Visualisation and Software to Communicate Data Preprocessing Decisions

Lydia R. Lucchesi

May 2024

A thesis submitted for the degree of
Doctor of Philosophy
of The Australian National University.

Except where otherwise indicated, this thesis is my own original work.

Lydia R. Lucchesi
20 May 2024

Acknowledgements

I would like to acknowledge and thank the following people and organisations.

First and foremost, Professor Lexing Xie, for her dedicated supervision, guidance, and support, which made this thesis possible. I feel extremely grateful to have had the opportunity to learn from Lexing.

Dr. Petra Kuhnert, for her supervision and support throughout the PhD, for the initial invite to visit Australia and CSIRO, and for the many fun and informative conversations about R.

Professor Jenny L. Davis, for bringing a unique perspective to the supervisory panel, for her support, and for patiently teaching me about concepts/methods from the social sciences.

The Australian National University (ANU), CSIRO's Data61, ANU's Humanising Machine Intelligence research initiative, and Dr. Justine Lacey (Director of CSIRO's Responsible Innovation Future Science Platform), for funding support that helped make this work possible.

Bill Venables, for recommending that `smallsets` be extended to R Markdown and providing feedback on the software implementation of it. Professor Kate Henne, for suggesting the use of focus groups. Dr. Cecile Paris, for providing comments on a manuscript draft.

The focus group participants, for sharing their time, experiences, opinions, and thoughts, to help the project progress. Future improvements to `smallsets` will be in large part thanks to their input and feedback.

Fellow students of the Computational Media Lab (including Alasdair Tran, Alexander Soen, Josh Nguyen, Minjeong Shin, and Ziyu Chen) and of the Humanising Machine Intelligence research initiative (including Glen Berman, Jake Stone, Lachlan McGinness, Rachel Aalders, and Xueyin Zha), for the many great (interdisciplinary) research discussions and for providing feedback on this work, in meetings and seminars.

Benjamin Altmann and Uwe Ligges from the R CRAN team, for reviewing `smallsets` and providing recommendations on how to make the software better. Anonymous reviewers for the ACM FAccT 2022 conference, for thoughtful comments on a manuscript, which also influenced subsequent work. Three anonymous thesis examiners, for their thoughtful feedback and meticulous checks of figures/text, which improved this document.

The Midwest Uncertainty Collective at Northwestern University—and in particular, Priyanka Nanayakkara, Abhraneel Sarma, and Dr. Matthew Kay—for the opportunity to spend a day working at the lab and present on `smallsets`; discussions there contributed to transformative revisions in `smallsets` version 1.0.0. Dr. Chenhao Tan, director of the Chicago Human+AI lab, for the opportunity to present on this work at the University of Chicago.

Professor Chris Wikle and Dr. Spencer James, for the mentorship and research opportunities that opened doors for me and helped prepare me to do doctoral research.

My parents Dave and Roxanne and my sister Emma, for their continued support and for making the long trip from the United States to Australia to visit me. Polly Logmans & Peter Caley and Ellen Picard & John Bromilow, for their kindness and support, for making me feel at home in a new country, and for teaching me about Australian wildlife. My friends in Australia, for the many great memories. My friends in the United States and abroad, for staying connected over phone calls and coordinating across big time differences to do so.

Abstract

This thesis is concerned with the communication of data preprocessing. Data preprocessing is a crucial intermediate stage in quantitative data analysis. During this stage, data practitioners decide how to resolve dataset issues and transform, clean, and format the dataset(s). It can be a challenging stage, full of decisions that have the potential to influence analytical outcomes. Yet, data preprocessing is often treated as behind-the-scenes work and overlooked in research dissemination. This discrepancy, in the practice and presentation of data analytics, is limiting when it comes to replicating, interpreting, and utilising research outputs.

This work makes several contributions to advance the communication of data preprocessing decisions. The first contribution is a new operational view of data preprocessing. It demarcates data preprocessing within the broader data pipeline and avoids the need to list out the wide variety of tasks that data preprocessing can encompass. The two most central contributions include Smallset Timelines and `smallsets`. The Smallset Timeline is a static and compact visualisation, documenting the sequence of decisions in a preprocessing pipeline; it is composed of small data snapshots of different preprocessing steps. The `smallsets` software builds a Smallset Timeline from a user's data preprocessing script, containing structured comments with snapshot instructions. Together, Smallset Timelines and `smallsets` are designed to support the production of accessible data preprocessing documentation, for research dissemination. The final two contributions are four case studies and a focus group study. The former demonstrates use of `smallsets`, in a range of research problems, which rely on diverse data sources (e.g., citizen science data and home loan data). The latter is a formal evaluation of `smallsets`, which gathered feedback from prospective users on the software's utility/usability and data on experiences with preprocessing communication, more broadly.

Published research outputs

The work in this thesis has resulted in two published research outputs, thus far.

Lydia R. Lucchesi, Petra M. Kuhnert, Jenny L. Davis, and Lexing Xie. 2022. Smallset Timelines: A Visual Representation of Data Preprocessing Decisions. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 1136–1153. `https://doi.org/10.1145/3531146.3533175`

Lydia R. Lucchesi (2023). smallsets: Visual Documentation for Data Preprocessing. R package version 2.0.0. `https://CRAN.R-project.org/package=smallsets`

Contents

List of Figures

List of Tables

List of Code

Chapter 1

Introduction

Data preprocessing is a crucial intermediate stage in quantitative data analysis but is regularly overlooked in the documentation and dissemination of research. This thesis is concerned with the communication of data preprocessing. Specifically, it presents a novel visualisation and software tool for communicating data preprocessing decisions. This chapter begins by considering the importance, difficulties, and interests in communicating data preprocessing (Section 1.1). Then, the principal research question is discussed (Section 1.2). That discussion is followed by an overview of the thesis (Section 1.3), which includes a summary of its original contributions (Section 1.3.1).

1.1 Motivation

This section first explores the question: What is significant about data preprocessing, that makes communicating it important? Then, difficulties in communicating data preprocessing are discussed. Finally, this section discusses the growing interest in data provenance.

The significance of data preprocessing

In general, data preprocessing does not have a reputation as being the exciting or interesting part of data analytics. In fact, its reputation tends to be just the opposite [Sambasivan et al., 2021]. Yet, data preprocessing requires data practitioners to make decisions about

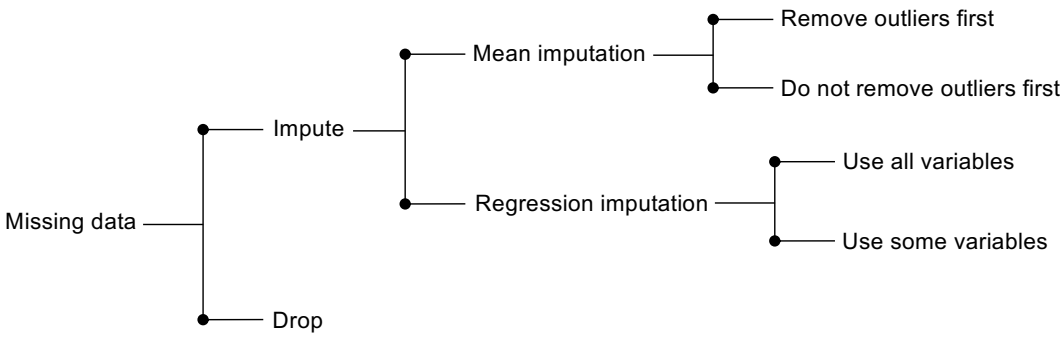


Figure 1.1: Data preprocessing tasks often involve one or more challenging (and sometimes consequential) decisions. The decision tree above shows several example choices that could be involved in a case of missing data. See Section 1.1 for a complete discussion.

how to address dataset issues, and the decisions are often not straightforward. Decisions in preprocessing can involve making assumptions and compromises, using domain and statistical knowledge, and being resourceful. Consider the simple scenario shown in Figure 1.1, in which a data practitioner must decide whether to impute or drop missing data.[1] Although the choice to drop looks the easiest, it may require a *strong* assumption about the underlying reason for missing data.[2] Alternatively, an initial choice to impute then requires *more* decisions about *how* to impute. And note that dealing with missing data might be just one of several preprocessing tasks, each requiring one or more decisions to be made.

