely participants clicked on the subgoals tab to view the subgoal graphics, and used the in-line comments in their stead. This was consistent with participants’ own reflections when asked about the subgoals:

“First I was using subgoals. But then **I kind of stopped using them** because I was referring to things I had done before that worked.” (P6)

“The graphics are good and they are very helpful in how you have a visual of the dataframes. *././* **I looked at the graphics when I was reading the question** at the beginning and in the script part of the program. I was relying on the little bit there to guide me as well.” (P7)

“I think, honestly, the way this has been laid out is really good. **I did have a tendency to go straight to the taxonomy without really looking at the subgoals that much**, but that is my own fault. *././* **When I remembered to look at the graphics, I found them helpful**, but I mostly relied on the verbal cues.” (P1)

When asked about whether her inattention to the subgoals were due to their location, P1 suggested replacing the in-code subgoals with a simple “Step 1” and “Step 2” to remind people to look at the subgoals, as well as to “remind them that you don’t have to tackle all of this in one go, there are separate steps that you can take”. This suggests that the subgoals could play a psychological role in making the task feel more surmountable.

Several other participants also expressed a positive perception of the subgoal graphics:.

“They are very helpful. **It is good to see a graphical representation** before you give it a go. It kind of hints... whereabouts to look for it. Because you kind of get an idea of what it is you are trying to do before you want to do it. /.../ **I definitely think the graphical things helped**, to explain when you know you wanna do something but not understanding the steps that you need to take to do that.” (P2)

Discussion. To increase participants’ engagement with the subgoal graphics, P1’s idea of removing the full subgoal labels from the IDE was adopted. Moreover, to spatially collect all hints, the hints were moved from the strip at the bottom of the IDE, to the sidebar where they were placed next to each subgoal.

Do participants appreciate the task?

Despite its intrinsic complexity in applying programming to data wrangling problems, participants appear to have found Part 3 relevant and helpful. P6 noted that “These exercises kind of throw you at the deep end, which is good for learning”. Participants appeared to agree that the level of difficulty was reasonable, even when it was challenging:

“Definitely a reasonable difficulty. I found it quite challenging but it’s useful things, things you would like to use if you were coding. /.../ Maybe if you are a complete beginner it might be tricky to begin with but that would depend on the user. **I like the level of difficulty because I would have to think a bit for all of them.**” (P2)

The opportunity to choose whether or not to use hints was positively received among participants who had some prior experience:

“I enjoyed on being able to rely more on running it in the shell, because that **feels more like what you’d really be doing if you were trying to program**. So the opportunity to use that to debug instead of just looking at the hints was very helpful. /.../ It is definitely important to have a place where people can practice without hints to work it out for themselves.” (P1)

Since the only hints provided indicated where in the menu to look (apart from the subgoals), quite extensive verbal guidance was often necessary to ensure participants solved the exercise within a reasonable time. There were occasional worries that, without guidance external to the app, they would be stalled:

“That was **the thing I struggled with, when I couldn’t work it out**. But the information was definitely there for me.” (P1)

Discussion. It seems like this sample of participants enjoyed the challenge, but we cannot say whether their attitude would have been different if the human guidance was removed, as is the case in the main study. It was in response to such concerns that the hint provision was expanded upon, as mentioned earlier, to prevent participants from getting stuck.

8.1.5 Conclusion

The usability study, despite its small size and the difference in circumstances compared with those faced by main study participants, proved to be an effective way of improving the interface and refining the materials. We were confident that the instructions and the interface were clear enough for the exercises to be solvable in practice. There were also strong indications that the graphical elements were subjectively helpful, and that through the modifications to SLICE N DICE implemented in response to the study, we had increased the likelihood of participants engaging meaningfully with them. What remained was now to trial SLICE N DICE and ascertain whether the data it generated were appropriate for the research questions.

8.2 Quantitative pilot study

After implementing the changes suggested by the usability study, the next step was to run a miniature version of the study in our local institution. This pilot study took place the semester before SLICE N DICE was due to be launched, and has been the subject of a 2021 publication [314]. The primary goal was to catch any further software issues that may arise in less supervised sessions, and to verify that the schema of data collection was appropriate for the research questions. The purpose of the pilot was *not* hypothesis testing, nor will its results be used for calculating the sample size of the follow-up study. This is because the group sizes will be far too small to yield an accurate estimate. The objective was also not to gauge how promising the intervention seemed, since making the choice of whether to conduct higher-powered studies contingent on preliminary (and inaccurate) estimates risk introducing follow-up bias [315]. For these reasons, it is primarily the overall distributions that are of interest - not the group differences.

8.2.1 Method

The version of SLICE N DICE presented to participants in the pilot was largely the same as that of the main study, including the graphics and exercises, with some notable exceptions. As with the usability study, what the main version refers to as Part 2 (programming) and Part 3 (programmatically data wrangling) were merged into a single Part 2. We will use “Part 2/3” to refer to this part. Other differences, some of which were directly in response to findings from the pilot, included:

- **Part 1 scoring:** in the pilot, participants only receive feedback on Part 1 exercises once they have submitted their solutions, whereas in the main study they need to re-attempt each subgoal until it is correct.
- **Part 2 documentation:** in the pilot, the relevant documentation entry was not automatically opened in introductory programming exercises, but instead showed as a card before the programming exercise was presented.
- **Hint navigation:** in the pilot, hints regarding where in the menu to find a relevant command did not work as hyper-links that they could click on to directly open the documentation entry.
- **Evaluation surveys:** the pilot study's evaluation surveys were shorter, and did not include questions that asked explicitly about the features of interest (e.g. subgoal graphics).
- **No unscaffolded exercises:** the pilot study presented subgoal scaffolding alongside every single data wrangling programming, but the main additionally created 3 temporarily unscaffolded problems.
- **Data back-end:** the pilot study collected fewer interaction data variables and, unlike the main study, did not record time off task (e.g. the tab losing focus), which leads to inflated time on task measurements.

Finally, graphics and exercise wordings were at this stage less refined, and were subject to various minor modifications before the main study.

Design

The pilot study's SLICE N DICE version technically had the same built-in experimental structure as the main study, involving two binary independent variables - the presence of subgoal graphics (SG/-SG) and thumbnail graphics (TG/-TG) - however we will mostly aggregate data across these variables and inspect overall dependent variable distributions and disaggregate them by exercise. The main performance metrics of interest are time on task and the number of attempts, but we will also explore subjective evaluation data.

Participants & recruitment

Following ethics approval by the local institution, participants were recruited from the local undergraduate population via an introductory CS0 programming course, a preparatory module for master's students about to begin a CS conversion course, and a campus-wide email campaign to the STEM-related departments. In the two courses, participation was voluntary and non-credit bearing. In the CS0 course, participation was completely opt-in, while for the conversion

course, SLICE N DICE was integrated as a voluntary classroom laboratory. As with the final study, respondents were offered a data wrangling e-book as an incentive to complete the study, but unlike the final study, they also received a £10 voucher as compensation. This payment was to accelerate recruitment, and to encourage completion even in the presence of bugs, however the choice to offer it implies that the pilot study is likely to overestimate the completion rate and recruitment success of the follow-up study.

Procedure

Due to COVID-19, all participants undertook the study remotely and asynchronously, by accessing the URL at a time of their own convenience. A channel was set up in a learning management system where participants could ask questions and inform us about bugs or confusions. The only exception was the conversion course students, for whom participation started off synchronously with a short lecture, delivered by me, and two tutorial sessions in which I was available to answer questions.

8.2.2 Part 1 results

Part 1 contains 64 operation cards and, interspersed among them, 9 non-programmatic data wrangling exercises. We were interested to see if participants progressed smoothly through the deck or whether they are ever stalled, and whether the performance measures produce a wide enough distribution to differentiate different levels of ability, without floor or ceiling effects.

Participant characteristics

Out of 55 participants who completed at least one exercise in Part 1, 44 participants completed Part 1 fully. There does not appear to be a particular point at which the participants chose to withdraw. As revealed by the initial demographic survey, among the 44 participants, 24 were female and 18 male, with 2 of unspecified gender. Participants represented a broad range of disciplines, from chemical physics, to digital media, economics, and zoology. 7 reported being CS students: based on the recruitment efforts, these would presumably be in the beginning of their first-year introductory course. 36 chose to do it in Python, while 8 chose R. Students' self-ratings of their ability to program in a language, which were Likert-scale items rated from 1 (*Not at all*) to 5 (*Advanced*), indicate that those who signed up and completed Part 1 generally had some prior exposure to programming in either procedural Python or R (see Figure 8.5). This probably reflects the fact that those who have been exposed to programming are more likely to see its value, and are therefore intrinsically motivated to sign up.

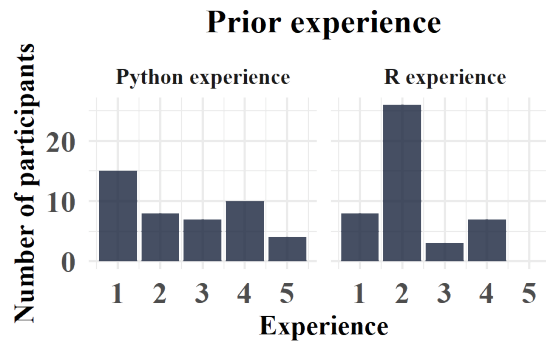


Figure 8.5: The self-rated prior experience in Python and R (1=*Not at all*, 2=*A little bit*, 3=*Beginner's level*, 4=*Intermediate*, 5=*Advanced*).

Operation cards

Part 1 contains a deck of 64 operation cards. The time taken to read each card was recorded and summed into a total reading time (shown in Figure 8.6a), to give us a sense of how quickly they made progress through the deck. The median total reading time was 1387s (≈ 23 min, IQR=725s). As common with response time distributions, we find suggestions of a long tail. 5 participants took less than 6 minutes - interestingly *all* of these were in the TG condition and thus had explanatory graphics on the card.

When looking at duration times card by card and sorting them by their order of occurrence (Figure 8.6b), we find that individual cards are generally read in less than 20 seconds, and that these became shorter and more stable with time, which could reflect both fatigue or learning. A few cards take exceptionally long to read: these include calculating a matrix through pairwise arithmetic, group-wise aggregation, merging, sorting a dataframe by column, and pivoting. Since these cards are presented in a sequential context, and later cards are presumably faster to read due to learning effects, these reading times do not provide an absolute metric by which to measure the complexity of an operation. It is interesting, however, to see intuitively more complex operations appear more time-consuming to read.

Time on task

The time spent solving an exercise, from when the exercise is presented until the participant hits *Submit*, would inform us about is participants are stalling at an exercise, and how easy one person finds the task overall relative to others.

Participants' total time on task data, seen in Figure 8.7a, indicates a slight positive skew, centred at around 19 min (median=1148s, IQR=991s), but with 6 participants taking more than double that amount. These could reflect confusion over the interface, or them taking a break, but interestingly all but 1 of these outliers were completely without graphics (\neg TG/ \neg SG). Note that if, on average, the exercises take 19 minutes and the operation cards take 23 minutes, Part 1 could comfortably be completed within 1 hour. When time on task is plotted exercise by exercise

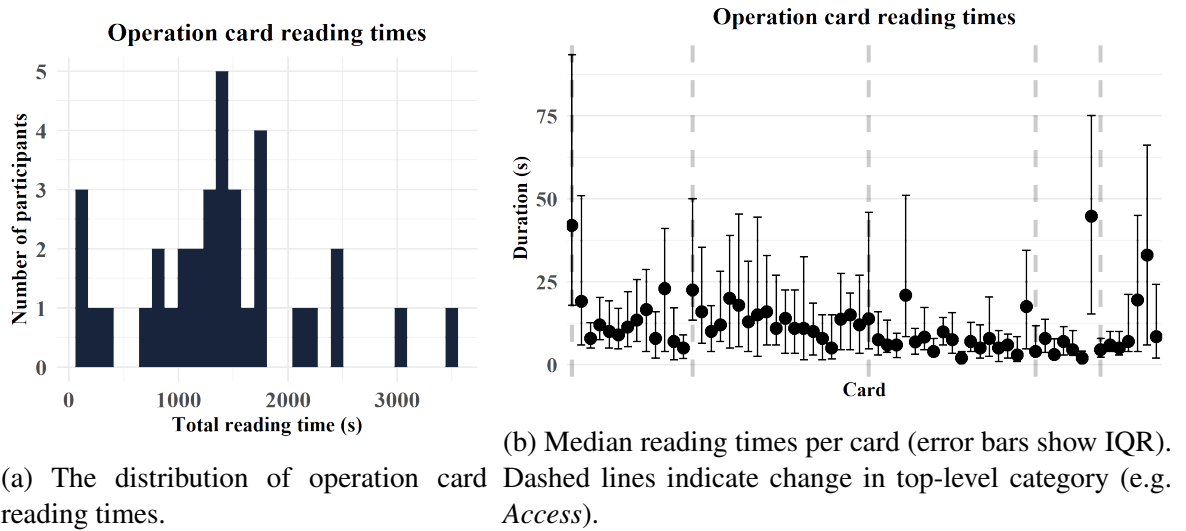


Figure 8.6: Results relating to operation card reading times in Part 1.

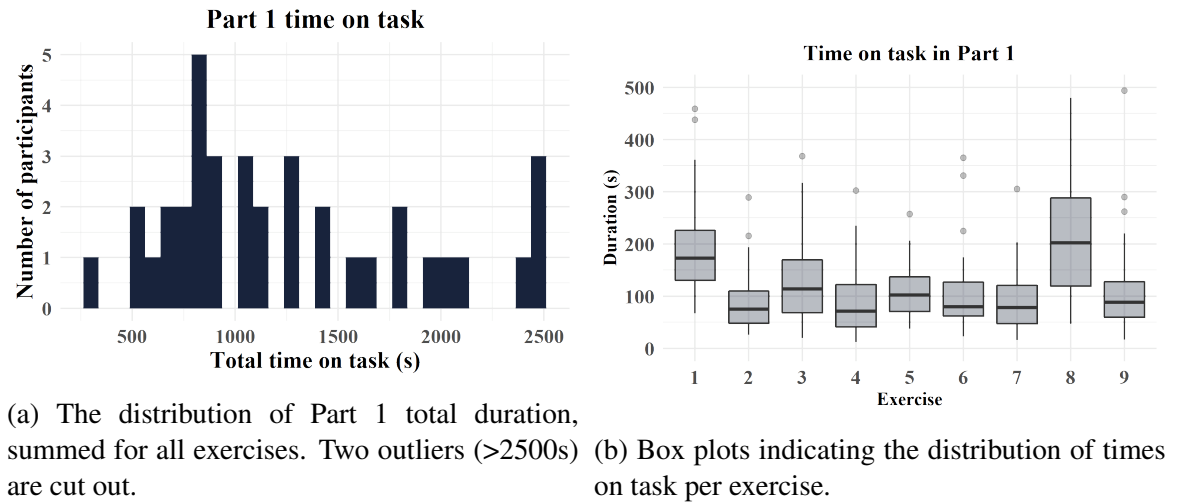


Figure 8.7: Results relating to time on task in Part 1.

(Figure 8.7a), we should remember that exercises contain different numbers of subgoals (in order: 3, 4, 2, 3, 2, 2, 2, 4, 2). One finds that exercises on average take under 2 min to complete, except the first exercise, where participants presumably are familiarising themselves with the nature of the task, and exercise 8, which at 4 subgoals is relatively complex.

Number of correct operations

Unlike the final study, where feedback regarding the subgoal correctness is given on a subgoal-by-subgoal basis, the pilot only gave feedback once the participant had given an answer to every subgoal. Consequently, performance could only be operationalised as the number of correctly selected operations (“correct subgoals”) on the first attempt, and not as the number of incorrect attempts. The maximum possible score was 24, but the maximum score achieved was 21 (median=14.5, IQR=6.75). The distribution appears concentrated towards the upper bound, with few

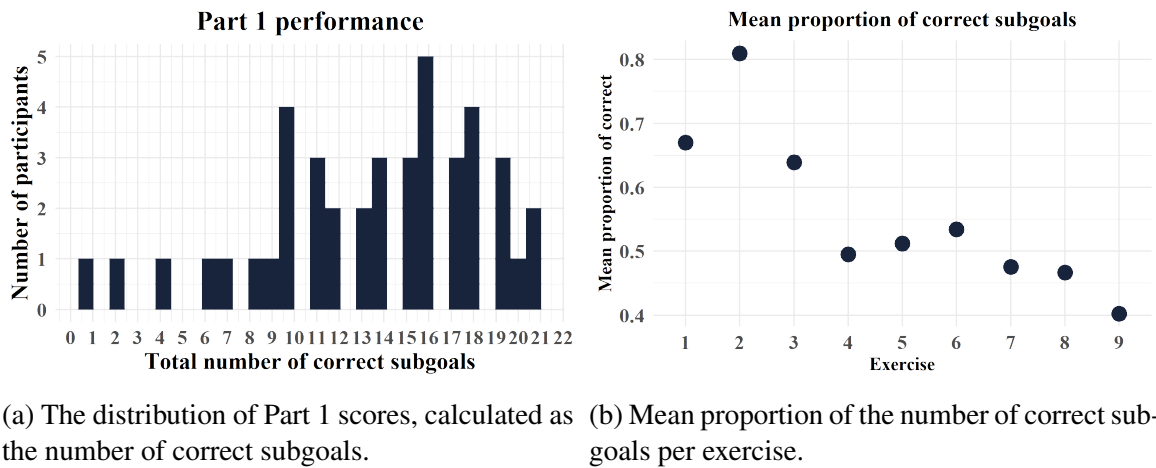


Figure 8.8: Results relating to correctness scores in Part 1.

(7) people achieving scores below 10. The skew is not extreme enough to constitute a ceiling effect however: the measurement still reveals a range of performance levels.

The left tail may be more problematic. It could be in spite of genuine effort: out of 7 participants who scored less than 10, all but 1 were in the \neg SG condition, which could mean that an absence of subgoal graphics has a crippling effect on performance. However, it is also possible that it reflects a lack of genuine effort. Since in this version, the participant is allowed to proceed regardless of correctness, they may be tempted to choose random operations, which also would undercut the informativeness of the time on task metric.

Upon looking at performance broken down by subgoal, we must remember that exercises contain a different number of subgoals. For that reason, the proportion of correct subgoals per exercise has been averaged across all participants: the result is shown in Figure 8.8b. We find that correctness goes down over time, from .8 to .4, which could be due to fatigue, reduced motivation to perform, but also due to intrinsic exercise complexity. Modifying the performance metric into number of incorrect attempts before finding the correct solution would presumably be less confounded by motivation-related factors.

Attitudinal data

In the evaluation survey given at the end, participants were asked to rate how concentrated and motivated they felt, as well as how enjoyable, effortful, and worthwhile they found the task to be. These ratings, which ranged from 1 (*Not at all*) to 5 (*Very much*), are summarised in Figure 8.9. Participants mostly gave a middle score of 3 when rating their concentration and the effort that the task demanded, suggesting that the difficulty level was well-calibrated and did not require excessive exertion. The enjoyability and motivation distributions tended towards a rating of 4. The high motivation could be due to monetary compensation, but the enjoyability scores are especially encouraging, as they reflect a more intrinsic property that would likely improve participant retention. Finally, the flat distribution of worthwhileness ratings suggests

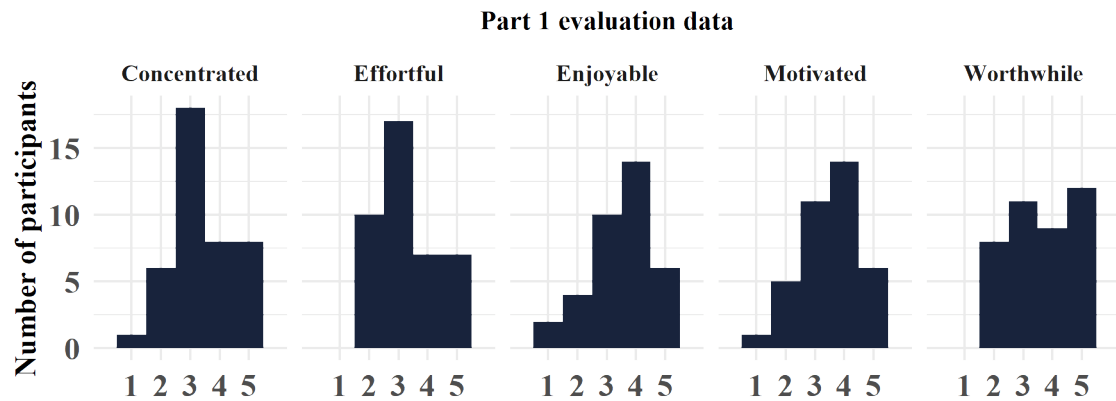


Figure 8.9: Results from five items in the evaluation survey, rated 1 (*Not at all*) to 5 (*Very much*).

disagreement. It is conceivable that the choice of featuring a non-programmatic introduction would polarise people based on dimensions such as prior coding experience.

Open-ended feedback

14 participants chose to leave feedback in the free-text field of the survey. Among these, two people commented on graphics specifically, although it is not exactly clear whether it is subgoal graphics or thumbnail graphics they refer to (emphasis added):

“**The graphic were really helpful** for answering the questions! And I think the interactive parts have really made me understand the concepts.”

“I tried not to rely on the graphics because **I found them to act somewhat as a ‘match the shape’ type problem**. Possibly would have been better if the graphics were able to be toggled.”

The second person comparing it with a “match the shape” problem was in the TG/SG condition, which to them may have introduced excessive scaffolding.

The nature of the task itself - that of operation selection - interestingly did not receive much commentary, except for the following:

“The drag and drop thing in the exercises was a bit confusing at first but it made sense after a while.”

Two people requested that Part 1 would introduce programming immediately:

“Would have liked to have seen how each of the operations could be done in Python, like what should be typed in when performing each operation.”

“I think it would be nice to be able to perform each action while you’re learning about it to make it more clear what it does.”

A specific suggestion was the ability to try again when a subgoal was incorrect:

“Could be more helpful if you got to have a 2nd attempt without the answer showing up - to encourage problem solving.”

Four participants asked for the problems and in particular their wordings to be clarified.

Implications

In light of the feedback, all exercises and their wordings were reviewed for clarification, with one data science novice consulted for feedback on the revised wordings. Additionally, the feature request of permitting multiple attempts was addressed, leading to the subgoal-by-subgoal feedback mechanism of the final SLICE N DICE iteration. However, the more drastic change of adding code execution to Part 1 was not implemented, since the time that would take was not available. To gather more feedback directly relevant to the subgoals and graphics, additional items were added to the evaluation surveys.

8.2.3 Part 2/3 results

Recall that, for this version, Part 2 and 3 were merged together. This meant a total of 21 programming exercises, the first 6 of which were introductory programming exercises, covering concepts like variables and functions.

Participant characteristics

Out of 41 participants who solved at least one exercise, only 16 finished all 21 exercises. The drop-off in participants is visualised in Figure 8.10. There appears to be two points at which several chose to drop out: the first is during exercise 3, which is about expressing logical conditions, while the second is during exercise 18, which is when the data wrangling exercises begin to get relatively complex, featuring more than 4 operations per solution.

6 of the participants who completed the study were females and 10 were males. These participants did not appear to over-represent those with prior experience, or a particular degree major: only 1 was a CS student. This allays some fears about potential attrition biases where the most experienced participants persevere.

Time on task

Given the small sample size, it is difficult to make projections of what the time on task distribution will look like. What emerges from the observed times on task (shown in Figure 8.11b) is

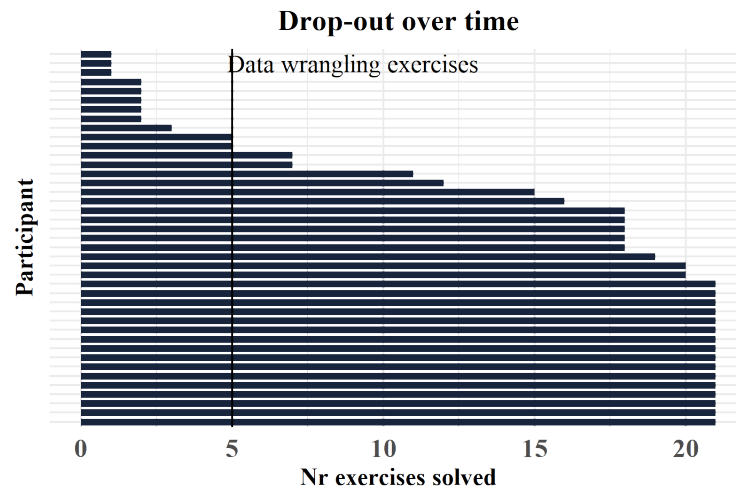


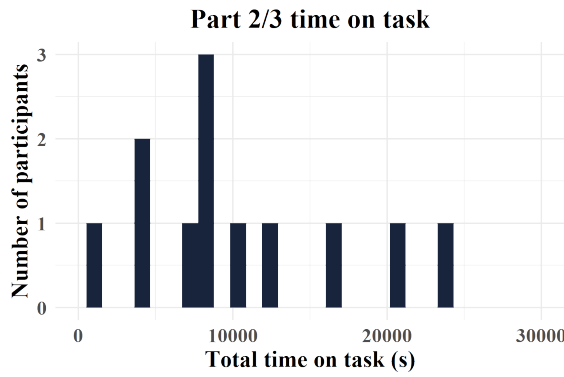
Figure 8.10: Each horizontal bar represents the persistence of a participant, thus showing the attrition of participants over time. The vertical line shows when basic programming exercises turn into multi-command data wrangling exercises and is inclusive.

a positively skewed distribution with a median of 11336s ($\approx 3\text{h } 9\text{min}$). While this number may seem high, it is actually close to the estimated duration, which is 4h for the entire study (since Part 2 and 3 are merged here, they would together take up 3h). Additionally, these recorded times do not factor in mid-exercise breaks (i.e. signs of inactivity). Nevertheless, we should be mindful of the fact that 7 participants recorded times on task of more than 4 hours, although 4 of those are outlier values ($>7\text{h}$) that almost certainly reflect breaks.

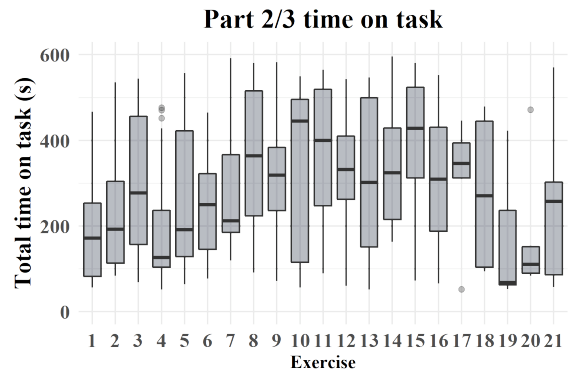
If we break down time on task data on an exercise basis, as shown in Figure 8.7a, and also include incomplete observations (i.e. participants who did not complete all 21 exercises) we find that exercises generally take between 100-500s to solve, and that the time required grows as the exercises become more complex. However, it is notable that, although the last few exercises are more difficult, time on task ceases to grow. This could be a reflection of a greater reliance on hints due to fatigue, a learning effect, or a selection effect (the sample of the later exercises is smaller).

Code executions and hint usage

Mid-way through the pilot study, the logging system was equipped with functionality to collect code executions and the number of times hints were referenced. Due to it being added later, the data is available for only for 36 participants, 10 of whom provided complete data. In the boxplot charts of Figure 8.12, these metrics are shown for each exercise on a per-participant basis (since hints only existed for data wrangling exercises, Figure 8.12b only features exercise 6-21). The charts indicate that, while time on task went down for the last exercises, so did the number of attempts, but the hint usage increased. This suggests that, for the later exercises, participants did indeed choose to rely unproductively on hints, rather than attempt to solve the exercises in

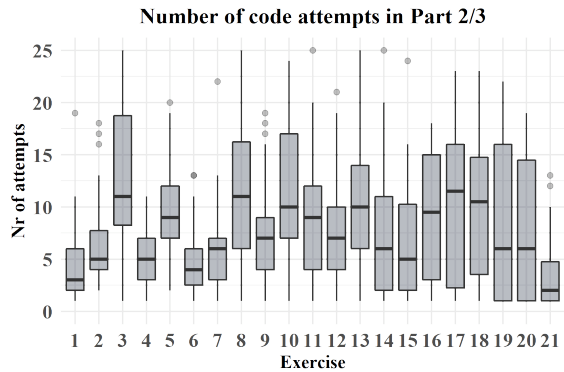


(a) The distribution of Part 2/3 total duration, summed for all exercises. Four outliers (>25000s) are cut out.

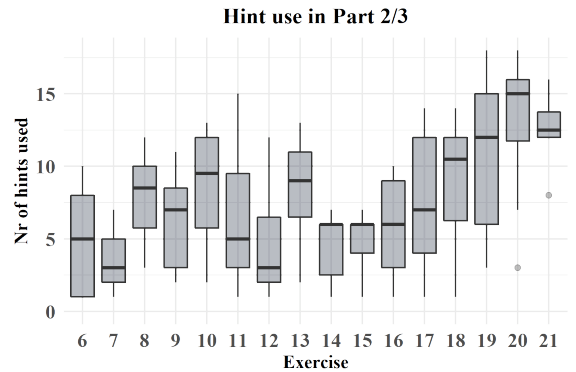


(b) Time on task (s) per exercise.

Figure 8.11: Results relating to time on task in Part 2/3.



(a) Number of code executions (attempts) per exercise and person.



(b) The number of times hints used per exercise and person.

Figure 8.12: Results relating to process metrics during Part 2/3.

earnest.

Attitudinal data

The evaluation survey at the end contained the same items as that after Part 1. Due to the time-out built into Part 2 which enables the user to proceed despite not completing an exercise, more participants completed the evaluation survey than finished each exercise: the sample size was 26. Compared with Part 1, it shows similarly middling scores on concentration, but a marked movement towards the higher bound in how effortful the task was rated: most participants rated it as a 5 (*Very much*). The mode rating in enjoyability is no longer a 4, but a 3. The programmatic data wrangling exercises are intrinsically more difficult than the operation selection tasks of Part 1, so the high effort is not surprising, but may play a part in the attrition. It is also a cause of some concern that motivation and worthwhileness levels have shifted somewhat leftwards compared with Part 1, although the difference is slight.

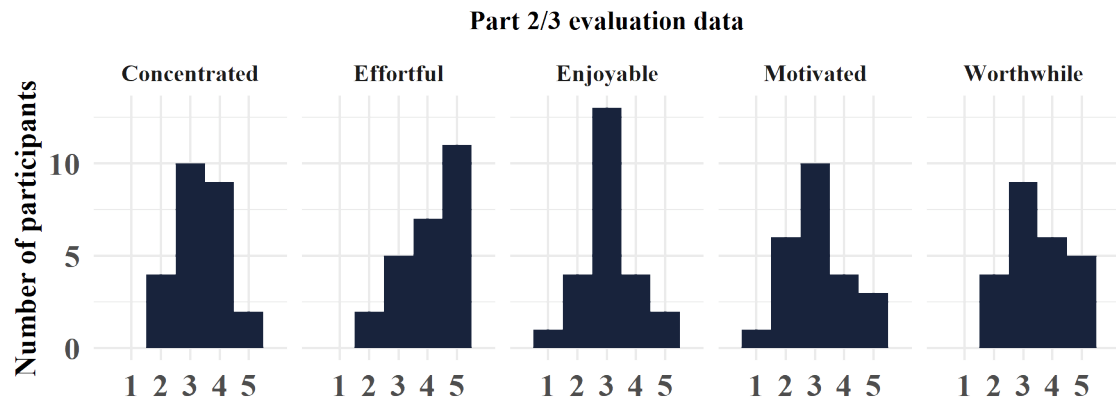


Figure 8.13: Results from five items in the evaluation survey, rated 1 (*Not at all*) to 5 (*Very much*). Since Part 2 and 3 were merged into one, this survey refers to both.

Open-ended feedback. Only 8 participants chose to leave free-text feedback. Of these, two expressed enjoying the task:

“I have really enjoyed this so far! I am not a strong programmer and struggle to remember the necessary steps so this is great for me.”

The following participant (TG/−SG) provides a clear example of the target audience, and explicitly commented on the subgoals:

“I study Chemistry and I have been looking at some data analysis graduate jobs which look really interesting but I have no experience in programming. I was considering trying to find somewhere to learn the basics when you emailed about your app! I thought the interactive exercises were so useful for understanding the concepts and **breaking the task down into subgoals really helped**, so did the hints for if you were really stuck. I did find it really difficult towards the end, I think the hardest part for me was trying to use the right brackets. I found with a lot of them I basically had the right thing but it was just the type of brackets or use of ” that were confusing me. But overall I thought the app was really great, very user friendly, clear instructions and **the graphics helped so much when trying to decide which strategy to use.**”

However, not everyone agreed with this sentiment. Two participants requested better formulation of the questions and one person requested the ability to review past answers in order to reuse strategies. The following participant (SG/−TG) felt demotivated due to the use of API commands they did not get to understand in depth:

“I thought the course was useful however the exercises jumped massively in difficulty from the intro courses to the actual programming. All of the sudden **being**

asked to use complicated functions never seen before was challenging to a degree that I found it very demotivating.”

The following participant, who in the \neg TG/ \neg SG condition lacked graphics completely, wrote the following:

“**There was no graphics, which I found very difficult** and it made it hard for me to follow what was happening to the dataframes/ numbers. As a complete beginner to python, I found this course quite difficult and confusing at times! I didn’t always understand what the questions were asking me to do and **found finding information in the taxonomy to be difficult sometimes**, as there is a lot of jargon and information isn’t always where I would expect it to be as a newbie!”

It is plausible that thumbnail graphics could have remedied some of their navigation issues, and subgoal graphics remedied their issues of “following what was happening”.

Implications

The results relating to Part 2/3 suggest that the time estimate for completing SLICE N DICE is reasonable and that several participants found the subgoals and graphics helpful. One cause of concern was found in how, for later exercises, the effort appears to go down, since hint usage increased while the time on task and number of code executions decreased. This could be due to fatigue or a feeling of defeat. There ratings of perceived effort were also high, and in the discussion, some participants reported feelings of low motivation. This prompted another round of revisions to the exercises and feedback, but may mean that the later exercises will be less reflective of true effort.

8.3 Chapter summary

- Two pilot studies were conducted in order to determine that the front-end was usable, whether the graphics were perceived as helpful, and how various process metrics changed over the course of Slice N Dice.
- The qualitative usability study prompted improvements in the clarity of the Part 1 interface and the displacement of where the subgoals were located in Part 3 to ensure that participants engaged with it more. The interviews also suggested a broad appreciation for all three types of graphics (subgoal, thumbnail, and operation card graphics).
- The quantitative pilot study prompted improvements in the wording of various questions, the back-end data schema, and raised some concerns regarding low motivation in the last few exercises.

Chapter 9

Slice N Dice validation studies

The results of the SLICE N DICE usability study, documented in Chapter 8, gave us confidence that the graphical elements were perceived as helpful and appreciated. However, except as a possible motivational boost, this does not imply that the graphics are *objectively* helpful. To be the latter, then at a minimum the graphics must be *interpretable*. By this term we mean that, upon reading them, a person is able to infer the behavioural logic of the operation that the graphic is intended to represent. Establishing such interpretability by comparing user interpretations with a ground truth is essential, as learners' own meta-cognitive intuitions are not necessarily accurate. Furthermore, without such assurances, a null effect would not necessarily mean that graphics added no further pedagogical value - it could also mean that they simply were not comprehensible.

Interpretability is difficult to define operationally. It is highly extrinsic, in the sense that it depends on factors beyond their graphical characteristics: for example, prior experience, familiarity with various graphical conventions, and whether interpretation is completely open-ended or constrained.

To be clear, we do not have strong *a priori* reasons for doubting the interpretability of the graphics. Firstly, Part 1 was deliberately designed as an on-ramp for familiarising participants with the graphical system. Secondly, the graphical notation is “literal” in how it represents data structures in the same way as the API conceptualises them (i.e. as 1 or 2-dimensional tables). It introduces few syntactic novelties beyond employing colour using conventions that are already ubiquitous in everyday life: to highlight, to connect, and to signify relative magnitude. Thirdly, if the interpretability of the graphics were seriously in question, we would expect at least one participant from the studies in Chapter 8 to have voiced concerns. Nevertheless, interpretability is a fundamental requirement of the graphics, and should be addressed directly.

For this reason, this chapter presents two studies, for probing the validity of subgoal graphics and thumbnail graphics, respectively. Although both were administered as surveys, they tested validity in different ways: subgoal graphics were interpreted through participant-authored subgoal labels, and thumbnail graphics were interpreted through multiple-choice items.

9.1 Subgoal graphic validation study

The goal of subgoal graphics is to facilitate a participant’s comprehension of a solution in terms of the data structure manipulations they require (i.e. plan composition), inspiring ideas of which operations to search for in documentation resources. Ideally, this comprehension would also transfer to unseen problems. In this study, we sought to ascertain their interpretability, in particular wondering if participants’ understanding of the operation depicted in a subgoal graphic matches the logic we intended to illustrate. This feeds into RQ1, on the pedagogical effects of subgoal graphics.

9.1.1 Method

The study took the form of an online survey (a copy of which is in Appendix E). The survey began with a participant information form and consent form. Within this form, our stated goal was to “evaluate the usability of graphics that describe data operations”. It was emphasised that it was the graphics that were evaluated, and it was not an aptitude measure of any kind. This was followed by a short demographic section that asked participants about their previous programming experience in Python, SQL, and Excel, followed by two sections that are described below.

Background section

The survey needed to establish some baseline knowledge of concepts and terminology before it could ask of participants to interpret graphics in terms of it. Ideally, they would have been taught the behaviour of *all* data operations in a non-graphical way. However, to prevent participant fatigue, we limited ourselves to a background section that explained the fundamental logic of a vector, matrix, and dataframe, the three main methods for indexing array structures (by index, by mask, by condition), the three categories of arithmetic (an aggregation, element-wise arithmetic, and arithmetic with scalars), and three methods for combining dataframes (merging, concatenating, and adding a new column). This section illustrated all concepts using minimalist diagrams that visualised the pre- and post-operation state of a data structure, populated with values, and without the colour highlights or other graphical elements used in subgoal graphics (see Figure 9.1). The omission of further graphical elements was by design: although the operations needed to be exemplified somehow, it was important not to prime them with the more idiosyncratic style choices proposed for subgoal graphics.

Exercises

To gauge the interpretability of the subgoal graphics, we gave participants a sample of five exercises from SLICE N DICE, chosen to represent the range in difficulty. For each exercise, the

You can access values from a vector via this index, for example, here you access the value 5 by specifying the index 2.

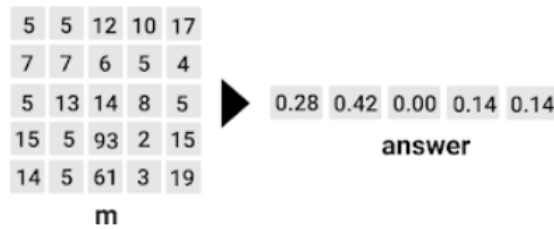


Figure 9.1: A snippet from the background section. Note the minimalist style of the graphics, carefully designed to avoid priming the participant.

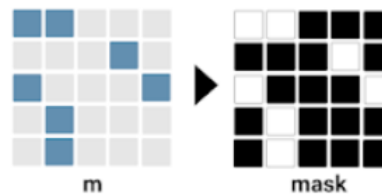
participant is presented with a problem description and a list of subgoal graphics, but without any subgoal labels. Instead, the participant is asked to write their own subgoal labels in open-ended text fields. We reasoned that if their subgoal labels are reasonably equivalent with ours, the labels can be regarded as interpretable. To clarify what was asked of them, the five exercises were preceded by a worked example. The instruction was formulated as follows: “Try to interpret the graphic in terms of what action it represents and provide a text label for it. There is no exact right answer, and no programming syntax is needed: simply provide human-friendly labels like *Rotate the matrix*, *Take the row whose B-column equals 3*, *Calculate the mean*.” An example of a subgoal label field is shown in Figure 9.2. Some annotations had been removed from the graphics to prevent the task from being overly simple, but data structure labels were kept. There were 5 exercises and 18 subgoals to label in total. For each exercise there was also an open text field to provide general feedback.

To measure the equivalence of the participant-authored and instructor-authored subgoal labels, we decided upon an ordinal scale that rated each subgoal label (i.e. interpretation) as follows: (0) entirely incomplete or incorrect, (1) mostly incomplete or incorrect, (2) somewhat incomplete or correct, or (3) largely correct. This scale recognises that provided labels can differ on at least two dimensions: completeness, since participants may vary in the detail or ambiguity they formulate their labels with (e.g. saying *Aggregate* instead of *Calculate the mean of each country’s income*) and correctness, since the inferred behaviour may be plainly wrong (e.g. *Sort* instead of *Flatten*). Since participants had not received an introduction to all the operations, we could not expect them to specify exact operations. Likewise, since subgoal labels themselves have numerous degrees of freedom in their formulation, we could not expect word by word equivalence, but rather focus on the deeper meaning being expressed. Although we did rate the provided responses, we view this mainly as a qualitative study, allowing us to flag potential issues in how novices comprehend the graphics.

What proportion (out of all 5s in the entire matrix) does each column in matrix m have?



1.1. What action does the following graphic suggest? *



Ditt svar

Figure 9.2: The first subgoal graphic in the first exercise in the survey.

Participants & Recruitment

The study was advertised during a lecture in a CS0 course offered at the local university, about 4 weeks into the semester. This CS0 course was aimed at complete beginners in programming, predominantly for freshmen, and at this time point they would be expected to have a basic understanding of procedural Python. The only exclusion criterion applied was that participants had to be over 18 years old. 15 participants ended up completing the survey online. Via a Likert scale, all participants reported having less than 1 year of experience in Python, all but two had experience with SQL, and a median of 1 year of experience in spreadsheets. Respondents entered a lottery where 4 people won £30 each in vouchers.

9.1.2 Results

The subgoal labels were rated according to the above-mentioned scheme. When summed for each participant, the mean score was 38.47 (out of a maximum possible score of 54, $SD=14.7$), which is more than 70%. Aggregates of ordinal data provide a very coarse measure, however. If we disaggregate the data based on subgoal, as done in Figure 9.3, we find a large variance among them. Since the score distribution of any given subgoal reflects differences in both inherent complexity and the specific graphic, any two subgoals are not directly comparable. Nevertheless, 6 subgoals stand out as particularly error-prone: 1.1, 1.4-2.2, 3.2, and 5.2. These involve matrix masking, element-wise arithmetic with single value, pair-wise arithmetic, matrix flattening, row-wise aggregation and group-wise aggregation, respectively. Several of these are intrinsically

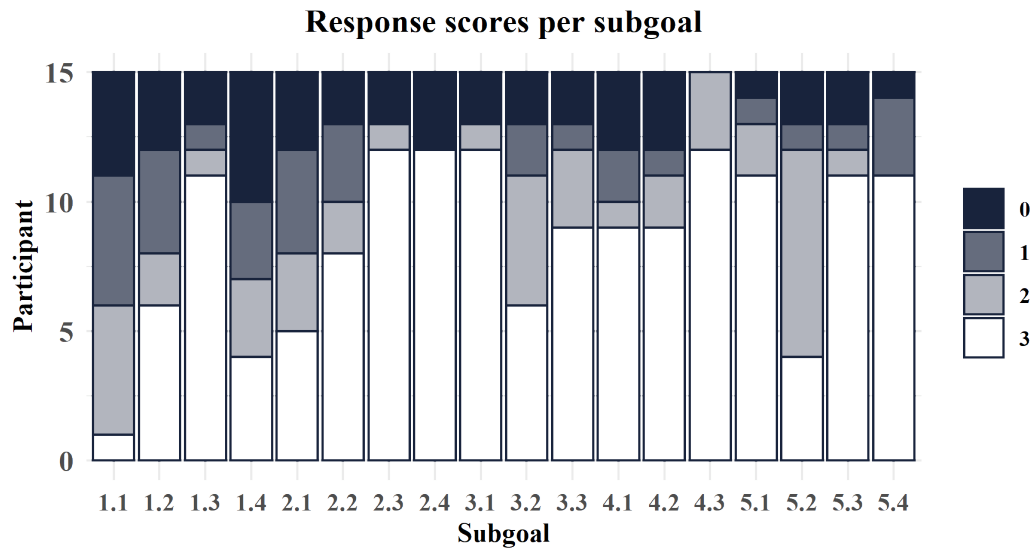


Figure 9.3: The relative proportions of 0-3 ratings for each subgoal exercise.

complex, which accounts for some of the misinterpretations. From inspecting the submitted labels further, we notice the following three sources of error:

The need for annotations. Recall that subgoal graphics within SLICE N DICE occasionally contain annotations to clarify which calculation is being performed (e.g. whether it is addition or multiplication) but that these were removed in this survey to prevent the task from being too simple. To some participants, it is clear that these annotations were necessary: for one graphic that displayed a vector and a single value being transformed into another vector, 5 participants expressed being confused by what it meant, which an annotation could have remedied.

Overly literal interpretations. There were several instances of interpretations that reasoned that the colours represented literal colours. For example, in the subgoal shown in Figure 9.2, one person interpreted it as “Filter the graph by blue as true and grey as false” while another person interpreted it as “Leave 5s blank, anything else black”. For these participants, it was apparently not evident that black represented `False` and blue represented the presence of 5s. While a small minority, it indicates that graphical conventions may not be as universal as presumed.

Confusion about colour. Since colour represented different relationships depending on context, this inconsistency occasionally led to incorrect references. For example, in a graphic representing column-wise aggregation, every column had a different shade of blue, but because this shading increased in intensity for each column, two participants interpreted it as meaning that it somehow involved sorting.

9.1.3 Discussion

Though we have seen that the graphics are not universally self-evident, we believe that the proportion of correct interpretations is overall adequate, especially given the participants' low prior experience and the lack of realism in the context that the graphics were presented in. Whereas actual SLICE N DICE participants would become accustomed to the available operations and the graphical style throughout Part 1, and moreover have the command menu available to constrain the possible interpretations, participants of the survey would only have had a 5-minute whirlwind introduction to the concept of tabular data and operations thereof. Their interpretations would therefore be much less constrained, hence the validation test is relatively severe and likely underestimates the graphics' interpretability in a more realistic context.

9.2 Thumbnail graphic validation study

Recall that thumbnail graphics are small, simple graphics that depict an operation but that do not contain data, in part due to the spatial constraints of a computer menu and in part for the sake of making the thumbnails readable at a glance. Unlike subgoal graphics, they are also not tailored to specific exercises, but rather illustrate archetypal toy examples of an operation. We could imagine that there is a trade-off in the speed and interpretability, since readability increases with less information, and interpretability with more information. In the survey study described below, we pose the following research questions, which all feed into RQ2, on the effect of thumbnail graphics:

RQ2a: How easily can learners interpret the meaning of a thumbnail graphic?

RQ2b: What is the effect on interpretability in adding data to the graphics?

RQ2c: Which graphic design do learners prefer as thumbnails?

9.2.1 Method

Similar to the subgoal graphics survey, the thumbnail validation survey (a copy of which is in Appendix F) began with a participant information form and consent form. Within this form, our stated goal was to “evaluate the usability of graphics that describe data operations” and it was emphasised that it was the graphics that were evaluated, and it was not an aptitude measure of any kind. This was followed by a short demographic section that asked participants about their gender and previous programming experience, followed by a number of sections that are described below.

Matching problems To establish the baseline knowledge for the task, the survey contained the same background section as the subgoal survey, except there were also 5 **matching problems**. For such problems, the participant is presented with three thumbnail graphics, and is asked to select which of three easily confused operations that each graphic is meant to represent. As an example, see Figure 9.5a. Such a line-up could be viewed as testing the ambiguity of the graphic: if a graphic repeatedly gets interpreted as representing the wrong operation, then such a graphic is not fit for purpose.

The reasons for why we included matching problems instead of just multiple-choice questions (what we call **interpretation problems**, see below) are three-fold. Firstly, they are more efficient in how they cover multiple operations at once and therefore make the survey less laborious to complete. Secondly, they reflect the fact that the meaning of certain graphics depend on the user contrasting them with other graphics. For example, the graphic for positional indexing (*Access by index*, see the bottom graphic in Figure 9.5a) has a small circle denoting the index, while *Access by condition* does not. As a result, the *absence* of an element that exists in a related graphic becomes meaningful, and this absence would only be noticed when juxtaposed with *Access by index*. Thirdly, the spatial juxtaposition of related graphics is representative of the context in which interpretation will ultimately take place, namely in a menu with adjacent thumbnails.

We are also conscious of the drawbacks matching problems carry: since within an exercise, every pairing reduces the number of options in the next pairing, a process of elimination could be applied. Matching problems are consequently much easier to solve than if graphics were interpreted in isolation from each other. For this reason, matching problems were only used to validate graphics that were fundamental enough that other operations depended on them. To avoid repetition, these questions will be described in more detail in the results (Section 9.2.2).

Interpretation problems without data In the second section, the participant is presented with 8 multiple choice *interpretation problems*. These problems present a single thumbnail graphic and asks of the participant to select which operation (out of 4 options) that the graphic represents. It is thus different from a matching problem in that they could no longer adopt a strategy of elimination, by pairing up the most obvious correspondences first. Instead it features 3 distractor items, all of which were chosen to be seriously plausible candidates. Compared with the matching problems, the interpretation problems therefore offer a relatively severe test of a graphical readability. An example of such a problem is shown in Figure 9.5b. Note that participants do not receive feedback to their response.

It would be infeasible and superfluous to pose this question for every single operation in the ontology. Since 15 had already been covered by the matching problems, and many graphics across data structures are similar enough, we selected 8 operations deemed to be at particular risk of ambiguity. These judgements were based on earlier feedback from the usability study and

from a more exhaustive version of the survey given to 4 participants as a formative mini-pilot.

Interpretation problems with data In the third part, the exact same 8 interpretation problems are presented again - in the same order, and with the same options - except the thumbnail graphics are now populated with data, similar to the tooltips and operation cards in SLICE N DICE. This provides a way of measuring intra-rater consistency, as a way of gauging response quality, and also a way of measuring whether the additional information helps resolve the remaining ambiguity.

Variation exploration In the next section, we sought to probe the design space more widely, by asking participants directly which graphical style, out of a set of 4 variants, that they prefer. The 4 variants are illustrated by example graphics displayed in Figure 9.4, and are characterised by the following:

Image 1: This is the representation most typically seen in textbooks and API documentation, namely how the data structure prints in the standard output. It consequently has no highlight colour or shading to separate values from indices. Vectors are printed horizontally, and indices are shown regardless of their relevance to the operation.

Image 2: This style is the same as used in the graphics of the background section. The dataframe header and elements are visually distinct, vectors are vertical to emphasise the correspondence with the column, and the graphic is populated with sample data. Importantly, there are no colour highlights.

Image 3: This style is similar to Image 2, but with colour highlights to further emphasise the relationship between the pre-operation state and the post-operation state.

Image 4: This style is similar to Image 3, but without data.

Participants were asked to select which style they preferred in two different contexts: “A thumbnail in a computer menu for accessing the corresponding computer command” and “A textbook illustration to explain how an operation works”. Our expectation was that Image 3, which combines highlights with data, would be preferred for textbooks, but that the smaller and more minimalist Image 4 would be preferred as thumbnails.

End section Towards the end, participants were asked to rate, on a scale of 1 (*Not helpful at all*) to 5 (*Very helpful*), how helpful they found the graphics without data, and the graphics with data.



Figure 9.4: The four different variants presented in the variant exploration question.

Which image represents the following access operations? *

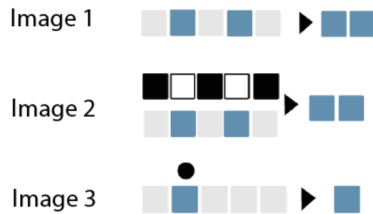


	Image 1	Image 2	Image 3
Access from vector by index	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Access from vector by mask	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Access from vector by condition	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

3. What does the following graphic represent? *

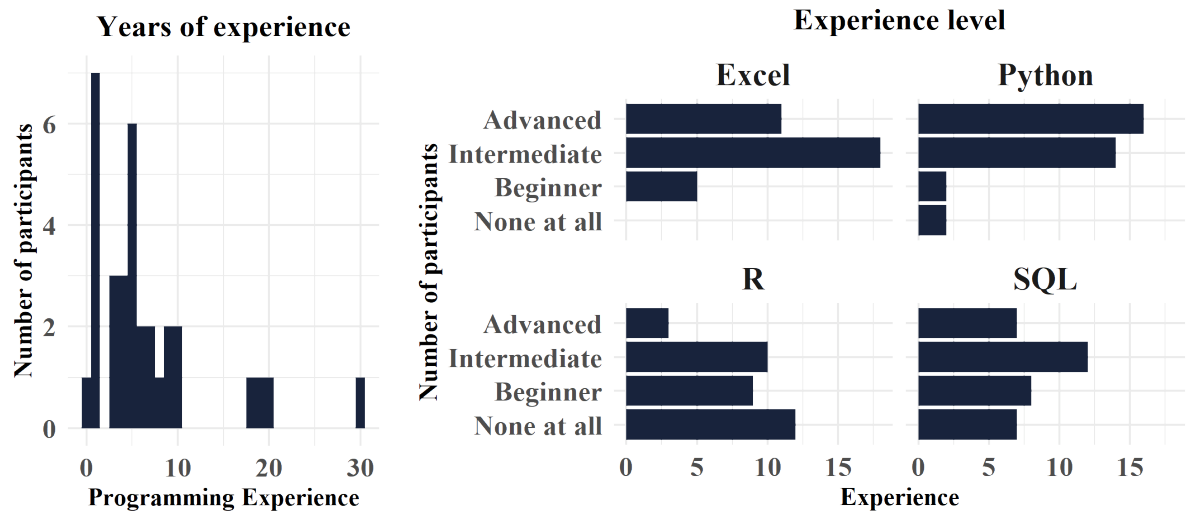


- Shuffling a vector
- Reversing the order in a vector
- Sorting a vector
- Duplicating a vector

(a) An example of a **matching problem**, which asks of the participant to match three graphics with three operations.

(b) An example of an **interpretation problem**, which asks of the participant to select one of four operations as the graphic's intended meaning.

Figure 9.5: The two types of problems contained in the survey.



(a) Years of programming experience.

(b) Self-rated experience in specific programming languages.

Figure 9.6: The prior experience of survey participants.

Procedure & Recruitment

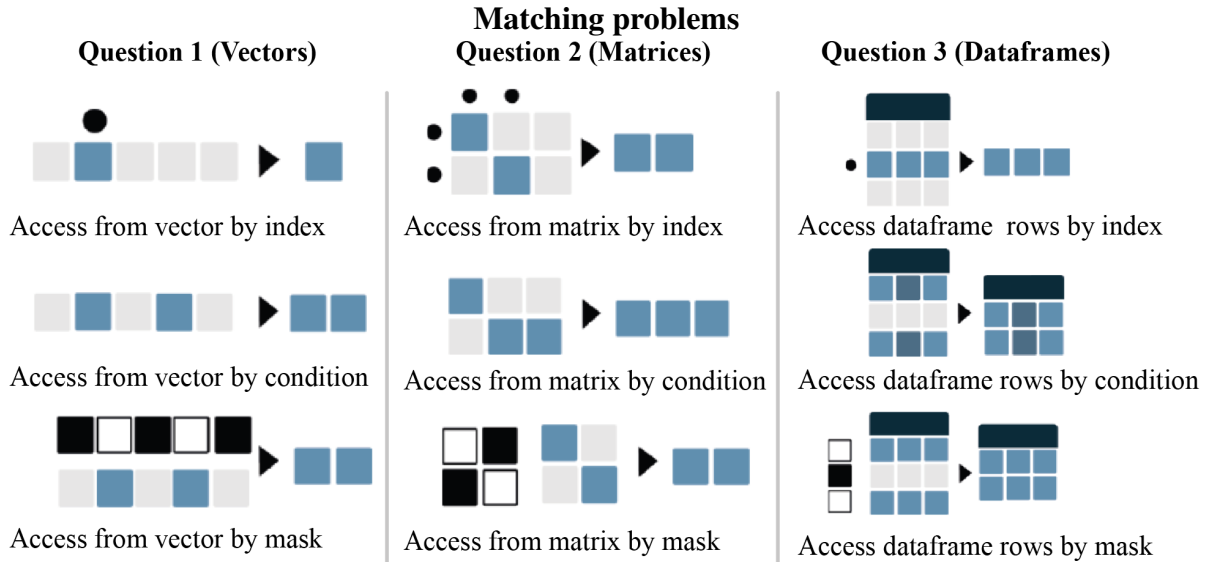
A convenience sample of acquaintances were recruited ($n=35$), from friend networks in Egypt, Finland, Sweden and United Kingdom. Participation was anonymous and remote, via Google Forms. Participants did not receive monetary compensation. The survey had received ethical approval from our local ethics board.

9.2.2 Results

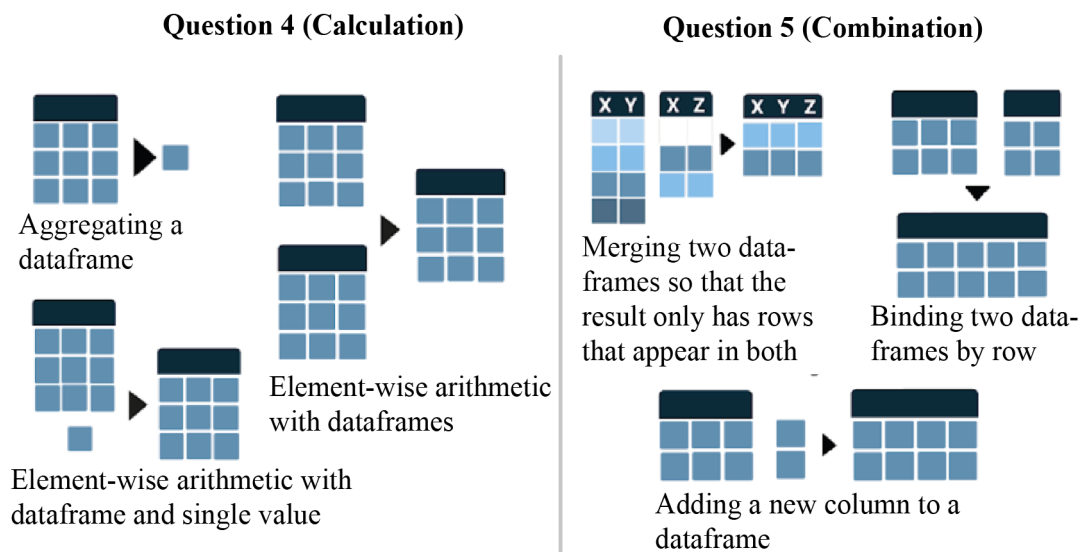
Among the participants, 19 were female and 15 were male (1 was of unspecified gender). Importantly, the participants were *not* complete beginners: the median years of programming experience was 5 (see Figure 9.6a for the distribution). They were especially experienced with general Python, but generally also had intermediate Excel, R, and SQL experience (Figure 9.6b).

Matching problems

The matching problems are shown in Figure 9.7. The operation-to-graphic mappings given by participants were overwhelmingly correct. For Q1-Q3, which concerned methods for accessing values, 32 responses (out of 35) were completely accurate. For Q4 (about distinguishing aggregations from element-wise arithmetic) and Q5 (distinguishing merging from concatenation), 30 were completely accurate. We therefore find that, when juxtaposed with each other, these thumbnail graphics were straightforward to interpret.



(a) The first three questions, which all concerned *Access* operations.



(b) The last two exercises, concerning calculation and combination, respectively.

Figure 9.7: The correct operation-graphic mappings for the five matching problems. The order of options was random and operations verbally formulated to be as self-explanatory as possible.

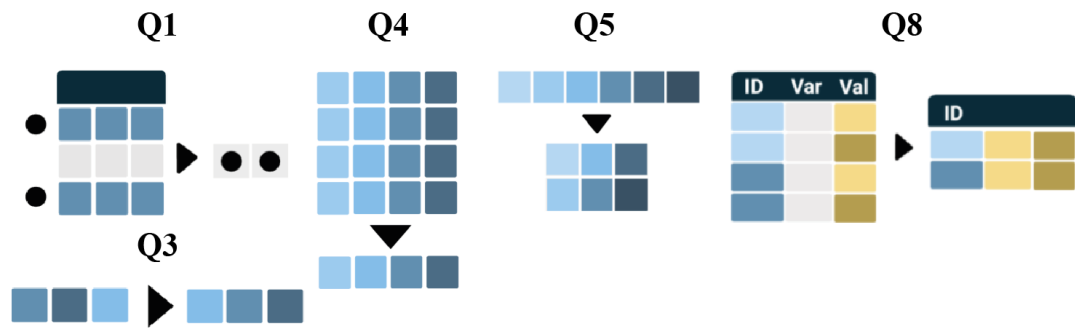


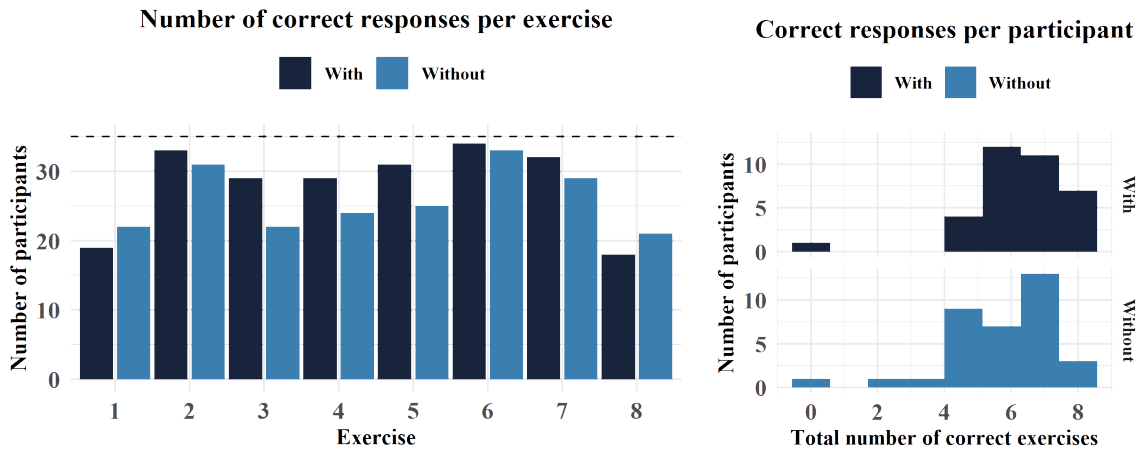
Figure 9.8: The graphics that were most commonly misinterpreted.

Interpretation problems

Recall that we had 8 graphics tested using interpretation problems, first without data and then with data. Each problem had 4 options, one of which was correct. Looking at the proportion of correct responses broken down by exercise (see the light blue bars in Figure 9.9a), we find that it ranges from 21 to 33 (out of 35), with a mean of 25.9. This means that between 60% and 94% of responses were correct. The graphics with the lowest accuracy were the following (see the graphics in Figure 9.8):

- Q1:** 22 correctly interpreted it as *Access indices from a dataframe based on a condition*, but 7 participants interpreted the graphic to mean *Access rows from a dataframe by index*. This appears to be a failure to notice the black circles, representing indices in the post-operation state.
- Q3:** 22 interpreted it correctly as *Sorting a vector*, but 10 misinterpreted it as *Shuffling a vector*. This appears to be a failure to attend to the shade of blue, which is incrementally darker in the post-state.
- Q4:** 24 correctly interpreted it as *Aggregating each column in a matrix* while 5 misinterpreted it as *Aggregating each row in a matrix*. This could be due to a misunderstanding of “aggregating each row” and meaning “producing a row of column aggregates”.
- Q5:** 25 correctly interpreted it as *Folding a vector into a matrix column-by-column*, but 4 misinterpreted it as being row-by-row. This again seems to be a failure to notice the systematic shade, which here is used to indicate correspondence of values from pre- to post-state.
- Q8:** 21 correctly interpreted it as *Pivoting a dataframe so that each unique ID has its own row* (long-to-wide) while 7 misinterpreted it as *Aggregating dataframe rows group-wise*. This could be due to confusion over what *Var* contains, and participants over-relying on the heuristic that a smaller post-operation state implies an aggregation.

In terms of the influence of introducing data to the graphics, we find that the median number of correct responses per participant shifts from 6 to 7 as participants receive graphics with data



(a) The number of correct responses in total, per graphical type, out of 35. (b) The total score per participant and graphical type.

Figure 9.9: The performance in interpretation problems, grouped by graphical type. *With* means with data, *Without* means without data.

(see Figure 9.9b). When broken down by exercise (Figure 9.9a), we find that adding data to the graphics improved response accuracy for all exercises except Q1 and Q8. In mean terms, it increased the number of correct responses by 2.25 points.

Variant exploration

In presenting the participant with four different graphical styles, we had expected that they preferred the minimalist Image 4 for the purpose of a menu thumbnail, and the more informative Image 3 for the purpose of a textbook illustration. The survey item allowed participants to select multiple versions, so the number of responses exceeded 35.

The results are shown in Figure 9.11. They show, interestingly, that participants prefer Image 3 in *both* contexts. It is worth observing that, for textbook illustration purposes, Image 3 is clearly dominant, and the format most typically seen - Image 1 - is the *least* popular. This suggests that textbook and documentation authors can make their materials much more appealing by changing the visual appearance of their examples. We find the thumbnail preferences surprising, however: Image 1 - what we had considered the least suitable design for thumbnails - is second to Image 3 in popularity, albeit with the simple Image 4 as a close third.

Ratings

The final section asked participants directly about how “clear” they would rate the two types of graphics (with or without data) to be. It is obvious from the response distributions of Figure 9.10 that participants found the graphics *with* data to be clearer, as the mode rating shifts from 3 to 5. 24 out of 35 gave the graphics with data a rating of 5.

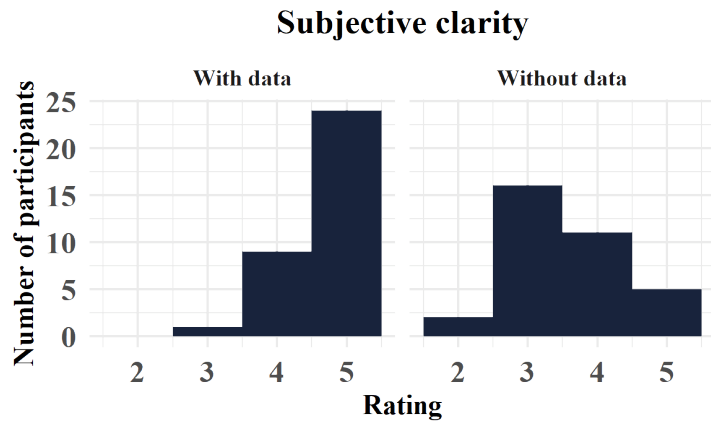


Figure 9.10: The response distributions on a Likert scale of 1 (*Not helpful at all*) to 5 (*Very helpful*) asking them how clear they thought the graphics were, with and without data in them.

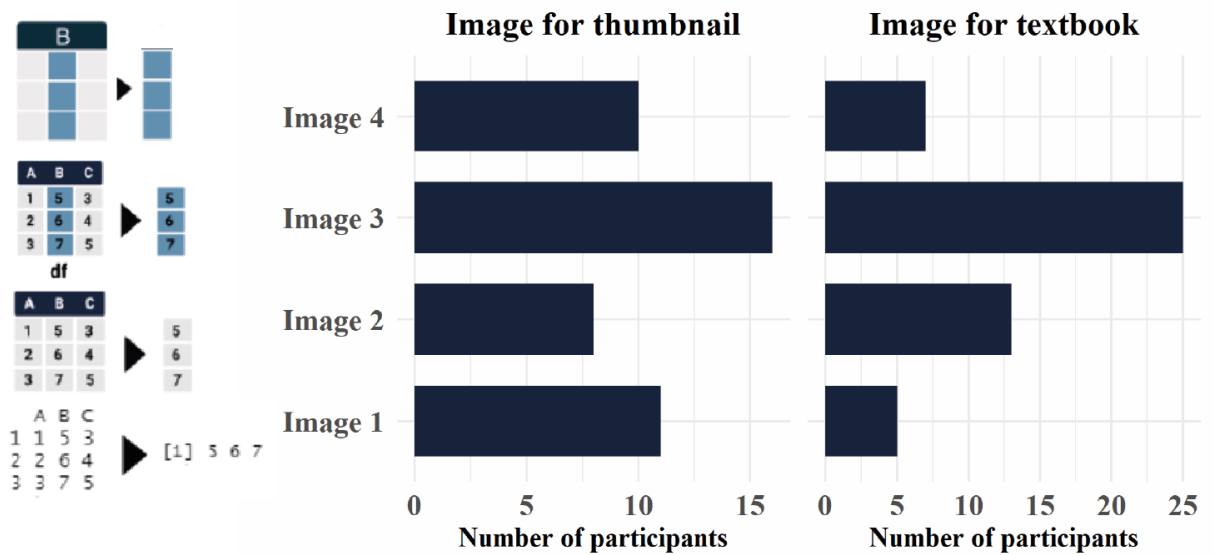


Figure 9.11: Which of 4 variants that the participant prefers in a textbook explanation and thumbnail graphic.

9.2.3 Discussion

The primary objective of this validation study was to gauge whether the thumbnail graphics were clear enough for the behaviour of represented operation to be easily interpreted. We know that indexing methods, calculation operations and combination operations are trivially easy to identify in the context of matching problems. We also know that, even with three carefully selected distractors, at least 60% always chose the correct interpretation for thumbnail graphics in interpretation problems. We would expect this clarity to be even greater in the context of SLICE N DICE, as that would afford a longer and more immersive engagement with the graphical system, compared with a 15-minute survey. Overall, we are satisfied that the thumbnail graphics are fit for purpose.

With or without data?

As a secondary research question, we were interested in whether interpretations become more successful when the graphics have data. When exercises are considered on an individual basis, the proportion of correct responses generally increases when data are added to the graphics. We also find this to translate into a slight improvement in the number of correct responses per participant. Given that the order in which graphics with or without data were presented was not counter-balanced, it is important to remember that this improved accuracy could have been an order effect: by the time they see the graphics for the second round, they could simply have become more comfortable interpreting them.

There were also two sources of subjective data: the variant preferences and the final clarity ratings. The variant preference data reveal that, for both thumbnails and textbook illustrations, people prefer graphics with colour highlights and with data. From the ratings it is furthermore evident that the graphics with data are subjectively clearer. This is not a trivial result: populating graphics with data adds potentially distracting details, but also more information for resolving ambiguity. We should note, however, that the ratings difference could also have been influenced by order effects: the choice to present the graphics with data last could have influenced them to think of them as a corrective.

Overall, we view these results as supporting our decision to augment thumbnails with tooltips: despite their simplicity, it is clear that participants wish to have tooltips available for clarification. This makes sense if we remember that the highlights are still available and could be relied upon, and the data could be ignored if necessary, but that their presence offers more information if the highlights proved insufficient.

Threats to validity

This validation study was not without weaknesses. For one, we did not test their understanding of operations independently from their association with graphics. It is possible that they did

not fully comprehend the notion of an *aggregation*, and that they relied on a more superficial comparison with the examples in the background section. Another weakness is the lack of formal equating of the difficulty posed by the distractor set in interpretation problems. It is possible that some items had much more dismissible distractors. Arguably, the most concerning weakness is that of the participants' prior experience: although 8 (out of 35) had programmed for at most a year, the majority were seasoned programmers, who likely already had a well-organised mental model of the available operations. This is likely to have made interpreting the graphics easier, and therefore a sample of novices would probably have been less accurate in their responses.

9.3 Summary

- In a survey study validating subgoal graphics, a sample of 15 programming novices were asked to provide their own interpretations of a sequence of subgoal labels. Response ratings indicate that their interpretation accuracy was generally adequate, but also observed several recurring sources of confusion, such as overly literal interpretations and misinterpretations of colour. We believe these errors are less likely in the context of SLICE N DICE.
- Another study validated thumbnail graphics by asking 35 people of varying programming experience to interpret thumbnail graphics without and with data. These responses were given through multiple choice items. The results indicate that participants tended to choose the correct interpretation, and that accuracy increased somewhat once populated with data. Subjectively, participants appear to prefer graphics with data for both thumbnails and textbook illustrations.

Chapter 10

Slice N Dice results

In Chapters 4, 6 and 7 we documented the groundwork of a large, summative capstone study addressing our two main research questions: what is the pedagogical effect of subgoal graphics (RQ1) and thumbnail graphics (RQ2)? In this chapter we describe the procedure used when administering SLICE N DICE to an international, multi-institutional sample of almost 300 participants, and present the results comparing the control group with the graphical group on a variety of pedagogically relevant metrics. The subgoal-related results [316] and thumbnail-related results [317] have been accepted for separate publications.

10.1 Method

10.1.1 Design

Recall that SLICE N DICE incorporates all the components of an RCT: participants are randomly assigned to one of 2x2 conditions, which persist throughout and constitute between-subjects variables. As dependent variables, a plethora of behavioural data are collected under the hood, along with self-reported survey data.

Independent variables

The two main independent variables are whether subgoal graphics are provided (**SG**, **¬SG**) and whether thumbnail graphics are provided (**TG**, **¬TG**). As noted in Chapter 7, these manipulations mostly imply a comparison between the presence and absence of graphics, thus effectively measuring the “added value” provided by graphics, which conceivably could be both redundant or counter-productive. There were two minor departures from the clean presence/absence divide: the thumbnail manipulation also changed the appearance of Part 1’s operation cards and the menu’s tooltips, which in TG featured graphics, and in ¬TG featured a shell-like representation of how a particular data operation works. Thus, the TG condition can be seen as a more consistently graphical menu experience.

Dependent variables

In Chapter 4 we discussed our intentions to capture data for a wide set of dependent variables. These dependent variables could be divided into two sets:

- **Performance metrics:** These are generally *process metrics*, reflecting the learning process as it unfolds [271]. We have noted previously that SLICE N DICE does not follow a post-test design, but rather focuses on measuring the speed and efficiency of the learning process. The data collected two variables that, as a matter of face validity, measure performance: **time on task** (i.e. the time taken to finish all exercises) and **number of incorrect attempts** (i.e. the attempts taken before getting the exercise solution correct).
- **Mediating metrics:** In addition to performance, a number of other metrics were collected that could not be said to capture data wrangling performance as a construct in its entirety. For example, the fewer times a user solicits a tooltip for clarification, the better their understanding of data operations could be assumed to be. However, an understanding of data operations is not the only factor that determines the ability to write accurate data wrangling programs. These metrics could be said to be mediating variables: themselves conceivably influenced by the graphics, but in turn also influencing the programming outcome.

The metrics are listed in Table 10.1. Except for the number of exercises completed, a smaller number represents a desirable outcome: the less time on task, the better; the fewer tooltip events, the better. Insofar as we expect graphics to have a broadly positive impact on motivation, API lookup efficiency, and overall performance, we hypothesise graphics to positively impact the corresponding metrics. Throughout our presentation of results in Sections 10.2 and 10.3, we will make plain the auxiliary assumptions of why we interpret a metric a particular way, and predict a particular result pattern. However, because our hypotheses are not continuous with a prior research programme of replications, few of them rest on a solid bedrock of prior estimates: they are best characterised as educated hunches. The analysis therefore resides in the middle of the confirmatory-exploratory continuum.

10.1.2 Procedure

The study was completed online and remotely. It happened during COVID-19, and as a result most participants presumably participated from their home. Since random assignment was automated and participants took part in the study on their own, the study is double-blinded.

Upon signing up, participants are automatically allocated to one of the 2x2 cells, using alternating assignment. Assignment can therefore be characterised as pseudo-random: by using alternating assignment, distinct recruitment groups are near-guaranteed to be equally distributed in the conditions. Since participants could drop out after signing up, sample sizes were not perfectly equal per cell.

Behavioural dependent variables analysed			
	Dependent variable	Hypothesis	Tested for subgoals
Part 1	Number of completed exercises	$G > \neg G$	Y
	Number of inactivity events	$G < \neg G$	Y
	Operation card reading times	$G < \neg G$	-
	Number of tooltip events	$G < \neg G$	-
	Time on task	$G < \neg G$	Y
	Number of attempts	$G < \neg G$	Y
Part 3	Number of tab changes	$G < \neg G$	Y
	Number of timeouts	$G < \neg G$	Y
	Number of exercises completed	$G > \neg G$	Y
	Number of tooltip events	$G < \neg G$	-
	Number of menu click events	$G < \neg G$	-
	Number of hints used	$G < \neg G$	Y
	Number of syntax errors	$G < \neg G$	Y
	Number of semantic errors	$G < \neg G$	Y
	Time on task	$G < \neg G$	Y
Un scaffolded	$G > \neg G$	Y	

Table 10.1: The dependent variables analysed in terms of group comparisons. G and $\neg G$ represent the central tendencies of the graphical and non-graphical control condition, respectively. The hypothesis column thus indicates which direction we expect to see in the effect, i.e. whether the graphical condition is larger or smaller than the control condition. Some variables, labelled -, are not compared based on subgoal graphic condition. Variables in **bold** are performance metrics. This table does not include self-reported variables.

All lecturers received materials with the correct responses to each question, so that they could help answer questions if students became frustrated. As a result, it had an aspect of blended learning to it, though anecdotally we saw few requests for such assistance. The participation was in all cases at least partially synchronous, leaving open the possibility that students coordinated with each other.

10.1.3 Participants

Sample size

Our aspiration was to obtain at least 200 complete observations at the beginning of the study. A sample size could be justified on many different grounds [318]. In justifying ours, we are influenced by CS educational norms, where the median sample size has been estimated to be 100 [271]. In practice, we are heavily constrained by available resources: while the administration of the experiment was easily scalable, but recruitment of participants was not.

In terms of empirical benchmarks for deciding expected effect sizes, there are few good options available. To the extent that effect sizes across heterogeneous studies can be aggregated, meta-reviews suggest that rigorous educational RCTs have on average a minuscule size of 0.06 *SD* [276], that intelligent programming tutor software has on average a “moderate” Hedges’ *g* of 0.46 [319]¹, while e-learning multimedia interventions on average have a “large” Cohen’s *d* of 1.67 [179]. As mentioned in Section 4.3.10, effect sizes are highly dependent on methodology, and unless the study is a replication², its expected effect size is purely speculative.

What is the smallest effect size of interest? Our inferential goal is to determine whether graphics impart added value beyond what the fixed scaffolding features (e.g. the menu and subgoal labels) already achieve. As such, both large and very small effects are of interest, since the low costs of creating graphics means even slight effects make them cost-effective.

Recruitment

To reach an adequate sample size, we approached lecturers from all across University of Glasgow and neighbouring universities to propose integrating SLICE N DICE into their teaching. Uptake was low, in part because the timing was inopportune, given that COVID-19 had already burdened many lecturers with the need to quickly move their teaching online. This led me to broaden my search for collaborators to universities outside of UK. In the end, four courses integrated the app in some measure, representing three different institutions from as many continents:

¹Hedges’ *g* corrects for an upward bias of Cohen’s *d* in small samples.

²On account of their imprecision, pilot studies are generally not considered a meaningful basis for effect sizes [318].

University of Glasgow : our local institution is a public research university in Scotland, UK.

It has a course called *Science Skills* aimed at introducing aspects of scientific practice to freshmen, and is open to students of any major. The course head agreed to dedicate one week's practical laboratory to SLICE N DICE. The laboratory was preceded by a guest lecture on data wrangling (delivered by me) which introduced the data structures but touched upon programming only briefly. The laboratories took place over Zoom, and I was present to answer questions. Students were free to continue in their spare time.

Namal Institute : this is a private university in Punjab, Pakistan. A lecturer agreed to introduce SLICE N DICE to two of his classes. One module, called *Introduction to Quantitative Reasoning*, mostly included non-majors who had already some procedural Python experience by the time they were recruited. Another class was a sophomore CS class in which students already had one year of Python programming experience, but who were new to data wrangling. In both cases, SLICE N DICE was marketed as an online "hackathon" event, where students were encouraged to complete it over a weekend. In addition to the cheat sheet e-book, participants also received a certificate as a token of gratitude.

University of Nebraska Omaha : this is a public research university in Nebraska, US. A lecturer in charge of a small module in introductory data science agreed to include it as coursework. The module was offered to teachers, some of whom had previously taught mathematics or computer science on a primary or secondary level.

Performance data from SLICE N DICE were not shared with the lecturers or factored into course grades. As previously noted, SLICE N DICE supports both Python and R in order to widen its appeal. However, most of the instructors who integrated it preferred students to do it in Python. As Python emerged as the majority choice, I chose to emphasise Python in later marketing and recruitment efforts, so as to avoid an equal language split, which would have increased the homogeneity in the sample.

In addition, SLICE N DICE was advertised to a large number of courses, via email campaigns, in-class marketing, or website notices. At University of Glasgow, it was advertised to first-year CS1 students, psychology students, a digital humanities course, and first-year students in a software-engineering apprenticeship course. It was furthermore added to `MOOC.fi`, a free MOOC platform overseen by University of Helsinki in Finland, and `dshub.ml`, an online data science network in Egypt. In all of these cases, participation was asynchronous and without any contact with me or a course leader.

Exclusion criteria

We had an inclusive approach towards admitting participants into the study and retaining them for the analysis. We assumed that anyone with solid programmatic data wrangling experience

would drop out of their own accord, given the multi-hour duration and lack of monetary incentives. We furthermore expected the tutorial generally (and the graphics specifically) to be beneficial for programming novices, data wrangling novices, and those inexperienced in both. All observations that were complete for either Part 1 or Part 3 were therefore included in our primary analysis.

Validity concerns

Due to the change in recruitment focus, and lack of financial compensation, we do not expect double recruitment of participants from the pilot studies to pose a risk. As noted in Chapter 4, we elected to make the study voluntary and non-credit bearing, due to a mixture of ethical and pragmatic concerns. There is consequently a risk for self-selection bias, particularly in who completes it. Such potential biases will be addressed in Chapter 11.

10.1.4 Analytical approach

Being an RCT, we are primarily interested in group differences between the graphical conditions and non-graphical conditions. Our analysis therefore takes the form of a series of significance tests comparing the groups for each independent and dependent variable in isolation. We will also conduct a few interaction analyses on the key process variables (time on task and the number of incorrect attempts) to see whether the provision of both graphics act synergistically, since it would allow them to employ a matching strategy of first looking at the subgoal graphic and then finding the matching thumbnail graphic.

Why the use of significance tests? We are aware of debates within the meta-science community regarding when hypothesis testing is appropriate, and not just an empty ritual. For example, Scheel et al. [320] have argued that confirmatory testing should be reserved for more mature theories with validated measurements. Since our focus is on design evaluation rather than theory development (see Section 4.1), this study is best characterised as an exploratory experiment of various pedagogical effects. However, given our wish to causally attribute any observed differences to graphics, we will still report statistical significance.

Statistical techniques

Group differences are usually analysed using a two-sample *t*-test. However, we had reason to believe the assumptions of such a test would be violated, in particular the assumption of normally distributed residuals. We anticipated that our dependent variables would exhibit mostly non-normal, positively skewed distributions, for several reasons. They are the most commonly encountered distribution in cognition-related metrics [321], they are pervasive in time on task estimates especially [322], and in count data when there is a lower bound. There was also a risk

that the heterogeneous sample led to a mixture of distributions in the sample. Irrespective of cause, this calls the appropriateness of a t -test into question.

Several approaches exist for comparing non-normally distributed groups. The first is to apply a non-linear transformation to the data, which has the drawback of obfuscating the findings' interpretation [323]. A second approach is to replace or trim outliers, which carries the risk of biasing the estimate [323]. A third approach is to use linear models despite violating the normality assumption, since simulation studies suggest such models can be surprisingly robust if the sample size is large [324]. A fourth approach is to use non-parametric methods, which could be rank-based or permutation-based. Where non-normal, we therefore opted for a non-parametric approach, using the following two statistical tests:

Wilcoxon rank sum test: Also known as the Mann-Whitney U test, this tests the null hypothesis that the distributions are equal, such that there is a 50% probability that a randomly sampled value from one group (or rather, its population) exceeds a randomly sampled value from the other group. All values are rank-transformed from low to high and these ranks are then summed for one group. This rank sum is used to produce a test statistic called W . A significant Wilcoxon test could be interpreted as meaning that there is a *location shift* where one distribution is shifted to the side of the other. This location shift - the effect size - is known as the Hodges-Lehmann (HL) estimator. Using the implementation of R's *stats* package, this means that the median of the difference between a sample from one group versus the other is significant.

Permutation test: Under the null hypothesis, it should not matter whether a participant is in the graphical condition or the control condition. Therefore, if we repeatedly take the raw data column and randomly reassign data points to one of the two groups (without replacement), and calculate the t -statistic for each iteration, these t -statistics form an empirical sampling distribution. If the observed t -value (of the original assignment) falls outside the middle 95% of this distribution, the test is significant at the .05 level. The implementation we use is the R package *lmPerm* (version 2.1.0) [325]. Though permutation tests are sometimes preferred since they do not discard raw data to the same extent, simulation studies generally indicate that Wilcoxon tests have higher statistical power than permutation tests [326,327].

Due to the non-normality, we will report robust descriptive statistics like medians and inter-quartile ranges instead of means and standard deviations. As advised by Robertson [1], standardised effect sizes will be computed using the non-parametric *probability of superiority* (PS), which represents the probability of a randomly sampled member in group A having a higher value than a randomly sampled member from group B. As inferential criteria we will apply two-tailed analyses, to allow for effects of unexpected directions to be detected as well.

Choice of α -level

As a matter of convention, we will test our hypotheses at a an α -level of .05. We are aware of ongoing meta-science debates on whether this threshold should be universally lowered or justified based on a cost-benefit analysis [328], but will for now adhere to established protocols.

Since we will test a large number of dependent variables, this will inflate the so-called *family-wise error rate*, which giving the appearance of a fishing expedition. As a result, researchers are widely encouraged to lower the nominal α -threshold, using for example Bonferroni-adjustment, which would imply an α -level of $.05/32$ ($\approx .0016$), since there are 32 implied hypotheses for the behavioural metrics (see Table 10.1). This would severely reduce the statistical power. What constitutes best practices is very much a delicate philosophical matter: some argue that significance tests are meaningless in exploratory analyses, while others argue that alpha-corrections should only be done for metrics that test the same hypothesis [329]. Since we consider each metric as its own hypothesis, we will report the outcomes in terms of an α of .05. Perhaps most important is avoiding outcome-reporting bias and ensuring that the results of all tests are reported, even if they are non-significant [330], and to supplement significance tests with more nuanced interpretations based on visual inspection.

Pre-registration

The hypotheses for performance variables (time on task, number of attempts, number of exercises completed) were pre-registered on the Open Science Framework [331]. In this registration, which occurred at an early stage of the data collection, only subgoal-related hypotheses were listed. This was because it was completed in anticipation of a subgoal-focused paper. Our terminology has since been revised: in the pre-registration, the SG (subgoal graphics) condition is referred to as GS (graphical subgoals). The hypotheses, listed in the order presented in pre-registration, map onto Sections 10.3.4, , 10.3.10, 10.3.7/10.3.8, 10.2.5, 10.2.6 and 10.2.1/10.3.1, respectively.

We also note a number of deviations from the planned analytical protocol, brought about by changes in our reasoning regarding best practices. The first deviation involves outlier exclusion policy, which the pre-registration indicated would be based on Z-scores, but which in our analysis is based on visual inspection and assumptions about the data-generating process. A second deviation regards a stated plan to normalise time on task data, where we instead employed non-parametric methods without any distributional transformations. A third deviation regards our plan to only use complete data, and explore data imputation for missing exercises: for Part 3 we opted instead to ignore the last 8 exercises to boost sample size. Our data is therefore complete with respect to 10 exercises, not 18. A fourth deviation was a plan to use "number of attempts per subgoal" as the outcome metric: we instead calculated the total number of attempts across the exercises. Finally, results will be presented with an unadjusted α -level and the performance in the unscaffolded condition was modelled as an outcome variable rather than a within-subjects

variable.

10.2 Part 1

Part 1 consists of a deck of operation cards, interspersed with 9 non-programmatic exercises where the user is tasked with mapping a given subgoal with an operation from the menu.

10.2.1 Number of exercises completed

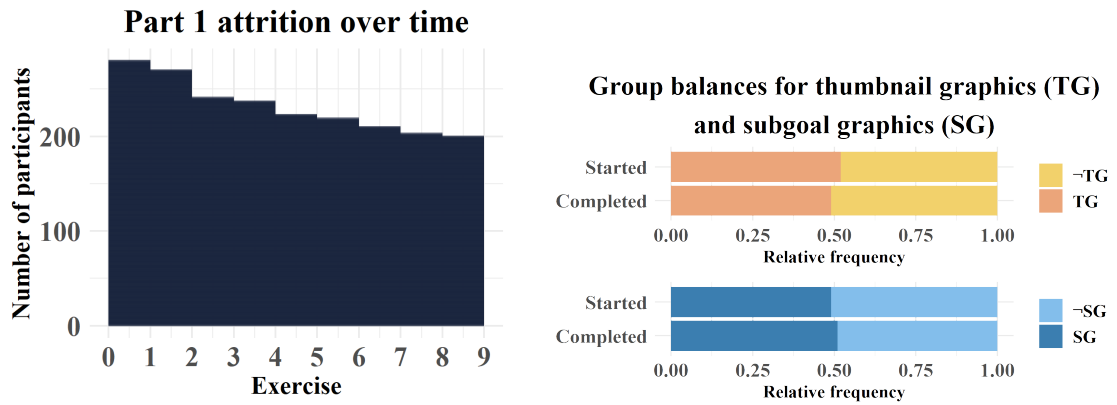
The number of completed exercises can be interpreted as a proxy of learner motivation. Since Part 1 is expected to take less than 1 hour, it is presumably not very vulnerable to fatigue- or boredom-induced boredom. However, it is also non-programmatic, which could reduce the perceived relevance of the task, and render it vulnerable to attrition induced by low motivation. If graphics make the task more enjoyable or motivating, we would **expect that participants in graphical conditions complete more exercises**.

Out of 288 unique participants who *engaged* with at least one Part 1 exercise, 197 *completed* all exercises, producing a completion rate of 68%. As shown in Figure 10.1a, the participant number dropped off steadily, with one discernible decline after the first two exercises (the *Create* category). With respect to the experimental manipulations, no condition appears to be over- or underrepresented among participants who completed all 9 exercises, as the final group ratios stayed close to the initial ratios for both independent variables (see Figure 10.1b). Therefore, it does not appear that the provision of thumbnail or subgoal graphics influenced participants' likelihood to complete the exercise set. Both conditions completed a median of 9 exercises and, in mean terms, between 7 and 8 exercises. Therefore there does *not* appear to be a difference in the number of Part 1 exercises completed that are linked to subgoal or thumbnail graphics, and no significance test was therefore conducted.

10.2.2 Inactivity events

Two types of inactivity data were collected over the course of the study. If the SLICE N DICE tab loses focus (i.e. the user goes to either another browser tab, another browser window, or another application), this will be recorded as a tab change, which in the case of Part 1 probably indicates distraction. If the time between any two successive key strokes or mouse movements exceeds 60 seconds, this will be recorded as a time-out event, assumed to reflect inattention. The two will be aggregated as a single category of *inactivity events* (an assumption that will not be repeated for Part 3, where the two are differentiated by how tab events could reflect usage of external documentation sources).

Metrics relating to attention or inactivity have a complex interpretation. As discussed in Section 4.3.5, they are mediating variables: they could both be influenced by the graphics and



(a) Attrition in the number of participants who completed exercise 1-9. (b) Representation of conditions among those who started and finished in Part 1.