Decisions, like those shown in Figure 1.1, matter because they can affect the outcome of a data analysis. For example, Little et al. [2012] emphasise the importance of minimising the occurrence of missing data in clinical trials (through good trial design), as handling missing data well—and not compromising downstream inferences—is difficult. Works in psychology [Steegen et al., 2016], political science [Denny and Spirling, 2018], ecology [Johnston et al., 2021], computing [Ding et al., 2021, Friedler et al., 2019, Ghotra et al., 2015], and neuroscience [Robbins et al., 2020, Shirk et al., 2017] have quantified the impact of data preprocessing decisions on analytical outcomes. Many of these works show that data preprocessing decisions can affect study outcomes. Some show that entirely different conclusions can be drawn, given different data preprocessing decisions (e.g., Denny and Spirling [2018]

[1]This example scenario does not present an exhaustive set of options for dealing with missing data but rather puts forward several plausible options that draw attention to the decision-making aspect of the task.

[2]Namely, it may require the assumption that the data are missing completely at random (MCAR), so dropping them will not change the underlying data distribution related to the target variable.

and Steegen et al. [2016]).

Another significant aspect of data preprocessing to consider is its evolving role in modern data analytics. Today, researchers can access many pre-existing datasets online or scrape the web itself, to facilitate an analysis. Yet, to use this found data, researchers must often mold (i.e., preprocess) it into a workable form, for the estimation or modelling task at hand. In other words, data preprocessing is increasingly being embraced as a means of harnessing readily available data, not immediately fit for purpose. For example, to conduct a demographic analysis of social media behaviour, researchers have inferred gender from first names provided in online profile data from platforms like Facebook [Tang et al., 2011] and Twitter [Mislove et al., 2021]. That preprocessing task typically involves finding *another* dataset, with both name and gender information, to predict gender in the dataset of interest.

Difficulties in communicating data preprocessing

Despite the challenge, impact, and pertinence of data preprocessing (as established above), it is often treated as behind-the-scenes work and overlooked in research dissemination [Leahey, 2008, Meng, 2021, Miceli et al., 2021, Passi and Jackson, 2017, Sambasivan et al., 2021]. This discrepancy, in the practice and presentation of data analytics, is limiting when it comes to replicating, interpreting, and utilising research outputs. It also sends misleading signals about what data analytics really is and what is involved in producing data outputs [Passi and Jackson, 2017]. There is not necessarily one specific reason for the lack of communication, per se. However, the lack of expectation and/or obligation *to* communicate data preprocessing seems to be a key contributing factor, from which different reasons stem.

For example, in situations in which disclosing dataset issues and preprocessing fixes is not mandatory, researchers may wonder if doing so might create more undesirable pushback, in both the peer review and public reception of work, than it is worth [Meng, 2021]. This may especially be the case for early-career researchers, who may face greater scrutiny over these types of decisions [Leahey, 2004]. Additionally, dataset documentation work that is not mandatory may continually get relegated to spare time, if there is any [Miceli et al., 2021]. On top of this, producing comprehensible descriptions of dataset decisions is a non-trivial task, that demands more of data practitioners than, say, making preprocessing code open source [Boulton et al., 2012]. Put simply, there are a variety of reasons that data preprocessing documentation can fall through the cracks.

Growing interests in data provenance

There has, however, been a notable uptick in work and initiatives related to data provenance, which refers to where a dataset comes from and how it was produced (and which data preprocessing documentation falls under). The uptick largely pertains to interests in transparency, accountability, and reproducibility in data analysis. For example, new documentation templates have been proposed for machine learning research—including datasheets [Gebru et al., 2021], Dataset Nutrition Labels [Holland et al., 2018], and model cards [Mitchell et al., 2019]—to support informed use of data and models. Some economic journals are now requiring replication packages with publications [Vilhuber, 2021]. There have been collaborative efforts to establish new data management standards, such as the FAIR Guiding Principles [Wilkinson et al., 2016] and *intelligent openness* [Boulton et al., 2012]. And one of the premier academic conferences on machine learning and artificial intelligence (NeurIPS) started a *Datasets and Benchmarks Track*, to incentivise work on datasets [Vanschoren and Yeung, 2021]. This thesis contributes to the growing body of work on data provenance, focusing exclusively on the communication of data preprocessing decisions, where gaps remain.

1.2 Principal research question

Attention to data provenance is increasing. However, practical tools designed to assist specifically with the communication of data preprocessing decisions remain sparse. This is a notable gap, as creating accessible data preprocessing documentation is important yet difficult (see Section 1.1). In turn, this thesis focuses on the following research question: **What tool can be designed, that results in accessible data preprocessing documentation for data consumers but minimises undue hassle for data producers?** The objective is to promote the effective communication of data preprocessing decisions, by making the communication task simple, easy, and worthwhile for data producers.

1.3 Thesis overview

Figure 1.2 presents a high-level outline of the thesis. It shows the topics of interest, the original contributions, and the connections between them. Namely, it shows that there is one contribution related to the broad topic of data preprocessing and four contributions related

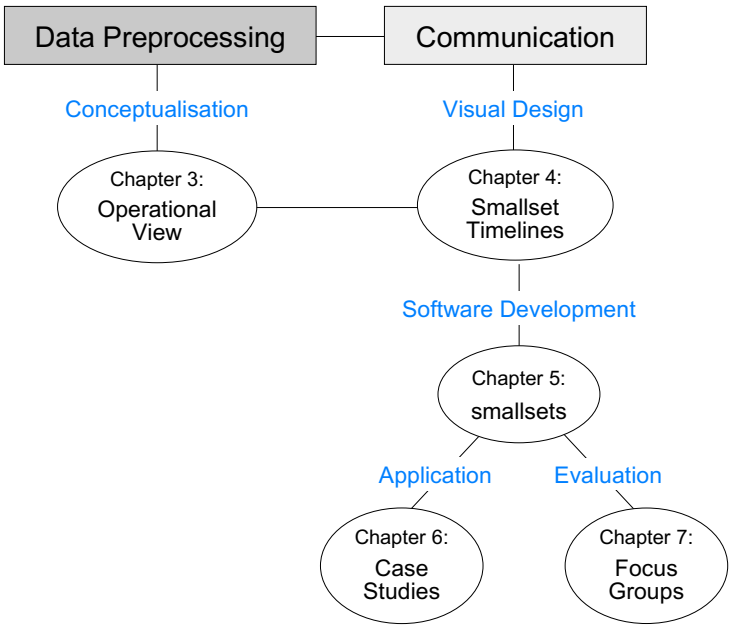


Figure 1.2: A graph outline of the thesis. The dark gray box is the broad topic: data preprocessing. The light gray box is the specific topic: communication of data preprocessing. Ovals are thesis contributions (with listed chapters). Edges show connections between topics and contributions. Blue labels indicate the predominant type of work, involved in each contribution. A full overview of the thesis can be found in Section 1.3.

to the *communication* of data preprocessing. As indicated by the edges in Figure 1.2, all contributions are connected, either directly or indirectly. Following the order of Figure 1.2 and starting with Chapter 3, this section provides an overview of the thesis. Note that Chapter 2 is about related work on data preprocessing and data provenance, including the documentation and visualisation of dataset changes.

The main focus of the thesis is data preprocessing communication. However, Chapter 3 takes a step back, to first consider data preprocessing itself. What is *data preprocessing*? It is an important question to answer early, to ensure the focus and scope of subsequent contributions are clear. Despite the ubiquity of data preprocessing in data analytics, there is not necessarily one widely-accepted definition of the term. Moreover, different terms are regularly used to refer to the same type of work involved in data preprocessing. Consider *data processing* or *data wrangling*, for example. Does this thesis pertain to those practices, too? Chapter 3 answers both questions, with a new operational view of data preprocessing.

It is a generalised view (used thereafter in the thesis) that defines data preprocessing not by the different types of tasks it may entail but by its procedural boundaries.

Next, Chapters 4 and 5 present the Smallset Timeline visualisation and `smallsets` software, respectively. These are the two most central contributions of the thesis and together aim to address the principal research question, presented in Section 1.2. Specifically, the Smallset Timeline is designed to result in accessible data preprocessing documentation for data consumers, while `smallsets` is designed to minimise undue hassle for data producers. The Smallset Timeline is a static, compact visualisation—composed of small data snapshots of different preprocessing steps—that documents the sequence of decisions in a preprocessing pipeline. The `smallsets` R software package builds a Smallset Timeline from the user's R *or* Python preprocessing code; users simply add structured comments with snapshot instructions to the code.