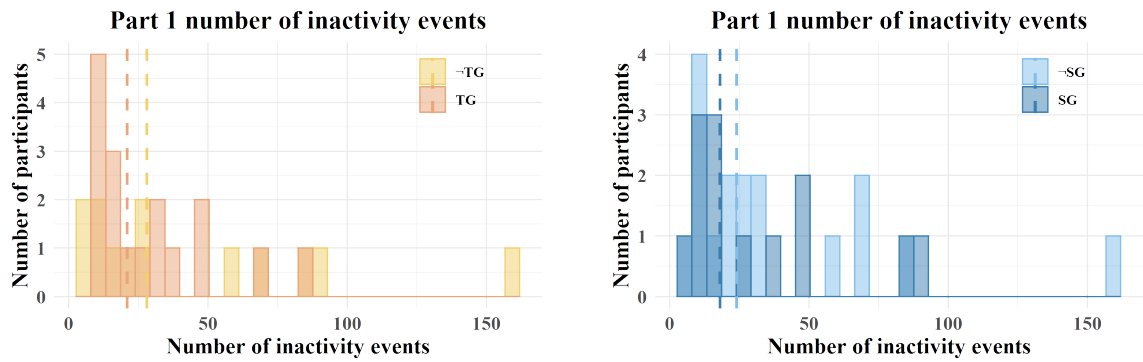
Figure 10.1: Although only 68% of Part 1 participants completed it in full, the ratio between experimental conditions did not appear to change markedly upon completion.

thus influence the outcome. They could also be conceptualised as a moderating co-variate, influenced by external factors such as a participant's ability for sustained attention, or extrinsic disturbances (especially relevant given that it was completed at home). In our case, we conceptualise attention as a dependent variable: based on the assumption that graphics make materials more engaging, **we expect that the graphical conditions will exhibit fewer inactivity events.**

Over the course of Part 1, participants were in general concentrated. Most participants did not record *any* tab event (169 out of 197) or timeout events (167 out of 197). Our assumption - one which will be invoked repeatedly throughout this analysis - is that those that do not register any inactivity events are systematically different. In this case, people who do not register inactivity events may be committed to treating SLICE N DICE as the experiment it is. However, in excluding them from further analysis, we are left with a sample size of 30, which is too small to permit inferential analysis. Due to the paucity of data, we will limit ourselves to descriptive statistics. The marginal distribution has no clear shape except a positive skew (1.7) that a Shapiro-Wilks test confirms is non-normal ($p < .001$). The \neg TG condition (Figure 10.2a) seems too widely dispersed to suggest an effect, albeit with a median difference of 7 events (TG=21, IQR=25; \neg TG=28, IQR=57). With subgoal graphics, whose distribution is shown in Figure 10.2b, there is a shift of 6 events (SG=18, IQR=34; \neg SG=24, IQR=46), but this appears to be caused by an outlier.

Conclusion

With both types of graphics, the graphical condition registers less inactivity, in median terms. However, with so few participants recording inactivity events, we refrain from drawing any strong inferences.



(a) The number of inactivity events, as grouped by thumbnail condition. Lines show medians.

(b) The number of inactivity events, as grouped by subgoal condition. Lines show medians.

Figure 10.2: Tab changes and timeout events were only recorded for 30 participants, which is too few to base inferences on.

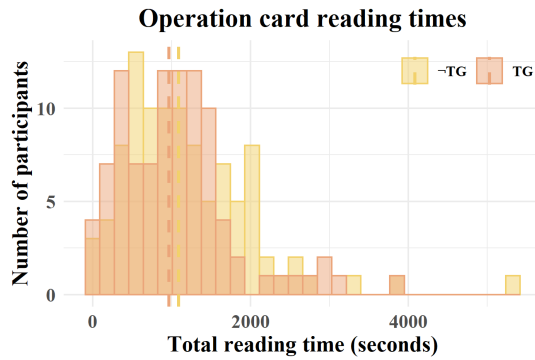
10.2.3 Operation card reading times

Recall that for TG, the operation cards display a graphic in the same style as the thumbnail graphics. The \neg TG group is instead presented with an example showing the state of the data structure before and after the operation, but in a format that resembles printed shell output. If graphics are easier to parse, we would **expect that thumbnail graphics lead to shorter total reading times**. This assumes that the reading time - recorded from the moment of presentation until they click the button to reveal the next card - reflects self-assessed comprehension times.

Following the deduction of time spent inactive, 84 records (out of 13916) were deemed abnormal, due to being greater than 200 seconds in length. These observations were replaced with the participant's median reading time, before being summed into a total reading time. As shown by Figure 10.3a, the distributions are positively skewed. The marginal distribution has a skewness of 1.58 and is significantly different from a normal distribution, as shown by a Shapiro-Wilk test ($W=.89$, $p<.001$), which is why we use non-parametric tests. Visually, the TG distribution *does* appear to be shifted to the left of \neg TG. In median terms, a TG group participant read all cards 119 seconds faster (≈ 2 min). A Wilcoxon rank sum test is non-significant ($W=5366$, $p=.2$) with a 95% confidence interval of $[-66.1, 322.1]$, a location shift of 128.3 and PS of .55. A permutation test is non-significant ($p=.27$).

Conclusion

We find descriptive evidence that the TG group reads the cards faster at a practically meaningful level, but are unable to rule out the null hypothesis.



	<i>n</i>	median	IQR	W	<i>p</i>
TG	97	969	770	5366	.2
-TG	100	1088	997		

(a) The times to read all cards, as grouped by thumbnail condition. Lines show medians.

(b) Descriptive and inferential statistics for reading time (s).

Figure 10.3: The TG group reads all cards 2 minutes faster on average, but this is not significant.

10.2.4 Tooltip events

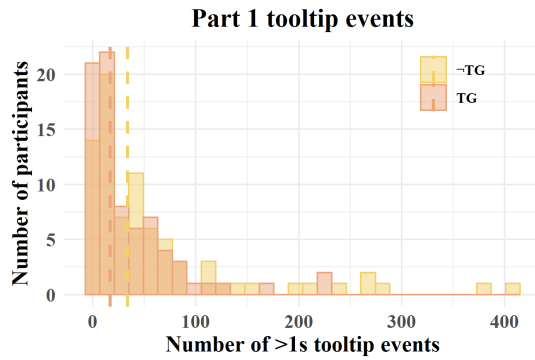
When users hover their mouse over a thumbnail for longer than 1 second (regardless of experimental condition), this triggers a tooltip to be displayed. 1 second was judged long enough to reflect an intention to read the tooltip and therefore a need for clarification. We would expect a more useful thumbnail to lead to fewer tooltip solicitations. In other words, we **hypothesise that thumbnail graphics lead to fewer tooltip events**. We see no strong theoretical grounds for claiming that subgoals would influence this need, and therefore will not test for this.

Out of the 197 participants who completed all exercises, 157 hovered at least once over a thumbnail across the entire session. We assume that the 40 participants who neglected the tooltips did not realise they existed as an affordance (even though they were explicitly mentioned in the interactive tour). We will therefore exclude those participants from the analysis, leaving samples of size 77 (TG) and 80 (-TG). Overall, the remaining participants hovered a median of 25 times (IQR=52). Looking at Figure 10.4a, it is clear that the distribution is highly positively skewed (skewness=2.7). Although there is little visible separation between conditions, TG appears more concentrated at the very bottom in tooltip usage.

In our summary statistics (see Table 10.4b), we find that the -TG group has a higher median, on average needing the tooltip twice as often (34 times versus 17). A Wilcoxon rank sum test results in a location shift of 8, a 95% confidence interval of [1,19] and a PS of .6. It is significant ($W=3707, p=.028$) at the unadjusted α of .05, as is a permutation test ($p<.01$).

Conclusion

We find convincing evidence that thumbnail graphics lead participants to request clarification via the tooltip less often, consistent with our hypothesis.



(a) The number of tooltip events per participant, grouped by thumbnail condition. Lines show medians.

Tooltip event statistics					
	<i>n</i>	median	IQR	W	<i>p</i>
TG	77	17	44	3707	.028*
¬TG	80	34	56		

(b) Descriptive and inferential statistics for tooltip events. * means significant at $\alpha=.05$.

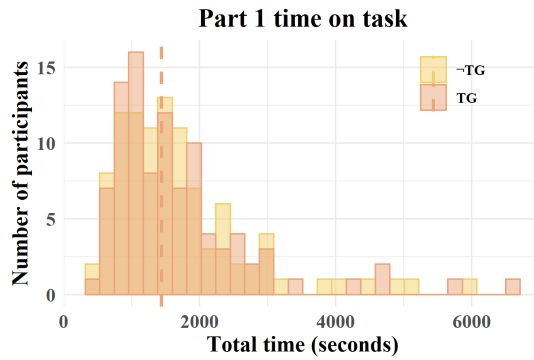
Figure 10.4: The TG group uses the tooltip only half as often as the control group, an effect that is significant.

10.2.5 Time on task

The time on task is measured from the time an exercise was presented until the time all subgoals were correct. The learner is notified when all are correct and allowed to proceed. We distinguish it from *time off task* - time off SLICE N DICE where the participant has either timed out or is in a different tab or non-browser application (see Section 10.2.2), which has already been deducted. Still, any low-stakes setting time on task data are likely to contain time in which the learner is wholly or partially disengaged.

It should be noted that time on task has an ambiguous interpretation: a long time on task could be deliberate (due to fastidiousness) or unavoidable (due to low ability). It could be both positively associated with an outcome, and negatively. According to Goldhammer et al. model for time on task in technology-rich environments, the association's direction depends on whether the task involves automatic or controlled mental processes [332]. SLICE N DICE requires the latter, which Goldhammer predicts has a *positive* association with ability. However, although time on task in both Part 1 and 3 is spontaneous (i.e. freely chosen by the learner, due to the lack of imposed time limit), it also has a definite end-point: the participant is informed if their solution is correct. As a result, there is little to be gained from a cautiously controlled problem-solving strategy. Moreover, since time on task in our case is summed across multiple exercises, if early meticulousness leads to speeds gains in later exercises, that should already have been accounted for. Overall we assume that, all else being equal, the more exercises a person completes per unit of time, the faster their performance.

As a measure, time on task encompasses a multi-stage process: the time taken to become familiarise oneself with the exercise, process the materials, produce a solution, and author the solution. It also includes recurring cycles of API lookup, example adaptation, and solution testing. Assuming thumbnail graphics make API lookup faster, and subgoal graphics make



Part 1 time on task statistics

	<i>n</i>	median	IQR	W	<i>p</i>
TG	97	1438	967	4890	.92
-TG	100	1432	933		

(a) The time on task per participant, grouped by thumbnail condition. Lines show medians.

(b) Descriptive and inferential statistics for time on task.

Figure 10.5: The difference in time between the two thumbnail conditions is negligible.

subgoal comprehension faster, we would **expect that graphics lead to shorter time on task**.

The distribution of time on task follows a positively skewed distribution (skewness=2.06), consistent with response time patterns [322], see Figure 10.5a. It is significantly different from a normal distribution, as determined by a Shapiro-Wilk test ($W=.81, p<.001$). In median terms, the participants took approximately 24 minutes to complete all 9 exercises, with an inter-quartile range of 964.1 (this does not include the reading of the operation cards). No observations were excluded for being outliers.

The effect of thumbnail graphics

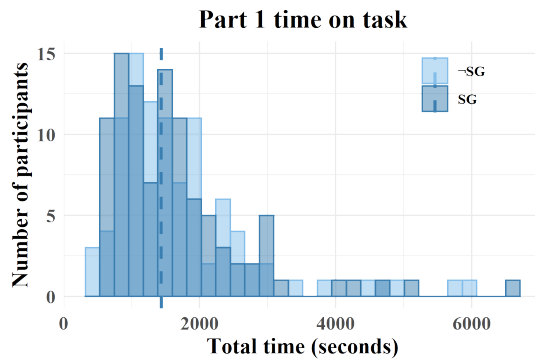
The histograms of Figure 10.5a suggest that the difference in the time taken between thumbnail groups is negligible: specifically, thumbnail graphic group was slower by 6 seconds (1438 vs. 1432 seconds). It was not significant for a Wilcoxon test ($W=4890, p=.92$) or a permutation test ($p>.99$).

The effect of subgoal graphics

With regards to subgoal graphics, the group medians are near-identical, although the SG data are now spread more widely: the SG group took 6 seconds *longer* time to finish the 8 exercises (1438 versus 1432 seconds), which was not significant ($W=5110, p=.52$). In the histograms of Figure 10.6a, we find no separation. The results, summarised in Table 10.6b, suggest that the null hypothesis cannot be rejected.

Interaction analysis

We may also investigate whether the two variables interact with each other, such that the four different cells vary meaningfully from each other. Boxplots are shown in Figure 10.7a. A permutation-based ANOVA found no significant interaction ($p=.72$).



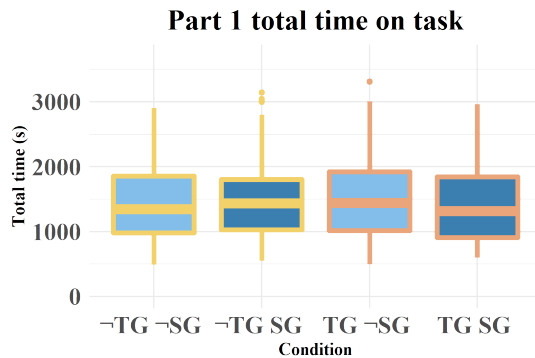
(a) The time on task per participant, grouped by subgoal graphics condition. Lines show medians.

Part 1 time on task statistics

	<i>n</i>	median	IQR	W	<i>p</i>
SG	100	1432	1007	5110	.52
-SG	97	1438	940		

(b) Descriptive and inferential statistics for time on task.

Figure 10.6: The difference in time between the two subgoal conditions is negligible.



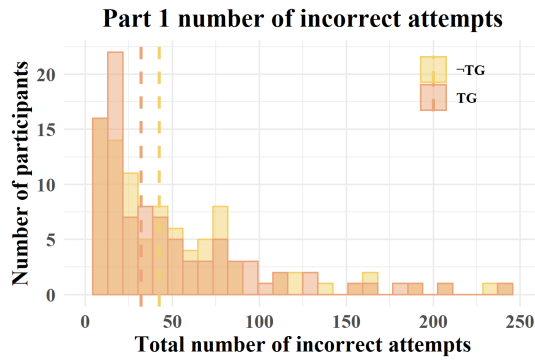
(a) Boxplots showing how the total time on task varies depending on condition

Part 1 total time statistics

Condition	<i>n</i>	median (s)	IQR	
TG	SG	48	1387	1041
	-SG	49	1454	940
-TG	SG	52	1438	947
	-SG	48	1430	890

(b) Descriptive statistics for the total time on task.

Figure 10.7: Data relating to a potential interaction in Part 1 time on task.



Part 1 number of attempts statistics

	<i>n</i>	median	IQR	W	<i>p</i>
TG	97	32	59	5231	.34
-TG	100	42.5	58		

(a) The number of attempts per participant, grouped by thumbnail condition. Lines show medians.

(b) Descriptive and inferential statistics for the number of incorrect attempts.

Figure 10.8: The -TG group commits 33% more errors, but it is not significant.

Conclusion

Neither thumbnail graphics nor subgoal graphics appear to measurably impact time on task.

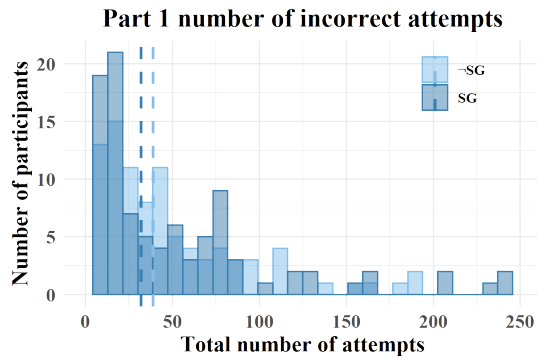
10.2.6 Number of attempts

To measure performance, the total number of *incorrect* attempts (i.e. the act of dragging and dropping an operation into a subgoal slot) were summed together for all exercises. We would expect overall attempt numbers to be low, since the first failed attempt per subgoal will always trigger a hint to display. We assume that the fewer errors, the better, and the more likely a participant is to possess an accurate understanding of which operation a subgoal corresponds to. If graphics influence this understanding, we would **expect the graphical conditions to record fewer incorrect attempts**.

Only participants who completed all exercises were included ($n=197$). The marginal distribution is, consistent with expectations, positively skewed (skewness=2.74). The median number of incorrect attempts is 35 (IQR=58).

The effect of thumbnails

With thumbnail graphics, the experimental group commits fewer incorrect attempts in median terms (32 versus 42.5). Although -TG commits 33% more errors with a location shift of 4 (PS=54), this is not significant, based on a Wilcoxon test ($W=5231, p=0.34$), whose 95% confidence interval was [-4,13], and a permutation test ($p=.9$). Therefore, although the direction is in line with the alternative hypotheses, the null hypotheses cannot be rejected. Results are summarised in Table 10.8b.



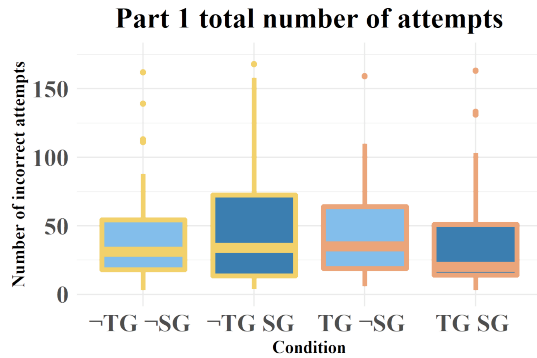
(a) The number of attempts per participant, grouped by subgoal condition. Lines show medians.

Part 1 number of attempts statistics

	<i>n</i>	median	IQR	W	<i>p</i>
SG	100	32	60	5286	.28
¬SG	97	39	55		

(b) Descriptive and inferential statistics for the number of incorrect attempts.

Figure 10.9: The ¬SG group commits 7 more errors, but it is not significant.



(a) Boxplots showing how the number of semantic errors varies depending on condition

Part 1 number of attempts statistics

Condition	n	median	IQR
TG SG	48	23.5	59
TG ¬SG	49	38	55
¬TG SG	52	43	59
¬TG ¬SG	48	42.5	56

(b) Descriptive statistics for the number of incorrect attempts.

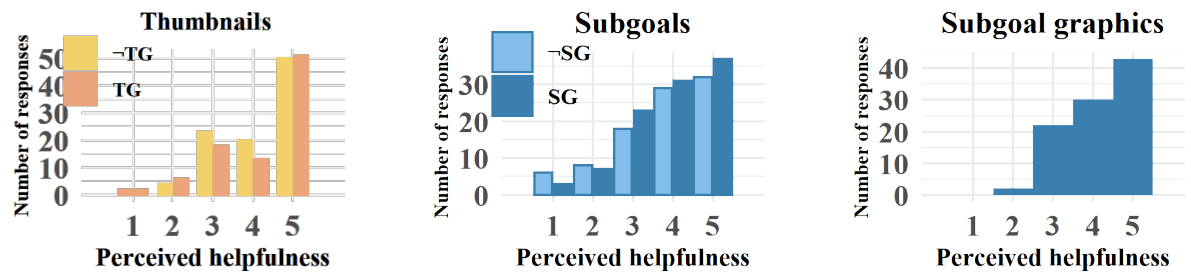
Figure 10.10: Data relating to a potential interaction in Part 1 number of attempts.

The effect of subgoal graphics

In looking at the number of incorrect attempts committed by each participant, as grouped by subgoal condition, the SG group committed on average 7 fewer of them (a median of 32 versus 39) with a location shift of 5 ($PS=.55$) but not significantly so for a Wilcoxon test ($W=5286, p=.28$), also with a 95% confidence interval of $[-14,12]$. It also was not significant for a permutation test ($p=.32$). Visually, Figure 10.9 shows no obvious group separation.

Interaction analysis

To probe whether the two variables interact with each other, we plotted the four distributions as boxplots, shown in Figure 10.10a. The boxplots indicate no pattern consistent with a synergistic interaction, since the TG/SG distribution is not noticeably lower in values compared with the rest. A permutation-based ANOVA found no significant interaction ($p>.99$).



(a) Perceived helpfulness of thumbnails ($n=186$).

(b) Perceived helpfulness of subgoals ($n=193$).

(c) Perceived helpfulness of subgoal graphics ($n=97$).

Figure 10.11: Subjective ratings of how helpful participants found various scaffolding elements of Part 1.

Conclusion

For both variables, the central tendencies are meaningfully different in directions that align with our hypothesis - suggesting that graphics *do* reduce the number of incorrect attempts - however the groupings are not visually separable, and not statistically significant. Therefore, we do not have enough evidence to reject the null hypothesis.

10.2.7 Evaluation

After the end of Part 1, a short evaluation survey was given, containing Likert-scale items rated 1 (*Not at all*) to 5 (*Very much*).

The helpfulness of thumbnail graphics

All participants, regardless of condition, were asked *How helpful did you find the thumbnails (and the informative box that pops up) to be for your learning?*. Note that this is technically a two-pronged question that asks about both tooltips and thumbnails in conjunction, and does not explicitly inquire about graphics. 186 people answered this questions. Looking at the ratings of Figure 10.11a, we find that most people regard the thumbnail graphics as *very helpful* but there does not appear to be any marked group differences.

The helpfulness of subgoal graphics

The SG group (but *not* the \neg SG group) was asked directly about how helpful they perceived the subgoal graphics to be. Compared with thumbnail graphics, we see in Figure 10.11c's a less dramatic skew: although a 5 rating was still the most common response, responses were more varied, suggesting that the subgoal graphics were less necessary than thumbnail graphics to accomplish the task.

All participants were asked about how helpful the *subgoals* were, a question that for the \neg SG group would be interpreted to mean the labels but which for the SG group would presumably be interpreted as labels and graphics in conjunction, thus allowing us to probe the differences in perception in a less leading way. However, in Figure 10.11b we find no meaningful in ratings.

Other metrics

Participants were also asked about their experience on a number of other dimensions, including how concentrated and motivated they felt, and how effortful, enjoyable and worthwhile they felt the task to be. These results are summarised in Figure 10.12 (grouped by subgoal condition) and Figure 10.13 (grouped by thumbnail condition).

Concentrated: Most participants rated their concentration to be at least a 3. We do not find any strong group differences associated with either independent variable

Effortful: The distribution of effort ratings was relatively symmetric, again without meaningful group differences.

Enjoyable: Enjoyability ratings were negatively skewed, suggesting that participants overall enjoyed the task. The TG group is over-represented among ratings of 5 while \neg TG more frequently rate it as 3, suggesting that thumbnail graphics could improve the enjoyability. With subgoal graphics, the response patterns are less consistent.

Motivated: Motivation ratings show a similar tilt as enjoyability ratings. No group differences are seen for subgoal graphics, but interestingly 5-ratings by TG participants far exceed \neg TG ratings, which tend more towards the middle, suggesting that thumbnails improve motivation.

Worthwhile: The worthwhile distribution is again negatively skewed, with no differences for subgoal graphics, but again a TG mode of 5 and a lower \neg TG mode of 4.

Overall, these response patterns suggest a greater influence of thumbnail graphics than subgoal graphics. This association is probably partially due to the nature of the Part 1 task, which involved operation selection, but it is still surprising that the relatively small feature of thumbnails carried more weight than the more visible subgoal graphics.

Open-ended feedback

The survey concluded with a free-text field where the participant could voice feedback, issues and ideas. 27 people wrote a response. Out of these, 12 were positive, for example one person wrote:

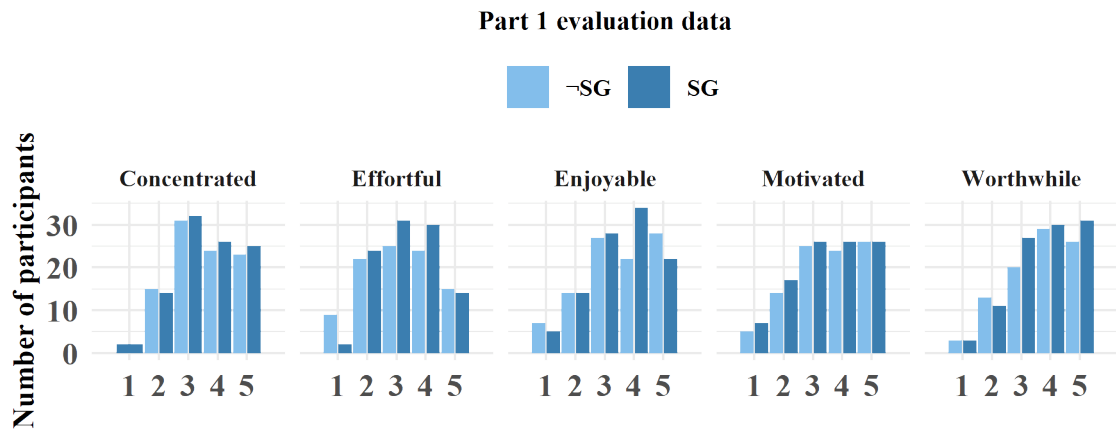


Figure 10.12: Evaluation survey items grouped by subgoal condition.

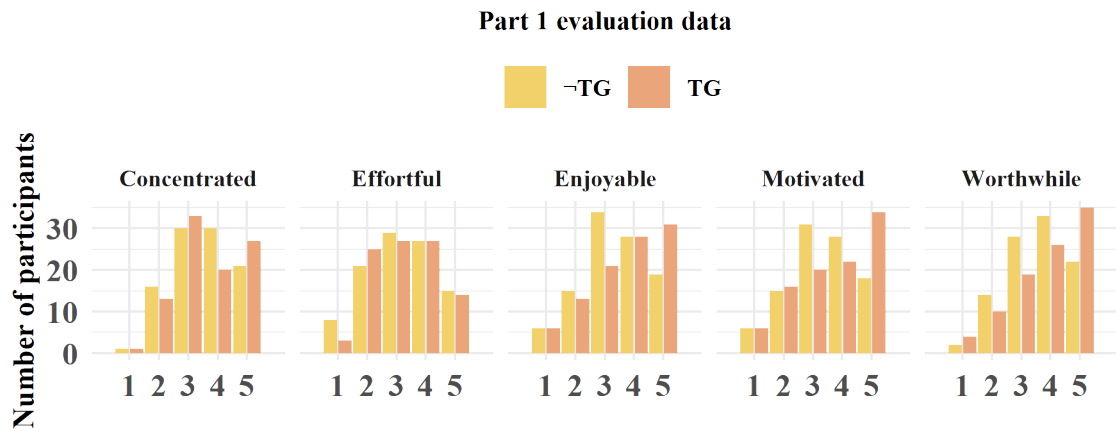


Figure 10.13: Evaluation survey items grouped by thumbnail condition.

“Great way of learning! Rather than throwing someone in the deep end, this was very simple to start with and so I ended up doing it fast, otherwise I’d have to carve time out of my work/studies to do it.”

Feedback on graphics 3 people commented specifically on the graphics (presumably the subgoal graphics):

“The graphics for each operation are very helpful.”

“The visuals were the reason it was understood what one did.”

“I thoroughly enjoyed the subgoals and thumbnails used.”

Interestingly, another person in the \neg SG condition themselves suggested what sounds like subgoal graphics:

“An illustration should be offered for each subtask leading up to the final task.”

Perceived low utility Some questioned the utility of a non-programmatic task and suggested providing code instead:

“I feel it would be better to run the actual program in combination with the guide.”

“there should be some sample of python code that might help in understanding more clearly.”

“I struggled to understand why I was doing this and why I needed to understand the different drag and drop options.”

“I would recommend that you should design some proper and logical, type of exercises because for me these exercises were not useful at all.”

Too simple hints Two people commented on the hints being too easy:

“The hints were good but sometimes they were too helpful by giving the answer away too easily.”

“hints should not be more clear because it effects our thinking capacity and we immediately find the answer.”

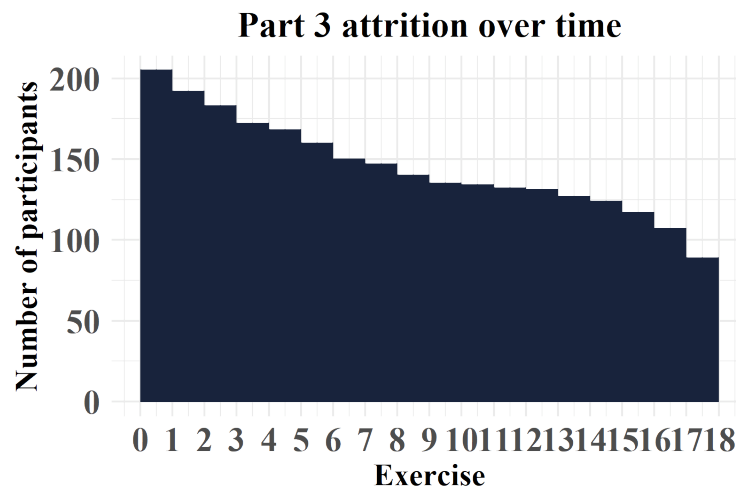


Figure 10.14: Each vertical bar represents the number of people completing that exercise.

10.3 Part 3

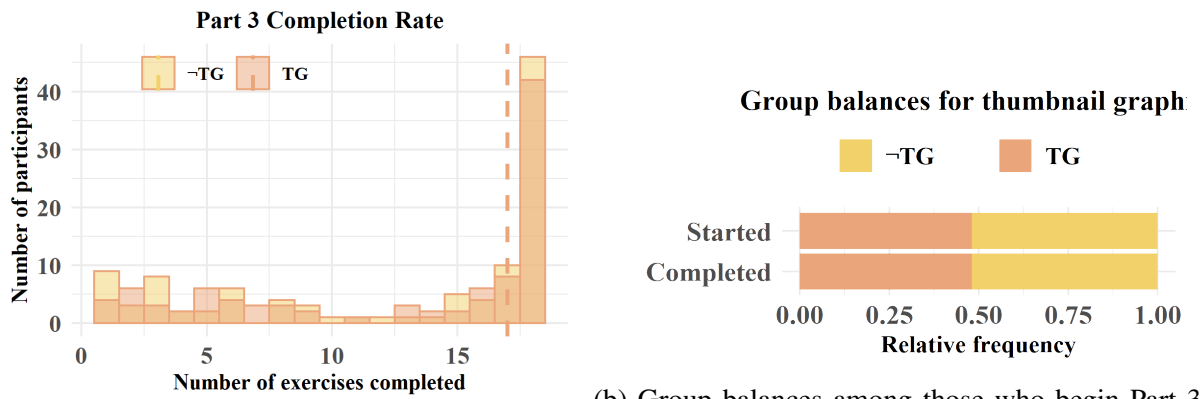
Since Part 2 included purely programming-relating topics, we proceed immediately to Part 3, which is focused on programmatic data wrangling, and consists of 18 programming exercises that participants complete in either Python or R. 3 out of these exercises are unscaffolded, i.e. without subgoals for 10 minutes.

10.3.1 Number of exercises completed

When inspecting the number of completed exercises, 204 participants completed at least 1 exercise, but only 88 participants completed all 18 exercises³. Figure 10.14 shows the decline in participation. Interestingly, it appears flat between the 8th and 15th exercise, followed by an accelerated plunge. This is surprising since a progress bar is visible throughout Part 3, which means the participants were aware of how close to the ending they were.

The attrition of participants creates a dilemma in terms of missing data. Like in Part 1, to compare aggregate performance across a range of exercises with different difficulty levels, the sum is most appropriate. This limits the analysis to participants who completed an equal number of exercises. As we have seen, only 88 participants completed the entire set, but if the last few exercises are excluded, this will boost the sample size. The cut-off point must balance data loss with respect to individuals, versus data loss with respect to exercises. 10 non-test exercises or more appeared as a good compromise, since it boosted the sample size to 123, and later exercises are vulnerable to fatigue, as shown by the pilot study (Chapter 8). This sample includes 114 participants that did it in Python, and 9 who did it in R.

³Recall that, in Part 3, there is a timeout mechanism where it becomes technically possible to proceed despite not having correctly solved the exercise. If we count the number of participants who reached the end, regardless of mechanism, the number is 121.



(a) Number of exercises solved per person.

(b) Group balances among those who begin Part 3 ($n=204$) and finish it ($n=88$).

Figure 10.15: Thumbnail-related group differences in tendency to complete Part 3.

Due to its length and intrinsic complexity (combining data wrangling conceptual skills with programming skills), Part 3 is vulnerable to a variety of dropout risks. In spite of our efforts to mitigate them, there are risks of boredom and frustration. Both participants who find it too easy or too difficult are likely to drop out.

Does the condition appear to impact the likelihood of completing all exercises? If graphics improve engagement and lower the cognitive load of the activity, then **we would expect the graphical conditions to be associated with more completed exercises**. The distribution of completed exercises per participant is clearly non-normal (see Figure 10.15a), with most participants concentrated at the higher end, but also at the beginning.

The effect of thumbnail graphics

Regarding thumbnail design, both the experimental and control condition complete a median of 17 exercises, both with an IQR of 12. Thumbnail design appears to exert no such influence, as shown in Figure 10.15a. Neither a Wilcoxon test ($W=5103$, $p=.83$) nor a permutation test ($p=.72$) are significant. The rightmost bar in the histogram shows the ratio of TG and -TG among those completed exercises: it is clear that the ratio upon completion remains almost the same as the ratio upon beginning (46 -TG and 42 TG completed). A Chi-squared test comparing the observed number of completions per group with that expected based on cell sizes (i.e. 97 and 107), also yields a non-significant result ($\chi^2=.001$, $p=.97$). These results are charted in Figure 10.16.

The effect of subgoal graphics

With subgoals the case is less clear-cut. The experimental group SG completed - in median terms - 2 more exercises than the control. They are also over-represented among participants who completed all exercises: 52 versus 36. This is more clearly conveyed by Figure 10.16b,

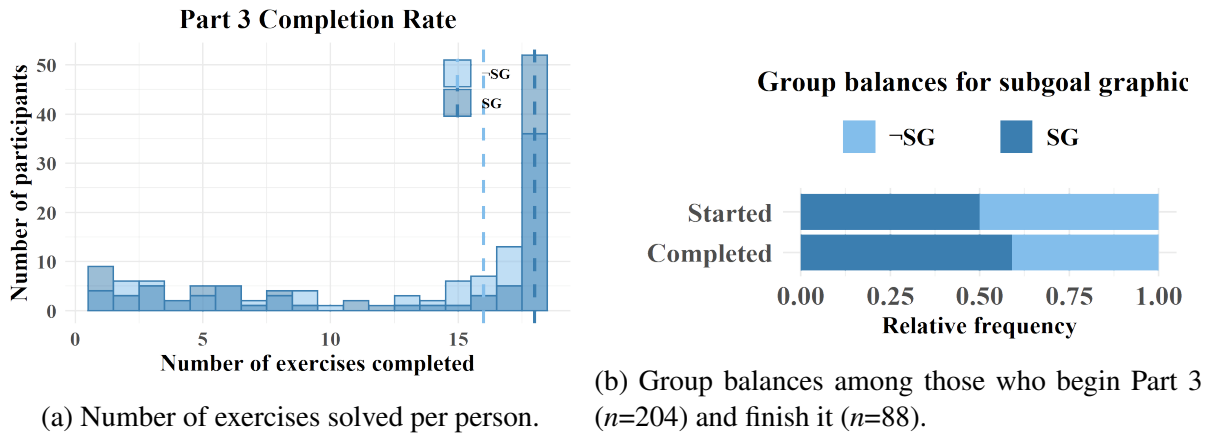


Figure 10.16: The SG condition is over-represented among participants who finish all 18 exercises.

Part 3 exercises completed statistics

	n	median	IQR	W	p
SG	103	18	12	4768	.28
\neg SG	101	16	11.5		

Table 10.2: The number of exercises completed depending on condition. W = Wilcoxon rank sum test.

which shows how the ratio of conditions changes among participants who started versus completed Part 3. The differences in exercises completed were not significant when subjected to a Wilcoxon rank sum test ($W=4769, p=.28$) with a location shift of -6.6 and 95% CI of [-10,1.5] or a permutation test ($p>.99$).

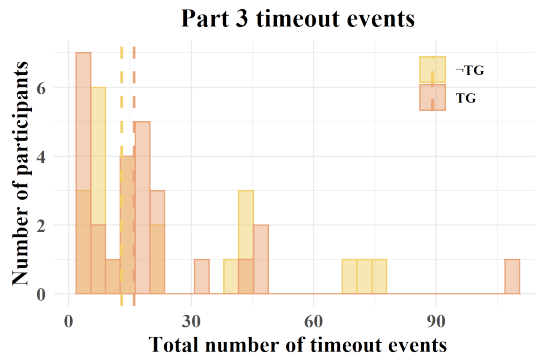
When performing a Chi-squared test, the deviation from expected counts of completions per condition fell just shy of significance ($\chi^2=3.23, p=.07$). Consequently, the null-hypothesis cannot be rejected, but we observe that the data aligns directionally with our prediction.

Conclusion

There does not appear to be any effect of thumbnail graphics on participants’ perseverance. We do have visual, arguably striking suggestions that subgoal graphics increase the likelihood of completing Part 3, but it is non-significant.

10.3.2 Timeout events

Recall that when the time between two consecutive key strokes or mouse movements exceeds 60 seconds, this records a timeout event, which could be seen as a measure of inattention. It is an imperfect metric, given that deep concentration may also cause apparent inactivity, but if graphics make the programming experience more engaging, we would **expect that the graphical**



	<i>n</i>	median	IQR	W	<i>p</i>
TG	27	16	22	318.5	.88
¬TG	23	13	23		

(a) The impact subgoal graphics on the number of tabs. Lines show medians.

(b) Descriptive and inferential statistics for the number of timeout events.

Figure 10.17: The TG is slightly more inactive, but not significantly or meaningfully so.

conditions exhibit fewer inactivity events.

We find that only 50 out of the 134 participants registered timeout events, and that the distribution of these 50 participants is concentrated towards the lower bound, with an overall median of 15 events (IQR=15.75).

The effect of thumbnail graphics

Due to the small sample size, the histogram of Figure 10.17a is sparse. In median terms, the TG group records 4 fewer events, but this is not significant when using a Wilcoxon rank sum test ($W=318.5, p=.88$).

The effect of subgoal graphics

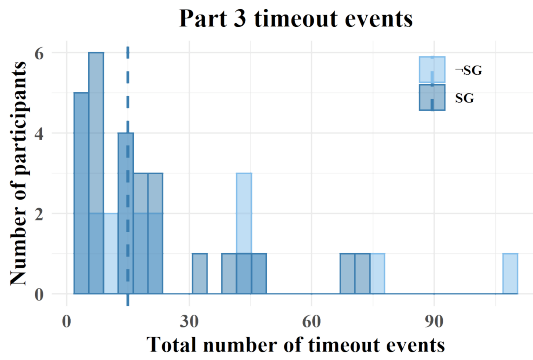
The histogram showing timeout events as grouped by subgoal condition is shown in Figure 10.18a. In median terms, they record the same number of events (15), which is non-significant using a Wilcoxon test ($W=319.5, p=.87$).

Conclusion

There appears to be no effect on the number of timeouts.

10.3.3 Tab events

In Part 1 we collapsed the two inactivity categories of timeouts and tab changes, since both could be considered proxies of inattention. In Part 3, there is the added caveat that tab changes could also indicate Google searches, or usage of other external resources as a substitute for the API menu. We will therefore consider it as its own category. We again **hypothesise that the graphical conditions will register fewer tab change events.**



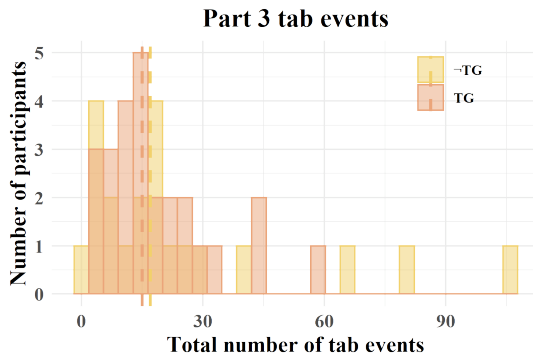
(a) The impact subgoal graphics on the number of timeout. Lines show medians.

Part 3 number of timeouts statistics

	<i>n</i>	median	IQR	W	<i>p</i>
SG	27	15	15.5	319.5	.87
¬SG	23	15	25		

(b) Descriptive and inferential statistics for the number of timeouts.

Figure 10.18: There are no evident group difference, although the ¬SG condition appears more dispersed.



(a) The impact thumbnail manipulations on the number of tabs.

Part 3 number of tab changes statistics

	<i>n</i>	median	IQR	W	<i>p</i>
TG	26	15	14.25	279	.91
¬TG	21	17	19		

(b) Descriptive and inferential statistics for the number of tab changes.

Figure 10.19: The TG is slightly lower in its tab changes, but not significantly or meaningfully so.

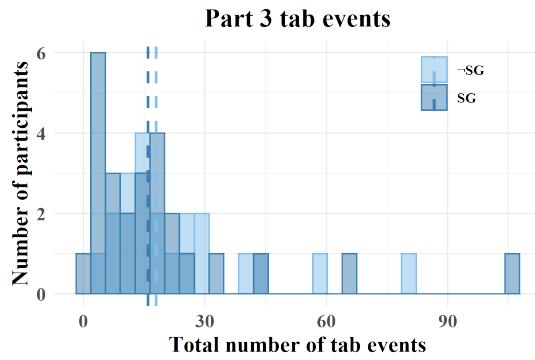
Only 47 participants in total registered tab changes, with a median of 16 events (IQR=16). Once again, we find that the distribution is heavily positively skewed.

The effect of thumbnail graphics

As shown by the histograms in Figure 10.19a, there are no strong group separation to suggest a difference. The TG group registers, on average, 2 fewer events (15 vs. 17), but this is not significant given a Wilcoxon test (W=279, *p*=.91).

The effect of subgoal graphics

Looking at the histograms in Figure 10.20a, we find a slightly clearer group separation, although this appears to be in part due to a histogram binning artefact. The SG group registers, on average, 2 fewer events (15 vs. 17), but this is not significant given a Wilcoxon test (W=337, *p*=.17).



Part 3 number of tab changes statistics

	<i>n</i>	median	IQR	W	<i>p</i>
SG	26	16	16	337	.17
¬SG	21	18	16		

(a) The impact subgoal graphics on the number of tabs.

(b) Descriptive and inferential statistics for the number of tab changes.

Figure 10.20: The SG is does slightly fewer tab changes, but not significantly or meaningfully so.

Conclusion

The provision of graphics does not appear to affect the number of tab changes.

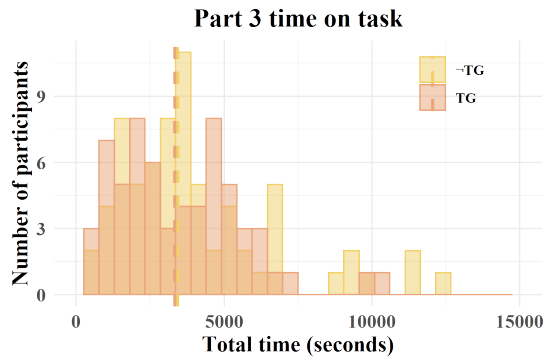
10.3.4 Time on task

As with Part 1, time on task is recorded as the time between an exercise is completed until the time it has been submitted and correct (*not* until the user clicks the *Next* button to proceed). We are concerned with *total* time on task, summed across the 10 exercises. Time spent off-task (timed out or with the tab out of focus) has been deducted as part of the data cleaning process. We noted in Section 10.2.5 that time on task has a complex interpretation but that, given the binary success outcome and automatic feedback thereof, a participant has nothing to gain from caution, except perhaps if they very deliberately seek to memorise syntax. All else equal, a shorter time on task is better, and is assumed to reflect efficiency in data wrangling-related activities. We **expect graphics to be associated with lower times on task**.

Participants completed the first 10 exercises within 65 minutes and 45 seconds, in median terms. The distribution is again positively skewed (skewness=2.16) and non-normal, as determined by a Shapiro-Wilks test ($W=0.82, p<.001$). We therefore proceed with non-parametric methods.

The effect of thumbnail graphics

In terms of median time on task, the thumbnail graphics condition is in median terms faster than the control group, but not by much, as seen in Figure 10.21. Participants receiving thumbnail graphics took 1.72 minutes less time (3330 versus 3434 seconds), with a location shift of 201 and 95% CI of [-561,958]. This is a small difference and wide interval, and a Wilcoxon test is *not* significant ($W=2369, p=.57$), nor is a permutation test ($p>.99$). Hence, no associated null

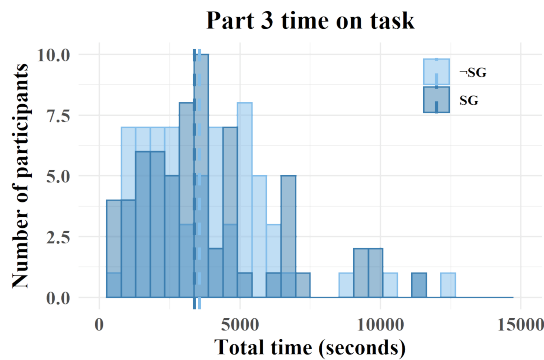


(a) The impact thumbnail manipulations on time on task in Part 3. Lines show medians.

	<i>n</i>	median	IQR	W	<i>p</i>
TG	64	3330	2955	2369	.57
¬TG	70	3434	2882		

(b) Descriptive and inferential statistics for the time on task.

Figure 10.21: The TG is slightly faster, but not significantly or meaningfully so.



(a) The impact subgoal manipulations on time on task in Part 3. Lines show medians.

	<i>n</i>	median	IQR	W	<i>p</i>
SG	66	3376	2529	2263	.93
¬SG	68	3558	3184		

(b) Descriptive and inferential statistics for the time on task.

Figure 10.22: The SG is faster by 3 minutes, but not significantly or meaningfully so.

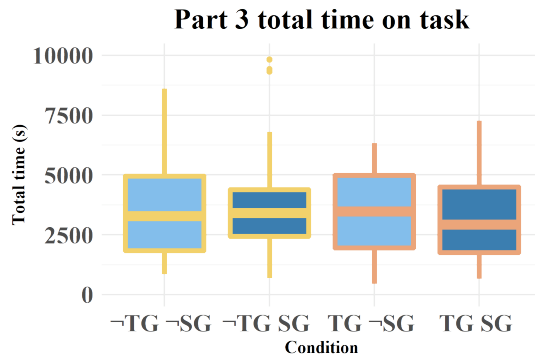
hypothesis can be rejected.

The effect of subgoal graphics

Participants with subgoal graphics are faster in median terms (3376 vs 3755 seconds, i.e. 3 minutes longer), but this was not significant ($W=2263, p=.93$), with a location shift of 46.7 and 95% CI of [-748,774]. There is no clear separation in Figure 10.22. Consequently we cannot reject the null hypothesis.

Interaction analysis

To probe whether the two variables interact with each other, we plotted the four distributions as boxplots, shown in Figure 10.23a. No significant interaction was detected using a permutation-based ANOVA ($p>.99$).



(a) Boxplots showing how total time on task varies depending on condition

Condition	<i>n</i>	median (s)	IQR
TG	SG	3061.8	2908
	¬SG	3658	3237
¬TG	SG	3434	2046
	¬SG	3520	3165

(b) Descriptive statistics for the total time on task.

Figure 10.23: Data relating to a potential interaction in Part 3 total time on task.

Conclusion

While the conditions with thumbnail conditions or subgoal conditions are both faster than their controls, inferential tests and visual inspection suggests that this could very well be flukes. The null hypotheses are not rejected.

10.3.5 Tooltip events

Recall that hovering over an operation in the menu will trigger an explanatory tooltip, which we interpret as the need for clarification as to how an operation works. We **expect thumbnail graphics to be associated with fewer tooltip events**.