After an in-depth description of `smallsets` in Chapter 5, `smallsets` is put to use in four data preprocessing case studies in Chapter 6. The first case study is about software defect detection research. The second is about using citizen science data in ecological modelling. The third is about measuring algorithmic fairness. And the fourth is about home loan lending audits. The case studies are diverse in terms of research domain, but all highlight the critical role that data preprocessing plays, for a particular research problem. Two case studies also involve analyses that quantify the downstream effects of data preprocessing decisions on analytical outcomes. In total, this chapter presents six example Smallset Timelines (built with `smallsets`) visualising real-world data preprocessing scenarios.

Chapter 7 presents a focus group study, an initial step in formally evaluating `smallsets`. The focus group questions were designed to initiate discussion on three aspects of `smallsets`: 1) its deployment context, 2) its utility/usability, and 3) its output (the Smallset Timeline). There were four focus groups, with a total of 13 data practitioners with preprocessing experience. The findings highlight strengths and weaknesses of `smallsets` and where to direct future software development efforts, to best meet the needs of data practitioners. The study also generated qualitative data on real-world experiences with preprocessing communication, from both the perspective of data producer and consumer.

Chapter 8 presents conclusions of this thesis and discusses future work.

1.3.1 Key contributions

The original contributions of the thesis are summarised below. As illustrated in Figure 1.2, the contributions involve different research types and approaches, including conceptual work, visual design, software development, applied research, and qualitative research.

- A new operational view of data preprocessing, diagramming the location and boundaries of data preprocessing within the broader data pipeline (Chapter 3).
- A new static and compact visualisation, the Smallset Timeline, that documents the sequence of decisions in data preprocessing (Chapter 4).
- `smallsets`, open-source software for building Smallset Timelines for tabular datasets preprocessed in R, R Markdown, Python, or Jupyter Notebooks (Chapter 5).
- Four diverse data preprocessing case studies that demonstrate use of `smallsets` and illustrate downstream effects from data preprocessing decisions (Chapter 6).
- A focus group study with data practitioners that gathered feedback on `smallsets` as well as data on preprocessing communication, more broadly (Chapter 7).

Chapter 2

Related Work

This chapter provides an overview of related work in data preprocessing and data provenance. First, Section 2.1 highlights different views of data preprocessing found in the literature. Next, Section 2.2 details work quantifying the downstream effects of data preprocessing decisions on analytical outcomes. Then, Section 2.3 presents related work on data provenance—specifically, the documentation and visualisation of dataset changes. Each section ends with a discussion on how the contributions of this thesis build on and differ from related work.

2.1 Varying views of preprocessing

This section discusses the various ways that data preprocessing is described and defined in the literature. The discussion is limited to literature containing the word *preprocessing*. It starts with literature that describes preprocessing for different sources and types of data. From there, it shifts to broader and more conceptual views of preprocessing.

Domain-specific views of data preprocessing

For data practitioners, the word *preprocessing* may bring to mind a specific set of tasks, related to the type of data they regularly work with. For example, for ecologists working with Landsat data (satellite imagery), the word *preprocessing* might bring to mind geometric, solar, atmospheric, and topographic corrections [Young et al., 2017]. Meanwhile, for political

scientists working with political texts as data, it might bring to mind choices about stemming, stopword removal, and n-gram inclusion [Denny and Spirling, 2018]. Table 2.1 presents seven sets of preprocessing tasks, extracted from articles focused either entirely or mostly on data preprocessing. Individually, these articles detail what preprocessing means for a particular source or type of data. Collectively, they highlight the wide variety of preprocessing tasks that exist across different domains.

Broad conceptualisations of data preprocessing

Some works zoom out to discuss data preprocessing more generally. This includes in the context of data mining [García et al., 2015], data analysis [Famili et al., 1997], and statistics curricula [Zhu et al., 2013]. Generalised views of preprocessing are not linked to a certain domain or dataset. Yet, like the views in Table 2.1, they tend to focus heavily on preprocessing tasks (but at a high-level). For example, in the context of data mining, García et al. [2015] define data preprocessing in terms of high-level tasks, split into two categories. The first category is *data preparation* and consists of *data cleaning*, *data transformation*, *data integration*, *data normalization*, *missing data imputation*, and *noise identification*. The second category is *data reduction* and consists of *feature selection*, *instance selection*, *discretization*, and *feature extraction and/or instance generation*.

Discussion

From Table 2.1, we can see that the term *data preprocessing* does not necessarily put data practitioners on the same page, if their understanding of the term is tied to domain-specific tasks. Therefore, the meaning of data preprocessing used for this thesis is clarified in Chapter 3. Specifically, a new operational view of data preprocessing is presented. It builds on generalised views of data preprocessing found in the literature. However, instead of categorising the types of tasks that preprocessing entails, the new view demarcates preprocessing from other stages in the data pipeline. Then, for each data analysis, data-related tasks can be classified as preprocessing, based on their location and purpose within the broader data pipeline, rather than on what they are called. This helps to make the generalised view robust to the wide variety of preprocessing tasks that exist across domains (see Table 2.1).

Data source: content	**Example data preprocessing tasks**	**Example estimation/modelling task**	**Source(s)**
Landsat data: satellite imagery	Georeferencing; co-registration; conversion to radiance; solar, atmospheric, topographic, and relative radiometric corrections	Spectral indices to highlight different land features	[Young et al., 2017]
Congressional bills: unstructured text	Remove punctuation, numbers, stopwords, and infrequent terms; stemming; lowercasing; n-gram inclusion	Latent Dirichlet Allocation (LDA) to identify topics	[Denny and Spirling, 2018]
EEG data: electrical brain activity (brain waves)	Remove artefacts, e.g., eye blinks and faulty sensors	Event-related potential (ERP) effects for a memory task	[Shirk et al., 2017]
Medical Information Mart for Intensive Care (MIMIC-III) dataset: electronic health records	Unit conversions; aggregate timestamps and medical codes; derive a medical intervention feature	Logistic regression model to predict mortality	[Wang et al., 2020]
NASA Metrics Data Program (MDP) datasets: software module specifications	Remove constant and repeated attributes; replace missing values; run integrity checks; remove duplicate and inconsistent cases	Random forest classifier to predict defectiveness	[Gray et al., 2011, 2012, Petrić et al., 2016, Shepperd et al., 2013]
eBird data: bird observations by citizen scientists	Spatial subsampling; create effort covariates; apply effort filters	Species distribution model to map estimated occurrence	[Johnston et al., 2021, Strimas-Mackey et al., 2023]
Home Mortgage Disclosure Act (HMDA) data: home loan applications and lending decisions	Address non-pricing issues (e.g., geographic issues), missing race or ethnicity data, missing income data, and pricing issues (e.g., yield curve changes)	Linear probability model to assess lending fairness	[Avery et al., 2007]

Table 2.1: A series of example data sources, related preprocessing tasks, and estimation/-modelling scenarios, discussed in works with a particular focus on data preprocessing. See Section 2.1 for a discussion on varying views of preprocessing and Chapter 6 for more information on examples 5-7, which form case studies.

2.2 Downstream preprocessing effects

As outlined above, preprocessing involves different data-related tasks. However, preprocessing is not simply executing these tasks. Rather, part of preprocessing is deciding which tasks to implement and how. Research about preprocessing effects investigates if study outcomes are sensitive to these decisions. This section highlights three areas of related work on preprocessing effects: 1) generalised methods for measuring effects, 2) graphical support for assessing effects, and 3) domain-specific studies of effects.

Generalised methods for measuring effects

There have been different methods proposed for measuring preprocessing effects in data analytics. Blocker and Meng [2013] draw on the statistical concept of *multiphase inference* to develop a theoretical framework for evaluating data preprocessing. Steegen et al. [2016] propose the *multiverse analysis*. This involves running the same hypothesis test on all plausible preparations of a dataset, to check how sensitive the analytical outcome is to preprocessing decisions. Denny and Spirling [2018] propose the *preText* method, for unsupervised learning in document analysis. This method looks at how the pairwise distances between documents change, with the application of different preprocessing operations. Software support for the *multiverse analysis* and *preText* are available in the `multiverse` [Sarma et al., 2023] and `preText` [Denny and Spirling, 2021] R packages, respectively.