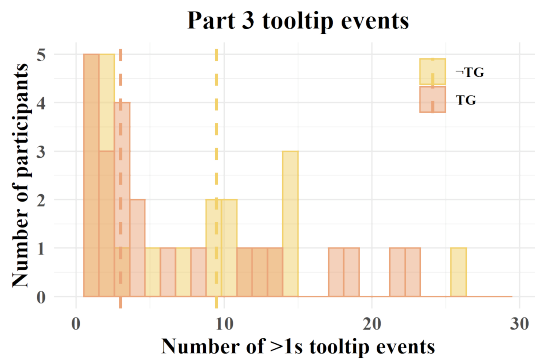
The number of people who made use of the tooltip feature was dramatically lower in Part 3 than in Part 1: only 53 out of 134 participants. As with Part 1, we exclude those who never used it, yielding a positively skewed distribution (skewness=2.62). Some of those who did not use tooltips here may have previously used the tooltips in Part 1. However, given the difference in the task, may have forgotten about them or assumed they were no longer available.

The effect of thumbnail graphics

The median tooltip usage among those with thumbnail graphics was 3 versus 9.5. This result was non-significant for a Wilcoxon rank sum test ($W=418.5, p=.19$), with a location shift of 2 and 95% CI of [-1,9] (PS=.61). However, it *was* significant for a permutation test ($p=.038$).

Conclusion

We have some evidence that, as in Part 3, the thumbnail graphic group uses the tooltip less often than the control group, consistent with the hypothesis.



(a) The number of tooltip events, grouped by thumbnail condition. Lines show medians.

Part 3 tooltip events statistics

	<i>n</i>	median	IQR	W	<i>p</i>
TG	23	3	9.5	418.5	.19
¬TG	30	9.5	12.75		

(b) Descriptive and inferential statistics for tooltip events.

Figure 10.24: Tooltip usage appears less common in the TG group, and this is significant for a permutation test, but not a Wilcoxon rank sum test.

10.3.6 Menu click events

When a user clicks on a leaf node in the menu, the menu will slide away within the sidebar panel to reveal a documentation entry with corresponding syntax information. The number of documentation clicks, summed across the 10 exercises, is therefore a proxy for the number of requests for syntax information. As with tooltip events, we assume that thumbnail graphics reduce the number of times a person needs to click to determine the entry is relevant. Hence, **we expect that participants with thumbnail graphics will click on the menu less often.**

Upon inspecting the data, it emerges that only 91 participants out of 134 ever clicked on the menu to reveal syntax. This is surprising - after all, participants depend on syntax information to author a program - and could mean one of two things. Firstly, it is possible that some participants exclusively accessed syntax information via the hints (recall that, for each subgoal, the first hint hyperlinked to the documentation and the third provided the expected syntax). This is confirmed by data - many participants who registered no clicks made extensive use of hints. A second possibility is that they relied upon external resources such as search engines, which would lead them to record tab changes. Figure 10.25 shows scatterplots probing both of these hypotheses. From these it is clear that, while many of the zero-click participants made extensive use of hints, only a few recorded many tab changes. This suggests a reliance on hints.

If we limit our analysis to participants who did not exclusively rely on hints (i.e. registered at least one menu click) does thumbnail condition matter? The graphical condition clicked, on average, 31.5 times versus 35 times, however this was not significant for a Wilcoxon ranked-sum test ($W=1029.5$, $p>.99$), with a location shift of 3.3 and a 95% CI of $[-12,13]$. We thus cannot reject the null hypothesis. Results are tabulated in Table 10.26b and plotted in Figure 10.26.

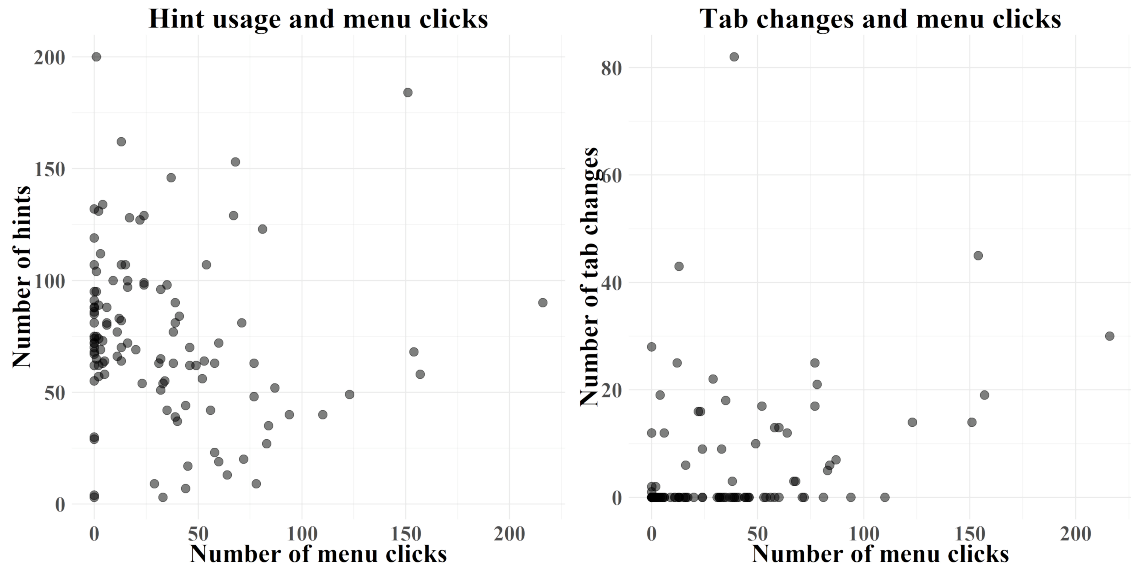
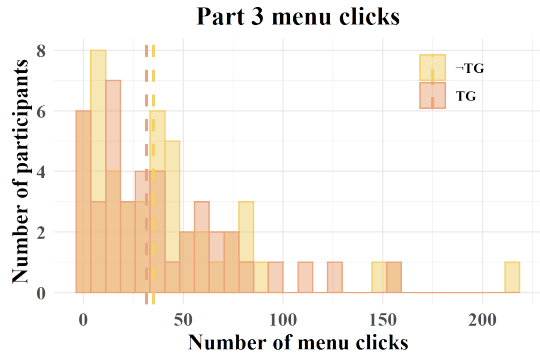


Figure 10.25: Why did so few participants click on the menu in order to access syntax information? The left scatterplot, which shows hint usage in relation to menu clicks, suggests that they relied on hints instead. The scatterplot to the right shows it in relation to tab changes, for example reflecting googling of external resources.



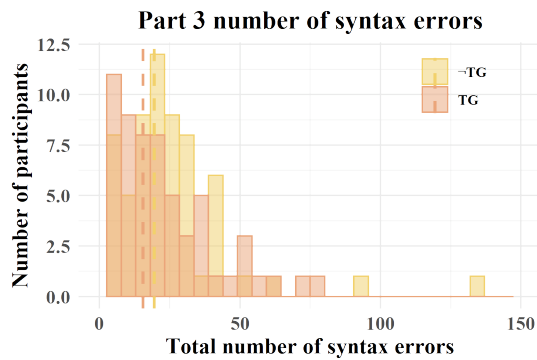
Part 3 click events statistics

	<i>n</i>	median	IQR	W	<i>p</i>
TG	42	31.5	44.5	1029.5	1
-TG	49	35	43		

(a) The number of times a participant clicks on the menu to request syntax information. Lines show medians.

(b) Descriptive and inferential statistics for click events.

Figure 10.26: The thumbnail condition clicks on the menu less often than the control condition, but only negligibly so.



(a) The number of syntax errors, grouped by thumbnail condition. Lines show medians.

	n	median	IQR	W	p
TG	64	15.5	225	2353	.61
-TG	70	19.5	21.5		

(b) Descriptive and inferential statistics for syntax errors.

Figure 10.27: The thumbnail condition commits fewer syntax errors, but not meaningfully or significantly so.

Conclusion

Despite the median difference, we cannot confirm a significant association between thumbnail graphics and menu click numbers.

10.3.7 Syntax errors

When a user runs or submits their code, and this results in a run-time error, we regard this as a *syntax error*. A high number of syntax errors could reflect haphazardness, a mindless trial-and-error approach, as well as a genuine confusion over syntax details. It could in some instances also reflect a more conceptual confusion over how to combine data wrangling operations in a coherent sequence. Assuming graphics help with both locating syntax information and understanding this sequence, we **expect that the graphical conditions show fewer syntax errors**.

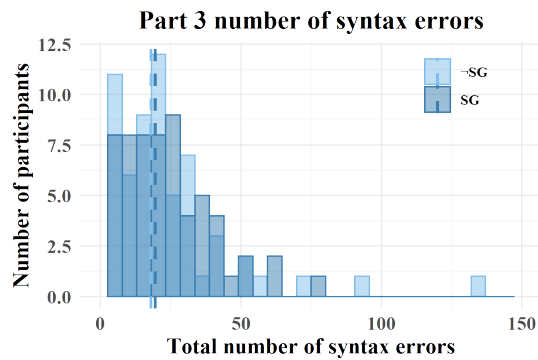
When this is summed for 10 exercises in Part 3, we find that the median number of exercises solved is 19 (IQR=21) and that the distribution is positively skewed (skewness=2.21).

The effect of thumbnail graphics

The TG group commits, on average, 4 fewer syntax errors, however not significantly so ($W=2353$, $p=.61$), with a location shift of 1 and a 95% CI of $[-4,6]$. It is also not significant for a permutation test ($p>.99$), leaving us unable to reject the null hypothesis. Results are shown in Figure 10.27 and Table 10.27b.

The effect of subgoal graphics

The subgoal graphic group commits on average 1.5 *more* syntax errors (19.5 versus 18), which is inconsistent with the expectation. However, it is not significant for a Wilcoxon test ($W=2103.5$, $p=.53$), nor a permutation test ($p>.99$). The results are summarised in Figure 10.29.



(a) The number of syntax errors, grouped by subgoal condition. Lines show medians.

	n	median	IQR	W	p
SG	66	19.5	23.5	2105	.53
-SG	68	18	20.5		

(b) Descriptive and inferential statistics for syntax errors.

Figure 10.28: The thumbnail condition commits fewer syntax errors, but not meaningfully or significantly so.

Conclusion

We find no strong evidence to suggest that graphics reduce the number of syntax errors.

10.3.8 Semantic errors

When a user runs or submits their code and the code does not cause a run-time error, but fails the submission correctness test suite, this is counted as a *semantic error*. It means that their solution is syntactically correct, and therefore that either the operation(s) were wrong, the arguments were wrong, or the solution is incomplete. Subgoal graphics are intended to help students identify the correct operation sequence, and thumbnail graphics are meant to help them find it in the documentation. We **expect that the graphical conditions show fewer semantic errors** than the control.

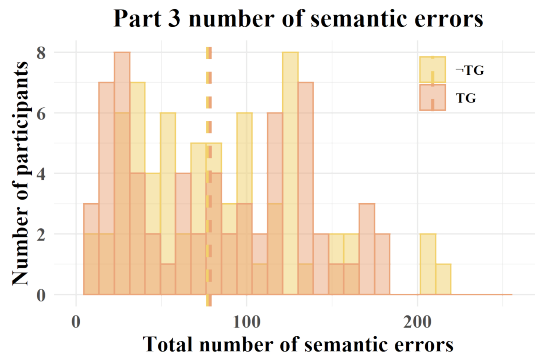
Looking at the marginal distribution (see Figure 10.29a), the distribution shows an almost bimodal distribution, although this is partially an artefact of binning the data: the distribution is positively skewed (skewness=7.7). Across the 10 exercises, the median number of semantic errors is 77.5 (IQR=85). Semantic errors are therefore much more common than syntax errors.

The effect of thumbnail graphics

The graphical condition commits 1.5 more semantic errors, in median terms, which is not significant for a Wilcoxon test ($W=2339.5$, $p=.66$) or permutation test ($p=.32$). It is also not evident in the distributions, shown in Figure 10.29a.

The effect of subgoal graphics

In median terms, the group provided with subgoal graphics committed 12 fewer semantic errors (73 versus 85) but this did not translate to a visual separation (Figure 10.30a) or a significant



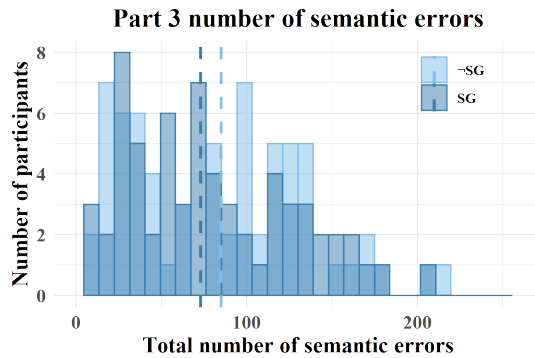
(a) The number of semantic errors, grouped by thumbnail condition.

Part 3 semantic error statistics

	<i>n</i>	median	IQR	W	<i>p</i>
TG	64	78.5	92	2339.5	.66
¬TG	70	77	79		

(b) Descriptive and inferential statistics for semantic errors.

Figure 10.29: The thumbnail condition commits *more* semantic errors, but this is not significant.



(a) The number of semantic errors, grouped by subgoal condition. Lines show medians.

Part 3 semantic error statistics

	<i>n</i>	median	IQR	W	<i>p</i>
SG	66	73	84	2305.5	.79
¬SG	68	85	87		

(b) Descriptive and inferential statistics for semantic errors.

Figure 10.30: The graphical condition commits fewer semantic errors, but not significantly so.

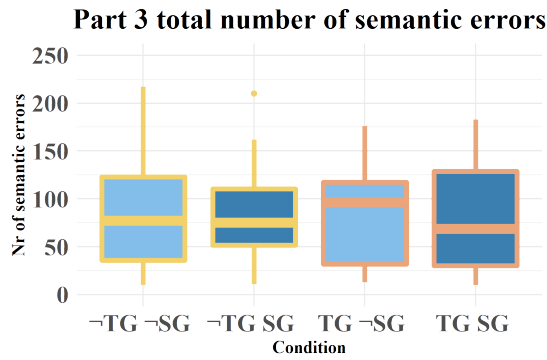
effect in a Wilcoxon test ($W=2305.5, p=.79$), or permutation test ($p=.44$). Our null hypothesis therefore cannot be rejected.

Interaction analysis

To probe whether the two variables interact with each other, we plotted the four distributions as boxplots, shown in Figure 10.31a. No significant interaction was detected using a permutation-based ANOVA ($p>.99$).

Conclusion

Although in the case of subgoal graphics there is a meaningful difference in central tendency, there is not sufficient evidence to reject the null hypothesis.



Part 3 nr of semantic errors statistics

Condition	<i>n</i>	median	IQR	
TG	SG	30	68.5	98.5
	¬SG	34	96.5	85.75
¬TG	SG	36	75	64.5
	¬SG	34	80.5	87

(a) Boxplots showing how the number of semantic errors varies depending on condition

(b) Descriptive statistics for the total semantic errors.

Figure 10.31: Data relating to a potential interaction in Part 3 total semantic errors.

10.3.9 Hint usage

In Part 3, the participant has three hints available for every subgoal that they are free to make use of. 22 out of 134 participants did not use a single hint. Consistent with previous analytical decisions, we chose to exclude these from the analysis, since it may indicate that they never discovered hints as a feature. Because graphics are designed with the intention to *reduce* the need for support such as hints, we **expect that the graphical conditions use fewer hints** compared with controls.

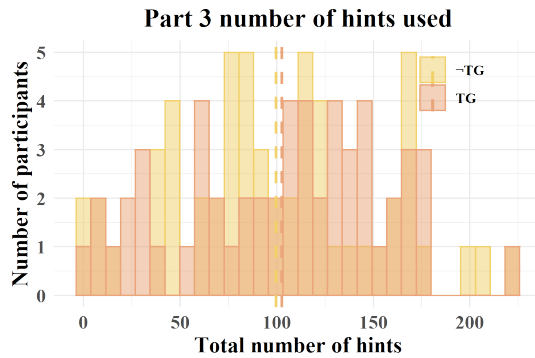
Hints are summed across the 10 exercises, and repeated access of the same hints are regarded as distinct events, and therefore factored into the hint usage count. Unlike previous metrics, hint usage appears to follow a symmetrical distribution, and a Shapiro-Wilk suggests that it is *not* significantly different from a normal distribution ($W=.98, p=.17$). We will therefore use parametric statistics. The mean hint usage is high - 101 - with a standard deviation of 52.

The effect of thumbnail graphics

In mean terms, the TG group used 3 more hints (103 versus 100), an effect that was not significant for a Welch Two Sample *t*-test ($t(109.85)=-0.3, p=.77$) with a 95% CI of [-22.6,16.7]. See Figure 10.32a and Table 10.32b. The null hypothesis cannot be rejected.

The effect of subgoal graphics

The graphical condition used on average 8 more hints, but is less widely dispersed than the control group. This did not translate into a significant effect ($t(108.6)=-.81, p=.42$), with a 95% CI of [-27.4, 11.5], leaving us unable to reject the null hypothesis.



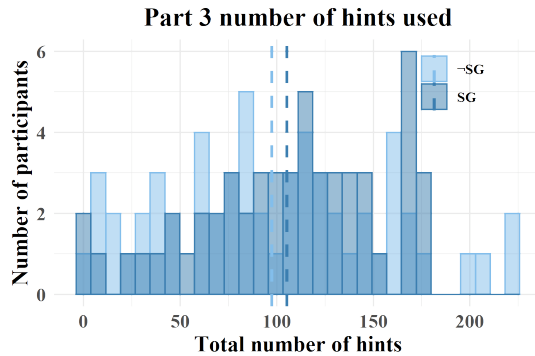
Part 3 hint usage statistics

	<i>n</i>	mean	SD	<i>t</i>	<i>p</i>
TG	55	103	52.4	-0.3	.77
-TG	57	100	52.5		

(a) The number of hints used, grouped by thumbnail condition. Lines show means.

(b) Descriptive and inferential statistics for hint usage.

Figure 10.32: The thumbnail condition uses hints slightly more often, but this is not significant.



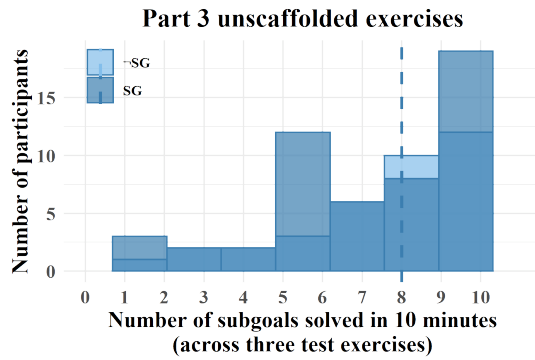
Part 3 hint usage statistics

	<i>n</i>	mean	SD	<i>t</i>	<i>p</i>
SG	54	105	47	-0.8	.42
-SG	58	97	57		

(a) The number of hints used, grouped by subgoal condition. Lines show means.

(b) Descriptive and inferential statistics for hint usage.

Figure 10.33: The subgoal condition uses hints *more* often than the control, but this is not significant.



Part 3 unscaffolded statistics

	<i>n</i>	median	IQR	W	<i>p</i>
SG	52	8	5	998	.6
-SG	36	8	2.25		

(a) The number of correct operations, grouped by subgoal condition. Lines show medians.

(b) Descriptive and inferential statistics for hint usage.

Figure 10.34: The subgoal condition uses hints *more* often than the control, but this is not significant.

Conclusion

We do not have a sufficient inferential basis to reject the null hypothesis that either graphical type reduced the number of hints used.

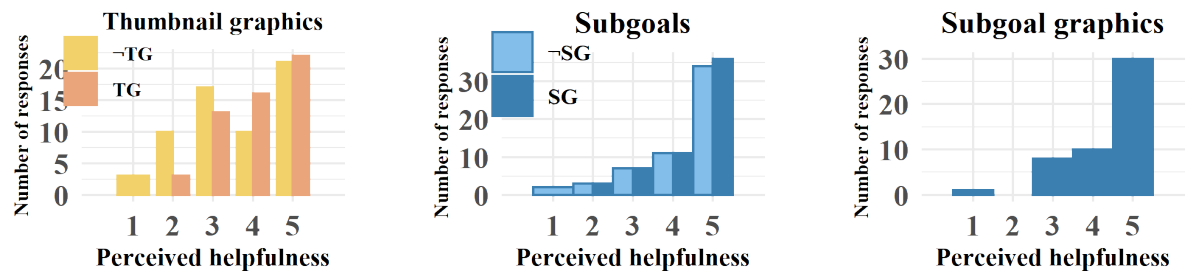
10.3.10 Unscaffolded exercise performance

3 of the 18 exercises in Part 3, 2 of which are included in the 10 exercises that previous analyses limited themselves to, were unscaffolded. For the first 10 minutes, the subgoals and associated hints and graphics were removed, although the menu remained the same. The 10 first minutes can thus be conceptualised as a test condition, where it is possible to measure the extent to which previous subgoal graphical exposure may or may not have translated to improved problem-solving capabilities once those training wheels are taken off. One way to operationalise it is to look at code executions within that timeframe and see whether they contain the correct operations. For every person’s submissions, we check for syntax tokens and function names that indicate that the participant has identified the correct operation. **We expect that the subgoal condition includes more correct operations** than the control.

This measure has a maximum of 10 points. Since we are restricted to complete data, the sample size is 88 and no longer balanced (SG=52, -SG=36). We find these to have a skew towards the upper bound (Figure 10.34). The subgoal graphic group solved the same median number of subgoals (median=8, IQR=5) than the control (median=8, IQR=2.25) and this was not significant using a Wilcoxon test (W=998, *p*=.6).

Conclusion

Subgoal graphics does not appear to impact the number of correct operations used in the first 10 minutes of unscaffolded exercises.



(a) Perceived helpfulness of thumbnail graphics ($n=115$).

(b) Perceived helpfulness of subgoals ($n=114$).

(c) Perceived helpfulness of subgoal graphics ($n=49$).

Figure 10.35: Subjective ratings of how helpful participants found various scaffolding elements of Part 3.

10.3.11 Evaluation data

The evaluation survey after Part 3 was answered by 121 people. This number includes people who may have skipped past an exercise using the timeout mechanism (see Section 10.3.1). A participant was free to leave some items untouched in the survey. As with Part 1, the items were Likert-scale (1=*Not at all*, 5=*Very much*) and phrased as *How helpful did you find...?*.

The helpfulness of thumbnail graphics

All participants, regardless of condition, were asked “How helpful did you find the thumbnails (and the informative box that pops up) to be for your learning?”. 115 participants responded to this question. We had **hypothesised that the graphical versions would be perceived as more helpful than the controls**. From the response distribution in Figure 10.35a, it can be noted that the \neg TG control group outnumbers TG responses in the lower ratings (1-3), while the opposite happens with higher ratings 4-5. This patterns lends some support the hypothesis, but it should be noted that both groups see a negative skew and mostly (37% of responses) view the thumbnails as *very helpful*.

The helpfulness of subgoal graphics

The group with subgoal graphics (but *not* \neg SG) was asked explicitly about how helpful they found the subgoal graphics to be. 49 people responded to this question, and we **had expected that subgoal graphics would be perceived as very helpful by an overwhelming majority**. While we cannot compare with the control group, it is clear from the distribution (Figure 10.35c) that a majority (71%) of respondents found the subgoal to be *very helpful*. Although there is certainly a risk of demand characteristics and social desirability bias that are hard to wholly dismiss, we can therefore say firmly that participants perceive the subgoal graphics as valuable, more unitedly so than thumbnail graphics.

Another evaluation survey item tests the perception of subgoal graphics more obliquely, by

asking how helpful they found the subgoals to be. We reason that the \neg SG is likely to interpret this as the subgoal *labels* (since those are the only elements available) while the SG condition would interpret it as labels and subgoals in conjunction. **We had predicted that subgoals would be perceived as more helpful if associated with graphics.** However, looking at the distribution (Figure 10.35b), we find no differences in response pattern between the two conditions. We therefore cannot say with certainty that, subjectively, subgoal graphics provide value beyond what subgoals already do.

Other metrics

The survey also asked participants about other metrics, such as how effortful, worthwhile and enjoyable they found the task to be, and how concentrated and motivated they were while completing it. These distributions are shown as grouped by thumbnail graphics in Figure 10.36, and subgoal graphics in Figure 10.37. For all of these items, the sample size was $n=121$.

Concentration: Overall, participants reported being at least moderately concentrated during Part 3. No striking group separation is seen for either thumbnails or subgoals, although while for SG the distribution grows monotonically, most \neg SG rate their concentration as a 3.

Effortful: It is clear that participants found Part 3 *very effortful*. No stark differences are visible upon grouping by thumbnail condition, but with subgoals, there are more *very effortful* responses for the \neg SG condition.

Enjoyable: The enjoyability distribution appears relatively uniform. The \neg TG group appears more concentrated in the lower ratings than TG, suggesting that thumbnail graphics enhance the enjoyability, but this is less pronounced with subgoal graphics.

Motivated: The mode value of the motivation ratings is a 3. For the \neg TG group, the second most common rating is 2, but there are no strong consensus or overall group differences.

Worthwhile: Responses concentrate around 3 and 5, and participants with graphics of either kind appear more concentrated towards the middle than the upper bound.

Open-ended feedback

There was additionally a free-text field where participants were solicited for general feedback and ideas. Only 26 participants chose to leave such feedback in Part 3. 9 of these comments were generically positive without specifying anything in particular. Constructive criticisms noted a few remaining glitches in the code editor, requested features such as videos, the ability to revisit past exercises, and a search bar for the menu. Graphics were never explicitly mentioned, but one

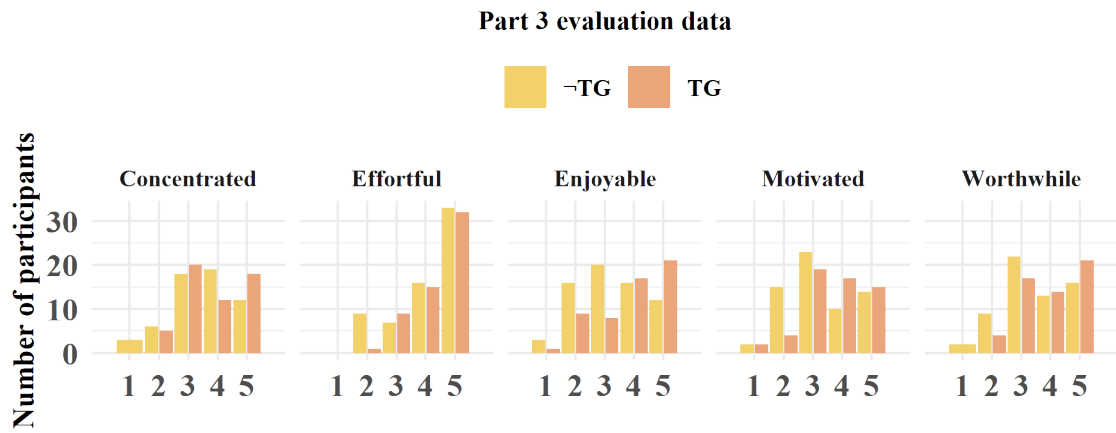


Figure 10.36: Evaluation survey items grouped by thumbnail condition.

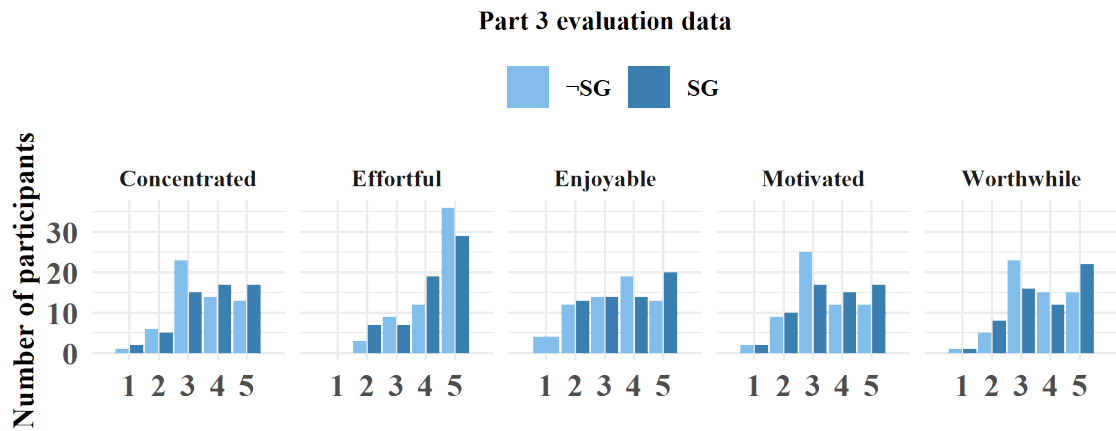


Figure 10.37: Evaluation survey items grouped by subgoal condition.

participant (in TG/SG) noted that the “subgoals were helpful” while another person (in \neg SG) noted that they “difficult at times to understand why some of the subgoals were necessary”. Two recurrent themes emerged from the remaining feedback:

Disagreements regarding level of scaffolding Unsurprisingly given the variety in background, there was disagreement about the difficulty gradient of the tutorial, with some finding it appropriate, and others finding it discouraging:

“I really enjoyed how the difficulty went up with time and slowly there were way more things you had to figure out during a single subgoal.”

Interestingly, in contrast with our worries that scaffolding-less testing would cause people to drop out, one person requested it, perhaps in order to concretise the skills:

“there should be introduce a little test in last of the course to check the skills .the test must have no subgoals or hints.”

It is clear that some participants would have benefited from reduced scaffolding and fewer hints, and that others would have benefited from more practice opportunities to consolidate their skills before proceeding.

“There should be less hints so that we put more effort for getting the solution.”

Low perceived utility A second quote suggests that letting the learner repeatedly look up information may not feel like actual learning:

“The first tasks were challenging but possible to do, hence I enjoyed doing them. However, then the tasks got too complex and I completely lost motivation to do them, as it felt that I will not remember the used operations anyway, so why should I put effort into learning them.”

It suggests that the lack of retention requirements and recall-based testing may have led to the perception that no durable skills were gained. It may be that the ability to utilise API documentation does not feel as concrete as, for example, the ability to recall Python syntax rules. The same sentiment is found in another quote, by a person completely without graphics (i.e. in \neg TG, \neg SG):

“I felt like the format of question prompt/subgoals with answers didn’t prioritise instilling the skills. It became more of a puzzle of looking up the expected answer and blindly filling out the gaps. obviously no offence but i have finished part 3 and have no clearer understanding of the subject matter. i expect i will have to revisit this and bring along my own researched guidance from elsewhere to figure out the "why" of each of these questions and not just the how. i do not feel prepared to answer any further question off my own back.” [sic]

It may be that, without graphics and the structural understanding that graphics are hypothesised to impart, lookup-based exercises simply felt too mechanical or mindless. It may be particularly important to offer a deeper understanding of the API data structures and paradigms, and instil the robust API knowledge written about by Thayer et al. [115]. This need for an underlying understanding of how commands work is seen in the following quote:

“I didn’t and still don’t really understand how to use and combine different types of commands in one string of code, but overall good.”

This suggests that graphics and lookup-based activities should be preceded by a more technical introduction, for example on how method chaining in Python works and what is happening under the hood.

Conclusion

Much of the of the open-ended feedback mirrors that of the pilot studies. It is clear that the task was perceived as highly effortful and that opinion on its utility was split, with the main issue being an absence of more in-depth explanations. The ratings data suggest that, while thumbnail graphics are viewed by most as very helpful, and subgoal graphics are viewed as very helpful by a clear majority, there are no strong differences in opinion regarding thumbnails and subgoals.

10.4 Summary

From the results presented above, and summarised in Table 10.3, many group differences in central tendency aligned directionally with our hypothesis, but only a few were convincingly large or significant. Distal performance variables like time on task and number of attempts did not appear to be sensitive to the experimental conditions, but in both Part 1 and 3, the more proximal variable of tooltip events did indicate a benefit of thumbnail graphics. Moreover, in Part 3, subgoal graphics appeared to meaningfully increase the likelihood of completing the entire exercise set. We were particularly surprised that thumbnail graphics did not have a greater impact on operation card reading times, and that subgoal graphics did not influence the number of semantic errors more markedly, although both received descriptive support. The results will be discussed in greater detail in Chapter 12.

Results from analysis						
	Dependent variable	Hypothesis	TG vs. ¬TG	PS	SG vs. ¬SG	PS
Part 1	Number of completed exercises	G>¬G	–	-	–	-
	Number of inactivity events	G<¬G	–	.57	–	.55
	Operation card reading times	G<¬G	<	.55	–	.53
	Number of tooltip events	G<¬G	< *	.6	–	.52
	Time on task	G<¬G	>	.5	<	.52
	Number of attempts	G<¬G	<	.54	<	.54
Part 3	Number of tab changes	G<¬G	–	-	–	-
	Number of timeouts	G<¬G	–	-	–	-
	Number of exercises completed	G>¬G	–	.49	>	.46
	Number of tooltip events	G<¬G	< *	.61	–	.61
	Number of menu click events	G<¬G	<	.50	–	.51
	Number of hints used	G<¬G	<	.47	<	.44
	Number of syntax errors	G<¬G	<	.53	>	.47
	Number of semantic errors	G<¬G	>	.52	<	.51
	Time on task	G<¬G	<	.53	<	.50
	Un scaffolded	G>¬G	–	-	–	-

Table 10.3: Results. The *hypothesis* columns indicates whether graphics were expected to reduce or increase the value of the metric. *TG vs. ¬TG* and *SG vs. ¬SG* indicate the direction of the group difference, where – indicates a lack of noteworthy group difference. *PS* indicates effect sizes, given in probability of superiority [1]. A dash (-) in *PS* indicates that no inference test was pursued on account of descriptive statistics. Asterisk indicates significance at $\alpha=.05$ (Part 3 tooltip events were significant for permutation tests, but not Wilcoxon rank sum test). None was significant at the Bonferroni-adjusted level ($\alpha=.0016$). Bold metrics indicate distal performance variables.

Chapter 11

Slice N Dice secondary analyses

In the introductory chapter we articulated our intention to minimise data waste by conducting a secondary set of analyses. These analyses will not compare observations on the basis of their experimental condition, but rather explore their associations with learner attributes and other variables.

11.1 Volunteer bias

Any study that depends on volunteers is vulnerable to *volunteer bias*, such that the people who choose to dedicate hours to a non-credit bearing study may differ systematically from the general student population. Volunteer bias would ideally be measured by comparing metrics of those who participate versus those who do not. In the absence of such data, we could measure differences among those who *complete* a part versus those who *began* the study. The most relevant variables are presumably those that tap into prior experience and motivation.

In SLICE N DICE, participants were asked in the demographic survey to rate their previous experience in Excel, Python, and R on a five-point Likert-scale (1=*Not at all*, 2=*A little bit*, 3=*Beginner's level*, 4=*Intermediate*, 5=*Advanced*). Figure 11.1a shows the distribution differences among those who began and completed Part 1 using a clustered bar chart. For all three technologies, the distributional shape remains roughly the same. This metric does not show any strong evidence for an experience-related selection bias.

Another proxy for experience was the 8-item programming pretest given just after the demographic survey, asking participants about whether they have experience with 8 foundational programming concepts, such as Booleans, functions and dataframes. It was thus ultimately a matter of self-assessment. The number of positive responses were summed together for participants who begin and complete Part 1, and are charted in Figure 11.1b. Here we find a more pronounced difference in shape, in that the relative reduction in participants appear much greater in higher-scoring participants - suggesting the opposite pattern that high-ability learners are *more* likely to abandon the platform.

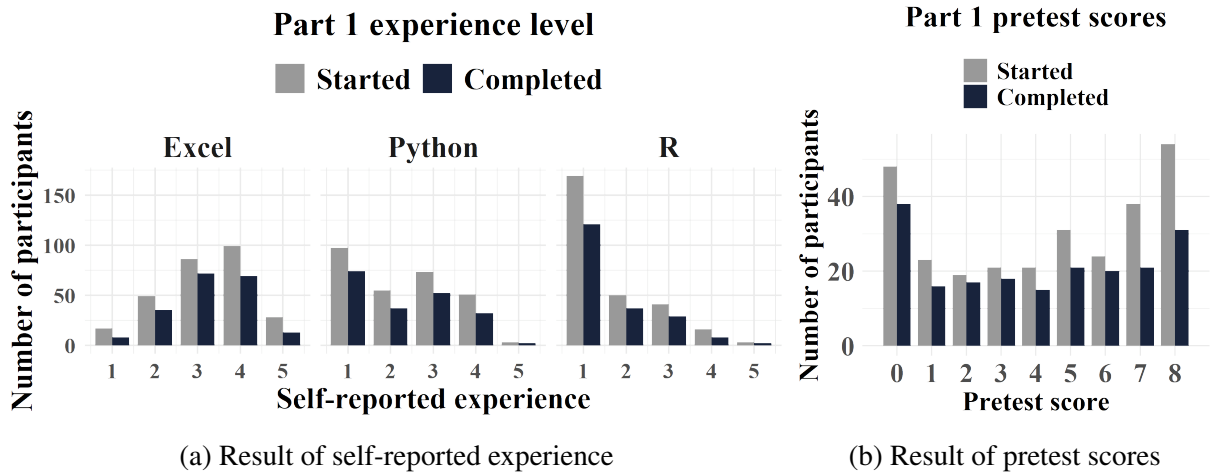


Figure 11.1: Experience-related differences among people who began and completed Part 1

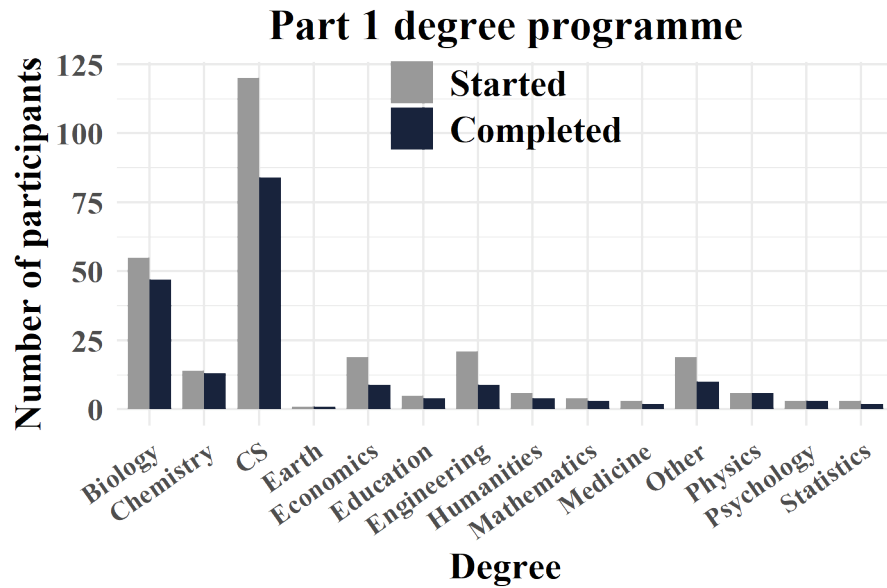


Figure 11.2: The degree programmes that participants were enrolled in.

Participants also indicated which university degree they were studying, if any. From Figure 11.2, it is clear that most participants were CS majors, likely reflecting volunteers from the Namal Institute, and that a larger proportion of them dropped out before completing compared with biology majors. This is consistent with the previous finding that high-ability students may judge the Part 1 task to be too simple to be useful.

11.2 Bug analysis

In the literature review, we mentioned that two previous bug analyses had been conducted from data wrangling sessions in R. Analysing the relative frequency of bugs and inferring the misconceptions underpinning them presents an effective strategy of generating actionable points of

improvements. We therefore conducted our own analysis of the data produced through SLICE N DICE.

Error proportions were calculated by aggregating errors raised in the Part 3 programming logs for *all* participants doing it in Python (i.e. not just complete observations). This resulted in 9,840 errors, which were then grouped by type. Semantic errors were operationalised as incorrect submissions (i.e. when they click at *Submit* not on *Run*). The result is visualised in Figure 11.3.

NameError: Among syntactic errors, `NameError` was the most frequent, which are raised when a variable or function is used before it is declared, likely due to accidentally misspelling or forgetting to declare a variable. This is consistent with the findings of Rafalski et al. [109] and Yarygina [171]. Manual inspection showed that recurring `NameErrors` included missing quotation marks around strings and uncapitalised Booleans, incorrect or missing library alias (e.g. `prod` instead of `np.prod`) and mixing up attributes with methods and functions (e.g. `size(x)` instead of `x.size`). This suggests that data wranglers would benefit from a light introduction into object-oriented programming, and how methods, functions, and attributes differ.

SyntaxError: `SyntaxError` exceptions are raised when the program cannot be parsed, and so usually relates to delimiters. Within the data, recurring causes were the use of incorrect brackets (e.g. `df.sum[]`), a confusion about why quotation marks are needed in square bracket notation but not in dot notation (e.g. `students.'absence'`), combinations of the two notations (e.g. `students.['raw']`), and issues with nested brackets during subsetting, especially when using compound conditions. These highlight usability issues with Pandas, which still relies on indexing operators instead of functions for subsetting rows or columns. Pandas would probably benefit from a more Tidyverse-like syntax that allows `df[df.A==5]` to be replaced with `df.filter(A==5)`¹.

There were also clear coordination issues (as per Ko et al.'s classification [158]) where participants did not understand how to combine operations. For example, there were several instances of `shape[0]mat` or `.shapemat` when `mat.shape[0]` was expected. This relates to Kelleher and Ichinco's observation that documentations (including ours) tend to treat operations in isolation from one another [3], and Thayer et al.'s recommendation for documentation to feature API usage patterns that explicate common combinations [115].

AttributeError: These are raised when the programmer attempts to access a non-existing attribute from an object. In our data, these were mostly due to mistyped NumPy functions, a very common one being `np.arrange` instead of `np.arange`.

¹In Pandas 1.3.3, there is a `filter` method, but it subsets based on labels rather than contents.

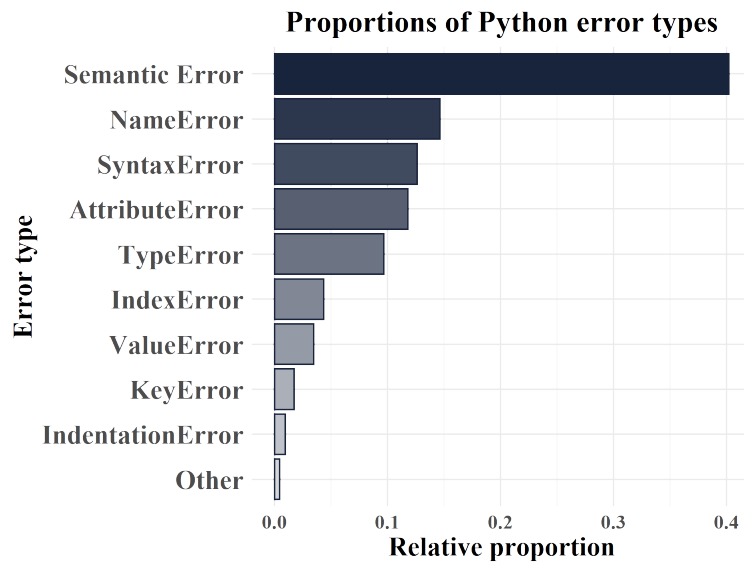


Figure 11.3: The relative proportions of error types, among all Part 3 exceptions raised by errors programming in Python.

TypeError: `TypeError` is raised when an operation is performed on objects with an incorrect or unsupported type. The most common cause of this error was failures of example adaptation, where inappropriate arguments were passed to a function. Another common cause of this error included attempts to subscript functions (e.g. `np.repeat[twice, 2]`) or using round brackets in order to subset (e.g. `labels(both)` where `labels` is a matrix and `both` is a mask), suggesting a confusion about the difference between functions and the indexing operator. Other instances of this error appear to reflect a failure to track the contents of a variable. For example, in one case the learner attempted to use an `&` on two variables that contain scalars, presumably believing it contained a mask. These are precisely the kind of issue that subgoal graphics could remedy.

Implications

Overall these findings testify to the complex design of the NumPy and Pandas APIs and suggest that code examples will be inadequate without a deeper explanation of the object-oriented concepts involved, and an understanding of how operators and functions relate to one another.

11.3 Correlational analysis

11.3.1 Process metrics

With so many process metrics collected, all believed to capture some aspect of learners' data wrangling ability and therefore to be associated with it, there is a risk of redundancy, such that the set of metrics collected can be reduced. For this reason, we conduct a correlational analysis.

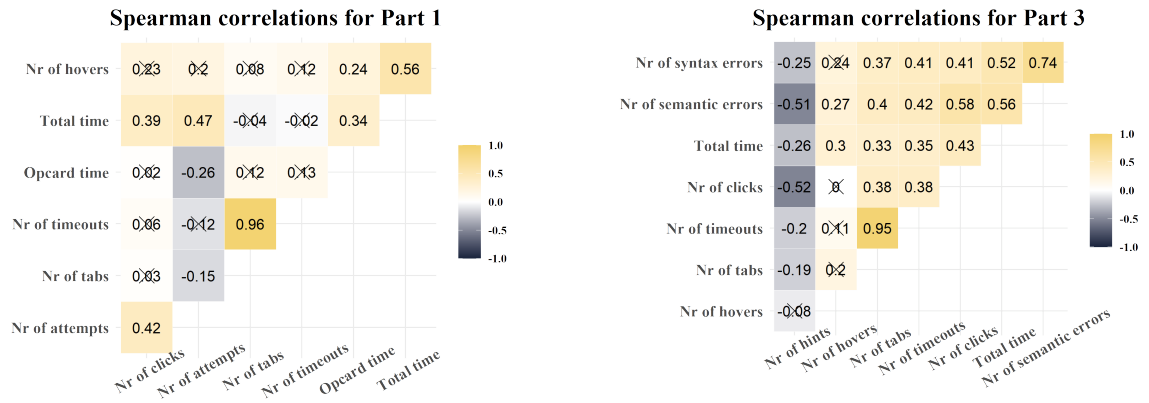
Since the distributions are skewed, we will perform this using Spearman's rank correlation coefficient, which is non-parametric and based on the rank-transformed data. For Part 1, the metrics we analyse are: the number of hovers (i.e. tooltip events), tabs, timeouts, incorrect attempts, as well as total operation card reading time, and total time taken solving the exercises. For Part 3, we include the number of hovers, tab changes, timeouts, menu clicks, syntax errors, semantic errors, and total time. Note that all of these have been assumed to be negatively associated with the underlying competence, since all else equal, a competent data wrangler would presumably complete all exercises in less time, with less inactivity, and fewer menu clicks. The results are visualised as a correlation matrices in Figure 11.4. Note that the sample sizes are the same as in Chapter 10.

For Part 1 (Figure 11.4a) we find that the time on task is moderately correlated ($\rho > .3$) with the number of menu clicks, tooltip events and incorrect attempts, which is expected: if you struggle more with finding the right answers, you will also take more time. It is also correlated with the amount of time spent reading operation cards, which could be explained through inexperience: an inexperienced person is likely to take longer to read, but also to make more mistakes. The correlation between the time spent on operation cards and number of attempts is negative, reflecting the fact that correlations are not necessarily transitive, which could mean that taking the time to absorb the meaning of each operation could lead to a better understanding of which operation to select. Timeouts and tab changes are highly correlated: this makes sense, since a highly distracted person is likely to disengage in either way.

For Part 3 (Figure 11.4b), we find that the number of hints used is negatively correlated with all other variables: although both hint usage and the other variables could be conceived of as indicators of low ability, it is also the case that hints reduce the time and mistakes necessary to solve it. We again find a near-perfect correlation between the number of tabs and timeouts. It is interesting that syntax errors are highly correlated with semantic errors: it suggests that the ability to integrate examples and spot typos, and the ability to select operations and plan solutions, are not as dissociated as we may think. The other variables are positively correlated with one another, which could indicate that they are all aligned with an underlying construct of low data wrangling abilities, supporting our assumption about using these as process metrics to measure competence and learning success.

Implications

Correlation matrices often present complex data patterns that it can be difficult to construct post-hoc narratives around. Our matrices suggest that the metrics of tabs and timeouts are mutually redundant and that, with the exception of hint usage, the metrics are positively correlated with each other, but not so highly as to be superfluous.



(a) Pair-wise Spearman correlation coefficients (ρ) among participant-level process metrics in Part 1. (b) Pair-wise Spearman correlation coefficients (ρ) among participant-level process metrics in Part 3.

Figure 11.4: The crossed out coefficients indicate non-significant results ($\alpha=.05$) for Bonferroni-adjusted p -values. n depends on the variable, since entries of 0 are excluded.

11.3.2 The association between SLICE N DICE parts

In Chapter 7 we explained how a major reason for the segmented design of SLICE N DICE was that, for theory-building reasons, we wanted to isolate the measurement of *conceptual data wrangling knowledge* (Part 1) from that of *general programming skills* (Part 2), both of which are presumed to influence the *programmable data wrangling skills* needed to succeed in Part 3. This presents an opportunity to investigate how performance in one part predicts performance in another part. We will focus our analyses on two metrics: time on task and number of semantic errors. Recall that, in total, 197 participants finished Part 1 completely, 148 participants finished Part 2 completely, and 88/134 participants finished Part 3 (depending on whether you cut off the last 7 exercises). Our exact sample size will depend on the overlap of these.

Time on task

Time on task can be interpreted as efficiency in the problem-solving. If largely influenced by prior experience or general intelligence, we would expect this to be highly correlated across parts. On the other hand, if the operation select and API navigation of Part 1 is a distinct skill from the example adaptation and debugging required in Part 2, then we would expect them to be less associated.

The results are summarised in Figure 11.5. It is clear that time on task in Part 1 is closely associated with that in Part 2 (Figure 11.5a). Running a linear regression, we find that Part 1 is a significant predictor of Part 2 ($\beta=.88$) [$F(1,136)=55.7, p<.001$]. The association between Part 2 and Part 3 (Figure 11.5b) interestingly appears more diffuse, though still significant ($\beta=.67$) [$F(1,106)=22.9, p<.001$]. The association between Part 1 and Part 3 is even flatter (Figure 11.5c), and non-significant ($\beta=.38$) [$F(1,109)=2.8, p=>.99$]. This suggests that performance in

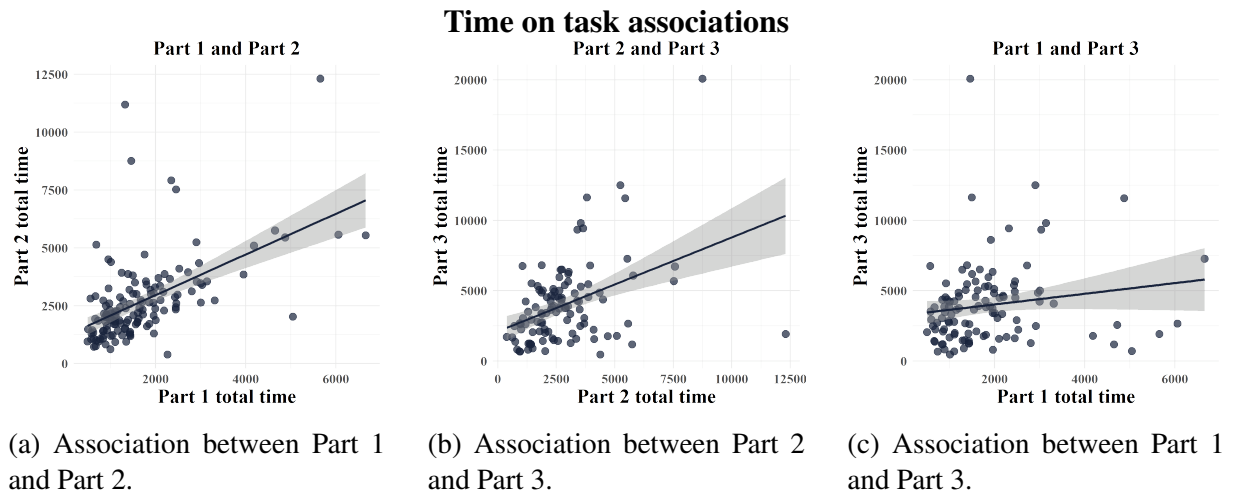


Figure 11.5: Associations between parts in time on task, with lines of best fit obtained through linear regression.

Part 3 is strongly determined by programming skills, and that the conceptual training of Part 1 had less influence on participants' ability to perform the data wrangling using actual syntax.

Number of semantic errors

The other metric, of semantic errors, is operationalised differently depending on the part. In Part 1, this refers to the total number of incorrect attempts they perform in the operation selection part. In Part 2 and 3, it refers to the number of code submissions that were syntactically valid but still incorrect.

The results are summarised in Figure 11.6. Compared with time on task, the correlations are here markedly weaker. Part 1 does not appear to predict Part 2 (Figure 11.6a), as a linear regression yields a non-significant coefficient ($\beta=.13$) [$F(1,137)=2.4, p=.12$]. There is some indication that Part 2 predicts Part 3 (Figure 11.6b), with a significant co-efficient ($\beta=.59$) [$F(1,107)=12.9, p<.001$]. With Part 1 and Part 3, the relationship is interestingly negative ($\beta=-.25$), though only barely significantly so [$F(1,109)=4.1, p<.05$]. This latter pattern could be due to the fact that the learning experiences of semantic errors in Part 1 lead to durable conceptual knowledge that they can take advantage of in Part 3, but it could also be due to hidden variables such as weaker learners being more likely to use hints.

11.3.3 The influence of experience on performance

In our sample we obtained a wide range of previous programming and data wrangling experience, which is likely to have influenced the performance in the programmatic data wrangling task of Part 3. Depending on the nature of their experience, an analysis of the experience-performance associations would be informative in inferring which competencies that programmatic data wrangling depend on.

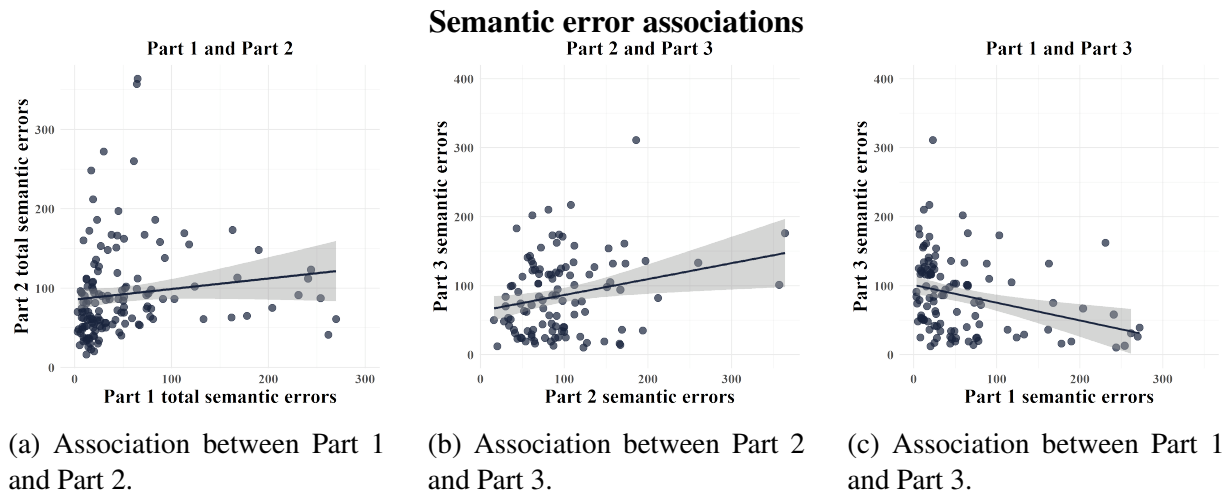


Figure 11.6: Associations between parts in the number of semantic errors, with lines of best fit obtained through linear regression.

Recall that the demographic survey of SLICE N DICE featured 3 main ways of measuring the participants' experience. The first is degree major, which can be coarsely classified into CS (*high*) and non-CS (*low*) as a proxy for programming experience. The second is their self-rated competence (1-5) in the Likert-scale items asking them about their abilities in R and Python NumPy/Pandas. This rating can bin participants into three groups, based on the rating of the language they chose to program in: *low* (<3), *middle* (3) and *high* (>3). Finally, we gave participants a short pretest asking them to check the boxes of various programming concepts to indicate whether they were aware of them. This score had the range 0-8 and could be binned as follows: *low* (<4), *middle* (3<,<5) and *high* (>5). We also used a wide set of metrics for performance: the total time on task, the number of hints used, the number of semantic errors, and the number of syntax errors.

To explore whether any of the experience measures appeared to meaningfully predict any of the performance variables, we created a boxplot grid, shown in Figure 11.7. If predictive, the boxplots would rise monotonically from *high* to *low*, since the metrics are presumed to be negatively correlated with programming ability. For degree major, we find that it appears predictive of the number of syntax errors, semantic errors, and time on task, though the degree of overlap in inter-quartile ranges are generally considerable. With pretest scores, we find that it appears to slightly predict hint usage, but the distributions are highly dispersed. Finally, with self-ratings semantic errors and time on task all appear to rise as the experience gets lower. Overall, degree major seems to be the most informative experience measure, perhaps because it gives participants a deeper understanding of programming syntax and a habit of debugging.

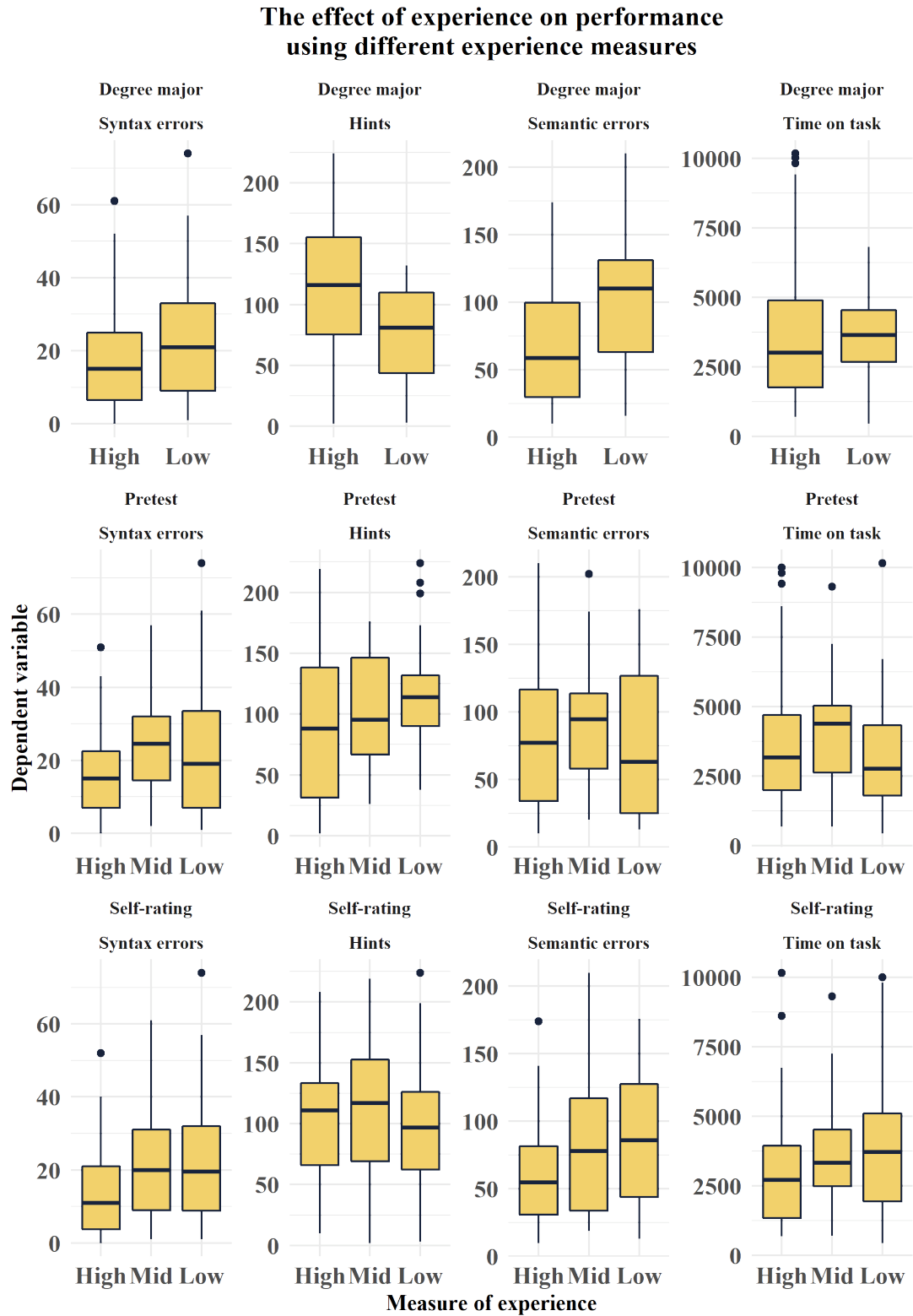


Figure 11.7: Boxplots showing the relationship between experience and performance, as measured by three different experience measures (their degree major, their pretest score, their self-rated knowledge) and four performance metrics (the number of syntax errors, the number of hints, the number of semantic errors, and the time on task, all in Part 3).

11.3.4 The influence of experience on the effect of graphics

While the control group in general did not differ from the experimental group in our SLICE N DICE, it remains possible that the impact is moderated by an expertise reversal effect. It is conceivable, for example, that more experienced participants were negatively impacted by the provision of graphics, cancelling out any positive impact on complete novices. To probe the presence of such an interaction, this section will disaggregate the data based on previous expertise. As a result of our analysis above, we will focus on degree major (CS or non-CS major) as our proxy for experience. Part 1 included 84 CS majors and 113 non-CS majors, while Part 3 included 82 CS majors and 52 non-CS majors. Regarding dependent variables, we will focus on the two main distal variables for Part 1 and 3: total completion time (time on task) and number of attempts. For Part 3, number of attempts represents the total number of errors, including both semantic and syntactic errors. This 2x4 matrix of groups will be analysed for both subgoal graphics and thumbnail graphics.

We display our results for subgoal graphics in Figure 11.8. In it we find descriptive evidence that for Part 1's number of attempts, subgoal graphics are associated with fewer errors in both experience groups. Part 3's number of attempts and Part 3's time on task are both consistent with expertise reversal, in that the non-CS condition is faster/less erroneous with graphics, while the CS majors are slower/more erroneous with graphics. For Part 1's time on task, neither experience group appears discernibly impacted by the intervention. None of these dependent variables exhibited a significant interaction between degree and condition, as determined by a permutation-based ANOVA ($\alpha=.05$).

The results for thumbnail graphics are shown in Figure 11.9. Here we find data patterns consistent with expertise reversal in Part 1's number of attempts and Part 1's time on task, but that graphics are consistently detrimental in Part 3's number of attempts. For Part 3's time on task, CS majors appear to benefit from graphics, but non-CS majors appear negatively affected. We again find no inferential evidence supporting the case for an interaction, since all were non-significant as per a permutation-based ANOVA ($\alpha=.05$).

We conclude that there is no consistent support for the hypothesis that the effect of graphics is experience-dependent, at least not on key performance variables, with the possible exception of subgoal graphics in Part 3 and thumbnail graphics in Part 1. The lack of consistent effect could be due to limitations to degree major as a proxy: it is self-reported, a choice among fixed options that may not map neatly on to all backgrounds (*Statistics, Medicine, Mathematics, Earth, Psychology, Physics, Humanities, Other*), and each group is likely to be heterogeneous, especially the non-CS group. It could also be that different outcome variables had different responses. For example, in the relatively simple Part 1, subgoal graphics may have been useful as a mechanical lookup cue for both experience groups, while Part 3 may have been complex enough for the graphics to be used in different ways depending on experience. For experienced programmers, it may then have been a redundant distraction, while for non-CS majors it may

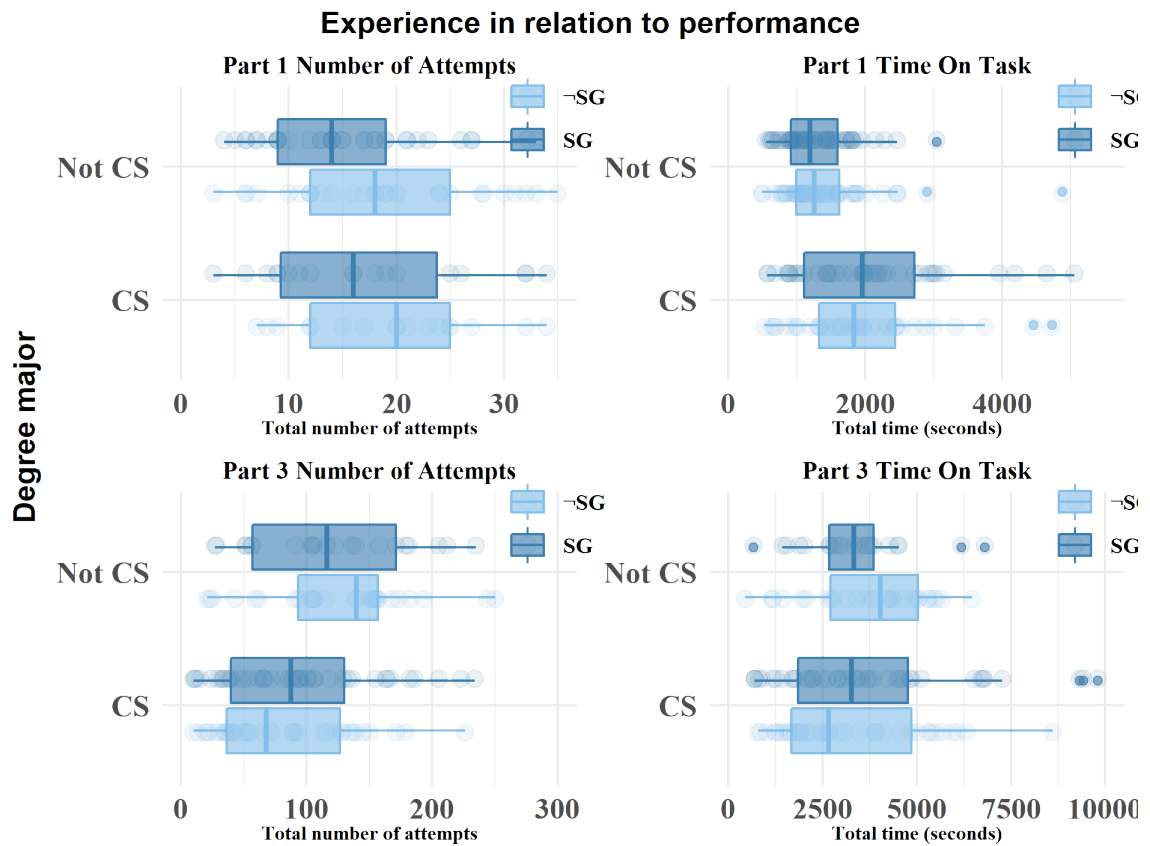


Figure 11.8: Box plots showing group differences associated with subgoal graphic condition. The box plots indicate medians and inter-quartile ranges. Each graph has been truncated along the x-axis, omitting between 1 and 6 outliers in order to display differences more clearly.

have provided crucial scaffolding.

In the future, finer interaction data such as eye-tracking would have been useful for quantifying the actual time spent engaging with the graphics, and whether the groups varied qualitatively in their behaviour. It is possible that CS majors inspected the graphics out of curiosity more than out of need, whereas non-CS majors returned to them more actively in search of help. Although our study included CS majors primarily due to resource limitations, it may be desirable for future researchers to either restrict inclusion criteria further so as to get a more homogeneous baseline, or to investigate potential expertise reversal more intentionally by giving participants a validated pre-test.

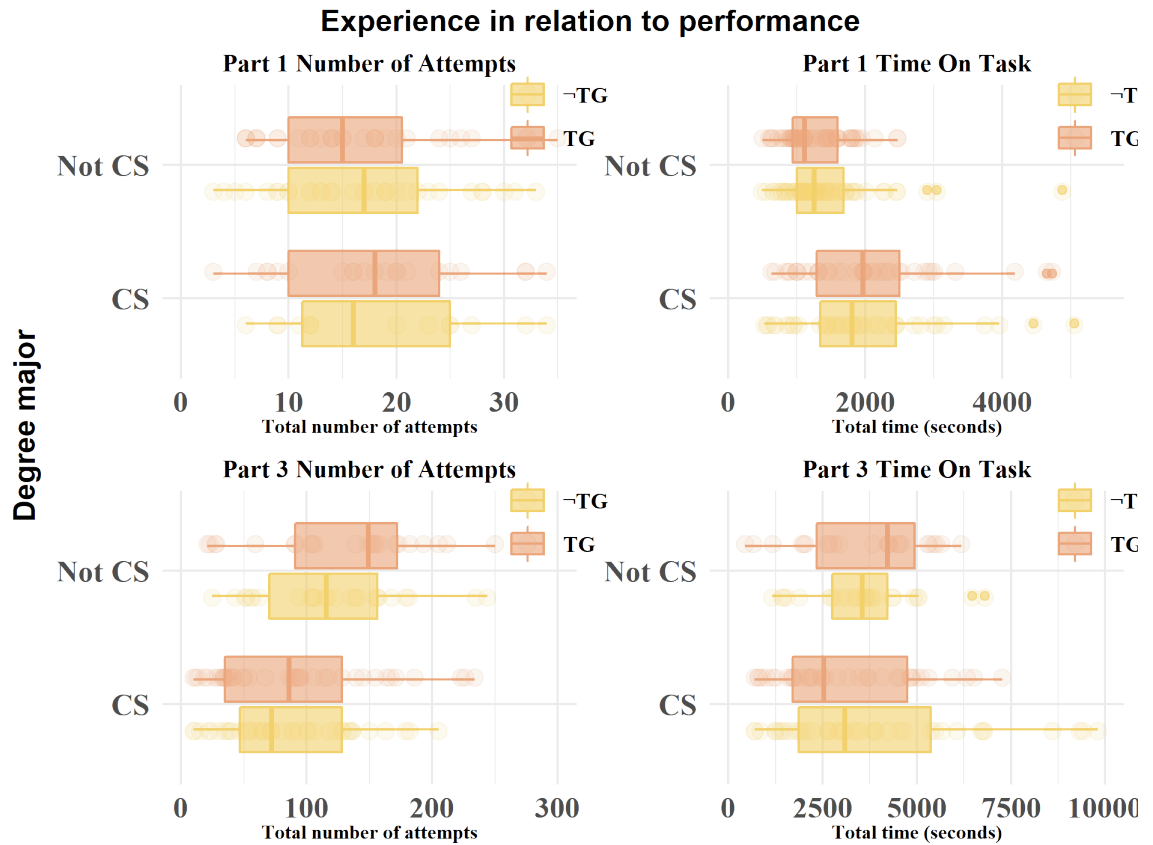


Figure 11.9: Box plots showing group differences associated with thumbnail graphic condition. The box plots indicate medians and inter-quartile ranges. Each graph has been truncated along the x-axis, omitting between 1 and 6 outliers in order to display differences more clearly.

Chapter 12

Discussion

This dissertation began by arguing that the need for effective data wrangling education is ever-growing, while the empirical research focused on addressing this demand remains minimal. It went on to argue that programmatic data wrangling comes with a distinctive set of barriers and opportunities. Barriers, because data wrangling APIs tend to be wide and complex, complicating the problem decomposition into API commands and requiring a continual search for code examples, which for a novice can be both difficult to find and integrate. Opportunities, because data wrangling involves two-dimensional data structures that are relatively straightforward to visualise. We proposed three types of graphics that plausibly could serve as scaffolding in a tutorial or e-learning context: subgoal graphics in support of plan composition, thumbnail graphics in support of API lookup, and parameter graphics in support of example adaptation. From this trio of interventions, three main research questions were posed - RQ1, RQ2, and RQ3 - about the pedagogical effects of subgoal, thumbnail and parameter graphics, respectively.

These research questions motivated two RCT studies: one small formative study on the effect of parameter graphics, and one large, multi-institutional capstone study involving a special-built tutorial platform called SLICE N DICE that simultaneously studied the effect of both subgoal and thumbnail graphics. While focused on causal attribution, the research maintained a broad perspective on the notion of “pedagogical effects”, and compared group differences in terms of a variety of dimensions, including both behavioural process metrics and subjective reflections. These RCTs were preceded by 3 pilot studies (1 for parameter graphics, 2 for SLICE N DICE) and further supported by 2 validation studies. We concluded the research project with a collection of smaller, exploratory analyses that bring added nuance to the SLICE N DICE data set.

In this discussion chapter, we recapitulate the key findings from each study and propose interpretations of the overall data pattern. We will also discuss what the results imply for the practices of instructors, interface designers, and researchers in data wrangling education. We will also reflect on how aspects of our research methods may have affected our results, how the space of methodological trade-offs and dilemmas could have been navigated differently, and

what a researcher could learn from those experiences. Finally, we will sketch out remaining research questions and propose how the research programme of data wrangling education could be expanded.

12.1 Summary of findings

Our three research questions asked about the effect of each graphical type on the learning of data wrangling, with the "effect on data wrangling" kept deliberately broad, and studied through the lens of multiple metrics. In this section we summarise the results of that multivariate analysis.

12.1.1 RQ1: The effect of subgoal graphics

Subgoal graphics were evaluated quantitatively in the final RCT, and qualitatively in the RCT, the preceding two pilot studies, and the survey-based validation study. In terms of positive findings, the **Part 3 completion rate appeared meaningfully higher among those with subgoal graphics**: 59% of those who completed Part 3 were in SG, despite SG participants only constituting 50% of participants who began the part. In both Part 1 and 3, the subgoal group committed fewer semantic errors, though not significantly so. In terms of negative findings, subgoals did not meaningfully reduce time on task in either part, or impact the performance in unscaffolded exercises. Moreover, **those with subgoal graphics appeared to use more hints**, though this was not significant.

The qualitative data indicated that **both the subgoal labels and the subgoal graphics were broadly perceived as helpful**, as evidenced by numerous unprompted comments. Participants reported using the graphics to verify their own understanding and when deciding which strategy to use. The Likert-scale data asking about subgoal graphics directly in SLICE N DICE suggested that a clear majority of respondents found the feature "very helpful". There are also some indications in the rating data that subgoal graphics led to improved levels of concentration and motivation.

12.1.2 RQ2: The effect of thumbnail graphics

Thumbnail graphics were evaluated quantitatively in the final RCT, and qualitatively in the RCT, the preceding two pilot studies, and a validation study. Quantitatively, graphics do appear to reduce the operation card reading times, consistent with expectations, but this was non-significant. In both Part 1 and Part 3, the group with thumbnail graphics sought significantly fewer clarifications via the tooltips. In Part 1, thumbnails were associated with fewer errors, an effect that appeared visually but was not significant. On the negative side, thumbnails do not appear to impact distal variables such as completion rate, time on task, hint usage, or programming-related

errors. Surprisingly, they also did not reduce the number of menu clicks, which we had used as a proxy for navigational efficiency.

The qualitative data suggest that participants appreciated the graphics, although the tooltips appear to have been more helpful than the thumbnails themselves. These indications were also supported in the validation study, which showed that participants prefer thumbnails with data, which is exactly what tooltips provide. In the context of operation cards, pilot data suggest that the graphical version was generally preferred over the textual one. The TG appeared to rate the thumbnail/tooltip feature as more helpful in Part 3, though not dramatically so.

12.1.3 RQ3: The effect of parameter graphics

Parameter graphics were evaluated in a small-scale RCT, namely the Study 2 documented in Chapter 5, in which participants completed a paper-based tutorial and three sets of example adaptation exercises, which were aided by cheat sheets either with or without parameter graphics. It compared the groups' total time on task, which had been hypothesised to be shorter for the parameter graphic condition. Contradicting this hypothesis, the control group was significantly faster, with a large Cohen's d of -1.1, meaning that the difference was larger than $1SD$. This result should be interpreted in the context of considerable measurement error (time was recorded manually) and sample error ($n=44$), but is nonetheless noteworthy. In terms of accuracy scores, we found no evidence for a difference, as the control group was far more dispersed than the experimental group. Therefore, parameter graphics do not appear to impact accuracy.

12.2 Reflections

12.2.1 Subgoal graphics

Why the lack of effect on objective metrics?

Although subgoal graphics appear to have positively affected retention and possibly also the number of semantic errors, it is nonetheless surprising that they did not translate into shorter time on task. This could be due to inattention. We saw in the quantitative pilot study that participants rarely consulted them due to the subgoals' availability within the IDE. It is possible that, even after the interface changes that this prompted, people mainly relied on the subgoal labels. Alternatively, if participants carefully perused the graphics, this would have consumed time that other performance benefits may not have compensated for.

Another possibility is redundancy. SLICE N DICE had been carefully designed to provide an accelerated, frustration-free tutorial and employed multiple scaffolding devices to ensure recruitment, retention, and timely completion. It is possible that, in so doing, the scaffolding reached a saturation point, and that in the presence of subgoal labels and hints, graphics provided

no additional measurable performance gains. If the labels and hints had been removed, the group differences may have been starker.

It is interesting that, though insignificant, the difference in the number of semantic errors is more favourable to subgoal graphics than the difference in syntactic errors. This makes sense given how subgoal graphics were meant to affect plan composition and the selection of operations: the risk of mistypings would likely remain the same regardless.

Why the disparity between subjective and objective results?

Across the various studies, participants were consistently positive in their feedback about the subgoal graphics. It is clear that the subgoal graphics were well-liked and perceived as helpful. As a relatively conspicuous feature, they are likely to have noticed it. In providing an almost literal insight into the workings of the solution, it may have a motivating effect and relieved feelings of frustration, which could explain the strikingly higher completion rates among participants who received them. But why did this not translate into faster times on task?

One possible explanation is faulty meta-comprehension. There is a decent-sized literature documenting instances in which participants believe a pedagogical feature to be useful, but when tested, this understanding turns out to be illusory [333, 334]. Serra and Dunlosky have called the fact that learners generally endorse multimedia materials as the *multimedia heuristic* [335], suggesting that they use images as a cue to estimate the accuracy of their understanding, instead of more valid strategies such as self-explanation. However, this explanation is not entirely convincing, as participants received repeated and relatively immediate feedback, limiting the extent to which their perceived understanding could veer from the actual understanding. Another explanation is that they deemed the subgoal graphics “very helpful” in the sense of “theoretically they could be useful”, but that they never relied upon because of the presence of other scaffolding devices. Indeed, although the effect is non-significant, there were some indications that participants with subgoal graphics were more likely to use hints. Perhaps the presence of graphical hints made the usage of further hints seem more permissible.

12.2.2 Thumbnail graphics

Why fewer tooltip events?

The most remarkable finding from the thumbnail data was that in both Part 1 and Part 3, participants who received thumbnail graphics tended to use the tooltip less frequently. Since the tooltip contained further explanation and disambiguation, this suggests that there were repeated instances in which participants without thumbnails struggled to understand what an operation entry meant (despite the label) but where thumbnails would have provided a sufficient explanation. This, in turn, indicates that thumbnails are not redundant and that they do increase the usability of the menu.

This is not the only conceivable explanation. A less favourable account would be that, since the control condition provided a question mark icon in order to show where to trigger tooltip, the control condition provoked more curiosity, which led users to explore the tooltips even when they did not have to.

Another aspect is the tooltip quality. The control condition used the tooltips more frequently even though the tooltips themselves were hypothesised to be worse, since they represented examples textually instead of graphically. In the operation cards (which used the exact same graphics), those graphics also appeared marginally faster to read (albeit non-significantly so). On the other hand, if a user does not understand what an operation does, then it would not matter if the tooltip is suboptimal, as long as it is adequate in its explanation.

Why the lack of impact on performance levels?

The lack of effect on completion rate, time on task and number of errors (in Part 3) may not be that surprising, given the relative subtlety of the intervention compared with subgoal graphics. Menu-proximal variables like the number of clicks and tooltip requests appear somewhat impacted, but the ultimate performance is impacted by numerous factors, among which API lookup ability plays but a small role compared with, for example, debugging skills.

12.2.3 Parameter graphics

Why the longer time on task?

The time on task data on parameter graphics contradicted the hypothesis that the graphics would *accelerate* the activities. If assumed to be true, there are several possible explanations for this. Learning the meaning of icons incurs a certain time cost. The activity of mapping problem context elements to parameter graphics similarly consumed time, and by including the parameter graphics along with thumbnail graphics, the cheat sheets contained more information to read. The novelty, combined with their more symbolic (as opposed to depictive) qualities, may have meant that parameter graphics exemplify a counter-productive instance of graphical scaffolding.

How useful are cheat sheets?

Our single-group study on parameter graphics, Study 1 in Chapter 5, found that 50% of complete novices managed to complete the three cheat sheet-based exercises in less than 2 hours, without sacrificing accuracy. In terms of scores, SQL exercises were solved with significantly higher accuracy than Python Pandas and Tidyverse. While completion rates are hard to interpret in the absence of a benchmark, it suggests that the exercises could be easily slotted into an introductory tutorial session, and that SQL education in general could benefit from incorporating cheat sheets.

12.3 Implications

12.3.1 Should instructors use subgoal graphics?

Our data do not give us reasons to expect that the inclusion of subgoal graphics would lead to dramatic acceleration in data wrangling skill acquisition. However, even in the absence of consistent positive results in the objective metrics, subgoal graphics were remarkably well-received on a subjective level. The perceived helpfulness could be intrinsically useful and likely did play a motivational role in making SG participants persist for longer, which explains why they are over-represented among those who completed Part 3. Instructors should therefore consider incorporating subgoal graphics into their teaching if the motivation among learners is low, and possibly also when students' apprehension or frustration levels are high.

12.3.2 Should instructors use thumbnail graphics?

Based on the repeated finding that thumbnail graphics reduce the need for clarification of what a particular operation does, we recommend the use of thumbnail graphics. These could be used in software menus, but also paper-based cheat sheets, textbooks and other documentation sources. Data from our validation survey suggests that - assuming that the space exists - users will prefer a graphic that additionally contains data, as this helps disambiguate the meaning of a graphic if colour highlighting and shapes are not sufficient.

12.3.3 Should instructors use parameter graphics?

Our data provide no strong grounds for recommending the usage of parameter graphics. However, this does not mean that the underlying reasoning should be dismissed. The act of explicating *which* pieces of information to look for in a problem statement, and of standardising documentation so as to clarify *where* to put those pieces, are probably still worthwhile pursuits, and can be done without graphics, as exemplified by Qian's cue-focused approach [291]. It also does not mean that graphical scaffolding for query writing should be outright dismissed: as reviewed in Chapter 3, several promising approaches exist and await evaluation.

The most serious drawback to parameter graphics, and the main reason for why we chose not to pursue it further, is their lack of extensibility. Although they can easily be designed for basic relational operations, extending the gallery of icons to encompass an entire data wrangling API would not be practical, since this would require them to effectively become arbitrary symbols, and therefore no better than opaque parameter names.

12.4 Limitations

The capstone study had many methodological strengths. Being randomised and controlled, it allowed for rigorous causal attribution. Being multi-institutional, reflective of typical MOOC conditions, and relatively large compared with most programming education RCTs [271], it provided generalisability. Both of these advantages come with a flip-side: rigorous experimental designs and multi-institutional studies have a tendency of reducing effect sizes and increasing sample error, respectively [37, 264]. In light of this, the scarcity of positive results may be attributable to low power, but strengths in the design mean that the results are relatively reliable. Beyond statistical power, a number of limitations of the SLICE N DICE study can be identified, regarding the tutorial's scaffolding, interface, and data collection.

12.4.1 Tutorial improvements

The most important weakness of the study overall related to potential over-scaffolding. Even absent any graphics, SLICE N DICE provided informative test cases and error messages, subgoal labels, and 3 hints for each subgoal. It did so for valid reasons: past experiences with conducting studies had made us well aware of the difficulty in recruiting and retaining volunteers for studies, especially since the study had to be prolonged (to cover all contents), challenging (to avoid floor effects) and without the peer support of in-person tutorials (due to COVID-19). To minimise frustration, we vested the tutorial with several scaffolding devices, which may have attenuated the added pedagogical value of graphics. Therefore we would recommend that, for researchers who have the opportunity and ethical clearances to do so, embed their e-learning tutorial within a course such that participants' performance impacts their grade. An alternative method that we could have used would be to gamify it and penalise students for using hints other than graphics. Given the time, the assistance dilemma could also have been addressed through a more advanced, adaptive scaffolding.

Another significant weakness was more of a wasted opportunity than a direct threat to validity. Although we did provide exercises that were completely unscaffolded for 10 minutes, this was probably a quite insensitive measure, and it would have been more interesting to include exercises that were unscaffolded for the entire duration. This would provide more unequivocal *product metrics*, giving insight into more durable effects of graphics beyond the efficiencies revealed by process metrics. After all, scaffolding is defined as an intervention that enables a learner to do something they could not, once the scaffolding is removed [186]. Relatedly, it would have been interesting to present tasks with less scaffolding after a delay, to gauge how durable the knowledge is. The reason why we put a time limit on the scaffolding removal was due to the recruitment quandary, but the open-ended feedback did reveal requests for less scaffolded testing, suggesting that our fears of frustration-induced dropout may have been misplaced.

12.4.2 Possible interface improvements

There are several interface aspects that in hindsight could be improved. The fact that the menu was collapsible meant that participants did not have exposure to thumbnails unless they had already expanded the super-ordinate node. It is possible that thumbnail-related effects would be amplified if the menu was constantly expanded. The interface's layout could theoretically have been changed so that the subgoals were in a separate panel and therefore permanently visible: the fact that the subgoals and documentation were shuffled in and out may have made participants less inclined to use the former. For analytical purposes, it would have been useful if the *Run* and *Submit* button in the programming area were replaced by a single button, or that usage of the *Submit* button were limited, so that we could separate semantic errors from playful code experimentation more cleanly. In hindsight, we probably should have removed the hyperlinks from the hints to avoid having such indirect documentation access confound our menu clicking data.

Regarding usability, there were certain feedback that was repeatedly voiced but not acted upon due to various trade-offs. Among them were the ability to backtrack to previous solutions, to provide a larger programming area, and add code execution capabilities to Part 1.

12.4.3 Possible data collection improvements

There were also data we wish we had collected but did not. Although gaze position and cursor movements are not perfectly correlated [336], it would nonetheless have been useful to collect more mouse movements. Currently we only collected clicks on graphics, for example, but it is likely that many participants attended to the graphics while mousing over them, though not clicking on them. Similarly, it would have been useful to track the cursor's motion within the documentation and the coding area, as a proxy for active engagement.

Another choice we regret was to only collect programming snapshots when the participant ran or submitted their code. This meant that participants reluctant to execute their code without first completing it would have registered fewer semantic and syntactic errors. It would have been more useful to record a snapshot of the code's state at regular intervals.

Finally, although we had reasons for doing so (a wish to avoid front-loading the study), we wish we would have given participants a more fine-grained pretest instead of the Likert-scale ratings and checkbox-based pretest, as these produced a rather coarse measure of their prior experience.

12.5 Future work

There remains several areas of low-hanging fruit to explore by future researchers and practitioners. The graphics could be incorporated into tooltip plug-ins or documentation sources and thus

be put to use in more realistic environments. A study of public code repositories could help chart the distribution of API command usage for NumPy/Pandas and Tidyverse/Base R, which in turn could help focus pedagogical efforts.

Beyond addressing the above-mentioned limitations, it remains to be determined how reproducible data wrangling graphics are, in the sense of whether other instructors can re-create similar graphics on their own to use in their classes. If the graphics are tied to those hard-coded into SLICE N DICE, adoption would be compromised. Although we have not attempted it ourselves, we see no reason why the graphics in principle cannot be generated automatically. Finally, we would like to see graphics extended to a wider variety of commands, for example visualising lambda functions and statistical programming, which often makes use of loops.

Another dimension that we have not explored relates to the constructivist continuum, and whether self-generated subgoal graphics are more helpful than instructor-provided graphics. It would moreover be interesting to see how learners “in the wild” visualise data wrangling procedures, either through mental representations or external sketches, and whether this imagery matches the graphics in SLICE N DICE.

SLICE N DICE included several non-graphical scaffolding devices that it did not experimentally manipulate. Some of these have been examined in previous literature, such as hints and error messages, but the main ones are largely unexplored: the use of a command menu in programming environments, and the use of subgoal labels to scaffold exercises, as opposed to worked examples. We would encourage more research into these, and especially into how command menus compare with search engines. Programmers’ web usage has been researched before (e.g. [153, 244]) but only rarely in the context of novices. Furthermore, due to the shortcomings of parameter graphics, we are calling for more research into the barriers novices encounter during their code example adaptation.

12.6 Finale

The work of this dissertation presents an inquiry into the pedagogical effects of subgoal graphics, thumbnail graphics, and parameter graphics. We found that subgoals graphics were well-liked, perceived as helpful, and appeared to have boosted completion rates, but that objective, behavioural metrics did not otherwise register any significant or practically meaningful effects. We found that thumbnail graphics reduced the number of clarifications needed when participants navigated the API command menu, but that this did not translate into shorter time on task or fewer attempts. Finally, we found that parameter graphics appeared to *increase* time on task but not affect response accuracy.

Improving and optimising data wrangling education remains an important cause, and this dissertation has served to explore one part of its design space. As more learners across disciplines face expectations to learn data wrangling, providing them with an enjoyable, motivating,

and effective experience can lead to lasting confidence to pursue scientific computing. Sub-goal and thumbnail graphics may serve this role, of making the data wrangling domain appear comprehensible and tractable, and it remains for future research to identify supplementary interventions for improving learners' objective performance levels.

Appendix A

Parameter graphic tutorial

Let's start with some demographic info:

What is your gender?

- Female
 Male
 Other

What is your age?

How easy do I find it to read and understand English?

Not easy at all Generally OK Very easy

Other (please specify)

Which languages are you comfortable processing data in?

	No experience at all	Some exposure	I can do basic tasks	I can do moderately complex tasks	I have a very advanced experience
Excel	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
SQL	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
R	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Python	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Please proceed to read this booklet at your own pace.

Please avoid distractions to the greatest extent possible (smart-phones, etc.) so that you stay concentrated on the topic.

I hope you will find the guide interesting.

How tables store data:

This experiment is about how to retrieve information from tables. You are probably already familiar with tables, if you've for example used Excel, but the rules are worth repeating.

In the world, there are many different categories of things, for example movies, teachers and books.

Each category is represented as a table.



These categories have many different attributes. For example, every movie has a name, a director, and a release date.

Each attribute is a column in the table.



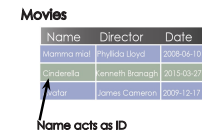
For each category, there are many cases. For example, many different movies.

Each case is represented as a row in the table.



The order of the rows and columns within a table doesn't carry any meaning.

All rows should be unique (without duplicates). This means that at least one column acts like a unique ID. For now, let's pretend that no two movies with the same name exist. That makes "Name" the key.



And by the way, let's symbolise a table as a purple rectangle:

Easy peasy! Now read on about how to get information from tables ("query" them) ->

Querying tables:

The following are the main operations used for queries. Each has an associated "component".

Selecting rows

Sometimes we wish to select certain rows based on a criterion. For example, we only want the cases whose Date is later than 2000-01-01.

Let's symbolise this criterion as a green row:

- Director is "James Cameron"
- Name is not "Mamma Mia!"
- Date is in January

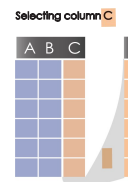


Selecting columns

We may only be interested in one or several attributes (columns). For example, we may ultimately only be interested in the Directors.

Let's symbolise this list of columns as an orange column:

- Date
- All
- Date, Name

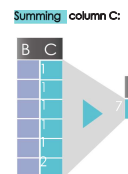


Aggregating columns

We often want to "aggregate" one or several columns. This means finding a single-value summary for it. For example, we may want to find the number (count) of all directors.

Let's symbolise this aggregation function as a cyan rectangle:

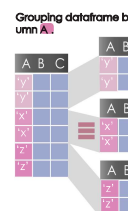
- Sum
- Average (mean)
- Max



Grouping rows before aggregation

Whenever we aggregate a column, we often wish to have a separate summary value for each separate group of data. For example, suppose the Movie table had a column "Genre". Then we may want a count of all movies in the thriller-genre, in the romcom-genre and so on.

Let's symbolise the column we "group by" as a red rectangle:



Piecing the operations together:

In order to process a query, we need to translate a natural English sentence into these types of query components. For example, a typical query would be:

"For all cases where B is above 4, what is the average C for all cases, grouped by A?"

Which rows are wanted? Which columns? Does it have to be aggregated? Grouped before aggregated? We can start annotating it...

"For all cases where B is above 4, what is the average C for all cases, grouped by A?"