Graphical support for assessing effects

There is also graphical support for assessing preprocessing effects in machine learning (ML). This includes the What-If Tool (WIT) [Wexler et al., 2020] and *fair-DAGs* [Yang et al., 2020]. WIT provides a graphical user interface that enables ML practitioners to interactively explore a model and its input data. Users can edit input data directly and re-run a model, to see if the model changes. The *fair-DAGs* tool is an open-source Python library for investigating the impact of preprocessing on ML pipelines, and in particular, algorithmic fairness. It builds a directed acyclic graph (DAG) visualising how data moves through the preprocessing steps. It also generates a report, detailing changes to sensitive variables (e.g., race and gender), with respect to the target variable, at the DAG vertices. The aim is to identify where biases against particular groups might be introduced, through operations like filtering or binning.

Domain-specific studies of effects

There are also domain-specific studies of preprocessing effects. These studies test if an analytical outcome changes, given the same (un-preprocessed) dataset and estimation/modelling strategy but different data preprocessing decisions. These studies have been conducted in a range of fields, such as neuroscience [Robbins et al., 2020, Shirk et al., 2017], machine learning [Ding et al., 2021, Friedler et al., 2019, Ghotra et al., 2015], political science [Denny and Spirling, 2018], ecology [Johnston et al., 2021], and psychology [Steegen et al., 2016].

Many of these studies show, quantitatively, that preprocessing decisions influence analytical outcomes. For example, Friedler et al. [2019] compared the performance of ML fairness interventions,[1] applied to different data preparations. The authors observed "a fairness-accuracy tradeoff which arises not from hyperparameters, but from *choice of preprocessing*" (italics in original) [p. 332]. For one data preparation, the model accuracy tended to be higher, while group fairness was lower. For the other data preparation, the reverse was true.

Discussion

Broadly speaking, this diverse body of literature on preprocessing effects is concerned with the impact of data preprocessing decisions on analytical outcomes. The domain-specific studies, in particular, underscore the importance of documenting data preprocessing decisions. These types of studies motivate the work presented in Chapters 4 and 5, on Smallset Timelines and `smallsets`, respectively. Studies assessing preprocessing effects are discussed again in Chapter 6, which presents four case studies. Two of the case studies in Chapter 6 also feature new analyses quantifying preprocessing effects. One is similar to the work by Friedler et al. [2019] (discussed above), in that it focuses on differences in group fairness in ML prediction tasks, due to variations in preprocessing. The topic of preprocessing effects emerges again in Chapter 7, as a noteworthy aspect of the data from the focus group study.

2.3 Data provenance tools

This next section focuses on data provenance tools. Data provenance tools are developed to record, organise, display, and/or visualise information about where a dataset comes from

[1] Fairness interventions aim to ensure that algorithmic decision aids do not discriminate by sensitive attributes, such as race, gender, age, etc.

and how it was produced. In this section, various types of data provenance tools are discussed. This includes broad documentation frameworks, software tools to automate provenance keeping, interactive visualisation systems, and both dynamic and static visualisations of data transformations.

Documentation templates and frameworks for ML

There have been various documentation tools proposed for machine learning (ML) research, including datasheets [Gebru et al., 2021], model cards [Mitchell et al., 2019], the Dataset Nutrition Label [Holland et al., 2018], data statements [Bender and Friedman, 2018], FactSheets [Arnold et al., 2019], and the REFORMS checklist [Kapoor et al., 2024]. These tools encourage data producers to document and share dataset information, to support informed use of data and models.

Several modules in the Dataset Nutrition Label have graphical components to explore variable distributions (e.g., histograms) and correlations (e.g., heat maps). Otherwise, the tools listed above represent text-based approaches to documentation. For example, the datasheet [Gebru et al., 2021] contains 57 questions to answer about a dataset, with four dedicated to preprocessing.[2] This datasheet template has since been adapted for specific types of data, such as health and speech datasets [Papakyriakopoulos et al., 2023, Rostamzadeh et al., 2022].

Documentation software

In general, the expectation for many of the documentation tools discussed above is that data practitioners will manually fill out the templates/frameworks. There has also been related work, however, in automating the production of dataset documentation. For example, the Jupyter Lab extension DocML [Bhat et al., 2023] can export Markdown cells from a Jupyter Notebook directly to a model card [Mitchell et al., 2019], one of the ML frameworks mentioned previously. Bhat et al. [2023] found in a user study that DocML encouraged ML practitioners to fill out model cards more thoroughly and update them in real-time.

Another example is E2ETools [Lerner et al., 2023]. It is a suite of seven R packages for data provenance, with a particular focus on reproducible analyses. Users can incorporate

[2]For example, question 33 in a datasheet [Gebru et al., 2021] is as follows: "Was any preprocessing/cleaning/labeling of the data done (for example, discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)?" [p. 90-91].

the packages into an R data analysis, to collect and save three types of provenance information: *environment information* (e.g., R package versions), *course-grained information* (e.g., data inputs to and outputs from R scripts), and *fine-grained information* (e.g., a line-by-line log of object assignment). In E2ETools, provenance information can be viewed in a short report with `provSummarizeR` [Lerner, 2022] or visualised in a provenance graph with `provViz` [Lerner and Boose, 2014].

Interactive visualisation systems

There are also interactive visualisation systems, that both assist data practitioners with data work and have data provenance features. Wrangler [Kandel et al., 2011] was a system that assisted users with preprocessing, recommending relevant data transformations that could be executed with point-and-click commands. Users could also add annotations to transformations. These annotations were included as code comments in programming scripts exported from Wrangler. In VisTrails [Bavoil et al., 2005], users could generate, compare, and annotate data workflows and the production of scientific visualisations. Data Quality Provenance (DQProv) Explorer [Bors et al., 2019] is a visual analytics tool with three different displays for assessing data quality at different data states.

Animated visualisations of data transformations

Some works propose the use of animation to visualise data transformations, for data consumers. Khan et al. [2017] developed *data tweening*, which involves animating the transformations occurring between two database queries. It shows, step-by-step, how the structure of a table changes from one query to the next. Pu et al. [2021] developed *datamations*. A datamation animates individual data points moving through a data analysis pipeline. It is designed to show the data manipulation process required to produce a figure or table.

Static visualisations of data transformations

Other works propose static visualisations of data transformations. For example, the Data Comic [Zhao et al., 2015] uses traditional aspects of a comic strip—i.e., storyline, small panels, characters, speech bubbles, graphics, etc.—to tell an accessible and engaging story about data. Wang et al. [2021] use Data Comics to visualise the steps in a user study in Human-Computer Interaction (HCI). Specifically, they divide an HCI user study into ten stages and

dedicate one or more panels in the comic to each stage. One stage is *Data Transformations*, for explaining *assumption checks*, *transformations*, and *outlier removal*. In an example, they illustrate how a logarithmic transformation can be visually explained in a data transformation panel, by showing density and Q-Q plots before and after the transformation.

The `provViz` package in E2ETools [Lerner and Boose, 2014] and the *fair-DAGs* library [Yang et al., 2020]—mentioned previously in this chapter—also produce static visualisations of data transformations. Both use graphs of nodes and edges to visualise a programming script and how data moves through it. For the reporting of randomised controlled trials (RCTs), there is the CONSORT diagram [Begg et al., 1996, Moher et al., 2001, Schulz et al., 2010]. This is a flow diagram that visualises participant groups and numbers throughout the different stages of a clinical trial. The two most recent versions of the CONSORT diagram encourage researchers to document how many participants were excluded from the statistical analysis and why. The R software package `consort` [Dayim, 2023] is designed to help researchers build this diagram.

Discussion

Data provenance is broad, and many of the tools discussed above focus on multiple aspects of data provenance. In this thesis, the choice is made to focus exclusively on **data preprocessing decisions**. Specifically, Chapter 4 proposes a new visualisation, the Smallset Timeline, that documents the sequence of decisions involved in a data preprocessing pipeline. Similar to the animated visualisations mentioned above, it shows how a dataset changes, step-by-step. However, the Smallset Timeline is a *static* and *compact* depiction of steps, such that it can be easily shared in many different mediums, including print.

Chapter 5 proposes `smallsets`, an open-source software tool for building Smallset Timelines. Similar to *fair-DAGs* and `provViz`, `smallsets` captures the effects of code on data, but its output—the Smallset Timeline—visualises the process with small data snapshots and captions (instead of nodes and edges in a graph). The `smallsets` tool can be used to answer preprocessing prompts in broader data provenance templates like datasheets and model cards, which practitioners are typically expected to complete by hand and with text alone.

Chapter 3

Data Preprocessing Defined

As illustrated in Section 2.1, there are many ways that data preprocessing is presented in the literature. Thus, it is important to establish a mutual understanding of the term, to ensure the focus and scope of subsequent chapters are clear. In turn, this chapter presents an operational view of data preprocessing (Section 3.1), used in the remainder of the thesis. This chapter also touches on various synonyms of the term *data preprocessing* (e.g., *data cleaning* and *data processing*), justifying the selection of *data preprocessing* as opposed to one of these alternatives (Section 3.2).

3.1 A new operational view

This section presents a new operational view of data preprocessing. It builds on existing conceptualisations of data preprocessing, outlined in Section 2.1, but with several notable differences, which are discussed in Section 3.1.1. The operational view is diagrammed in Figure 3.1. It defines data preprocessing by drawing a boundary around it, within a (simplified) data pipeline. Therefore, everything within the boundary can be classified as data preprocessing. As shown in the diagram, a simple three-stage data pipeline is assumed. It consists of 1) data collection, 2) data preprocessing, and 3) estimation and modelling. Boundary definitions between adjacent stages establish the beginning and end of data preprocessing. In particular, the completion of data collection is defined, creating the left-hand boundary of

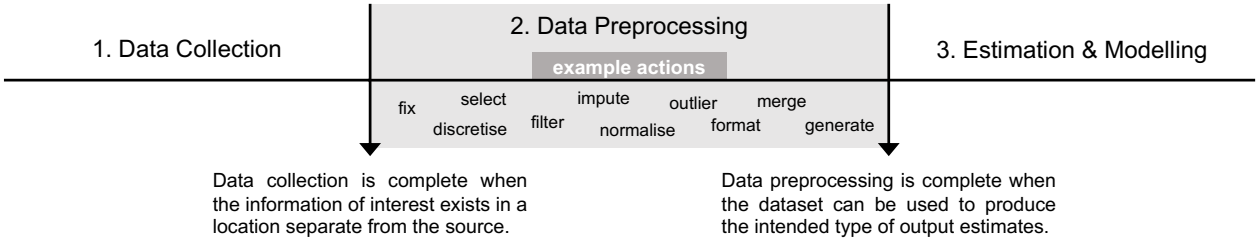


Figure 3.1: Operational view for data preprocessing, in which boundary definitions distinguish data preprocessing from neighbouring stages in the data pipeline. See Section 3.1 for further description of the operational view. Also note that this diagram uses a linear format, to focus on the boundaries of data preprocessing; however, in practice, iteration between stages may occur. A discussion on the limitations of this simplified representation of the data pipeline can be found in Section 3.1.2.

data preprocessing. What constitutes a completion of data preprocessing is also defined, creating the right-hand boundary. Thus, any actions altering the dataset(s) between the left and right boundaries can be classified as data preprocessing. Moreover, the decisions—occurring between the left and right boundaries—about *what* actions to implement (and *why*/*how* to implement them) can be classified as **data preprocessing decisions**.

3.1.1 Key design choices

There were three key choices that shaped development of the operational view proposed above (see Figure 3.1): 1) the view should not centre on preprocessing tasks, 2) it should elaborate on the data pipeline, and 3) it should avoid use of the words *raw* and *analysis* in certain contexts. These choices are discussed below.

Not **centring on preprocessing tasks**

As demonstrated in Section 2.1, it is common to present data preprocessing, in terms of the tasks it entails. However, the first key design choice was to *not* centre the view around a list or taxonomy of preprocessing tasks, given the immense variety of them (see Table 2.1) and the difficulty of producing something comprehensive. Moreover, existing lists/taxonomies are already inconsistent. For example, García et al. [2015] treat *data preparation* as a category of techniques that includes both *data transformation* and *noise identification*. Meanwhile, Famili et al. [1997] treat "preparation for data analysis" [p. 3] as one of two key reasons for

preprocessing and *data transformation* as one of three broad categories of techniques, under which *noise modelling* is categorised.

Instead, the boundary approach in Section 3.1 eliminates the need to list out or taxonimise all possible tasks (though some are provided in Figure 3.1 as examples). It also makes the view robust to variability in the location and purpose of tasks, within a data pipeline, as tasks seen in preprocessing are not necessarily exclusive to preprocessing. For example, *inference* and *prediction* can be used to generate new features in a dataset *or* produce the final output of an analysis. However, using the operational view in Figure 3.1, we know that if the inference or prediction task changes the dataset to facilitate the final estimation or modelling task, it is part of data preprocessing. Consider the example mentioned in Section 1.1, in which gender is inferred from online profile names, to facilitate the study of social media behaviour by demographic characteristics [Mislove et al., 2021, Tang et al., 2011]. Here, despite the name of the task, *gender inference* would be classified as data preprocessing, given that its purpose is to make the online profile data usable.

Elaborating on the data pipeline

Defining data preprocessing by its position in the data pipeline is also common. Following are two examples of this: 1) "[d]ata preprocessing consists of all the actions taken before the actual data analysis process starts" [Famili et al., 1997, p. 8] and 2) "data preprocessing bridges the gap from data acquisition to statistical analysis" [Zhu et al., 2013, p. 235]. The view in Figure 3.1 uses this approach but with added boundary definitions—to demarcate data preprocessing—and a diagram to visualise them. The boundary definitions provide additional information about pipeline stages. Notably, the second boundary definition imparts intention in data preprocessing work, conveying that data alterations are expressly motivated by the subsequent stage of estimation and modelling. The diagram helps to underscore the critical role of preprocessing in data analytics; preprocessing is placed alongside other important data stages and allocated substantial space within the diagram.

Avoiding the words *raw* and *analysis*

The operational view avoids use of the word *raw* in reference to data (i.e., "raw data"). *Raw* is a popular data descriptor, used to indicate that data have not been altered in any way since collection. In other words, it is a common way to allude to data that have not been

preprocessed. However, heeding warnings from the field of science studies,[1] the descriptor is avoided in this thesis. *Raw* implies that data exist naturally and then are "cooked," when in fact we create data through decisions about what information to collect and how to collect it [Gitelman, 2013]. In other words, the "cooking" of data starts before data preprocessing, so *raw* can be misleading. The term *un-preprocessed* is used instead.

Use of the word *analysis*—as in "data analysis" or "statistical analysis"—is also avoided, when referring to a stage in the pipeline distinct from data preprocessing. In this thesis, the preference is to only use *data analysis* as a broad term that encompasses multiple stages, including data preprocessing. In turn, data preprocessing is presented as a crucial part of a data analysis instead of as a precursor to it, where it is more likely to be overlooked and its effects underestimated.

3.1.2 Limitations

There are also limitations to the operational view presented in Figure 3.1. The diagram has a linear format, implying analyses are a straight and forward-moving process, when in practice some iteration between stages may be necessary. For certain analyses the boundary points may still seem blurry, despite the boundary definitions. Consider particle physics, where data may be preprocessed as they are collected to manage data volume [Radovic et al., 2018], or the R command `rpart()` for building regression trees [Therneau and Atkinson, 2023], which defaults to removing observations with missing values for the dependent variable.

Additionally, the title of the third stage—*Estimation & Modelling*—could also include terms like *Prediction*, *Forecasting*, or *Inference* to be more comprehensive. However, for brevity, two terms—that were not *analysis* (see Section 3.1.1) and that seemed general and comprehensive enough—were selected. Lastly, the diagram communicates what part of the data pipeline this thesis is focused on, but it does not provide significant insight into the nature of preprocessing work. Specifically, the view itself does not capture the ways in which data preprocessing often requires data practitioners to make challenging decisions, do data detective work, use creativity to resolve data problems, or employ domain and statistical knowledge.

[1] The study *of* science and how it is produced and used in society.