We only seem concerned with rows where B > 4

This is the aggregation function required

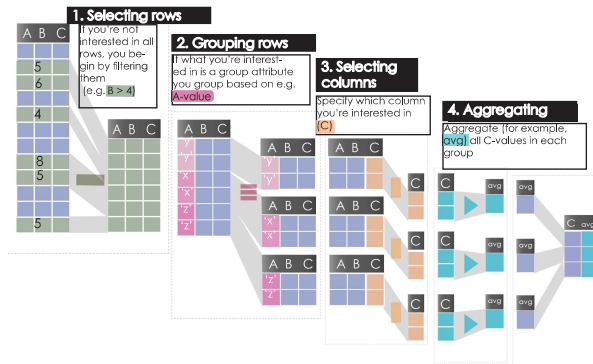
C is the column we eventually want the average

A must be a categorical attribute that splits all rows into groups

We can summarise what each types of component corresponds to:



Then, have a look at how these components are used for a sequence of operations that start with the original table and returns the information that the query asks for.



Don't care too much about ordering. Our focus is on the ability to translate English into our list of components.

Joining tables

Sometimes a query asks of us to select rows or columns based on information that is stored in a different table.

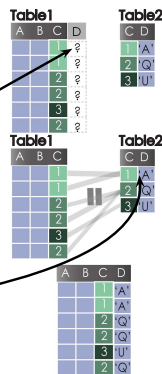
For example, suppose you want "All A where D='Q'". Column A is in Table1 but column D is in Table2 so we cannot execute the query based on Table1 alone.

This requires of us to join the tables together based on some shared column, that tells us which Table2-rows we are supposed to attach to which Table1-rows.

We find that both tables have C in them, representing the same attribute ("C=C"). We can let "C=C" act as a glue, or "joining condition". Let's symbolise it with two grey columns:

If a row in Table1 has C = 1, then we will glue the Table2 row with C = 1 on to it (giving us D = 'A').

Doing the query then becomes easy and we can go through the usual steps. We know that "All A where D='Q'" will return the A-value of rows where C = 2.



Let's clarify with two queries that require a combination of information from a Dogs table and Owners table.



Find the **name** of dogs whose **owners live in London**.



What are the **dogs** of the **owners with age above 80**?



Practicing colour-coding:

In the examples below, two queries have been provided for the Employees table. The query components have been written next to corresponding symbol:

Employee_id	Name	Income	Position	Contract

Find the **names** of all employees with **fulltime contract**.



Find the **average income** for all kinds of **contracts**.



To get into the habit of colour-coding queries, please do the same thing for these queries on the Dogs table:

DogName	Owner	Breed	Height	Weight

What are the **weights** of the dogs called **Fido**?



What is the **total weight** of all dogs called **Fido**?



How many dogs are there above height **50cm**?



What is the **number** of dogs for all different **breeds**?



Please try out joins yourself in the following two exercises.

Companies

Name	CEOName	YearFounded	Goods

CEOs

CEOName	Income	CarBrand

Find the **car brand** of the CEO of "H&M".



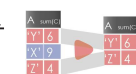
Find the **companies** whose CEO earns more than a **million pounds**.



Filtering aggregates

Suppose we did an aggregation-query that was only interested in some groups' aggregates, like "the sum of C where the sum is less than 7, grouped by A".

Selecting rows where sum(C) < 7



In other words, once we have aggregated groups, we *select aggregates the way we previously selected rows based on a criterion*. Let's symbolise these aggregation criteria as a red triangle.

Let's clarify selection of group aggregates using the same tables from before:

Dogs

DogName	Owner	Breed	Height	Weight

Owners

Owner	Age	Location

Find the **owners** of all dog breeds whose **tallest dog is above 60cm**.



Which dog breeds have an **average weight above 20 kg**?



Thank you for completing the training part of the study.

Please have a short stretch or break if needed, ask questions if you have any, then approach me for Part II.

Appendix B

Multilingual cheat sheets

SQL

Using the syntax guide to the left, please translate the following queries into SQL using tables Flights and Countries.

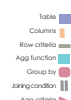
Flights				Countries		
Flight_id	Destination	Departure_from	Charter	Passengers	Destination_in_EU	Capital

Give the destinations of all flights.



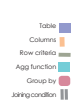
```
SELECT DISTINCT(Destination)
FROM Flights
```

What is the total number of passengers?



```
SELECT SUM(Passengers)
FROM Flights
```

Get all flights IDs for flights arriving in EU countries.



```
SELECT Flight_id
FROM Flights INNER JOIN Countries ON
Flights.Destination = Countries.Destination
WHERE in_EU = 1
```

Find the IDs for all charter flights.



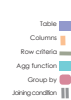
```
SELECT DISTINCT(Flight_id)
FROM Flights WHERE Charter = 1
```

Get the nr of flights per destination.



```
SELECT COUNT(*), Destination
FROM Flights GROUP BY Destination
```

Which non-EU countries are destination to more than 4000 passengers in total?



```
SELECT Flights.Destination
FROM Flights INNER JOIN Countries ON
Flights.Destination = Countries.Destination
WHERE in_EU = 0 GROUP BY Flights.Destination
HAVING SUM(Passengers) > 4000
```

Syntax:

Selecting columns



```
SELECT
FROM Table;
```

Selecting rows



```
SELECT *
FROM Table
WHERE Age > 18;
```

Selecting both



```
SELECT
FROM Table
WHERE Age > 18;
```

Aggregating columns



```
SELECT AVG(Age)
FROM Table;
```

Grouping rows before aggregation



```
SELECT AVG(Age)
FROM Table
GROUP BY Class;
```

Filtering aggregates



```
SELECT AVG(Age)
FROM Table
GROUP BY Class
HAVING AVG(Age) > 25;
```

Joining tables



```
SELECT *
FROM Table1 INNER JOIN Table2
ON Table1.Name = Table2.Name;
```

Example:

```
SELECT Name
FROM Table;
```

```
SELECT *
FROM Table
WHERE Age > 18;
```

```
SELECT Name
FROM Table
WHERE Age > 18;
```

```
SELECT AVG(Age)
FROM Table;
```

```
SELECT AVG(Age)
FROM Table
GROUP BY Class;
```

```
SELECT AVG(Age)
FROM Table
GROUP BY Class
HAVING AVG(Age) > 25;
```

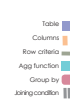
```
SELECT *
FROM Table1 INNER JOIN Table2
ON Table1.Name = Table2.Name;
```

R Dplyr

Using the syntax guide to the left, please translate the following queries into R Dplyr using tables Bands and Musicians.

Bands				Musicians				
Bandname	Nr_members	Income	Genre	Name	Instrument	Bandname	Startyear	Endyear

Find all the members of ABBA.



```
Musicians %>% filter(Bandname == "ABBA")
```

What is the total nr of genres?



```
Bands %>% group_by(Genre)
%>% summarise(n())
```

Which musicians are members of several bands?



```
Musicians %>% group_by(Name)
%>% summarise(n = n())
%>% filter(n > 1)
```

Find all instruments played by musicians with a null Endyear.



```
Musicians %>% filter(is.null(Endyear)) %>%
select(Instrument)
```

What is the mean band income per genre?



```
Bands %>% group_by(Genre)
%>% summarise(mean(Income))
```

Which guitarists are in bands that on average earn more than £100 000?



```
inner_join(Musicians, Bands,
by=c("Bandname", "Bandname"))
%>% filter(Instrument == "Guitar")
%>% filter(Income > 100 000)
```

Syntax:

Selecting columns



```
table %>% select()
```

Selecting rows



```
table %>% filter(==)
```

Selecting both



```
table %>% filter(==) %>%
select()
```

Aggregating columns



```
table %>% summarise(())
```

Grouping rows before aggregation



```
table %>% group_by(Class) %>%
summarise(())
```

Filtering aggregates



```
table %>% group_by(Class)
summarise(()) %>%
filter(=)
```

Joining tables



```
inner_join(table1, table2,
by=c("Name", "Name"))
```

Example:

```
Table %>% select( Name)
```

```
Table %>% filter(Age > 18)
```

```
Table %>% filter(Age > 18)
%>% select(Name)
```

```
Table %>% summarise(mean(Age))
```

```
Table %>% group_by(Class)
%>% summarise(mean(Age))
```

```
Table %>% group_by(Class)
summarise(mean(Age))
%>% filter(mean(Age) > 25)
```

```
inner_join(Table1, Table2,
by=c("Name", "Name"))
```


Python Pandas

Using the syntax guide to the left, please translate the following queries into Python Pandas using tables Recipes and Cookbooks.

Recipes

Name	Author	Nr_served	Type	Cost

Cookbooks

Bookname	Author	Cuisine	in_Paperback

Which recipes are of type dessert?



```
Recipes[Recipes.Type == "Dessert"]
```

What would it cost to cook all the recipes?



```
Recipes.Cost.sum()
```

How many authors that have written a book on Italian have written a recipe of type "dessert"?



```
df = Cookbooks.merge(Recipes, on='Author')
df[df.Type=="Dessert"].Author.count()
```

What are the cuisines of Jamie Oliver's books?



```
Cookbooks[Cookbooks.Author == "Jamie Oliver", Cuisine]
```

What is the average number of people served for the different recipe types.



```
Recipes.groupby('Type')['Nr_served'].mean()
```

Which authors that have at least one paperback book published have also written more than 80 recipes?



```
df = Cookbooks.merge(Recipes, on='Author')
df = df[df.in_Paperback == 1]
df.groupby('Author').filter(lambda x: x.Name.count() > 2).Name.unique()
```

Syntax:

Example:

Selecting columns



```
Table.Name
```

Selecting rows



```
Table[Table.Name > 18]
```

Selecting both



```
Table[Table.Age > 18].Name
```

Aggregating columns



```
Table.Name.mean()
```

Grouping rows before aggregation



```
Table.groupby('Class')['Age'].mean()
```

Filtering aggregates



```
Table.groupby('Class').filter(lambda x: x.Age.mean() > 25)
```

Joining tables



```
merge(Table1, Table2, on='Name')
```

Appendix C

Slice N Dice taxonomy

Create

vectors

- without names
- with names
- arithmetic sequence
- from smaller vectors
- by repeating vectors

matrices

- by folding a vector
- from multiple vectors

dataframes

- from vectors
- from a matrix

Access

vectors

- by index
- by name
- by condition
- by mask
- indices by condition

matrices

- single value
- multiple values
- rows or columns
- row and column combination
- by condition
- by mask
- indices by condition

dataframes

- column(s) by name
- element by index
- rows/columns by index
- rows by condition
- rows by mask
- indices by condition

Calculate

vectors

- with single value
- elementwise
- aggregation
- pairwise

matrices

- with single value
- elementwise
- aggregation
- mask
- rowwise aggregation
- columnwise aggregation

dataframes

- with single value
- elementwise
- calculating new column
- aggregation
- rowwise aggregation
- columnwise aggregation
- groupwise aggregation

Combine

matrices

- binding vertically
- binding horizontally

dataframes

- with vector
- binding vertically
- binding horizontally
- by merging

Restructure

vectors

- by sorting

matrices

- flattening
- rotating

dataframes

- sorting by column
- pivoting long to wide
- pivoting wide to long

Appendix D

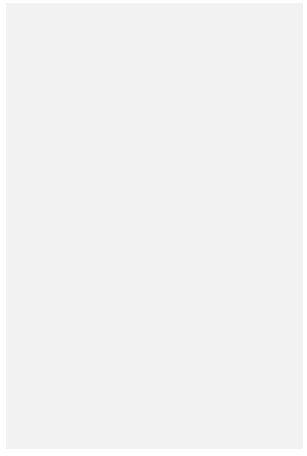
Slice N Dice exercises

Part 1

Which operations, in order, are required to transform vectors **g** and **s** into matrix **m**?

1. Join **g** and **s** together, into [6 8 1]
2. Duplicate **s**, into [8 1 8 1]
3. Join the two together into a matrix as rows

Create > vector > from smaller vectors
 Create > vector > by repeating vectors
 Create > matrix > from multiple vectors



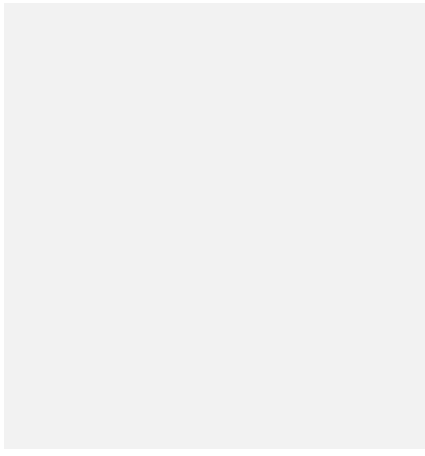
Which operations, in order, are required to create the dataframe **df** from scratch?

A	B	C	D
3	15	11	17
7	3	15	11
11	7	3	15

df

1. Form a series [3 7 11 15]
2. Turn that series into [3 7 11 15 3 7 11 15 3 7 11 15]
3. Wrap the longer series into a matrix, row by row
4. Turn the matrix into a dataframe

Create > vector > arithmetic sequence
 Create > vector > by repeating vectors
 Create > matrix > by folding a vector
 Create > dataframe > from a matrix



Which operations, in order, are required to access the vector **v** from matrix **m**?

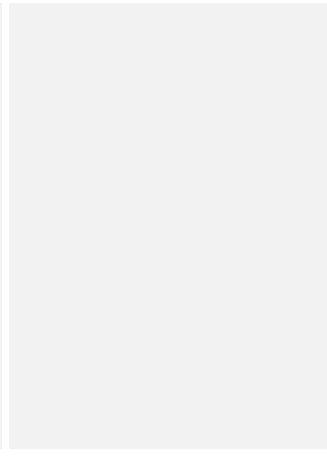
3	15	5	17
6	0	5	4
11	7	8	15

m

Vector **v** is [6 0 5 4 11 7 8]

1. Get the second row [6 0 5 4]
2. Get the first three columns of the third row, i.e. [11 7 8]
3. Join the two together into [6 0 5 4 11 7 8]

Access > from matrix > rows or columns
 Access > from matrix > row and column combination
 Create > vector > from smaller vectors



access2

Which operations, in order, are required to access the smaller dataframe **df2** from dataframe **df**?

A	B	C	D		A	C
6	15	"O"	"t"		8	"p"
8	3	"p"	"e"		8	"q"
8	7	"q"	"e"		6	"r"
6	7	"r"	"e"			

df

1. Get hold of the rows whose D-column value equals 'e'

2. From the result of step 1, retrieve A and C

df2

Access > from dataframe > rows by condition

Access > from dataframe > column(s) by name

calculate1

In the matrix **nr_bought**, each row represents a person, and each column represents a particular product that a particular person bought. The values represent the number of each product bought. The price at which each person bought each good is given in **price_per_item**.

Which operations, in order, are required to calculate the total amount of money that the purchases in **nr_bought** would cost?

	apple	banana	orange	strawberry	apple	banana	orange	strawberry			
person1	1	2	0	4	1	person1	2	1	5	6	4
person2	0	4	1	0	0	person2	2	3	3	4	5
person3	6	7	3	0	0	person3	7	2	3	6	1
person4	4	5	6	1	1	person4	1	3	5	5	1

nr_bought

price_per_item

You have two vectors, **widths** and **heights**. What operations, in order, are required to get the values of all areas that are below 100?

widths	heights
20	50
50	30
30	10
60	60
5	5
4	4
3	3

1. For each combination of width and height, calculate the area

2. Retrieve all values below 100

low_products

calculate2

Calculate > vectors > pairwise

Access > from matrix > by condition

combine

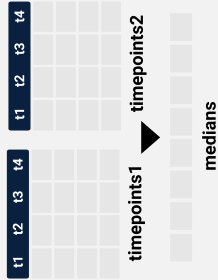
The dataframes **timepoints1** and **timepoints2** give the same data for data for two different sets of participants. t1-t4 represent performance scores at different timepoints.

1. Join the two dataframes together

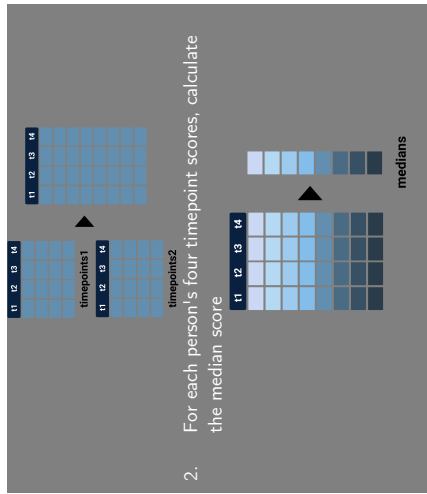
Combine > dataframes > binding vertically

Calculate > dataframes > rowwise aggregation

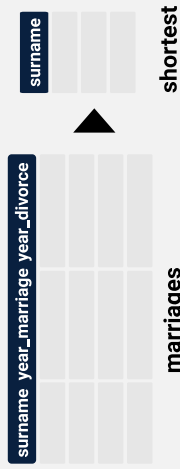
What is the medians performance for all participants?
Give the operations in order.



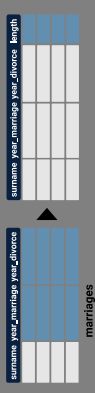
2. For each person's four timepoint scores, calculate the median score



Given the dataframe **df**, containing year for marriage and divorce for a set of couples, which surnames had the three shortest marriages?



1. Get hold of marriage years, the divorce years, and calculate the length of each marriage as a new column



2. Sort the marriages based on their length, from short to long



3. Take the first three marriages (i.e. the three shortest)



4. Retrieve the surnames



Calculate > dataframes > calculating new column

Restructure > dataframes > sorting by column

Access > from dataframe > rows-or columns by index

Access > from-dataframe > column(s) by name

Dataframe **prices** gives the prices at four different time points for a certain product in different countries. What is the mean price in different countries?

country	price1	price2	price3	price4	meanprice

1. Rearrange **prices** so that all the prices are stored in one column, and the timepoint number (1,2,3,4) is in its own column.

country	time	price

2. Compute the mean price for each country.

country	meanprice

Restructure > dataframes > pivoting wide to long

Calculate > dataframes > groupwise aggregation

Part 2

Starting out gentle, let's get a feel for how variables and variable assignment work.

- Using the documentation to the RIGHT, create a variable called **a** containing 5, and **b** containing 10. Multiply them together using *****.
- Now multiply them again, but save the result to a variable called **ab_prod**. Inspect it by simply typing **g ab_prod**.
- Do the same thing but in the area to the LEFT. Click on SUBMIT.
- In the scripting area to the left, create a variable called **amount** and assign it the value 900.
- In a new line, add 50 to **amount**, making sure the new value is saved to **amount**. Click RUN.
- Print out the contents of **amount** in the area to the right and hit RUN. Do you see what you expect?
- Click on Submit to confirm that it is correct.

- Create a variable called **a**, containing 5 (save to: **a**)
- Create a variable called **b**, containing 10 (save to: **b**)
- Create a variable called **ab_prod** (save to: **ab_prod**)

Variable assignment
Variable updating
Variable inspection
Read-eval-print loop
Arithmetic operators
The basics > variables

```
a<-5
b<-10
ab_prod<-a*b

a=5
b=10
ab_prod=a*b
```

- Create a variable called **amount** and assign it 900 (save to: **amount**)
- Add 50 to variable **amount** (save to: **amount**)
- Print **amount** to the console.

Variable inspection
Variable updating
Variable inspection
Read-eval-print loop
Arithmetic operators
The basics > scripts and consoles

```
amount<-900
amount <- amount + 50
amount

amount=900
amount = amount + 50
print(amount)
```


<p>1. A variable called <code>value</code> has already been loaded. Check its contents by writing <code>value</code> in the area to the right and click ENTER.</p> <p>2. Do the exercises in the scripting area, confirming the contents at each step by either printing them, or typing their name to the right.</p> <p>3. As always, make sure the steps are saved to the correct variable names and click on SUBMIT when you're done.</p>	<ol style="list-style-type: none"> Subtract 11 from <code>value</code> (save to: result1) <code>result1 <- value - 11</code> Divide <code>result1</code> by 10 (save to: result2) Raise <code>result2</code> to power of 2 (save to: answer) 	<pre>result1<- value-11 result2 <- result1/10 answer<-result2**2 result1= value-11 result2 = result1/10 answer=(result2**2)</pre>	<p><i>The basics > scalar arithmetic</i></p>
<p>Write a variable called <code>name</code> with the name <code>Joe</code> in it and another called <code>married</code> indicating that he married.</p>	<ol style="list-style-type: none"> Assign the value <code>Joe</code> to a variable called name Assign whether he married to a variable called married 	<pre>name<-'Joe' married<-TRUE name='Joe' married=True</pre>	<p><i>The basics > data types</i></p>
<p>Three variables have already been created: <code>name</code> containing a person's name, <code>age</code> containing the person's age, and <code>married</code> containing their marriage status as a <code>True/False</code> value.</p> <ol style="list-style-type: none"> Inspect their values by typing their names in the area to the right. Experiment with conditions there. While pretending that you don't know the values beforehand, compute the following logical conditions and save the results to the variable names requested: 	<ol style="list-style-type: none"> Is the name 'Peter'? (save to: step1) <code>step1 <- name=='Peter'</code> Is the name either 'Joe' or 'Peter'? (save to: step2) Is the age greater than 20? (save to: step3) Is the age greater than 20 and also less than 50? (save to: step4) Is he married? (save to: step5) 	<pre>step1 <- name == 'Peter' step2 <- name == 'Joe' name == 'Peter' step3 <-age > 20 step4 <- age > 20 && age < 50 step5 <- married == TRUE step1 = name == 'Peter' step2 = name == 'Joe' or name == 'Peter' step3 = age > 20 step4 = age > 20 and age < 50 step5 = married == True</pre>	<p><i>The basics > logical conditions</i></p>
<ol style="list-style-type: none"> Read first the entry about vectors. A vector called <code>old</code> has been pre-loaded. Inspect its value by typing its value in the right side area. Create a vector called <code>data</code> with values 80, 81 and 82. Explore the taxonomy entries under <i>Calculate > vectors > elementwise</i>. Add <code>old</code> to <code>data</code> and save to <code>total</code>. 	<ol style="list-style-type: none"> Create a vector with 80, 81, 82 (save to: data) Add <code>old</code> to <code>data</code> (save to: total) Add 8 to every element in <code>total</code> (save to: answer) 	<pre>data <- c(80,81,82) total <- data + old answer <- total + 8 data = np.array([80,81,82]) total = data + old answer = total + 8</pre>	<p><i>The basics > vector</i></p>

<p>6. Explore the taxonomy entries under <code>Calculate > vectors > with single value</code>. Add 8 to every element in total and save to <code>answer</code>.</p>		
<p>1. Read first the entry about functions.</p> <p>2. Use the function described in <code>Create > vector > arithmetic sequence</code> to create a vector called <code>vect</code> containing 5,8,...,26,29.</p> <p>3. Use the function described in <code>Calculate > vector > aggregation</code> to calculate the product of all values. Store the result in <code>answer</code>.</p>	<ol style="list-style-type: none"> 1. Create vector <code>vect</code> 5,8,...,26,29 (save to: <code>vect</code>) 2. Compute product (save to: <code>answer</code>) 	<pre>vect <- seq(5,29, by=3) answer <- prod(vect) vect = np.arange(5,30,3) answer = vect.prod()</pre> <p><i>The basics > functions</i></p>
<p>You have three vectors - <code>a</code>, <code>b</code>, and <code>c</code>, each with four elements.</p> <p>Inspect them either by printing them or typing their name to the right.</p> <p>Solve the exercises below. If you accidentally overwrite a vector, copy your code and refresh the console using the button at the right.</p>	<ol style="list-style-type: none"> 1. Access the last element in <code>a</code> (save to: <code>last_a</code>) 2. Assign <code>last_a</code> value to the first value in <code>b</code> 3. Double the second element in <code>c</code> 	<pre>last_a <- a[4] b[1] <- last_a c[2] <- c[2]*2 last_a = a[3] b[0] = last_a c[1] = c[1]*2</pre> <p><i>The basics > access, assign, modify</i></p>
<p>1. Two matrices called <code>m1</code> and <code>m2</code> have been pre-loaded. Inspect them.</p> <p>2. Using <code>Calculate > matrices > elementwise</code>, multiply the two together into matrix <code>prods</code>.</p> <p>3. Using <code>Access > from matrix > rows or columns</code>, access the first row, saving it to <code>answer</code></p>	<ol style="list-style-type: none"> 1. Multiply <code>m1</code> and <code>m2</code> together (save to: <code>prods</code>) 2. Access the first row (save to: <code>answer</code>) 	<pre>prods <- m1 * m2 answer <- prods[1,] prods = m1 * m2 answer = prods[0,]</pre> <p><i>The basics > matrix</i></p>
<p>A dataframe has been pre-loaded, called <code>df</code>. Inspect it.</p> <ol style="list-style-type: none"> 1. Access the <code>B</code> column and save to <code>df_B</code>. 2. Using <code>Access > from vector > by condition</code>, access all values in <code>df_B</code> that are above 4. Save to <code>answer</code>. 	<ol style="list-style-type: none"> 1. Access <code>B</code> (save to: <code>df_B</code>) 2. Access values over 4 (save to: <code>answer</code>) 	<pre>df_B <- df\$B answer <- df_B[df_B > 4] df_B = df.B answer = df_B[df_B > 4]</pre> <p><i>The basics > dataframe</i></p>

Part 3

vector1

You are given two vectors: x and y.

Calculate $2*x+4$, and calculate the differences between those values and the y-values. Square and sum the differences.

$$\begin{matrix} 34 & 30 & 29 & 38 & 70 & 73 & 67 & 75 \\ \mathbf{x} & & & & & & & \mathbf{y} \end{matrix}$$

$$\begin{matrix} 2* & +4 & & & & & & \\ \mathbf{x} & & & & & & & \mathbf{newx} \end{matrix}$$

$$\begin{matrix} - & & & & & & & \\ \mathbf{y} & & & & & & & \mathbf{newx} \end{matrix}$$

$$\begin{matrix} \text{sum} & & & & & & & \\ \mathbf{squares} & & & & & & & \mathbf{answer} \end{matrix}$$

$$\begin{matrix} \text{raise to} & & & & & & & \\ 2 & & & & & & & \\ \mathbf{squares} & & & & & & & \mathbf{squares} \end{matrix}$$

$$\begin{matrix} \mathbf{answer} \\ 135 \end{matrix}$$

- Multiply x by 2 and add 4 (save to: newx)
- Subtract newx from y and raise the result to 2 (save to: squares)
- Calculate the sum of squares (save to: answer)

```
newx <- 2*x + 4
squares <- (y - newx) ** 2
answer <- sum(squares)
newx = 2*x + 4
squares = (y - newx) ** 2
answer = squares.sum()
```

Calculate > vectors > with single value
 Calculate > vectors > elementwise
 The basics > scalar arithmetic
 Calculate > vectors > elementwise
 Calculate > vectors > aggregation

vector2

Create a vector called answer containing (2,2,4,4,9,9,2,4,4,9,9) given vector sub (2,4,9), sub is already in memory.

$$\begin{matrix} 2 & 4 & 9 \\ \mathbf{sub} & & \end{matrix}$$

$$\begin{matrix} 2 & 2 & 4 & 4 & 9 & 9 & 2 & 2 & 4 & 4 & 9 & 9 \\ \mathbf{answer} & & & & & & & & & & & \end{matrix}$$

- Duplicate sub to [2,4,9,2,4,9] (save to: twice)
- Repeat each element in twice twice (save to: answer)

```
twice <- rep(sub,each=2)
answer <- rep(twice,times=2)
twice = np.tile(sub,2)
answer = np.repeat(twice,2)
```

Create > vector > by repeating vectors
 Create > vector > by repeating vectors

vector3

Given vector arr, assign the value 5 to the second and fourth place

Add 2 to every second element, starting with the first.

$$\begin{matrix} 5 & 6 & 5 & 3 & 7 & 5 & 4 & 4 \\ \mathbf{arr} & & & & & & & \end{matrix}$$

$$\begin{matrix} 7 & 5 & 7 & 5 & 9 & 5 & 6 & 4 \\ \mathbf{answer} & & & & & & & \end{matrix}$$

- Assign the value 5 to the second and fourth place in arr (save to: arr1)
- Create a vector repeating [TRUE FALSE] four times, i.e. [TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE] (save to: ind)
- Access those elements from the arr you modified in step 1, and add 2 to them (save to: answer)

```
arr[c(2,4)] <- 5
arr1 <- arr
ind <- rep( c(TRUE,FALSE) , times=4 )
arr[ind] <- arr[ind] + 2
answer <- arr
arr[[1,3]] = 5
arr1 = arr
ind = np.tile( [True,False] , 4)
arr[ind] = arr[ind]+2
answer = arr
```

Calculate > vectors > mask
 Access > from vector > by mask
 Create > vector > arithmetic sequence
 Access > from vector > by index

Out of all numbers in vectors **x** and **y**, at which position does the value 12 come in terms of size? For example, the smallest value, with position 1, is 8.

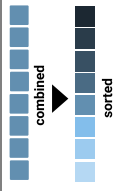
Save the answer to **answer**.



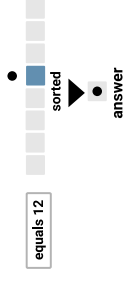
1. Glue the two vectors together (save to: **combined**)



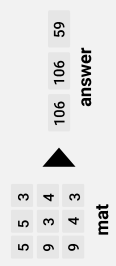
2. Sort **combined** from small to large (save to: **sorted**)



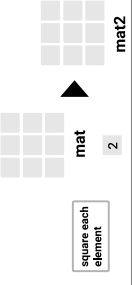
3. Find the index at which 12 is located in **sorted** (save to: **answer**)



Given matrix **mat**, square every number and sort the sums of each row from large to small.



4. Square each element in **mat** (save to: **squares**)



5. Calculate the sum of each row (save to: **sums**)

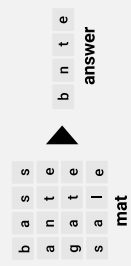


6. Sort sums from large to small (save to: **answer**)



How would you access all the diagonal elements from a matrix **m**, from the top-left to the bottom-right corner?

Pretend that you don't know the size of the matrix.



1. Get the number of rows in **mat** (save to: **N**)



2. Use **N** to create a sequence that contains every row index, up until **N** (save to: **seq**)



```
combined <- c(x,y)
sorted<-sort(combined)
```

```
combined = np.concatenate([x,y])
sorted = np.sort(combined)
```

```
squares <- mat **2
sums = apply(squares,1,sum)
answer = sort(sums, decreasing = TRUE)
squares = mat**2
sums = squares.sum(axis=0)
answer = np.sort(sums)[:-1]
```

```
N <- dim(mat)[1]
seq <- 1:N
ind <- cbind(seq,seq)
answer <- mat[ind]
N = mat.shape[0]
seq = np.arange(0,N)
ind = np.array([seq,seq])
```

Create > vector > from smaller vectors
Restructure > vectors > by sorting
Access > from vector > indices by condition

Calculate > matrices > with single value
Calculate > matrices > rowwise aggregation
Restructure > vectors > by sorting

Calculate > matrices > aggregation
Create > vector > arithmetic sequence
Create > matrix > from multiple vectors
Access > from matrix > multiple values

matrix3

```

5 5 12 10 17
7 7 6 5 4
5 13 14 8 5
15 5 93 2 15
14 5 61 3 19
m

```

What proportion of all 5s does each column have?

```

0.28 0.42 0.00 0.14 0.14
answer

```

matrix4

Find the three smallest sums out of all the sums generated from pairing up each value in vector **males** with each value in vector **females**.

3. Construct a matrix containing seq twice as rows, so as to indicate the row and column indices of the desired elements (save to: **ind**)

4. Use that matrix as index for mat (save to: **answer**)

1. Replace every value in m that equals 5 with 1, otherwise set it to be 0 (save to: **mask**)

2. Sum every column (save to: **sums**)

3. Get the total number of 5s (save to: **total**)

4. Divide sums by total (save to: **answer**)

1. Construct a matrix containing the sums of every possible male, female pair (save to: **sums**)

```

answer = mat[List(ind)]

```

```

mask <- ifelse(m==5,1,0)
sums <- apply(mask, 2, sum)
total <- sum(sums)
answer <- sums/total
mask = np.where(m == 5,1,0)
sums = mask.sum(axis = 1)
total = sums.sum()
answer = sums/total

```

```

sums <- outer(males,females,'+')
flat <- as.vector(sums)
sorted <- sort(flat)
answer <- sorted[1:3]

```

```

Calculate > matrices > mask
Calculate > matrices > columnwise aggregation
Calculate > vectors > aggregation
Calculate > vectors > with single value

```

```

Calculate > vectors > pairwise
Restructure > matrices > flattening
Restructure > vectors > by sorting

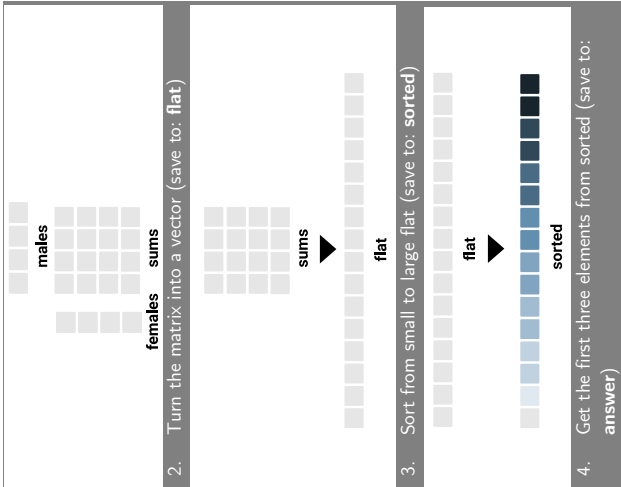
```

```
150 200 180 190
males
    250 260 270
answer
100 110 120 150
females
```

Suppose you have data representing 4x4 brain areas. Matrix **mat1** represents one measure while **mat2** represents another type of measure.

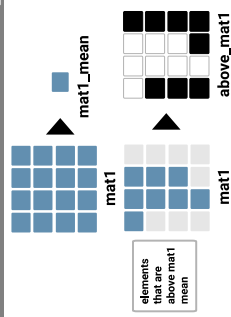
Matrix **label** gives the corresponding label of each brain area. You want the labels of all brain areas that show above-average activity in both measures.

```
26 21 26 8   -1  5  21 13
14 27 27 14   9  10 15  9
14 28 21 14   21  6 13  8
 7  21 19 19   7  20 15  5
mat1          mat2
A1  A5  A9  A13
A2  A6  A10 A14
A3  A7  A11 A15
A4  A8  A12 A16
label
```



```
sums = np.add.outer(males, females)
flat = sums.flatten()
sorted = np.sort(flat)
answer= sorted[0:3]
```

1. Compute a matrix indicating whether an element in **mat1** is above average (save to: **above_mat1**)



2. Compute a matrix indicating whether an element in **mat2** is above average (save to: **above_mat2**)

Access > from vector > by index

Access > from matrix > by mask
Calculate > matrices > aggregation

Access > from matrix > by mask
Calculate > matrices > aggregation

The basics > logical conditions

Access > from matrix > multiple values

Given matrix `mat`, calculate the means of every odd row and even row, and calculate whether each even row has a higher mean than the previous odd mean. Save the answer to `answer`.

```

14 11 17 7 15
16 14 11 8 6
9 10 13 20 8
15 9 8 13 8
20 19 9 21 25
24 23 20 7 8
mat

```

False False False
answer

3. Compute a matrix indicating whether an element is above average in both (save to: `both`)

4. Use it to access corresponding elements from `label` (save to: `answer`)

1. Access the odd and even rows (save to: `odd` and `even`)

2. Calculate the mean of every row in both `odd` and `even` (save to: `odd_means` and `even_means`)

```

odd <- mat[seq(1,6,2),]
even <- mat[seq(2,6,2),]
even_means <- apply(even,1,mean)
odd_means <- apply(odd,1,mean)
answer <- even_means > odd_means
odd = mat [np.arange(0,6,2),:]
even = mat [np.arange(1,6,2),:]
odd_means = odd.mean(axis=1)
even_means = even.mean(axis=1)
answer = even_means > odd_means

```

Create > vector > arithmetic sequence
Access > from matrix > rows or columns
Calculate > matrices > rowwise aggregation
The basics > logical conditions

dataframe1

Return the height of the rows in dataframe **data** whose genome is 'xx'. Ensure that the result is a dataframe.

genome	height	age	height
xx	180	18	180
xy	190	22	155
xx	155	31	
xy	170	24	

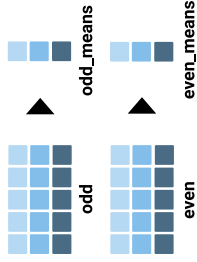
answer

Suppose you have a dataframe called **students** and want to find the maximum days of absence among students who failed their exam and are above 170cm in height.

ID	failed	height	absence
1	True	178	21
2	True	168	26
3	True	172	20
4	False	165	19
5	False	167	16

students

Suppose you have a dataframe (called **women**) containing the number of kids that each woman has, along with their income and the countries they lived in.



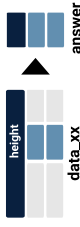
3. Write a condition that asserts that **even_means** is greater than **odd_means** (save to: **answer**)



1. Filter the rows based on 'xx' (save to: **rows**)



2. Retrieve the heights as dataframe (save to: **answer**)



1. Get hold of all students who both failed the exam and are above 170cm in height (save to: **failing_tall**)



2. Given those students, find the maximum days of absence (save to: **answer**)



1. Categorise each country's average number of kids and average income (save to: **means** with column names 'mean_kids' and 'mean_income')

Access > from dataframe > rows by condition

Access > from dataframe > column(s) by name

Access > from dataframe > rows by condition

Access > from dataframe > column(s) by name

Calculate > dataframes > groupwise aggregation

```
rows <- data[data$genome=='xx',]
answer <- rows['height']
rows = data[data.genome=='xx']
answer = rows['height']
```

```
failing_tall <- students[students$failed == TRUE & students$height > 170,]
answer <- max(failing_tall$absence)
failing_tall = students[(students.failed == True) & (students.height > 170)]
answer = failing_tall.absence.max()
```

```
means <- aggregate(list(mean_kids = women$nr_kids, mean_income = women$income), by=list(country = women$country), mean)
```

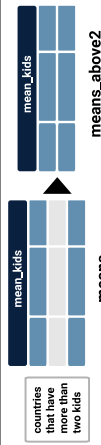

In the countries with more than 2 kids per woman on average, what is the mean income?

Ensure that the answer is a dataframe.

ID	country	nr_kids	income	mean_income
1	'C1'	3	200	
2	'C1'	2	300	
3	'C2'	0	100	250
4	'C2'	4	200	75
5	'C3'	5	50	
6	'C3'	5	100	



2. Get hold of the countries that have, on average, more than two kids (save to: `means_above2`)



3. Obtain those countries' income means as a single column dataframe (save to: `answer`)



```
means_above2 <- means[means$mean_kids > 2, ]
answer <- means_above2['mean_income']

means =
women.groupby('country')['nr_kids', 'income'].
mean().reset_index()

means.columns=['country', 'mean_kids', 'mean_in
come']

means_above2 = means[means.mean_kids > 2]
answer = means_above2['mean_income']
```

Access > from dataframe > rows by condition

Access > from dataframe > column(s) by name

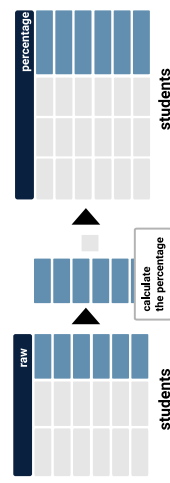
Suppose you have a dataframe with students in each row (called `students`). You also have one column for test scores called `raw`. You also know the gender of each student.

Assuming that 50 is the maximum possible score on the test, how many percent of all women scored above 70%?

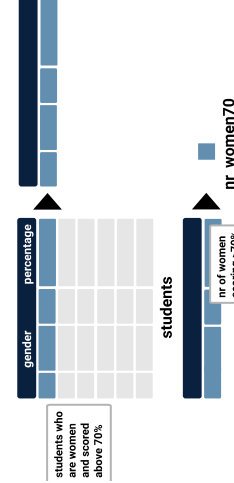
ID	gender	raw
1	'F'	47
2	'F'	28
3	'F'	34
4	'M'	16
5	'M'	45
6	'M'	38

33.333....

1. Compute the percentage for each student as $(\text{raw}/50)*100$ (save to: `students['percentage']`)



2. Compute how many students are women who scored above 70 percent (save to: `nr_women70`)



3. Compute how many students are women (save to: `nr_women`)

```
students['percentage'] <-
(students$raw/50)*100

nr_women70 <- nrow( students[(students$gender
== 'F') &
(students$percentage > 70), ] )

nr_women <- nrow( students[students$gender ==
'F', ] )

answer <- ( nr_women70 / nr_women ) * 100

students['percentage'] =
(students['raw']/50)*100

nr_women70 = len( students[(students.gender
== 'F') &
(students.percentage > 70)] )

nr_women = len( students[students.gender ==
'F', ] )

answer = ( nr_women70 / nr_women ) * 100
```

Calculate > dataframes > calculating new column

The basics > logical conditions

Access > from dataframe > rows by condition

Calculate > dataframes > aggregation

Access > from dataframe > rows by condition

Calculate > dataframes > aggregation

The basics > scalar arithmetic

Suppose you had a dataframe (called **citizens**) containing the height, weight, gender and country for a number of citizens from around the world.

Which country has the highest male average BMI? BMI = weight / height²

country	gender	weight	height
'C1'	'M'	70	1.80
'C1'	'M'	80	1.75
'C1'	'F'	60	1.65
'C1'	'F'	70	1.70
'C2'	'M'	100	1.70
'C2'	'M'	120	1.70
'C2'	'F'	70	1.60
'C2'	'F'	80	1.60

citizens

'C2'

students who are women

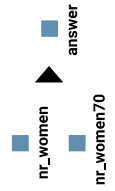
gender

students

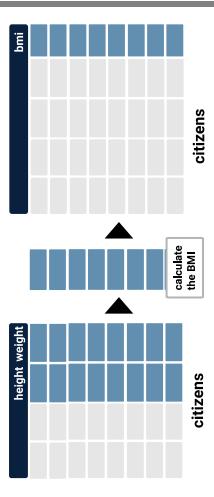
nr_of_women

nr_women

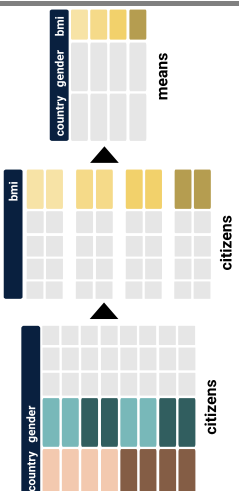
4. Divide the number of women scoring above 70 with the number of women overall, *100 (save to: answer)



1. Compute the BMIs of the citizens (save to: citizens['bmi'])



2. Get hold of the mean BMI for each combination of country and gender (save to: means)



3. Retrieve the maximum mean among male populations (save to: max_mean)

students who are women

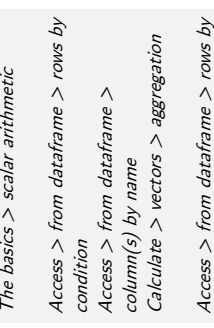
gender

students

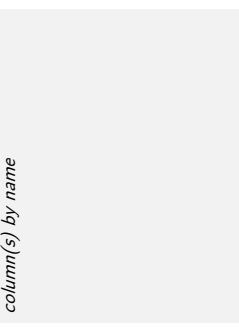
nr_of_women

nr_women

1. Compute the BMIs of the citizens (save to: citizens['bmi'])



2. Get hold of the mean BMI for each combination of country and gender (save to: means)



3. Retrieve the maximum mean among male populations (save to: max_mean)

```

citizens['bmi'] <- citizens$weight /
citizens$height**2

means <- aggregate(list(BMI = citizens$BMI),
by=list(country = citizens$country, gender =
citizens$gender), mean)

max_mean <- max(
means [means$gender=='M', ]$BMI)

answer <- means [(means$BMI == max_mean) &
(means$gender=='M'), ]$country

citizens['bmi'] = citizens.weight /
citizens.height**2

means =
citizens.groupby(['country', 'gender']).BMI.me
an().reset_index()

max_mean = means [means.gender=='M'].BMI.max()

answer = means [(means.BMI == max_mean) &
(means.gender=='M')].country.iloc[0]
    
```

Calculate > dataframes > calculating new column
 The basics > scalar arithmetic
 Access > from dataframe > rows by condition
 Access > from dataframe > column(s) by name
 Calculate > vectors > aggregation
 Access > from dataframe > rows by condition
 Access > from dataframe > column(s) by name

dataframe

Suppose you study population sizes of different bacteria types during four days. You have stored your data in three dataframes: the **data** has the first three days for type A and B, the **last** has the last day for A and B, and the **extra** has the daily sizes for type C.

You want to compute the mean size for each day and sort them from large to small based on mean.

day4	type	day1	day2	day3	day4	type	size
240	'C'	100	120	125	130	'B'	172.5
190	'A'	120	130	140		'A'	157.5
	'B'	100	150	250		'C'	118.75

data

extra	type	size
	'B'	172.5
	'A'	157.5
	'C'	118.75

answer

4. Find the country with the maximum male mean (save to: **answer**)

Find row with the max bmi as bmi

Find the max_bmi

1. Bind together the dataframes **data** and **last** (save to: **data2**)

2. Bind together the dataframes **data2** and **extra** (save to: **data3**)

3. Collapse **data3** so that sizes are stored in a single column, with the day stored in column "day" and actual size stored in column "size" (save to: **long_data**)

4. Take the mean for every type's mean size (save to: **means**)

```
data2 <- cbind(data,last)
data3 <- rbind(data2,extra)
long_data <- reshape(data3, idvar="type", v
arying=c('day1', 'day2', 'day3', 'day4'),
timevar='day',
v.names="size", direction="long")
means <- aggregate( list (size=
long_data$size),
by=list(type = long_data$type), mean)
answer <- means[ order(-means$size), ]
```

```
data2 = pd.concat([data,last],axis=1)
data3 = pd.concat([data2,extra],axis=0,
ignore_index=True)
long_data = pd.melt(data3, id_vars='type',
var_name='day', value_name='size')
means = long_data.groupby(['type'])['size']
.mean().reset_index()
answer = means.sort_values(by='size',
ascending = False)
```

Combine > dataframes > binding horizontally

Combine > dataframes > binding vertically

Restructure > dataframes > pivoting from wide to long

Calculate > dataframes > groupwise aggregation

Restructure > dataframes > sorting by column

dataframe7

Suppose you rent out cars and have a vector **rented** containing the IDs of the subset of cars currently being rented out.

You also have a dataframe **locations** containing the location-codes for all car IDs. You also have a dataframe **coords** containing the geographic X- and Y-coordinates for each location code.

You also have a matrix called **temp** with the current weather (in degrees).