3.2 Close synonyms

In this chapter on defining *data preprocessing*, it is worth acknowledging the term's many close synonyms, some of which are used widely and instead of *data preprocessing*. If there was a thesaurus of data science terms, the entry for *data preprocessing* might include the following: *data blending*, *data cleaning*, *data construction*, *data crunching*, *data editing*, *data franchising*, *data manipulation*, *data munging*, *data preparation*, *data processing*, and *data wrangling*. Across these terms, there is overlap in the type of data work they entail. Moreover, they are sometimes used interchangeably, which is why they are presented as close synonyms. However, there are also important nuances in meaning.

For example, *data cleaning* and *data blending* are more specific than *data preprocessing* and could be thought of as subsets of *preprocessing*. However, *data processing* could be seen as adjacent to *preprocessing*, given the prefix *pre* in the latter. In general though, the term *data processing* is used liberally. It has been used to refer to the preparation of datasets for storage in archives and use in secondary analyses [Plantin, 2019]. There, it involves *data cleaning* for some future use. At the Australian Bureau of Statistics, *data processing* involves "Despatch and Collection Control," "Data Capture and Coding," and "Editing" [Australian Bureau of Statistics, 2023]. The Cambridge Dictionary[2] simply defines it as "the use of a computer to perform calculations on data."

Many of the related works cited throughout this thesis use these synonyms. For example, Leahey [2004] uses *data editing*. Kasica et al. [2021] use *data wrangling*. Steegen et al. [2016] use *data construction* and *data processing*. For this thesis, I elected to use the term *data preprocessing*. As indicated above, some synonyms were either too specific (e.g., *cleaning* or *blending*) or too broad (e.g., *processing* or *wrangling*), for the purpose of this work. *Preprocessing*, though, is not too specific, in terms of the types of data operations it may entail, but also not too broad, in terms of the parts of a data analysis it may refer to. It is also not too niche, appearing in academic literature from a range of fields (e.g., statistics [Blocker and Meng, 2013], political science [Denny and Spirling, 2018], medicine [Robbins et al., 2020], and ecology [Young et al., 2017], to name a few). These features made it a suitable choice.

For consistency, the term *data preprocessing* is used exclusively in this thesis. Moreover, in the following chapters, the Smallset Timeline visualisation and `smallsets` software tool

[2] `https://dictionary.cambridge.org/dictionary/english/data-processing`

are proposed for communicating *data preprocessing* decisions. However, despite the choice of terminology, Smallset Timelines and `smallsets` are also applicable to these close synonyms, when used to refer to work falling within the boundaries of *data preprocessing* established in Section 3.1 (i.e., the shaded region in Figure 3.1). For example, a Smallset Timeline can be used to communicate *data cleaning* decisions that occur after data collection and before the estimation/modelling stage.

3.3 Summary

The objective of this chapter was to clarify the meaning of the term *data preprocessing* used in this work, as there is not necessarily one widely accepted definition of the term that makes its meaning explicitly clear. In turn, an operational view of data preprocessing was presented (Figure 3.1), which is used in all subsequent chapters. It assumes a three-stage data pipeline, in which data preprocessing is an intermediate stage occurring after data collection and before estimation and modelling. Boundary definitions between stages demarcate data preprocessing work. The following chapter proposes a new visualisation—the Smallset Timeline—focused specifically on the decisions made during data preprocessing.

Chapter 4

Smallset Timelines: A Visualisation of Data Preprocessing Decisions

The previous chapter clarified which part of the data pipeline is of interest in this thesis. In particular, the focus is on the data preprocessing stage, which occurs after data collection and before estimation and modelling (see Figure 3.1). This chapter presents a novel visualisation, the Smallset Timeline, designed to communicate what happens *during* this crucial stage, full of important decisions. To date, tools designed specifically for this purpose have been sparse, especially those utilising visualisation, a powerful mode of communication.

The Smallset Timeline is a static, compact visualisation composed of small data snapshots of different preprocessing steps. Figure 4.1 is an example Smallset Timeline, visualising preprocessing steps for the synthetic dataset *s_data*, used throughout this chapter for illustration.[1] The Smallset Timeline uses a small set of data—i.e., a *Smallset*—from the original dataset, to provide a step-by-step depiction of data preprocessing decisions. *Snapshots* of the Smallset—taken at different moments in the preprocessing stage—colour-encode examples of dataset changes due to preprocessing. Snapshot *captions*, supplied by the data producer, explain each step in greater detail.

The remainder of this chapter provides a detailed explanation of Smallset Timelines and is structured as follows. Section 4.1 introduces the intended users and design goals of the

[1]The s_data dataset has 100 rows and eight columns (C1-C8). More information on s_data can be found in Appendix A.

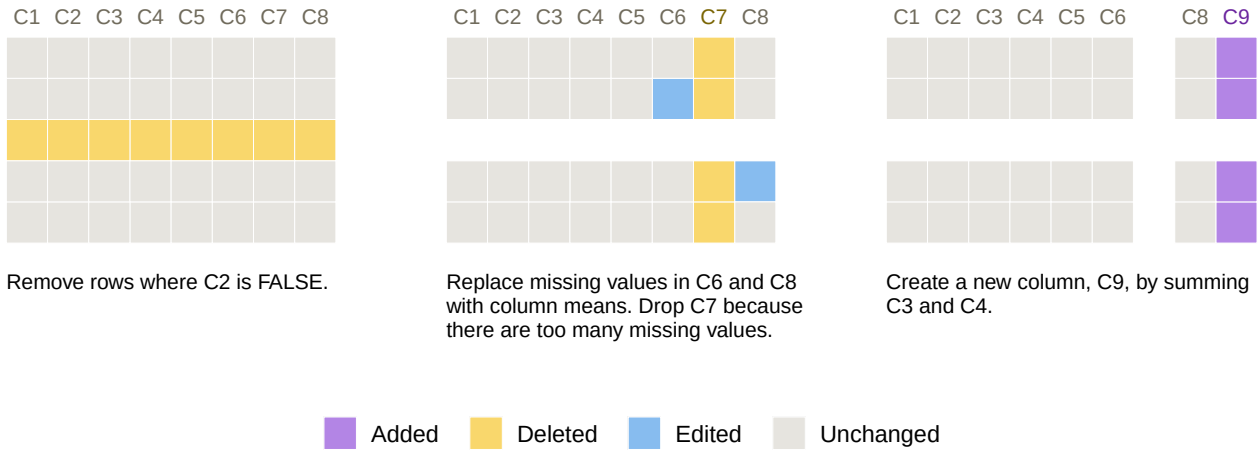


Figure 4.1: A Smallset Timeline for the s_data dataset and preprocessing scenario (see Appendix A). A Smallset of five rows was selected through random sampling.

Smallset Timeline, highlighting its utility for both data producers and consumers and its adaptability to different communication settings and information needs. Section 4.2 details the visual design. This includes the Smallset Timeline's three core visual components (the *Smallset*, *snapshots*, and *captions*) and four optional enrichment features, used to "enrich" the figure with additional dataset/preprocessing information. Section 4.3 presents automated methods for selecting Smallset rows, a crucial aspect in figure construction. Finally, Section 4.4 presents automated alternative (alt) text production, for making content in Smallset Timelines accessible to those with visual impairments.

4.1 Intended users and design goals

The Smallset Timeline was designed with two users and three design goals in mind. The users are *Timeline creator* and *Timeline reader*, and the three design goals relate to these users. Among Timeline creators, the Smallset Timeline is designed to encourage reflection on and reflexivity about data preprocessing decisions. Among Timeline readers, it is designed to support reproducibility and replicability as well as comprehension and evaluation of data preprocessing decisions. These design goals are summarised in Table 4.1 and discussed in more detail below.

The first design goal is to encourage Timeline creators (i.e., data practitioners) to reflect

	Reflect & Be Reflexive	**Reproduce & Replicate**	**Comprehend & Evaluate**
User: action	Data practitioner: create	Data practitioner: read	Target audience: read
Outcome	Asks practitioners to recount their decision-making process in the Smallset Timeline captions, encouraging reflection for the preprocessing stage and reflexivity about their influence on the work and outcomes.	Provides visual examples and written descriptions of preprocessing steps performed in a programming language. Documents information in a stable format that can be saved and shared.	Provides an accessible preprocessing narrative with enough information for decisions to be understood and assessed. Highlights the role that humans play in data production.
Presentation	Reflection and reflexivity occur while writing captions for the Smallset Timeline.	– Smallset with more rows – More snapshots – Detailed captions	– Smallset with fewer rows – Fewer snapshots – Succinct captions

Table 4.1: Design goals for Smallset Timelines, including users, utilities, and the corresponding format variations.

on and be reflexive about their data preprocessing decisions. To create a Smallset Timeline, Timeline creators must recount their decision-making process, in the snapshot captions (see Section 4.2.1). This ideally prompts reflection on their decisions and facilitates reflexivity. *Reflexivity* can refer to the research practice of examining how one's own subjectivities have impacted the formulation, trajectory, and outcome of an analysis [Mauthner and Doucet, 2003]. Traditionally, it has been viewed as an important practice for those working with qualitative data, where researcher subjectivity is acknowledged [Lumsden et al., 2019]. Today, there is a growing call to incorporate reflexivity into quantitative work as well [D'Ignazio and Klein, 2020, Elish and boyd, 2018, Miceli et al., 2021, Tanweer et al., 2021].

The second goal is to support reproducibility and replicability. If preprocessing information is not shared, it may be challenging to accurately reproduce an analysis. If preprocessing *code* is shared, the Smallset Timeline can serve as an access point to the code, which can be overwhelming to interpret on its own. Having the Smallset Timeline may be especially helpful when hitting runtime errors in open-source preprocessing code. As for replicability, missing preprocessing information may impede the replication of a result with different data, if the preprocessing decisions were consequential in the original analysis. In this situation, having *explanations* of each preprocessing decision is important, as it may be less about

executing the exact same code and more about replicating the decision rationale.

The third goal is to support comprehension and evaluation among Timeline readers. During data preprocessing, data practitioners typically encode preprocessing decisions in code. However, sharing the code alone will not necessarily make preprocessing information accessible to those unfamiliar with the programming language(s). Even for those with familiarity, the encoded decisions may still feel inaccessible, as parsing through another data practitioner's preprocessing script can be challenging. Thus, getting these decisions out of code—and into a practical and accessible format—is crucial for making them legible and open to evaluation. The Smallset Timeline is designed to do just that, bringing key preprocessing decisions into plain view with visuals and text.

At all stages in its life cycle, the Smallset Timeline is designed to serve a purpose, whether it is for Timeline creators or readers. The aspects of its construction—that would serve less of a meaningful purpose to do by hand—have been automated by the `smallsets` software (see Chapter 5). Note that the design goals discussed above are *intended* design goals. The findings from a focus group study, presented in Chapter 7, suggest that the utility of the Smallset Timeline likely extends beyond the goals outlined in Table 4.1. The use cases envisioned by focus group participants are discussed in Section 7.3.3.

4.2 Visual design

The Smallset Timeline was designed to be simple, practical, and accessible. The visualisation is static in nature and compact in size, making it easier to save, print, and share. It uses a familiar layout—the timeline—to present an ordered sequence of steps and familiar graphical elements—such as tables, colour, and text—to explain what is happening in each step. This section dives into the visual design of the Smallset Timeline, including in-depth discussions on its three core visual components (Section 4.2.1) and four optional enrichment features (Section 4.2.2).

4.2.1 Three core visual components

The Smallset Timeline has three core visual components. There is a *Smallset*, consisting of a small set of rows (5-15), from the original dataset. It contains examples of preprocessing changes. There are *snapshots* of the Smallset. These are taken at different moments in the

preprocessing pipeline and plotted sequentially to form a timeline of steps. Each snapshot visualises one or more preprocessing decisions and highlights dataset changes through colour. There are *captions*, for the snapshots, that provide written descriptions of the data preprocessing decisions. Figure 4.2 breaks down the example Smallset Timeline from Figure 4.1 into its three core visual components, to illustrate what a Smallset Timeline is comprised of. Each visual component is discussed in detail below.

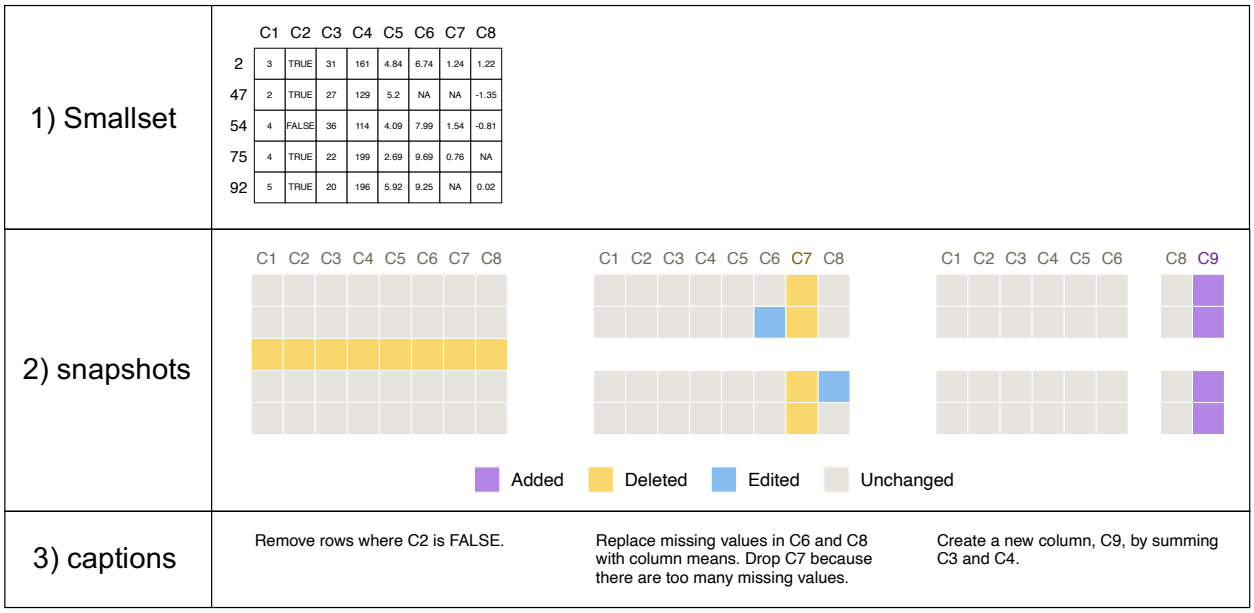


Figure 4.2: The Smallset Timeline in Figure 4.1 broken down into its three core visual components: the Smallset, snapshots, and captions. The Smallset consists of rows 2, 47, 54, 75, and 92 from s_data, printed in full in Table A.1. Three sequential snapshots of the Smallset highlight, with colour, example dataset changes, due to preprocessing. Three captions (one for each snapshot) describe the preprocessing steps in more detail. See Section 4.2.1 for a discussion on components.

The Smallset

A "Smallset" is a small set of observations (5-15) from the dataset of interest. The observations feature examples of dataset alterations due to preprocessing. The objective of a Smallset is not to create a representative sample of the dataset and how often data alter-

ations occur. Instead, the objective is to create a small dataset-like object that can be used to visually explain the preprocessing steps, at a manageable scale for comprehension and figure production. It is crucial to make this objective clear to Timeline readers, to avoid misinterpretation of the Smallset and what it can and cannot say about the full dataset.

Throughout this section, it is assumed that the Smallset rows are given. However, in practice the Smallset rows have to be selected from the dataset of interest. The Smallset selection criteria and automated selection methods are discussed in Section 4.3. For datasets with many columns, it may also be necessary to show only a subset of columns in the visualisation, to keep the Smallset Timeline compact in size. The current design of the Smallset Timeline is suited to tabular data only, meaning the Smallset has a table format. Each observation is a row. Each attribute is a column, and there is no nested data structure (e.g., lists or other key-value structures) in a cell.

Small tables like this have long been used in the programming community to explain coding commands for data manipulation. For example, the cheatsheet for the R `dplyr` package [Posit Software, PBC, 2023] uses little (empty) tables and colour to visually explain to data scientists what happens to the data object when a `dplyr` command is applied to it. There have also been online tools developed to teach programming commands through the use of interactive table visualisations supplemented with outlining, colour, and arrows [Lau et al., 2023]. With the Smallset, a similar technique is employed. However, now the focus is on the decisions *behind* the programming command. Specifically, the technique is used to demonstrate what happens to data as a result of data preprocessing decisions.