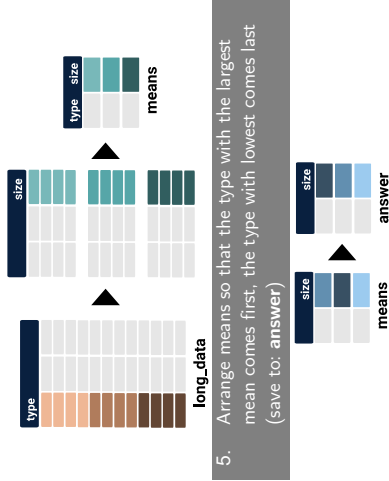
What are each rented car's location ID's temperature?

car_ID	loc	loc	row	col	loc	temp
1	10	1	1	4	27	36
2	6	2	2	1	36	22
3	24	3	3	4	25	19
4	3	4	2	3	21	34
5	4	5	4	2	36	21
6	4	6	2	2	31	34
7	3	7	3	3	39	14
8	5	8	4	1	27	28

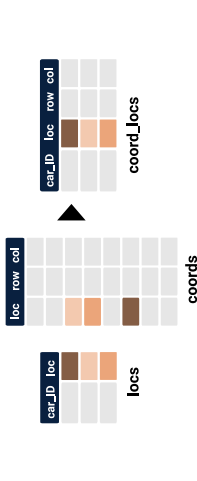
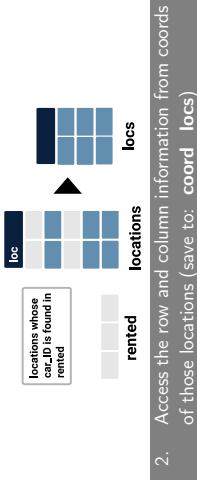
locations

locations	temp
2	4
4	5
8	4
1	1

answer



1. Return the location IDs of the cars whose car IDs are found in rented (save to: **locs**)



Access > from dataframe > rows by condition

Combine > dataframes > by merging

Access > from dataframe > column(s) by name

Create > matrix > from multiple vectors

Access > from matrix > multiple values

Create > dataframe > from vectors

```
locs <- locations[locations$car_ID %in%
rented,]
```

```
coord_locs <- merge(locs, coords, by="loc")
```

```
temps <- temp[ cbind(coord_locs$row,
coord_locs$col) ]
```

```
answer <- data.frame(loc=coord_locs$loc,
temp=temps)
```

```
locs =
locations[locations.car_ID.isin(rented)]
```

```
coord_locs = pd.merge(locs,coords,on='loc')
```

```
mat = np.array([coord_locs.row,
coord_locs.col])
```

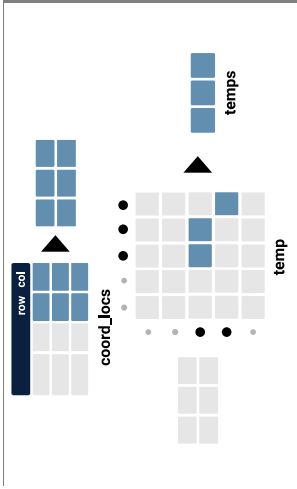
```
temps=temp[list(mat)]
```

```
answer =
pd.DataFrame({'loc':coord_locs['loc'],
'temp': temps}, columns=['loc', 'temp'] )
```

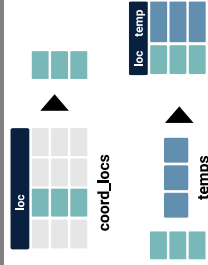
dataframe

Find the highest population density (people per unit area) out of all cities, saving the answer to **answer**.

ACity's area is incorrect and is actually 9000, so make sure to correct it first.



4. Join together the location IDs and temps into a dataframe (save to: **answer**)



1. Correct ACity's area value (update: **cities**)

```
cities[1,3]=9000
cities <- merge(cities,populations,by='name')
cities$density <- cities$population /
cities$area
answer<- max(cities$density)
```

Access > from dataframe > element
by index

Combine > dataframes > by
merging

name	population
"CityA"	89000
"CityB"	56000
"CityC"	78000
"CityD"	88000
"CityE"	65000
"CityF"	21000

populations

country	name	area
"C1"	"CityA"	10000
"C1"	"CityB"	15000
"C1"	"CityC"	9000
"C2"	"CityD"	7000
"C2"	"CityE"	12000
"C2"	"CityF"	14000

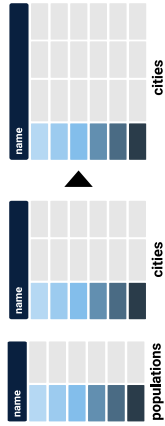
cities

33.333...
answer

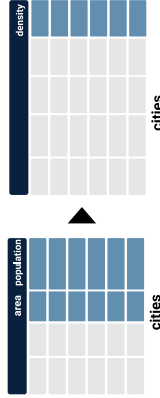
area
9000

cities

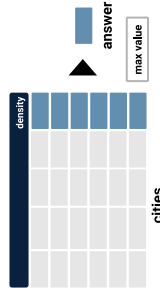
2. Join together **cities** and **populations** (save to: **cities**)



3. Create a new column named **density** with the result from dividing **population** by **area** (save to: **cities[density]**)



4. Take the maximum value out of the **density** column (save to: **answer**)



```
cities.iloc[0,2]=9000
```

```
cities = pd.merge( cities, populations,
on='name', how='inner')
```

```
cities['density']=cities.population /
cities.area
```

```
answer=cities.density.max()
```

Calculate > dataframes > calculating new column

Calculate > vectors > aggregation

Appendix E

Subgoal validation survey

Evaluating subgoal graphics

This survey is designed to evaluate a set of "subgoal graphics". These subgoal graphics are meant to illustrate the steps needed to solve various data wrangling problems. By "data wrangling", we mean the transformation of data tables from one format into another.

The survey will not involve programming and does not assume any prior knowledge. It is expected to take 20-25 minutes.

By participating, your e-mail will be placed in a lottery where every participant will have an equal chance of winning 1 of 4 £30 Amazon vouchers. The only condition is that the responses reflect an earnest effort. The winners will be announced on Wednesday the 27th of October, so make sure you complete it before then.

The study is voluntary and will have no effect on your grade. You can withdraw at any time. The data will be anonymised and used for a doctoral dissertation. It has been approved by the local ethics review board (app no. 300200145).

It is organised by me, Lovisa Sundin, a PhD-student at the School of Computing Science. It is supervised by Professor Quintin Cutts. Contact us at:

lsundin.1@research.gla.ac.uk
quintin.cutts@glasgow.ac.uk

*Obligatorisk

1. What is your student e-mail address? We will use this address for the lottery. *

2. What is your experience in... ? *

Markera endast en oval per rad.

	None	0-1 years	1-2 years	>2 years
Python	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
SQL	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Excel	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Background

Here we will go through a few concepts you should be aware of before proceeding.

<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHhY8H7oXslu2mj8kk/edit>

1/15

Alternatively, you can access values by using another vector, known as a "mask", which contains the value "True" for every value you want to access, and "False" for every value you want to ignore.



Matrices

A matrix is like a vector, but two-dimensional. In a matrix, you have a row and column index. Here, for example, you can access the value 2 by specifying the row index 1 and column index 2. You can also access values from a matrix by specifying a filtering condition ("give me all values that are even") or by using a mask.



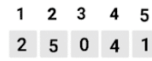
Dataframes

<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHhY8H7oXslu2mj8kk/edit>

3/15

Vectors

A vector is a list of multiple values, for example numbers or words. Every element in the vector has a numerical address known as an "index". Here the second element has index "2" and the value "5". (Sometimes vectors may be zero-indexed but here we assume that the first element has index 1).



Access by index

You can access values from a vector via this index, for example, here you access the value 5 by specifying the index 2.



Access by condition

You can also access values by expressing a condition, such as "give me all values that equal 5".



Access by mask

<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHhY8H7oXslu2mj8kk/edit>

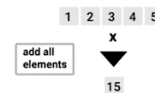
2/15

A dataframe stores measurements of different kinds. Every row represents a sample observation (for example, a person) and every column represents a measurement (for example, height or weight). Each column has a label. As with matrices, you can access rows, columns, and elements, using index, masks, or conditions. Since columns are named, you can also use access them by column name.

	A	B	C
'x'	1	4	
'y'	2	5	
'z'	3	6	

Aggregations

For a given data structure, you can "aggregate" the values. An aggregation means turning multiple values into a single value, for example taking the sum or mean. Here we add all the elements together.

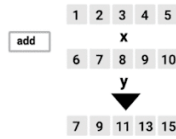


Element-wise arithmetic

<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHhY8H7oXslu2mj8kk/edit>

4/15

For a given data structure, you can also calculate element-wise arithmetic. "Element-wise" refers to the fact that you calculate arithmetic element by element. For example, here you add two vectors together.



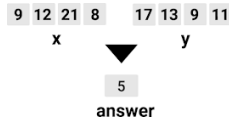
To summarise, we have the following concepts to be aware of:

- * Vectors
- * Accessing by index (numbers), conditions, and mask
- * Matrices
- * Dataframes
- * Aggregations
- * Element-wise arithmetic

Warm-up problem

In this section, we will warm up by looking at an example of a data wrangling exercise, and how it could be solved.

Out of all numbers in vectors x and y, at which position does the value 12 come in terms of size? For example, the smallest value, with position 1, is 8.



This exercise can be subdivided into the following steps:

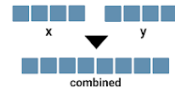
1. "Glue" the two vectors together
2. Sort the result from small to large
3. Find the index at which the value 12 is located, and return the index

<https://docs.google.com/forms/d/1D9jxvDFDEUgsC1DSWNh3pTO-BHfY8H7oXslu2mj8k/edit>

5/15

Now, let's try visualise each step:

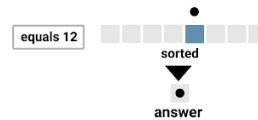
1. "Glue" the two vectors together



2. Sort the result from small to large



3. Find the index at which the value 12 is located, and return the index



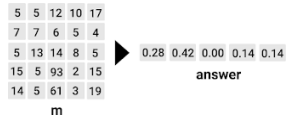
From now onwards, the graphics will be provided, but it will be your task to interpret them in terms of which steps they represent.

Problem 1

<https://docs.google.com/forms/d/1D9jxvDFDEUgsC1DSWNh3pTO-BHfY8H7oXslu2mj8k/edit>

6/15

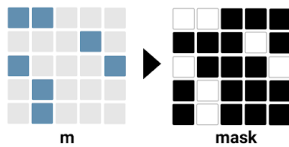
What proportion (out of all 5s in the entire matrix) does each column in matrix m have?



Below the solution has been split up into a number of ordered steps, represented through subgoal graphics. Try to interpret the graphic in terms of what action it represents and provide a text label for it.

There is no exact right answer, and no programming syntax is needed: simply provide human-friendly labels like "Rotate the matrix", "Take the row whose B-column equals 3", "Calculate the mean"...

3. 1.1. What action does the following graphic suggest? *



4. 1.2. What action does the following graphic suggest? *



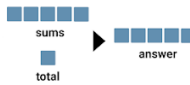
<https://docs.google.com/forms/d/1D9jxvDFDEUgsC1DSWNh3pTO-BHfY8H7oXslu2mj8k/edit>

7/15

5. 1.3. What action does the following graphic suggest? *



6. 1.4. What action does the following graphic suggest? *



7. Any feedback regarding the graphics? Is anything unclear about them?

Problem 2

Find the three smallest sums out of all the sums generated from pairing up each value in vector males with every value in vector females.



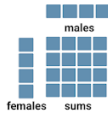
<https://docs.google.com/forms/d/1D9jxvDFDEUgsC1DSWNh3pTO-BHfY8H7oXslu2mj8k/edit>

8/15

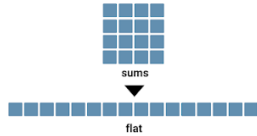
Below the solution has been split up into a number of ordered steps, represented through subgoal graphics. Try to interpret the graphic in terms of what action it represents and provide a text label for it.

There is no exact right answer, and no programming syntax is needed: simply provide human-friendly labels like "Rotate the matrix", "Take the row whose B-column equals 3", "Calculate the mean"...

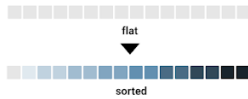
8. 2.1. What action does the following graphic suggest? *



9. 2.2. What action does the following graphic suggest? *



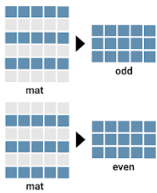
10. 2.3. What action does the following graphic suggest? *



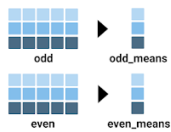
<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHYY8H7oXslu2mj8k/edit>

9/15

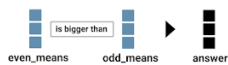
13. 3.1. What action does the following graphic suggest? *



14. 3.2. What action does the following graphic suggest? *



15. 3.3. What action does the following graphic suggest? *



<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHYY8H7oXslu2mj8k/edit>

11/15

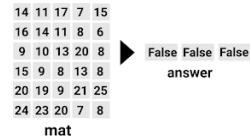
11. 2.4. What action does the following graphic suggest? *



12. Any feedback regarding the graphics? Is anything unclear about them?

Problem 3

Given matrix mat, calculate the means of every odd row and even row, and calculate whether each even row has a higher mean than the previous odd mean.



Below the solution has been split up into a number of ordered steps, represented through subgoal graphics. Try to interpret the graphic in terms of what action it represents and provide a text label for it.

There is no exact right answer, and no programming syntax is needed: simply provide human-friendly labels like "Rotate the matrix", "Take the row whose B-column equals 3", "Calculate the mean"...

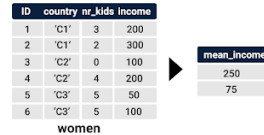
<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHYY8H7oXslu2mj8k/edit>

10/15

16. Any feedback regarding the graphics? Is anything unclear about them?

Problem 4

Suppose you have a dataframe (called women) containing the number of kids that each woman has, along with their income and the countries they lived in. For each of the countries with more than 2 kids per woman on average, what is the mean income? Ensure that the answer is a dataframe.



Below the solution has been split up into a number of ordered steps, represented through subgoal graphics. Try to interpret the graphic in terms of what action it represents and provide a text label for it.

There is no exact right answer, and no programming syntax is needed: simply provide human-friendly labels like "Rotate the matrix", "Take the row whose B-column equals 3", "Calculate the mean"...

17. 4.1. What action does the following graphic suggest? *



<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHYY8H7oXslu2mj8k/edit>

11/15

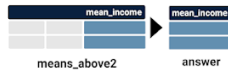
<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHYY8H7oXslu2mj8k/edit>

12/15

18. 4.2. What action does the following graphic suggest? *



19. 4.3. What action does the following graphic suggest? *



20. Any feedback regarding the graphics? Is anything unclear about them?

Problem 5

Suppose you had a dataframe (called citizens) containing the height, weight, gender and country for a number of citizens from around the world. Which country has the highest average BMI? BMI = weight / height²

country	gender	weight	height
C1	M	70	1.90
C1	M	80	1.75
C1	F	60	1.65
C1	F	70	1.70
C2	M	100	1.70
C2	M	120	1.70
C2	F	70	1.60
C2	F	80	1.60

citizens

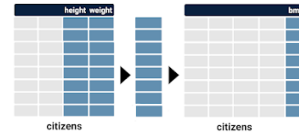
<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHhY8H7oXslu2mj8kk/edit>

13/15

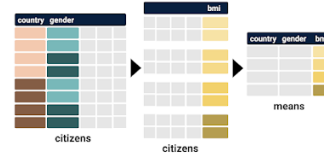
Below the solution has been split up into a number of ordered steps, represented through subgoal graphics. Try to interpret the graphic in terms of what action it represents and provide a text label for it.

There is no exact right answer, and no programming syntax is needed: simply provide human-friendly labels like "Rotate the matrix", "Take the row whose B-column equals 3", "Calculate the mean"...

21. 5.1. What action does the following graphic suggest? *



22. 5.2. What action does the following graphic suggest? *



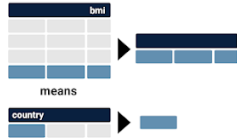
23. 5.3. What action does the following graphic suggest? *



<https://docs.google.com/forms/d/1D9jsxFDEUgsC1DSWNdH3pTO-BHhY8H7oXslu2mj8kk/edit>

14/15

24. 5.4. What action does the following graphic suggest? *



25. Any feedback regarding the graphics? Is anything unclear about them?

Thank you!

Thank you for contributing to this survey. We will be in touch on Wednesday the 27th to inform you of the lottery result.

If you have any more questions, contact me at l.sundin.1@research.qla.ac.uk

Det här innehållet har varken skapats eller godkänts av Google.

Google Formulär

Appendix F

Thumbnail validation survey

Evaluating data operation graphics

You are being asked to participate in a research study. The purpose of the research is to study methods of teaching data science. Specifically, we are interested in evaluating the usability of graphics that describe data operations. These graphics would, among other things, be used as thumbnails in a computer menu.

Importantly, you are not being evaluated - the graphics are what's being evaluated.

After a short explanation of key concepts, you will be asked to interpret the meaning of a series of graphics, by choosing one option out of a set of possible meanings. The same graphics will then re-appear in a slightly new version, and you will be asked to repeat the task.

* No computer programming background is necessary.

* The expected duration of your participation is 15 minutes.

* There are no risks or discomforts associated with this study.

* The information you provide for purposes of this research is confidential and anonymous.

* Questions about the research or your rights should be directed to Dr. Nouri Sakr at nouri.sakr@aucegypt.edu or Lovisa Sundin at l.sundin.1@research.gla.ac.uk.

* Participation in this study is voluntary. Refusal to participate will involve no penalty or loss of benefits to which you are otherwise entitled. You may discontinue participation at any time.

* The study has received ethical approval from the IRB at the American University in Cairo.

Please indicate your informed consent to proceed with the questionnaire:

***Obligatorisk**

1. I understand and agree to the terms above. *

Markera alla som gäller.

Yes

Background questions

First up, just a few questions regarding your previous background.

https://docs.google.com/forms/d/1ef8NKYD3JGIGxvx4bZusGy0iErIzVXb_B-HADjV3Y/edit

1/21

29/10/2021, 21:05

Evaluating data operation graphics

A vector is a list of multiple values, for example numbers or words. Every element in the vector has a numerical address known as an "index". Here the second element has index "2" and the value "5". (Sometimes vectors may be zero-indexed but here we assume that the first element has index 1).

1	2	3	4	5
2	5	0	4	1

You can access values from a vector via this index, for example, here you access the value 5 by specifying the index 2.

1	2	3	4	5
2	5	0	4	1

▶ 5

You can also access values by expressing a condition, such as "give me all values that equal 5"

2	5	0	4	1
---	---	---	---	---

▶ 5

Alternatively, you can access values by using another vector, known as a "mask", which contains the value "True" for every value you want to access, and "False" for every value you want to ignore.

True	True	False	False	True
mask				
2	5	0	4	1

▶ 2 5 1

https://docs.google.com/forms/d/1ef8NKYD3JGIGxvx4bZusGy0iErIzVXb_B-HADjV3Y/edit

3/21

2. How many years of programming experience do you have? (If none, write 0) *

3. What's your gender?

Markera endast en oval.

- Female
 Male
 Prefer not to say
 Other

4. What is your experience level with the following technologies? *

Markera endast en oval per rad.

	None at all	Beginner	Intermediate	Advanced
Excel / other spreadsheet software	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
SQL	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
R	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Python	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Background information

Please read through the following quick background. Do not make use of any external sources (e.g. Google) to answer the questions.

Vectors

https://docs.google.com/forms/d/1ef8NKYD3JGIGxvx4bZusGy0iErIzVXb_B-HADjV3Y/edit

2/21

29/10/2021, 21:05

Evaluating data operation graphics

5. Which image represents the following access operations? *

Image 1 

Image 2 

Image 3 

Markera endast en oval per rad.

	Image 1	Image 2	Image 3
Access from vector by index	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Access from vector by mask	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Access from vector by condition	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Matrices

A matrix is like a vector, but two-dimensional. In a matrix, you have a row and column index. Here, for example, you can access the value 2 by specifying the row index 1 and column index 2.

	1	2	3
1	3	2	5
2	6	5	6

▶ 2

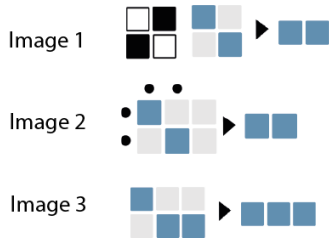
https://docs.google.com/forms/d/1ef8NKYD3JGIGxvx4bZusGy0iErIzVXb_B-HADjV3Y/edit

3/21

https://docs.google.com/forms/d/1ef8NKYD3JGIGxvx4bZusGy0iErIzVXb_B-HADjV3Y/edit

4/21

6. You can also access values from a matrix by specifying a filtering condition ("give me all values that are even") or by using a mask. Which image represents the following access operations? *



Markera endast en oval per rad.

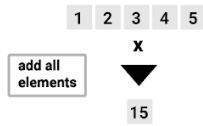
	Image 1	Image 2	Image 3
Access from matrix by index	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Access from matrix by condition	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Access from matrix by mask	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Dataframes

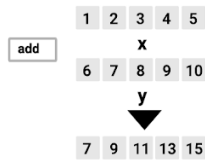
https://docs.google.com/forms/d/1efx8NKYD3JGIGxvx4bZusGy0iErIzVxb_B-HADjfv3Y/edit

5/21

For a given data structure, you can "aggregate" the values. An aggregation means turning multiple values into a single value, for example taking the sum or mean. Here we add all the elements together.



For a given data structure, you can also calculate element-wise arithmetic. "Element-wise" refers to the fact that you calculate arithmetic element by element. For example, here you add two vectors together.



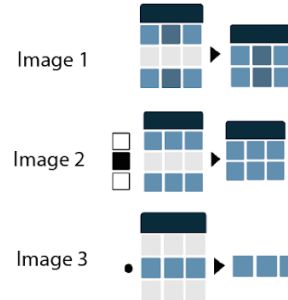
https://docs.google.com/forms/d/1efx8NKYD3JGIGxvx4bZusGy0iErIzVxb_B-HADjfv3Y/edit

7/21

A dataframe stores measurements of different kinds. Every row represents a sample observation (for example, a person) and every column represents a measurement (for example, height or weight). Each column has a label.

A	B	C
'x'	1	4
'y'	2	5
'z'	3	6

7. Which image represents the following access operations? *



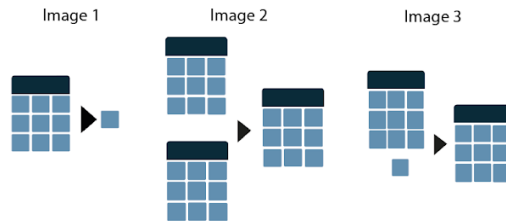
Markera endast en oval per rad.

	Image 1	Image 2	Image 3
Access rows from dataframe by index	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Access rows from dataframe by condition	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Access rows from dataframe by mask	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

https://docs.google.com/forms/d/1efx8NKYD3JGIGxvx4bZusGy0iErIzVxb_B-HADjfv3Y/edit

6/21

8. The same type of arithmetic could be done with dataframes. Which image represents the following access operations? *



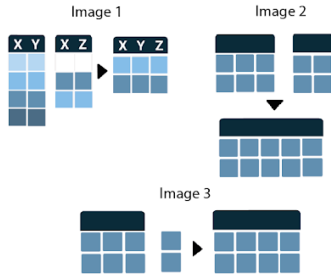
Markera endast en oval per rad.

	Image 1	Image 2	Image 3
Element-wise arithmetic with a dataframe and single value	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Aggregating a dataframe	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Element-wise arithmetic with dataframes	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

https://docs.google.com/forms/d/1efx8NKYD3JGIGxvx4bZusGy0iErIzVxb_B-HADjfv3Y/edit

8/21

9. Dataframes can be combined in various ways. Which image represents the following access operations? *



Markera endast en oval per rad.

	Image 1	Image 2	Image 3
Binding two dataframes together by row	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Merging two dataframes so that the result only has rows that appear in both	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Adding a new column to a dataframe	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Validating graphics

Below is a series of 9 graphics using a graphical notation. They're meant to serve as "thumbnails" on a computer menu for accessing information on which computer command to use.

Please look at each graphic and choose, from the list of option, which operation you believe the graphic represents.

If no option fits your interpretation, please write your interpretation under "other...".

Do not make use of any external sources (e.g. Google) to answer the questions.

10. 1. What does the following graphic represent? *



Markera endast en oval.

- Access the top and bottom row from a dataframe
- Access rows from a dataframe by index
- Access indices from a dataframe based on a condition
- Access values from a dataframe based on a condition
- Övrigt: _____

11. 2. What does the following graphic represent? *



Markera endast en oval.

- Replacing two columns in a dataframe by a new column
- Calculating a new column in a dataframe based on previous columns
- Accessing a column from a dataframe based on a condition
- Duplicating a column in a dataframe
- Övrigt: _____

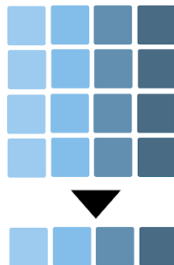
12. 3. What does the following graphic represent? *



Markera endast en oval.

- Sorting a vector
- Reversing the order in a vector
- Duplicating a vector
- Shuffling a vector
- Övrigt: _____

13. 4. What does the following graphic represent? *



Markera endast en oval.

- Aggregating each column in a matrix
- Aggregating each row in a matrix
- Accessing a row from a matrix by condition
- Accessing a row from a matrix by index
- Övrigt: _____

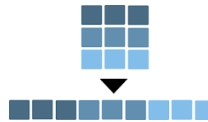
14. 5. What does the following graphic represent? *



Markera endast en oval.

- Folding a vector into a matrix column-by-column
- Folding a vector into a matrix row-by-row
- Flattening a matrix into a vector
- Creating a matrix from smaller vectors
- Övrigt: _____

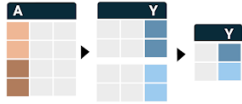
15. 6. What does the following graphic represent? *



Markera endast en oval.

- Flattening a matrix into a vector
- Folding a vector into a matrix
- Aggregating a matrix, e.g. calculating its mean
- Splitting a matrix into smaller vectors
- Övrigt: _____

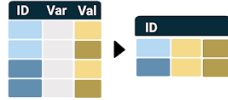
16. 7. What does the following graphic represent? *



Markera endast en oval.

- Splitting a dataframe into groups, and aggregating a column in each group
- Aggregating a dataframe row-wise
- Combining two dataframes vertically
- Accessing a column from a dataframe based on name
- Övrigt: _____

17. 8. What does the following graphic represent? *



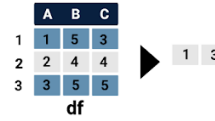
Markera endast en oval.

- Pivoting a dataframe so that every categorical variable is its own column
- Pivoting a dataframe so that each unique ID has its own row
- Accessing rows from a dataframe based on condition
- Aggregating dataframe rows group-wise, i.e. based on a column
- Övrigt: _____

Graphics with data

Now let's do this one more time, but this time with data. Do not make use of any external sources (e.g. Google) to answer the questions.

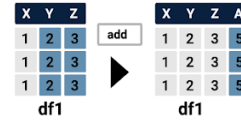
18. 9. What does the following graphic represent? *



Markera endast en oval.

- Access the top and bottom row from a dataframe
- Access rows from a dataframe by index
- Access indices from a dataframe based on a condition
- Access values from a dataframe based on a condition
- Övrigt: _____

19. 10. What does the following graphic represent? *



Markera endast en oval.

- Replacing two columns in a dataframe by a new column
- Calculating a new column in a dataframe based on previous columns
- Accessing a column from a dataframe based on a condition
- Duplicating a column in a dataframe
- Övrigt: _____

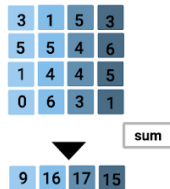
20. 11. What does the following graphic represent? *



Markera endast en oval.

- Sorting a vector
- Reversing the order in a vector
- Duplicating a vector
- Shuffling a vector
- Övrigt: _____

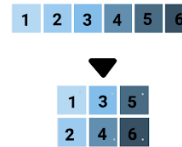
21. 12. What does the following graphic represent? *



Markera endast en oval.

- Aggregating each column in a matrix
- Aggregating each row in a matrix
- Accessing a row from a matrix by condition
- Accessing a row from a matrix by index
- Övrigt: _____

22. 13. What does the following graphic represent? *



Markera endast en oval.

- Folding a vector into a matrix column-by-column
- Folding a vector into a matrix row-by-row
- Flattening a matrix into a vector
- Creating a matrix from smaller vectors
- Övrigt: _____

23. 14. What does the following graphic represent? *

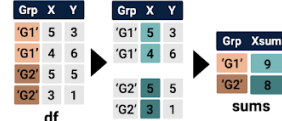
3	1	5	3
5	5	4	6
1	4	4	5
0	6	3	1

3	1	5	3	5	5	4	6	1	4	4	5	0	6	3	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Markera endast en oval.

- Flattening a matrix into a vector
- Folding a vector into a matrix
- Aggregating a matrix, e.g. calculating its mean
- Splitting a matrix into smaller vectors
- Övrigt: _____

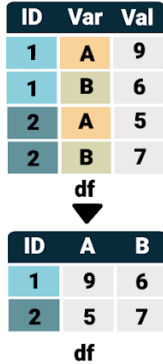
24. 15. What does the following graphic represent? *



Markera endast en oval.

- Splitting a dataframe into groups, and aggregating a column in each group
- Aggregating a dataframe row-wise
- Combining two dataframes vertically
- Accessing a column from a dataframe based on name
- Övrigt: _____

25. 16. What does the following graphic represent? *

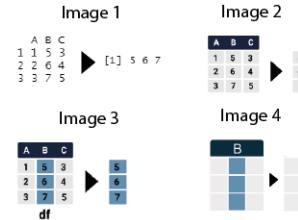


Markera endast en oval.

- Pivoting a dataframe so that every categorical variable is its own column
- Pivoting a dataframe so that each unique ID has its own row
- Accessing rows from a dataframe based on condition
- Aggregating dataframe rows group-wise, i.e. based on a column
- Övrigt: _____

Exploring different variants

26. The four images below all show you the operation "accessing a column from a dataframe based on name". Which of the image versions would you prefer as (select all that apply): *



Markera alla som gäller.

	Image 1	Image 2	Image 3	Image 4	None
A thumbnail in a computer menu for accessing the corresponding computer command	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A textbook illustration to explain how an operation works	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Final thoughts

Nearly done now - just a few concluding thoughts.

27. On a scale of 1 to 5, how clear do you think the graphics WITHOUT data were?

Markera endast en oval.

1 2 3 4 5

Not helpful at all Very helpful

28. On a scale of 1 to 5, how clear do you think the graphics WITH data were?

Markera endast en oval.

1 2 3 4 5
Not helpful at all Very helpful

29. If you have any other thoughts, please provide them here.

Thank you for your participation!

Your feedback is very helpful to the development of our tool. If you have any questions, contact Dr. Nouri Sakr at nouri.sakr@aucegypt.edu or Lovisa Sundin at lsundin.1@research.gla.ac.uk.

Det här innehållet har varken skapats eller godkänts av Google.

Google Formulär

Bibliography

- [1] J. Robertson, “Likert-type scales, statistical methods, and effect sizes,” *Communications of the ACM*, vol. 55, no. 5, pp. 6–7, 2012.
- [2] H. Wickham and G. Grolemund, *R for data science: import, tidy, transform, visualize, and model data*. " O’Reilly Media, Inc.", 2016.
- [3] C. Kelleher and M. Ichinco, “Towards a model of API learning,” in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 163–168, IEEE, 2019.
- [4] F. Voichick, G. Gao, M. Ichinco, and C. Kelleher, “Towards Validation of a Model of API Learning,” in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 267–269, IEEE, 2019.
- [5] B. Baumer, “A data science course for undergraduates: Thinking with data,” *The American Statistician*, vol. 69, no. 4, pp. 334–342, 2015.
- [6] T. Taipalus, “A Notation for Planning SQL Queries,” *Journal of Information Systems Education*, vol. 30, no. 3, pp. 160–166, 2019.
- [7] J. Danaparamita and W. Gatterbauer, “QueryViz: helping users understand SQL queries and their patterns,” in *Proceedings of the 14th International Conference on Extending Database Technology*, pp. 558–561, 2011.
- [8] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, “Wrangler: Interactive visual specification of data transformation scripts,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 3363–3372, 2011.
- [9] X. Zhang and P. J. Guo, “Ds. js: Turn any webpage into an example-centric live programming environment for learning data science,” in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pp. 691–702, 2017.
- [10] RStudio, “RStudio Cheat Sheets.” <https://github.com/rstudio/cheatsheets>, 2021. [Online; accessed 03-June-2021].

- [11] G. L. Nelson and A. J. Ko, “On use of theory in computing education research,” in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pp. 31–39, 2018.
- [12] W. S. Cleveland, “Data science: an action plan for expanding the technical areas of the field of statistics,” *International statistical review*, vol. 69, no. 1, pp. 21–26, 2001.
- [13] D. Donoho, “50 years of data science,” *Journal of Computational and Graphical Statistics*, vol. 26, no. 4, pp. 745–766, 2017.
- [14] I. Carmichael and J. Marron, “Data science vs. statistics: two cultures?,” *Japanese Journal of Statistics and Data Science*, vol. 1, no. 1, pp. 117–138, 2018.
- [15] DOMO, “Data Never Sleeps 6.0.” <https://www.domo.com/solution/data-never-sleeps-6/>, 2018. [Online; accessed 10-May-2021].
- [16] S. Kross and P. J. Guo, “Practitioners teaching data science in industry and academia: Expectations, workflows, and challenges,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–14, 2019.
- [17] A. E. C. Sotos, S. Vanhoof, W. Van den Noortgate, and P. Onghena, “Students’ misconceptions of statistical inference: A review of the empirical evidence from research on statistics education,” *Educational research review*, vol. 2, no. 2, pp. 98–113, 2007.
- [18] S. Lohr, “For big-data scientists, ‘janitor work’ is key hurdle to insights,” *New York Times*, vol. 17, p. B4, 2014.
- [19] CrowdFlower, “Data Science Report 2016.” <http://www2.cs.uh.edu/~ceick/UDM/CFDS16.pdf/>, 2016. [Online; accessed 10-May-2021].
- [20] Dodds, Leigh, “Do data scientists spend 80% of their time cleaning data? Turns out, no?.” <https://blog.ldodds.com/2020/01/31/do-data-scientists-spend-80-of-their-time-cleaning-data-turns-out-2020/>. [Online; accessed 10-May-2021].
- [21] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. Van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono, “Research directions in data wrangling: Visualizations and transformations for usable and credible data,” *Information Visualization*, vol. 10, no. 4, pp. 271–288, 2011.
- [22] Y. Zhu, L. M. Hernandez, P. Mueller, Y. Dong, and M. R. Forman, “Data acquisition and preprocessing in studies on humans: what is not taught in statistics classes?,” *The American Statistician*, vol. 67, no. 4, pp. 235–241, 2013.

- [23] T. Erickson, M. Wilkerson, W. Finzer, and F. Reichsman, “Data moves,” *Technology Innovations in Statistics Education*, vol. 12, no. 1, 2019.
- [24] J. E. Broatch, S. Dietrich, and D. Goelman, “Introducing data science techniques by connecting database concepts and dplyr,” *Journal of Statistics Education*, vol. 27, no. 3, pp. 147–153, 2019.
- [25] J. Hardin, R. Hoerl, N. J. Horton, D. Nolan, B. Baumer, O. Hall-Holt, P. Murrell, R. Peng, P. Roback, D. Temple Lang, *et al.*, “Data science in statistics curricula: Preparing students to “think with data”,” *The American Statistician*, vol. 69, no. 4, pp. 343–353, 2015.
- [26] T. Camp, S. Zweben, E. Walker, and L. Barker, “Booming enrollments: Good times?,” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pp. 80–81, 2015.
- [27] L. J. Sax, K. J. Lehman, and C. Zavala, “Examining the enrollment growth: Non-CS majors in CS1 courses,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pp. 513–518, 2017.
- [28] M. Han, Z. Li, J. S. He, and X. Tian, “What are the Non-majors Looking for in CS Classes?,” in *2019 IEEE Frontiers in Education Conference (FIE)*, pp. 1–5, IEEE, 2019.
- [29] H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemond, A. Hayes, L. Henry, J. Hester, *et al.*, “Welcome to the tidyverse,” *Journal of Open Source Software*, vol. 4, no. 43, p. 1686, 2019.
- [30] S. Liu, G. Andrienko, Y. Wu, N. Cao, L. Jiang, C. Shi, Y.-S. Wang, and S. Hong, “Steering data quality with visual analytics: The complexity challenge,” *Visual Informatics*, vol. 2, no. 4, pp. 191–197, 2018.
- [31] A. Bogatu, N. W. Paton, A. A. Fernandes, and M. Koehler, “Towards automatic data format transformations: data wrangling at scale,” *The Computer Journal*, vol. 62, no. 7, pp. 1044–1060, 2019.
- [32] T. Erickson, B. Finzer, F. Reichsman, and M. Wilkerson, “Data moves: one key to data science at school level,” in *Proceedings of the International Conference on Teaching Statistics (ICOTS-10)*, vol. 6, 2018.
- [33] M. H. Wilkerson, K. Lanouette, and R. L. Shareff, “Exploring variability during data preparation: a way to connect data, chance, and context when working with complex public datasets,” *Mathematical Thinking and Learning*, pp. 1–19, 2021.
- [34] C. Taylor, J. Spacco, D. P. Bunde, Z. Butler, H. Bort, C. L. Hovey, F. Maiorana, and T. Zeume, “Propagating the adoption of CS educational innovations,” in *Proceedings*

Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, pp. 217–235, 2018.

- [35] C. L. Hovey, L. Barker, and V. Nagy, “Survey Results on Why CS Faculty Adopt New Teaching Practices,” in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 483–489, 2019.
- [36] J. Hüffmeier, J. Mazei, and T. Schultze, “Reconceptualizing replication as a sequence of different studies: A replication typology,” *Journal of Experimental Social Psychology*, vol. 66, pp. 81–92, 2016.
- [37] G. Norman, “Results confounded and trivial: the perils of grand educational experiments,” *Medical education*, vol. 37, no. 7, pp. 582–584, 2003.
- [38] R. Kimball, “Data wrangling,” *Information Management*, vol. 18, no. 1, p. 8, 2008.
- [39] W. McKinney, *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.", 2012.
- [40] C. J. Wild and M. Pfannkuch, “Statistical thinking in empirical enquiry,” *International statistical review*, vol. 67, no. 3, pp. 223–248, 1999.
- [41] C. Franklin, G. Kader, D. Mewborn, J. Moreno, R. Peck, M. Perry, and R. Scheaffer, “Guidelines for assessment and instruction in statistics education (gaise) report,” 2007.
- [42] N. J. Horton, B. S. Baumer, and H. Wickham, “Setting the stage for data science: integration of data management skills in introductory and second courses in statistics,” *arXiv preprint arXiv:1502.00318*, 2015.
- [43] T. Rattenbury, J. M. Hellerstein, J. Heer, S. Kandel, and C. Carreras, *Principles of data wrangling: Practical techniques for data preparation*. " O'Reilly Media, Inc.", 2017.
- [44] J. Morcos, Z. Abedjan, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker, “Datax-former: An interactive data transformation tool,” in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 883–888, 2015.
- [45] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker, “Datax-former: A robust transformation discovery system,” in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pp. 1134–1145, IEEE, 2016.
- [46] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*, vol. 72. Springer, 2015.
- [47] D. Loshin, *Master data management*. Morgan Kaufmann, 2010.

- [48] A. Forte and M. Guzdial, "Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses," *IEEE Transactions on Education*, vol. 48, no. 2, pp. 248–253, 2005.
- [49] J. B. Rode and M. M. Ringel, "Statistical software output in the classroom: A comparison of R and SPSS," *Teaching of Psychology*, vol. 46, no. 4, pp. 319–327, 2019.
- [50] D. F. Shell and L.-K. Soh, "Profiles of motivated self-regulation in college computer science courses: Differences in major versus required non-major courses," *Journal of Science Education and Technology*, vol. 22, no. 6, pp. 899–913, 2013.
- [51] M. N. Giannakos, I. O. Pappas, L. Jaccheri, and D. G. Sampson, "Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness," *Education and Information Technologies*, vol. 22, no. 5, pp. 2365–2382, 2017.
- [52] W. Finzer, "The data science education dilemma," *Technology Innovations in Statistics Education*, vol. 7, no. 2, 2013.
- [53] Z. Jiang, E. B. Fernandez, and L. Cheng, "P2n: A pedagogical pattern for teaching computer programming to non-CS majors," in *Proceedings of the 18th Conference on Pattern Languages of Programs*, pp. 1–9, 2011.
- [54] S. Krishnamurthi and K. Fisler, "Data-centricity: a challenge and opportunity for computing education," *Communications of the ACM*, vol. 63, no. 8, pp. 24–26, 2020.
- [55] Swanstrom, Ryan, "The Data Science 101 Blog." <https://ryanswanstrom.com/colleges/>, 2021. [Online; accessed 24-May-2021].
- [56] R. D. De Veaux, M. Agarwal, M. Averett, B. S. Baumer, A. Bray, T. C. Bressoud, L. Bryant, L. Z. Cheng, A. Francis, R. Gould, *et al.*, "Curriculum guidelines for undergraduate programs in data science," *Annual Review of Statistics and Its Application*, vol. 4, pp. 15–30, 2017.
- [57] M. Çetinkaya-Rundel and V. Ellison, "A fresh look at introductory data science," *Journal of Statistics Education*, pp. 1–11, 2020.
- [58] A. F. Blackwell, "End-user developers—what are they like?," *New perspectives in end-user development*, pp. 121–135, 2017.
- [59] B. A. Nardi, *A small matter of programming: perspectives on end user computing*. MIT press, 1993.

- [60] M. T. Mullarkey, A. R. Hevner, T. G. Gill, and K. Dutta, “Citizen data scientist: A design science research method for the conduct of data science projects,” in *International conference on design science research in information systems and technology*, pp. 191–205, Springer, 2019.
- [61] E. Codd, “Derivability, redundancy and consistency of relations stored in large data banks,” 1969.
- [62] T. Taipalus and V. Seppänen, “SQL education: A systematic mapping study and future research agenda,” *ACM Transactions on Computing Education (TOCE)*, vol. 20, no. 3, pp. 1–33, 2020.
- [63] Muenchen, Robert A, “The popularity of data analysis software.” <http://r4stats.com/articles/popularity//>, 2019. [Online; accessed 10-May-2021].
- [64] M. R. Zynda, “The first killer app: A history of spreadsheets,” *interactions*, vol. 20, no. 5, pp. 68–72, 2013.
- [65] F. F. J. Hermans, “Analyzing and visualizing spreadsheets,” 2013.
- [66] L. A. Pace and K. A. Barchard, “Using a spreadsheet programme to teach introductory statistics: Reducing anxiety and building conceptual understanding,” *International journal of innovation and learning*, vol. 3, no. 3, pp. 267–283, 2006.
- [67] M. Niazkar and S. H. Afzali, “Application of Excel spreadsheet in engineering education,” in *First International and Fourth National Conference on Engineering Education, Shiraz University*, pp. 10–12, 2015.
- [68] S. K. Katoch, “MS-Excel spreadsheet applications in introductory undergraduate physics — a review,” *J. Sci. Technol.*, vol. 5, pp. 48–52, 2020.
- [69] A. McNamara, “Key attributes of a modern statistical computing tool,” *The American Statistician*, 2018.
- [70] R. Panko, “What we don’t know about spreadsheet errors today: The facts, why we don’t believe them, and what we need to do,” *arXiv preprint arXiv:1602.02601*, 2016.
- [71] J. M. Hellerstein, J. Heer, and S. Kandel, “Self-service data preparation: Research to practice,” *IEEE Data Eng. Bull.*, vol. 41, no. 2, pp. 23–34, 2018.
- [72] A. E. Bartz and M. A. Sabolik, “Computer and software use in teaching the beginning statistics course,” *Teaching of Psychology*, vol. 28, no. 2, pp. 147–149, 2001.

- [73] H. Davidson, Y. Jabbari, H. Patton, F. O'Hagan, K. Peters, and R. Cribbie, "Statistical software use in Canadian university courses: Current trends and future directions," *Teaching of Psychology*, vol. 46, no. 3, pp. 246–250, 2019.
- [74] N. W. Paton, "Automating data preparation: Can we? should we? must we?," 2019.
- [75] B. Wu and C. A. Knoblock, "Maximizing correctness with minimal user effort to learn data transformations," in *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pp. 375–384, 2016.
- [76] S. Gulwani, "Automating string processing in spreadsheets using input-output examples," *ACM Sigplan Notices*, vol. 46, no. 1, pp. 317–330, 2011.
- [77] R. Singh and S. Gulwani, "Synthesizing number transformations from input-output examples," in *International Conference on Computer Aided Verification*, pp. 634–651, Springer, 2012.
- [78] S. Bhupatiraju, R. Singh, A.-r. Mohamed, and P. Kohli, "Deep API programmer: Learning to program with APIs," *arXiv preprint arXiv:1704.04327*, 2017.
- [79] V. Le and S. Gulwani, "Flashextract: A framework for data extraction by examples," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 542–553, 2014.
- [80] W. R. Harris and S. Gulwani, "Spreadsheet table transformations from examples," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 317–328, 2011.
- [81] S. Gulwani, W. R. Harris, and R. Singh, "Spreadsheet data manipulation using examples," *Communications of the ACM*, vol. 55, no. 8, pp. 97–105, 2012.
- [82] Z. Jin, M. R. Anderson, M. Cafarella, and H. Jagadish, "Foofah: Transforming data by example," in *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 683–698, 2017.
- [83] S. Gulwani, J. Hernández-Orallo, E. Kitzelmann, S. H. Muggleton, U. Schmid, and B. Zorn, "Inductive programming meets the real world," *Communications of the ACM*, vol. 58, no. 11, pp. 90–99, 2015.
- [84] C. J. Weiss, "Scientific computing for chemists: An undergraduate course in simulations, data processing, and visualization," *Journal of Chemical Education*, vol. 94, no. 5, pp. 592–597, 2017.
- [85] J. O. Holman, "Teaching Statistical Computing with Python in a Second Semester Undergraduate Business Statistics Course," *Business Education Innovation Journal VOLUME 10 NUMBER 2 December 2018*, p. 104, 2018.

- [86] D.-H. Kim, “Teaching r to undergraduate business students.,” *Business Education Innovation Journal*, vol. 11, no. 1, 2019.
- [87] R. Li, “Teaching Undergraduates R in an Introductory Research Methods Course: A Step-by-Step Approach,” *Journal of Political Science Education*, pp. 1–19, 2019.
- [88] A. A. David, “Introducing Python Programming into Undergraduate Biology,” *The American Biology Teacher*, vol. 83, no. 1, pp. 33–41, 2021.
- [89] S. J. Eglén, “A quick guide to teaching R programming to computational biology students,” *PLoS Comput Biol*, vol. 5, no. 8, p. e1000482, 2009.
- [90] A. Wilhelm, “Use R for Teaching Statistics in the Social Sciences?!,” *56th Session of the International Statistical Institute, Lisbon*, 2007.
- [91] M. Mascaró, A. I. Sacristán, and M. Rufino, “Teaching and learning statistics and experimental analysis for environmental science students, through programming activities in R,” in *Constructionism and Creativity-Proceedings 3rd Intl. Constructionism Conf*, pp. 407–416, 2014.
- [92] D. Langan and A. Wade, “Guidance for teaching R programming to non-statisticians,” in http://iase-web.org/documents/anzcots/OZCOTS_2016_Proceedings, vol. 9, pp. 141–146, Statistical Society of Australia Inc.(SSAI), 2016.
- [93] A. Luxton-Reilly, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo, “Introductory programming: a systematic literature review,” in *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pp. 55–106, 2018.
- [94] E. Patitsas, “A Numpy-First Approach to Teaching CS1 to Natural Science Students,” in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 333–333, 2015.
- [95] K. E. Iverson, “A personal view of APL,” *IBM Systems Journal*, vol. 30, no. 4, pp. 582–593, 1991.
- [96] A. M. Olney and S. D. Fleming, “A cognitive load perspective on the design of blocks languages for data science,” in *2019 IEEE Blocks and Beyond Workshop (B&B)*, pp. 95–97, IEEE, 2019.
- [97] D. Qiu, B. Li, and H. Leung, “Understanding the API usage in Java,” *Information and software technology*, vol. 73, pp. 81–100, 2016.
- [98] H. Wickham, “Data transformation,” in *ggplot2*, pp. 203–220, Springer, 2016.

- [99] J. Fox and A. Leange, “R and the Journal of Statistical Software,” *Journal of Statistical Software*, vol. 73, no. 1, pp. 1–13, 2016.
- [100] J. Fox, M. Bouchet-Valat, L. Andronic, M. Ash, T. Boye, S. Calza, A. Chang, P. Grosjean, R. Heiberger, K. K. Pour, *et al.*, “Package ‘rcmdr’,” 2020.
- [101] J. Allaire, Y. Xie, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone, “rmarkdown: Dynamic documents for r,” *R package version*, vol. 1, no. 11, 2018.
- [102] Z. Ross, H. Wickham, and D. Robinson, “Declutter your R workflow with tidy tools,” *PeerJ Preprints*, vol. 5, p. e3180v1, 2017.
- [103] H. Wickham, R. Francois, L. Henry, and K. Müller, “Dplyr,” in *useR! Conference*, 2014.
- [104] H. Wickham and M. H. Wickham, “Package ‘tidyr’,” *Easily Tidy Data with ‘spread’ and ‘gather ()’ Functions*, 2017.
- [105] R. dos Santos Ferreira, V. Y. Kataoka, and M. Karrer, “Teaching probability with the support of the R statistical software.,” *Statistics Education Research Journal*, vol. 13, no. 2, 2014.
- [106] B. Baumer, M. Cetinkaya-Rundel, A. Bray, L. Loi, and N. J. Horton, “R markdown: Integrating a reproducible analysis tool into introductory statistics,” *arXiv preprint arXiv:1402.1894*, 2014.
- [107] B. Stemock and L. Kerns, “Use of commercial and free software for teaching statistics.,” *Statistics Education Research Journal*, vol. 18, no. 2, pp. 54–68, 2019.
- [108] A. Counsell and R. A. Cribbie, “Students’ Attitudes toward Learning Statistics with R.,” *Psychology Teaching Review*, vol. 26, no. 2, pp. 36–56, 2020.
- [109] T. Rafalski, P. M. Uesbeck, C. Panks-Meloney, P. Daleiden, W. Allee, A. Mcnamara, and A. Stefik, “A Randomized Controlled Trial on the Wild Wild West of Scientific Computing with Student Learners,” in *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pp. 239–247, 2019.
- [110] F. Perez, B. E. Granger, and J. D. Hunter, “Python: an ecosystem for scientific computing,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 13–21, 2010.
- [111] J. M. Perkel, “Why Jupyter Is Data Scientists’ Computational Notebook of Choice,” *Nature*, vol. 563, no. 7729, pp. 145–146, 2018.

- [112] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.
- [113] W. McKinney *et al.*, “pandas: a foundational python library for data analysis and statistics,” *Python for High Performance and Scientific Computing*, vol. 14, no. 9, pp. 1–9, 2011.
- [114] A. C. Bart, J. Tibau, E. Tilevich, C. A. Shaffer, and D. Kafura, “Blockpy: An open access data-science environment for introductory programmers,” *Computer*, vol. 50, no. 5, pp. 18–26, 2017.
- [115] K. Thayer, S. E. Chasins, and A. J. Ko, “A theory of robust API knowledge,” *ACM Transactions on Computing Education (TOCE)*, vol. 21, no. 1, pp. 1–32, 2021.
- [116] M. Campbell-Kelly, M. Croarken, R. Flood, E. Robson, *et al.*, *The history of mathematical tables: from Sumer to spreadsheets*. Oxford University Press, 2003.
- [117] C. Konold, W. Finzer, and K. Kreetong, “Modeling as a core component of structuring data.,” *Statistics Education Research Journal*, vol. 16, no. 2, 2017.
- [118] T. Bressoud and D. White, “Tabular model: Structure and formats,” in *Introduction to Data Systems*, pp. 145–173, Springer, 2020.
- [119] H. Wickham *et al.*, “Tidy data,” *Journal of statistical software*, vol. 59, no. 10, pp. 1–23, 2014.
- [120] A. Falbel and C. Hancock, “Coordinating sets properties when representing data: the group separation problem,” in *Proceeding 17nd Annual Meeting of the International Group for the Psychology of Mathematics Education*, vol. 2, pp. 17–24, 1993.
- [121] A. B. Markman and E. J. Wisniewski, “Similar and different: The differentiation of basic-level categories.,” *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 23, no. 1, p. 54, 1997.
- [122] L. Hajibayova, “Basic-level categories: A review,” *Journal of Information Science*, vol. 39, no. 5, pp. 676–687, 2013.
- [123] L. C. Haldar, N. Wong, J. I. Heller, and C. Konold, “Students making sense of multi-level data,” *Technology Innovations in Statistics Education*, vol. 11, no. 1, 2018.
- [124] H. Wickham *et al.*, “The split-apply-combine strategy for data analysis,” *Journal of statistical software*, vol. 40, no. 1, pp. 1–29, 2011.

- [125] S. Estrella, “Data representations in early statistics: Data sense, meta-representational competence and transnumeration,” in *Statistics in early childhood and primary education*, pp. 239–256, Springer, 2018.
- [126] H. Chick, “Tools for transnumeration: Early stages in the art of data representation,” in *Mathematics Education for the Third Millennium: Towards 2010. Proceedings of the Twenty-seventh Annual Conference of the Mathematics Education Research Group of Australasia*, pp. 167–174, 2004.
- [127] D. Miedema, E. Aivaloglou, and G. Fletcher, “Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study,” in *Proceedings of the 17th ACM Conference on International Computing Education Research*, pp. 355–367, 2021.
- [128] T. Taipalus, “Explaining Causes Behind SQL Query Formulation Errors,” in *2020 IEEE Frontiers in Education Conference (FIE)*, pp. 1–9, IEEE, 2020.
- [129] P. Reisner, “Use of psychological experimentation as an aid to development of a query language,” *IEEE Transactions on Software Engineering*, no. 3, pp. 218–229, 1977.
- [130] A. Ahadi, J. Prior, V. Behbood, and R. Lister, “Students’ semantic mistakes in writing seven different types of sql queries,” in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 272–277, 2016.
- [131] L. J. Worrall and R. J. Quinn, “Promoting conceptual understanding of matrices,” *The Mathematics Teacher*, vol. 94, no. 1, pp. 46–49, 2001.
- [132] T. Murphy and C. Williams, “Development of a matlab app for improving student learning of the use of arrays,” 2020.
- [133] J. Wrenn and S. Krishnamurthi, “Executable examples for programming problem comprehension,” in *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pp. 131–139, 2019.
- [134] J. Prather, R. Pettit, K. McMurry, A. Peters, J. Homer, and M. Cohen, “Metacognitive difficulties faced by novice programmers in automated assessment tools,” in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pp. 41–50, 2018.
- [135] M. J. Nathan, W. Kintsch, and E. Young, “A theory of algebra-word-problem comprehension and its implications for the design of learning environments,” *Cognition and instruction*, vol. 9, no. 4, pp. 329–389, 1992.

- [136] D. Bouvier, E. Lovellette, J. Matta, B. Alshaigy, B. A. Becker, M. Craig, J. Jackova, R. McCartney, K. Sanders, and M. Zarb, “Novice programmers and the problem description effect,” in *Proceedings of the 2016 ITiCSE Working Group Reports*, pp. 103–118, 2016.
- [137] E. Lovellette, J. Matta, D. Bouvier, and R. Frye, “Just the numbers: an investigation of contextualization of problems for novice programmers,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pp. 393–398, 2017.
- [138] M. Craig, J. Smith, and A. Petersen, “Familiar contexts and the difficulty of programming problems,” in *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, pp. 123–127, 2017.
- [139] J. Leinonen, P. Denny, and J. Whalley, “Exploring the effects of contextualized problem descriptions on problem solving,” in *Australasian Computing Education Conference*, pp. 30–39, 2021.
- [140] H. A. Simon, “The structure of ill structured problems,” *Artificial intelligence*, vol. 4, no. 3-4, pp. 181–201, 1973.
- [141] M. C. Linn and M. J. Clancy, “The case for case studies of programming problems,” *Communications of the ACM*, vol. 35, no. 3, pp. 121–132, 1992.
- [142] J. R. Anderson, F. G. Conrad, and A. T. Corbett, “Skill acquisition and the LISP tutor,” *Cognitive Science*, vol. 13, no. 4, pp. 467–505, 1989.
- [143] J. J. Van Merriënboer and F. G. Paas, “Automation and schema acquisition in learning elementary computer programming: Implications for the design of practice,” *Computers in human behavior*, vol. 6, no. 3, pp. 273–289, 1990.
- [144] E. Soloway, “From problems to programs via plans: The content and structure of knowledge for introductory LISP programming,” *Journal of Educational Computing Research*, vol. 1, no. 2, pp. 157–172, 1985.
- [145] R. S. Rist, “Knowledge creation and retrieval in program design: A comparison of novice and intermediate student programmers,” *Human-Computer Interaction*, vol. 6, no. 1, pp. 1–46, 1991.
- [146] F. E. V. Castro and K. Fisler, “Qualitative analyses of movements between task-level and code-level thinking of novice programmers,” in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pp. 487–493, 2020.
- [147] H. Glaser, P. H. Hartel, and P. W. Garratt, “Programming by numbers: a programming method for novices,” *The Computer Journal*, vol. 43, no. 4, pp. 252–265, 2000.

- [148] J. C. Spohrer and E. Soloway, “Novice mistakes: Are the folk wisdoms correct?,” *Communications of the ACM*, vol. 29, no. 7, pp. 624–632, 1986.
- [149] J. Spohrer and E. Soloway, “Alternatives to construct-based programming misconceptions,” in *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 183–191, 1986.
- [150] D. Norman, *The design of everyday things: Revised and expanded edition*. Basic books, 2013.
- [151] H. Fangohr, “A comparison of C, MATLAB, and Python as teaching languages in engineering,” in *International Conference on Computational Science*, pp. 1210–1217, Springer, 2004.
- [152] W. Wang, A. Kwatra, J. Skripchuk, N. Gomes, A. Milliken, C. Martens, T. Barnes, and T. Price, “Novices’ learning barriers when using code examples in open-ended programming,” *arXiv preprint arXiv:2104.11806*, 2021.
- [153] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, “Two studies of opportunistic programming: interleaving web foraging, learning, and writing code,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1589–1598, 2009.
- [154] B. Dorn and M. Guzdial, “Learning on the job: characterizing the programming knowledge and learning strategies of web designers,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 703–712, 2010.
- [155] B. Dorn, A. Stankiewicz, and C. Roggi, “Lost while searching: Difficulties in information seeking among end-user programmers,” *Proceedings of the American Society for Information Science and Technology*, vol. 50, no. 1, pp. 1–10, 2013.
- [156] G. Gao, F. Voichick, M. Ichinco, and C. Kelleher, “Exploring Programmers’ API Learning Processes: Collecting Web Resources as External Memory,” in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1–10, IEEE, 2020.
- [157] Y. Ben-David Kolikant and Z. ma’ayan, “Computer science students’ use of the internet for academic purposes: Difficulties and learning processes,” *Computer Science Education*, vol. 28, no. 3, pp. 211–231, 2018.
- [158] A. J. Ko, B. A. Myers, and H. H. Aung, “Six learning barriers in end-user programming systems,” in *2004 IEEE Symposium on Visual Languages-Human Centric Computing*, pp. 199–206, IEEE, 2004.

- [159] H. Nygren, J. Leinonen, and A. Hellas, “Tracking Students’ Internet Browsing in a Machine Exam,” in *Proceedings of the 6th Computer Science Education Research Conference*, pp. 91–95, 2017.
- [160] A. J. Ko and Y. Riche, “The role of conceptual knowledge in api usability,” in *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 173–176, IEEE, 2011.
- [161] B. Skudder and A. Luxton-Reilly, “Worked examples in computer science,” in *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148*, pp. 59–64, 2014.
- [162] J. Brandt, V. Pattamatta, W. Choi, B. Hsieh, and S. R. Klemmer, “Rehearse: Helping programmers adapt examples by visualizing execution and highlighting related code,” tech. rep., Citeseer, 2010.
- [163] B. Lennon, “Foo, bar, baz. . . : The metasyntactic variable and the programming language hierarchy,” *Philosophy & Technology*, vol. 34, no. 1, pp. 13–32, 2021.
- [164] M. Ichinco, K. J. Harms, and C. Kelleher, “Towards understanding successful novice example user in blocks-based programming,” *Journal of Visual Languages and Sentient Systems*, vol. 3, pp. 101–118, 2017.
- [165] T. Taipalus, M. Siponen, and T. Vartiainen, “Errors and complications in SQL query formulation,” *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 3, pp. 1–29, 2018.
- [166] P. Denny, J. Prather, and B. A. Becker, “Error message readability and novice debugging performance,” in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 480–486, 2020.
- [167] A. Ahadi, V. Behbood, A. Vihavainen, J. Prior, and R. Lister, “Students’ syntactic mistakes in writing seven different types of SQL queries and its application to predicting students’ success,” in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pp. 401–406, 2016.
- [168] R. McCauley, S. Fitzgerald, G. Lewandowski, L. Murphy, B. Simon, L. Thomas, and C. Zander, “Debugging: a review of the literature from an educational perspective,” *Computer Science Education*, vol. 18, no. 2, pp. 67–92, 2008.
- [169] R. Bednarik and M. Tukiainen, “Temporal eye-tracking data: evolution of debugging strategies with multiple representations,” in *Proceedings of the 2008 symposium on Eye tracking research & applications*, pp. 99–102, 2008.

- [170] J. B. Smelcer, "User errors in database query composition," *International Journal of Human-Computer Studies*, vol. 42, no. 4, pp. 353–381, 1995.
- [171] O. Yarygina, "Learning analytics of CS0 students programming errors: the case of data science minor," in *Proceedings of the 23rd International Conference on Academic Mindtrek*, pp. 149–152, 2020.
- [172] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cognitive science*, vol. 12, no. 2, pp. 257–285, 1988.
- [173] Z. Ozcinar, "The topic of instructional design in research journals: A citation analysis for the years 1980-2008," *Australasian Journal of Educational Technology*, vol. 25, no. 4, 2009.
- [174] L. Malmi, J. Sheard, R. Bednarik, J. Helminen, P. Kinnunen, A. Korhonen, N. Myller, J. Sorva, and A. Taherkhani, "Theoretical underpinnings of computing education research: what is the evidence?," in *Proceedings of the tenth annual conference on International computing education research*, pp. 27–34, 2014.
- [175] W. Schnotz and C. Kürschner, "A reconsideration of cognitive load theory," *Educational psychology review*, vol. 19, no. 4, pp. 469–508, 2007.
- [176] J. Sweller, "Some cognitive processes and their consequences for the organisation and presentation of information," *Australian Journal of Psychology*, vol. 45, no. 1, pp. 1–8, 1993.
- [177] D. Weintrop and U. Wilensky, "To block or not to block, that is the question: students' perceptions of blocks-based programming," in *Proceedings of the 14th international conference on interaction design and children*, pp. 199–208, 2015.
- [178] N. C. Brown, A. Altadmri, and M. Kölling, "Frame-based editing: Combining the best of blocks and text programming," in *2016 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, pp. 47–53, IEEE, 2016.
- [179] R. E. Mayer, "Using multimedia for e-learning," *Journal of Computer Assisted Learning*, vol. 33, no. 5, pp. 403–423, 2017.
- [180] J. Sweller, J. J. Van Merriënboer, and F. G. Paas, "Cognitive architecture and instructional design," *Educational psychology review*, vol. 10, no. 3, pp. 251–296, 1998.
- [181] S. Kalyuga, "Cognitive load theory: How many types of load does it really need?," *Educational Psychology Review*, vol. 23, no. 1, pp. 1–19, 2011.

- [182] K. R. Popper, “Science as falsification,” *Conjectures and refutations*, vol. 1, no. 1963, pp. 33–39, 1963.
- [183] P. Gerjets, K. Scheiter, and G. Cierniak, “The scientific value of cognitive load theory: A research agenda based on the structuralist view of theories,” *Educational Psychology Review*, vol. 21, no. 1, pp. 43–54, 2009.
- [184] J. D. Sneed, *The logical structure of mathematical physics*, vol. 35. Springer Science & Business Media, 2012.
- [185] M. Guzdial, L. Hohmann, M. Konneman, C. Walton, and E. Soloway, “Supporting programming and learning-to-program with an integrated cad and scaffolding workbench,” *Interactive Learning Environments*, vol. 6, no. 1-2, pp. 143–179, 1998.
- [186] D. Wood, J. S. Bruner, and G. Ross, “The role of tutoring in problem solving,” *Journal of child psychology and psychiatry*, vol. 17, no. 2, pp. 89–100, 1976.
- [187] A. Shvarts and A. Bakker, “The early history of the scaffolding metaphor: Bernstein, Luria, Vygotsky, and before,” *Mind, Culture, and Activity*, vol. 26, no. 1, pp. 4–23, 2019.
- [188] E. Ackermann, “Piaget’s constructivism, Papert’s constructionism: What’s the difference,” *Future of learning group publication*, vol. 5, no. 3, p. 438, 2001.
- [189] F. Hermans and M. Smit, “Explicit direct instruction in programming education,” in *Proceedings of the 29th Annual Conference of the Psychology of Programming Interest Group (PPIG 2018)*, pp. 86–93, 2018.
- [190] P. Kirschner, J. Sweller, and R. E. Clark, “Why unguided learning does not work: An analysis of the failure of discovery learning, problem-based learning, experiential learning and inquiry-based learning,” *Educational Psychologist*, vol. 41, no. 2, pp. 75–86, 2006.
- [191] J. Sweller, P. A. Kirschner, and R. E. Clark, “Why minimally guided teaching techniques do not work: A reply to commentaries,” *Educational psychologist*, vol. 42, no. 2, pp. 115–121, 2007.
- [192] I. Biederman and M. M. Shiffrar, “Sexing day-old chicks: A case study and expert systems analysis of a difficult perceptual-learning task,” *Journal of Experimental Psychology: Learning, memory, and cognition*, vol. 13, no. 4, p. 640, 1987.
- [193] tidyblocks.tech, “TidyBlocks.” <https://github.com/tidyblocks/tidyblocks>, 2021. [Online; accessed 21-Feb-2021].
- [194] M. K. Kjølsvik and E. H. Schultheis, “Getting messy with authentic data: Exploring the potential of using data from scientific research to support student data literacy,” *CBE—Life Sciences Education*, vol. 18, no. 2, p. es2, 2019.

- [195] A. C. Bart, “Situating computational thinking with big data: Pedagogy and technology,” in *Proceedings of the 46th ACM technical symposium on computer science education*, pp. 719–719, 2015.
- [196] A. C. Bart, R. Whitcomb, D. Kafura, C. A. Shaffer, and E. Tilevich, “Computing with corgis: Diverse, real-world datasets for introductory computing,” *ACM Inroads*, vol. 8, no. 2, pp. 66–72, 2017.
- [197] A. C. Bart, D. Kafura, C. A. Shaffer, and E. Tilevich, “Reconciling the promise and pragmatics of enhancing computing pedagogy with data science,” in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pp. 1029–1034, 2018.
- [198] M. Guzdial and J. Robertson, “Too much programming too soon?,” *Communications of the ACM*, vol. 53, no. 3, pp. 10–11, 2010.
- [199] E. Riese and S. Stenbom, “Experiences of assessment in introductory programming from the perspective of noncomputer science majors,” in *2020 IEEE Frontiers in Education Conference (FIE)*, pp. 1–9, IEEE, 2020.
- [200] M. Guzdial, “Does contextualized computing education help?,” *ACM Inroads*, vol. 1, no. 4, pp. 4–6, 2010.
- [201] A. Paivio, *Mental representations: A dual coding approach*. Oxford University Press, 1990.
- [202] J. Cuevas and B. L. Dawson, “A test of two alternative cognitive processing models: Learning styles and dual coding,” *Theory and Research in Education*, vol. 16, no. 1, pp. 40–64, 2018.
- [203] E. R. Tufte, N. H. Goeler, and R. Benson, *Envisioning information*, vol. 2. Graphics press Cheshire, CT, 1990.
- [204] J. H. Larkin and H. A. Simon, “Why a diagram is (sometimes) worth ten thousand words,” *Cognitive science*, vol. 11, no. 1, pp. 65–100, 1987.
- [205] D. Todorovic, “Gestalt principles,” *Scholarpedia*, vol. 3, no. 12, p. 5345, 2008.
- [206] S. Ainsworth, “Deft: A conceptual framework for considering learning with multiple representations,” *Learning and instruction*, vol. 16, no. 3, pp. 183–198, 2006.
- [207] F. Gobet, “Chunking models of expertise: Implications for education,” *Applied Cognitive Psychology*, vol. 19, no. 2, pp. 183–204, 2005.

- [208] J. Sorva, V. Karavirta, and L. Malmi, “A review of generic program visualization systems for introductory programming education,” *ACM Transactions on Computing Education (TOCE)*, vol. 13, no. 4, pp. 1–64, 2013.
- [209] J. Hidalgo-Céspedes, G. Marín-Raventós, and V. Lara-Villagrán, “Learning principles in program visualizations: a systematic literature review,” in *2016 IEEE frontiers in education conference (FIE)*, pp. 1–9, IEEE, 2016.
- [210] K. Romanowska, G. Singh, M. A. A. Dewan, and F. Lin, “Towards developing an effective algorithm visualization tool for online learning,” in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pp. 2011–2016, IEEE, 2018.
- [211] R. Kearns, S. Shead, and A. Fekete, “A teaching system for SQL,” in *Proceedings of the 2nd Australasian conference on Computer science education*, pp. 224–231, 1997.
- [212] M. Cembalo, A. De Santis, and U. Ferraro Petrillo, “SAVI: a new system for advanced SQL visualization,” in *Proceedings of the 2011 conference on Information technology education*, pp. 165–170, 2011.
- [213] S. W. Dietrich, D. Goelman, C. M. Borrer, and S. M. Crook, “An animated introduction to relational databases for many majors,” *IEEE Transactions on Education*, vol. 58, no. 2, pp. 81–89, 2014.
- [214] P. Garner and J. Mariani, “Learning SQL in steps,” *Learning*, vol. 12, p. 23, 2015.
- [215] K. A. T. Folland, “viSQLizer: Using visualization for learning SQL,” in *Norsk IKT-konferanse for forskning og utdanning*, 2016.
- [216] L. Mohan and R. L. Kashyap, “A visual query language for graphical interaction with schema-intensive databases,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 5, pp. 843–858, 1993.
- [217] G. Viehstaedt and A. L. Ambler, “Visual representation and manipulation of matrices,” *Journal of Visual Languages & Computing*, vol. 3, no. 3, pp. 273–298, 1992.
- [218] J. L. Leopold and A. L. Ambler, “A user interface for the visualization and manipulation of arrays,” in *Proceedings 1996 IEEE Symposium on Visual Languages*, pp. 54–55, IEEE, 1996.
- [219] D. Mason and K. Dave, “Block-based versus flow-based programming for naive programmers,” in *2017 IEEE Blocks and Beyond Workshop (B&B)*, pp. 25–28, IEEE, 2017.

- [220] M. Hofmann and R. Klinkenberg, *RapidMiner: Data mining use cases and business analytics applications*. CRC Press, 2016.
- [221] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: an update,” *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.
- [222] J. Demšar, T. Curk, A. Erjavec, Č. Gorup, T. Hočevár, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, *et al.*, “Orange: data mining toolbox in python,” *the Journal of machine Learning research*, vol. 14, no. 1, pp. 2349–2353, 2013.
- [223] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, and B. Wiswedel, “Knime—the konstanz information miner: version 2.0 and beyond,” *AcM SIGKDD explorations Newsletter*, vol. 11, no. 1, pp. 26–31, 2009.
- [224] K. Covington and A. Parikh, “The Red-R framework for integrated discovery,” *The Red-R Journal*, vol. 1, 2011.
- [225] N. Shrestha, C. Botta, T. Barik, and C. Parnin, “Here we go again: why is it difficult for developers to learn another programming language?,” in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pp. 691–701, IEEE, 2020.
- [226] I.-H. Hsiao and C. López, “Lessons learned from students’ cheat sheets: Generic models for designing programming study guides,” in *2016 IEEE 16th International Conference on Advanced Learning Technologies (ICALT)*, pp. 209–211, IEEE, 2016.
- [227] M. de Raadt, “Student created cheat-sheets in examinations: impact on student outcomes,” in *Proceedings of the Fourteenth Australasian Computing Education Conference*, vol. 123, pp. 71–76, 2012.
- [228] S. Hamouda and C. A. Shaffer, “Crib sheets and exam performance in a data structures course,” *Computer Science Education*, vol. 26, no. 1, pp. 1–26, 2016.
- [229] E. Soloway, “Learning to program= learning to construct mechanisms and explanations,” *Communications of the ACM*, vol. 29, no. 9, pp. 850–858, 1986.
- [230] D. Ginat, “Interleaved pattern composition and scaffolded learning,” in *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*, pp. 109–113, 2009.
- [231] D. Ginat, E. Menashe, and A. Taya, “Novice difficulties with interleaved pattern composition,” in *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, pp. 57–67, Springer, 2013.

- [232] A. Ebrahimi, “VPCL: a visual language for teaching and learning programming.(A picture is worth a thousand words),” *Journal of Visual Languages & Computing*, vol. 3, no. 3, pp. 299–317, 1992.
- [233] H. Al-Shuaily, *SQL pattern design, development & evaluation of its efficacy*. PhD thesis, University of Glasgow, 2013.
- [234] L. Margulieux and R. Catrambone, “Using learners’ self-explanations of subgoals to guide initial problem solving in app inventor,” in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pp. 21–29, 2017.
- [235] L. E. Margulieux, M. Guzdial, and R. Catrambone, “Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications,” in *Proceedings of the ninth annual international conference on International computing education research*, pp. 71–78, 2012.
- [236] L. E. Margulieux, R. Catrambone, and M. Guzdial, “Subgoal labeled worked examples improve K-12 teacher performance in computer programming training,” 2013.
- [237] L. E. Margulieux and R. Catrambone, “Improving problem solving performance in computer-based learning environments through subgoal labels,” in *Proceedings of the first ACM conference on Learning@ scale conference*, pp. 149–150, 2014.
- [238] L. E. Margulieux and R. Catrambone, “Improving problem solving with subgoal labels in expository text and worked examples,” *Learning and Instruction*, vol. 42, pp. 58–71, 2016.
- [239] L. E. Margulieux and R. Catrambone, “Finding the best types of guidance for constructing self-explanations of subgoals in programming,” *Journal of the Learning Sciences*, vol. 28, no. 1, pp. 108–151, 2019.
- [240] B. B. Morrison, L. E. Margulieux, and M. Guzdial, “Subgoals, context, and worked examples in learning computing problem solving,” in *Proceedings of the eleventh annual international conference on international computing education research*, pp. 21–29, 2015.
- [241] B. B. Morrison, L. E. Margulieux, B. Ericson, and M. Guzdial, “Subgoals help students solve Parsons problems,” in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pp. 42–47, 2016.
- [242] L. E. Margulieux, B. B. Morrison, and A. Decker, “Design and pilot testing of subgoal labeled worked examples for five core concepts in CS1,” in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 548–554, 2019.

- [243] P. J. Guo, S. Kandel, J. M. Hellerstein, and J. Heer, “Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 65–74, 2011.
- [244] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, “Example-centric programming: integrating web search into the development environment,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 513–522, 2010.
- [245] M. Ichinco, W. Y. Hnin, and C. L. Kelleher, “Suggesting api usage to novice programmers with the example guru,” in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pp. 1105–1117, 2017.
- [246] M. Ichinco and C. Kelleher, “The need for improved support for interacting with block examples,” in *2017 IEEE Blocks and Beyond Workshop (B&B)*, pp. 69–70, IEEE, 2017.
- [247] A. Woodruff, A. Faulring, R. Rosenholtz, J. Morrisson, and P. Pirolli, “Using thumbnails to search the web,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 198–205, 2001.
- [248] D. Wightman, Z. Ye, J. Brandt, and R. Vertegaal, “Snipmatch: Using source code context to enhance snippet retrieval and parameterization,” in *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pp. 219–228, 2012.
- [249] S. Gray, C. St. Clair, R. James, and J. Mead, “Suggestions for graduated exposure to programming concepts using fading worked examples,” in *Proceedings of the third international workshop on Computing education research*, pp. 99–110, 2007.
- [250] A. Dahotre, V. Krishnamoorthy, M. Corley, and C. Scaffidi, “Using intelligent tutors to enhance student learning of application programming interfaces,” *The Journal of Computing Sciences in Colleges*, p. 195, 2011.
- [251] O. Hazzan, N. Ragonis, and T. Lapidot, “Data science and computer science education,” in *Guide to Teaching Computer Science*, pp. 95–117, Springer, 2020.
- [252] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, “What makes a good code example?: A study of programming Q&A in StackOverflow,” in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pp. 25–34, IEEE, 2012.
- [253] M. Ichinco and C. Kelleher, “Towards block code examples that help young novices notice critical elements,” in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 335–336, IEEE, 2017.

- [254] M. Ichinco and C. Kelleher, “Towards better code snippets: Exploring how code snippet recall differs with programming experience,” in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 37–41, IEEE, 2017.
- [255] M. A. Kraft, “Interpreting effect sizes of education interventions,” *Educational Researcher*, vol. 49, no. 4, pp. 241–253, 2020.
- [256] J. Westfall and T. Yarkoni, “Statistically controlling for confounding constructs is harder than you think,” *PloS one*, vol. 11, no. 3, p. e0152719, 2016.
- [257] A. Lishinski, J. Good, P. Sands, and A. Yadav, “Methodological rigor and theoretical foundations of cs education research,” in *Proceedings of the 2016 ACM conference on international computing education research*, pp. 161–169, 2016.
- [258] D. A. Cook and T. J. Beckman, “Reflections on experimental research in medical education,” *Advances in health sciences education*, vol. 15, no. 3, pp. 455–464, 2010.
- [259] P. Hawe, A. Shiell, and T. Riley, “Complex interventions: how “out of control” can a randomised controlled trial be?,” *Bmj*, vol. 328, no. 7455, pp. 1561–1563, 2004.
- [260] L. J. Cronbach and K. Shapiro, *Designing evaluations of educational and social programs*. Jossey-Bass,, 1982.
- [261] G. M. Sullivan, “Getting off the “gold standard”: randomized controlled trials and education research,” *Journal of graduate medical education*, vol. 3, no. 3, pp. 285–289, 2011.
- [262] D. A. Cook, “If you teach them, they will learn: why medical education needs comparative effectiveness research,” 2012.
- [263] R. Slavin and N. A. Madden, “Measures inherent to treatments in program effectiveness reviews,” *Journal of Research on Educational Effectiveness*, vol. 4, no. 4, pp. 370–380, 2011.
- [264] A. C. Cheung and R. E. Slavin, “How methodological features affect effect sizes in education,” *Educational Researcher*, vol. 45, no. 5, pp. 283–292, 2016.
- [265] A. Decker and M. M. McGill, “A topical review of evaluation instruments for computing education,” in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 558–564, 2019.
- [266] R. A. Charter, “Study samples are too small to produce sufficiently precise reliability coefficients,” *The Journal of General Psychology*, vol. 130, no. 2, pp. 117–129, 2003.

- [267] M. C. Parker, M. Guzdial, and S. Engleman, “Replication, validation, and use of a language independent CS1 knowledge assessment,” in *Proceedings of the 2016 ACM conference on international computing education research*, pp. 93–101, 2016.
- [268] M. C. Morris and R. M. Nelson, “Randomized, controlled trials as minimal risk: an ethical analysis,” *Critical care medicine*, vol. 35, no. 3, pp. 940–944, 2007.
- [269] C. Cook and C. Sheets, “Clinical equipoise and personal equipoise: two necessary ingredients for reducing bias in manual therapy trials,” *Journal of Manual & Manipulative Therapy*, vol. 19, no. 1, pp. 55–57, 2011.
- [270] “Ethical Guidelines for Educational Research, fourth edition (2018) | BERA.”
- [271] L. Margulieux, T. A. Ketenci, and A. Decker, “Review of measurements used in computing education research and suggestions for increasing standardization,” *Computer Science Education*, vol. 29, no. 1, pp. 49–78, 2019.
- [272] P. Sedlmeier and G. Gigerenzer, “Do studies of statistical power have an effect on the power of studies?,” 1992.
- [273] R. Wetzels, D. Matzke, M. D. Lee, J. N. Rouder, G. J. Iverson, and E.-J. Wagenmakers, “Statistical evidence in experimental psychology: An empirical comparison using 855 t tests,” *Perspectives on Psychological Science*, vol. 6, no. 3, pp. 291–298, 2011.
- [274] E. Loken and A. Gelman, “Measurement error and the replication crisis,” *Science*, vol. 355, no. 6325, pp. 584–585, 2017.
- [275] A. Ahadi, A. Hellas, P. Ihtola, A. Korhonen, and A. Petersen, “Replication in computing education research: researcher attitudes and experiences,” in *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pp. 2–11, 2016.
- [276] H. Lortie-Forgues and M. Inglis, “Rigorous large-scale educational rcts are often uninformative: Should we be concerned?,” *Educational Researcher*, vol. 48, no. 3, pp. 158–166, 2019.
- [277] A. Simpson, “On the misinterpretation of effect size,” *Educational Studies in Mathematics*, vol. 103, no. 1, pp. 125–133, 2020.
- [278] L. Sundin and Q. Cutts, “Is it feasible to teach query programming in three different languages in a single session? a study on a pattern-oriented tutorial and cheat sheets,” in *Proceedings of the 1st UK & Ireland Computing Education Research Conference*, pp. 1–7, 2019.

- [279] D. Gentner, "Structure-mapping: A theoretical framework for analogy," *Cognitive science*, vol. 7, no. 2, pp. 155–170, 1983.
- [280] W. Pouw, G. Rop, B. De Koning, and F. Paas, "The cognitive basis for the split-attention effect.," *Journal of Experimental Psychology: General*, vol. 148, no. 11, p. 2058, 2019.
- [281] N. L. Schroeder and A. T. Cenkci, "Spatial contiguity and spatial split-attention effects in multimedia learning environments: A meta-analysis," 2018.
- [282] S. Puma, N. Matton, P.-V. Paubel, and A. Tricot, "Cognitive load theory and time considerations: Using the time-based resource sharing model," *Educational Psychology Review*, vol. 30, no. 3, pp. 1199–1214, 2018.
- [283] C. Zerby, *The devil's details: A history of footnotes*. Simon and Schuster, 2007.
- [284] C. I. Johnson and R. E. Mayer, "An eye movement analysis of the spatial contiguity effect in multimedia learning.," *Journal of Experimental Psychology: Applied*, vol. 18, no. 2, p. 178, 2012.
- [285] S. Kalyuga, P. Chandler, and J. Sweller, "Managing split-attention and redundancy in multimedia instruction," *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, vol. 13, no. 4, pp. 351–371, 1999.
- [286] E. Ozcelik, T. Karakus, E. Kursun, and K. Cagiltay, "An eye-tracking study of how color coding affects multimedia learning," *Computers & Education*, vol. 53, no. 2, pp. 445–453, 2009.
- [287] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.
- [288] O. Byrne, "The Elements of Euclid," *William Pickering, London*, 1847.
- [289] A. F. Toney, K. M. Slaten, and E. F. Peters, "Color work to enhance proof-writing in geometry," *Journal of the California Mathematics Project*, vol. 6, pp. 9–20, 2013.
- [290] T. Sułkowski. <https://twitter.com/sulco/status/1281545450273865730?s=20>, July 2020. Tweet (@sulco). Retrieved on 2021-08-30.
- [291] G. Qian, "Teaching SQL: a divide-and-conquer method for writing queries," *Journal of Computing Sciences in Colleges*, vol. 33, no. 4, pp. 37–44, 2018.
- [292] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 26, no. 1, pp. 64–69, 1983.

- [293] S. Sosnovsky and T. Gavrilova, "Development of educational ontology for C-programming," 2006.
- [294] B. Swartout, R. Patil, K. Knight, and T. Russ, "Toward distributed use of large-scale ontologies," in *Proc. of the Tenth Workshop on Knowledge Acquisition for Knowledge-Based Systems*, pp. 138–148, 1996.
- [295] N. F. Noy, D. L. McGuinness, *et al.*, "Ontology development 101: A guide to creating your first ontology," 2001.
- [296] R. de Almeida Falbo, "Sabio: Systematic approach for building ontologies.," in *ONTO.COM/ODISE@ FOIS*, 2014.
- [297] A. Conde, M. Larrañaga, A. Arruarte, J. A. Elorriaga, and D. Roth, "litewi: A combined term extraction and entity linking method for eliciting educational ontologies from textbooks," *Journal of the Association for Information Science and Technology*, vol. 67, no. 2, pp. 380–399, 2016.
- [298] P. Vrablecová and M. Šimko, "Supporting semantic annotation of educational content by automatic extraction of hierarchical domain relationships," *IEEE Transactions on Learning Technologies*, vol. 9, no. 3, pp. 285–298, 2016.
- [299] A. Jiomekong, G. Camara, and M. Tchunte, "Extracting ontological knowledge from java source code using hidden markov models," *Open Computer Science*, vol. 9, no. 1, pp. 181–199, 2019.
- [300] C. Yan and Y. He, "Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1539–1554, 2020.
- [301] P. Teetor, *R cookbook: Proven recipes for data analysis, statistics, and graphics*. "O'Reilly Media, Inc.", 2011.
- [302] B. Vickery, "Faceted classification for the web," *Axiomathes*, vol. 18, no. 2, pp. 145–160, 2008.
- [303] D. P. Miller, "The depth/breadth tradeoff in hierarchical computer menus," in *Proceedings of the Human Factors Society Annual Meeting*, vol. 25, pp. 296–300, SAGE Publications Sage CA: Los Angeles, CA, 1981.
- [304] J. I. Kiger, "The depth/breadth trade-off in the design of menu-driven user interfaces," *International journal of man-machine studies*, vol. 20, no. 2, pp. 201–213, 1984.

- [305] S. L. Wise and C. E. DeMars, "Low examinee effort in low-stakes assessment: Problems and potential solutions," *Educational assessment*, vol. 10, no. 1, pp. 1–17, 2005.
- [306] A. Wigfield and J. S. Eccles, "Expectancy–value theory of achievement motivation," *Contemporary educational psychology*, vol. 25, no. 1, pp. 68–81, 2000.
- [307] K. R. Koedinger and V. Aleven, "Exploring the assistance dilemma in experiments with cognitive tutors," *Educational Psychology Review*, vol. 19, no. 3, pp. 239–264, 2007.
- [308] S. Marwan, A. Dombe, and T. W. Price, "Unproductive help-seeking in programming: what it is and how to address it," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 54–60, 2020.
- [309] M. Roskes, A. J. Elliot, B. A. Nijstad, and C. K. De Dreu, "Time pressure undermines performance more under avoidance than approach motivation," *Personality and Social Psychology Bulletin*, vol. 39, no. 6, pp. 803–813, 2013.
- [310] K. Nolan and S. Bergin, "The role of anxiety when learning to program: a systematic review of the literature," in *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pp. 61–70, 2016.
- [311] A. L. Brown and R. A. Ferrara, "Diagnosing zones of proximal development," *Culture, communication, and cognition: Vygotskian perspectives*, pp. 273–305, 1985.
- [312] J. C. Campione, "Assisted assessment: A taxonomy of approaches and an outline of strengths and weaknesses," *Journal of learning disabilities*, vol. 22, no. 3, pp. 151–165, 1989.
- [313] S. L. Wise and X. Kong, "Response time effort: A new measure of examinee motivation in computer-based tests," *Applied Measurement in Education*, vol. 18, no. 2, pp. 163–183, 2005.
- [314] L. Sundin and Q. Cutts, "Introducing data wrangling using graphical subgoals-findings from an e-learning study," in *Proceedings of the Eighth ACM Conference on Learning@Scale*, pp. 267–270, 2021.
- [315] C. Albers and D. Lakens, "When power analyses based on pilot data are biased: Inaccurate effect size estimators and follow-up bias," *Journal of experimental social psychology*, vol. 74, pp. 187–195, 2018.
- [316] L. Sundin, N. Sakr, J. Leinonen, S. Aly, and Q. Cutts, "Visual recipes for slicing and dicing data: teaching data wrangling using subgoal graphics," in *21st Koli Calling International Conference on Computing Education Research*, pp. 1–10, 2021.

- [317] L. Sundin, N. Sakr, J. Leinonen, and Q. Cutts, “Facilitating api lookup for novices learning data wrangling using thumbnail graphics,” *Foundations of Data Science*, 2021.
- [318] D. Lakens, “Sample size justification,” 2021.
- [319] J. C. Nesbit, O. O. Adesope, Q. Liu, and W. Ma, “How effective are intelligent tutoring systems in computer science education?,” in *2014 IEEE 14th International Conference on Advanced Learning Technologies*, pp. 99–103, IEEE, 2014.
- [320] A. M. Scheel, L. Tiokhin, P. M. Isager, and D. Lakens, “Why hypothesis testers should spend less time testing hypotheses,” *Perspectives on Psychological Science*, p. 1745691620966795, 2020.
- [321] M. J. Blanca, J. Arnau, D. López-Montiel, R. Bono, and R. Bendayan, “Skewness and kurtosis in real data samples,” *Methodology*, 2013.
- [322] Q. Nguyen, “Rethinking time-on-task estimation with outlier detection accounting for individual, time, and task differences,” in *Proceedings of the Tenth International Conference on Learning Analytics & Knowledge*, pp. 376–381, 2020.
- [323] J. Pek, O. Wong, and A. Wong, “How to address non-normality: A taxonomy of approaches, reviewed, and illustrated,” *Frontiers in psychology*, vol. 9, p. 2104, 2018.
- [324] U. Knief and W. Forstmeier, “Violating the normality assumption may be the lesser of two evils,” *Behavior Research Methods*, pp. 1–15, 2021.
- [325] B. Wheeler and M. Torchiano, “Imperm: Permutation tests for linear models. r package version 1.1-2,” *Vienna, Austria*, 2010.
- [326] W. P. Van den Brink and S. G. van den Brink, “A comparison of the power of the t test, Wilcoxon’s test, and the approximate permutation test for the two-sample location problem,” *British Journal of Mathematical and Statistical Psychology*, vol. 42, no. 2, pp. 183–189, 1989.
- [327] M. Weber and S. Sawilowsky, “Comparative Power Of The Independent t, Permutation t, and Wilcoxon Tests,” *Journal of Modern Applied Statistical Methods*, vol. 8, no. 1, p. 3, 2009.
- [328] D. Lakens, F. G. Adolphi, C. J. Albers, F. Anvari, M. A. Apps, S. E. Argamon, T. Baguley, R. B. Becker, S. D. Benning, D. E. Bradford, *et al.*, “Justify your alpha,” *Nature Human Behaviour*, vol. 2, no. 3, pp. 168–171, 2018.
- [329] M. Rubin, “Do p values lose their meaning in exploratory analyses? it depends how you define the familywise error rate,” *Review of General Psychology*, vol. 21, no. 3, pp. 269–275, 2017.

- [330] T. D. Pigott, J. C. Valentine, J. R. Polanin, R. T. Williams, and D. D. Canada, “Outcome-reporting bias in education research,” *Educational Researcher*, vol. 42, no. 8, pp. 424–432, 2013.
- [331] L. Sundin, S. Aly, N. Sakr, J. Leinonen, and Q. Cutts, “Slice N Dice.” <https://osf.io/6q8ut>, Mar 2021.
- [332] F. Goldhammer, J. Naumann, A. Stelter, K. Tóth, H. Rölke, and E. Klieme, “The time on task effect in reading and problem solving is moderated by task difficulty and skill: Insights from a computer-based large-scale assessment.,” *Journal of Educational Psychology*, vol. 106, no. 3, p. 608, 2014.
- [333] A. J. Jaeger and J. Wiley, “Do illustrations help or harm metacomprehension accuracy?,” *Learning and Instruction*, vol. 34, pp. 58–73, 2014.
- [334] J. Wiley, A. J. Jaeger, A. R. Taylor, and T. D. Griffin, “When analogies harm: The effects of analogies on metacomprehension,” *Learning and Instruction*, vol. 55, pp. 113–123, 2018.
- [335] M. J. Serra and J. Dunlosky, “Metacomprehension judgements reflect the belief that diagrams improve learning from text,” *Memory*, vol. 18, no. 7, pp. 698–711, 2010.
- [336] M. C. Chen, J. R. Anderson, and M. H. Sohn, “What can a mouse cursor tell us more? correlation of eye/mouse movements on web browsing,” in *CHI’01 extended abstracts on Human factors in computing systems*, pp. 281–282, 2001.