Snapshots

A snapshot is a picture of the Smallset at a particular moment in the data preprocessing pipeline. The first and last snapshots are taken at the beginning and end of the data preprocessing pipeline, respectively. In other words, every Smallset Timeline has at least two snapshots. Snapshots in-between represent intermediary points in the preprocessing pipeline. These intermediate snapshot points are selected by the Timeline creator (see Figure 4.3). The snapshots are plotted sequentially in a timeline format.[2] When plotted this way, the snapshots break the preprocessing pipeline down into digestible pieces and mirror the sequence of programming instructions used to implement a data preprocessing strategy.

[2]For a horizontal Smallset Timeline, snapshots are arranged from left to right and can span multiple rows. For a vertical Smallset Timeline, snapshots are arranged from top to bottom and can span multiple columns.

Colours are used to highlight data changes between snapshots. This technique of colour-encoding differences between two tables appears in exploratory data tools [Fitzpatrick et al., 2023, Gaslam, 2021, Niederer et al., 2017, Tierney, 2017] and explanations of data transformations [Kasica et al., 2021, Posit Software, PBC, 2023]. The colours used in Smallset snapshots represent general changes undergone by a dataset: 1) it gets bigger, 2) it gets smaller, or 3) it stays the same size, but the contents change. Thus, the scheme has four categories: added, deleted, edited, and unchanged. It is limited to four, to minimise consultation with the colour legend while reading a Smallset Timeline. Timeline creators can choose a four-colour palette consistent with the visual style of their document. If a certain type of change is not part of the preprocessing steps, the category is left off the legend (e.g., see Figure 6.1).

As noted above, Timeline creators are responsible for selecting snapshot points. This selection will likely depend on the purpose of the Smallset Timeline (see Table 4.1). For example, in Figure 4.3, Alice chooses to take snapshots showing exactly one operation at a time. This approach emphasises the effects of each operation and helps prepare documentation for reproducing or replicating the steps. Alternatively, Bob groups related operations together as a composite preprocessing step. This approach conveys the conceptual outline rather than the details of preprocessing. It is suited to mediums in which space and reader attention span are limited (e.g., research article, white paper, or blog post). Note that if a data point has been altered more than once since the last snapshot, the cell colour will reflect the most recent change, i.e., one operation becomes hidden behind another. Here the choice to prioritise simplicity and minimise visual clutter was made.

Originally, in addition to colour-coding, snapshots included *stamps* in table cells undergoing an addition, deletion, or edit. A stamp was a small circle containing a letter or symbol, indicative of the data operation resulting in the data change. Figure 4.4 provides an example of stamps, where the letter *I* stands for *imputation*. The stamps feature was dropped because it added visual clutter. It was hard to select intuitive letters/symbols, and they had to be explained in the caption, where the change was being explained anyway. One alternative to stamps worth exploring is the option to assign colours to data operations more specific than *added*, *deleted*, and *edited*. This is discussed briefly in Chapter 7 and has been flagged as future work.

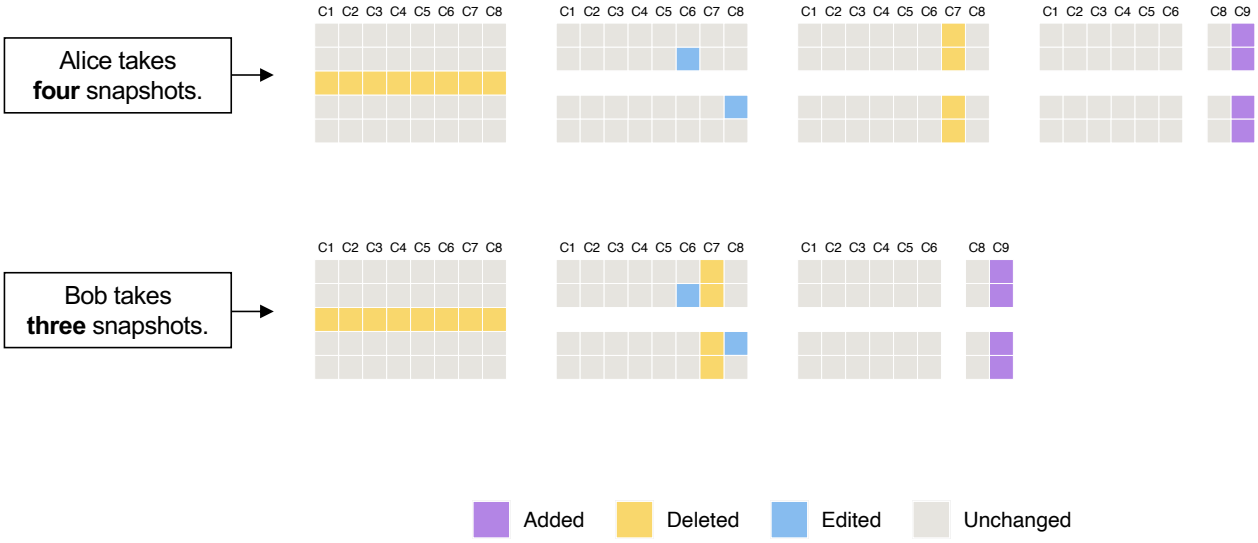


Figure 4.3: Diagram showing discretion in the selection of snapshot points. Alice generates one snapshot for each preprocessing decision, while Bob combines two related decisions in the second snapshot. Both sets of snapshots are based on the s_data example (see Appendix A).

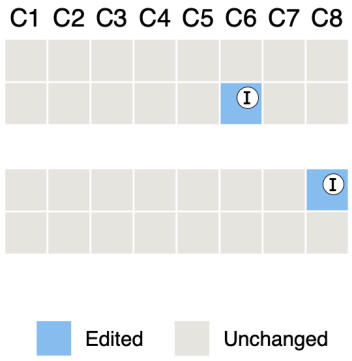


Figure 4.4: An example of the *stamps* visual feature, discarded early in the design process of the Smallset Timeline, due to the non-intuitive visual clutter it added to snapshots. Here, the letter I in the stamp stands for data *imputation*.

Captions

Captions accompany snapshots to provide information about the preprocessing steps. There is one caption per snapshot. Timeline creators are responsible for providing these captions, which should supply Timeline readers with information that enhances their understanding of the process. The caption text is usually located beneath snapshots. It can also be placed to the left or right of a snapshot, if the Smallset Timeline is arranged vertically.

At the most basic level, a caption says what was done in the preprocessing step. The colour categories for data changes are broad, so a caption allows the exact nature of the change to be stated. From there, the caption can be upgraded to also explain why the preprocessing step was done. Timeline creators can use the caption space to defend and discuss their preprocessing decisions and offer rationale. A detailed explanation is especially important if a decision deviates from a preprocessing norm. In some instances, it may be necessary to also specify how the preprocessing step was done. This information can be essential for Timeline readers trying to reproduce or replicate the preprocessing steps.

The caption style will depend on the purpose of the Smallset Timeline. To caption appropriately for general comprehension and evaluation of preprocessing decisions (Table 4.1 column 3), jargon is avoided, and the text is pared back to the most relevant parts to prevent information overload. For the purpose of reproducing or replicating data preprocessing decisions (Table 4.1 column 2), snapshot captions may be detailed, include jargon, and reference preprocessing code. Timeline readers aiming to reproduce or replicate steps likely have some familiarity with the topic, such that the amount and type of information are not overwhelming. In short, the content of the captions should be tailored to the target audience and the reason they need the information. Captioning for different audiences and purposes is explored further in Section 6.1.

4.2.2 Four enrichment features

In addition to the three core visual components, the Smallset Timeline has four enrichment features: *printed data*, *missing data tints*, *ghost data*, and *resume markers*. These are optional features that can be activated to augment snapshots and provide more information about the data or preprocessing pipeline. The utility of the enrichment features will depend on data privacy, the application of interest, the preferences of the Timeline creator, and the information needs of the Timeline reader.

Figure 4.5: Overview of three of the four Smallset enrichment features. See Section 4.2.2 for descriptions. Snapshots are based on s_data, detailed in Appendix A.

For the first three enrichment features (i.e., printed data, missing data tints, and ghost data), Figure 4.5 shows snapshots when the feature is not activated ("off") versus activated ("on"). Activating the feature means it will be applied in all snapshots in the Smallset Timeline; it cannot be applied selectively to some snapshots or activated part way through the Smallset Timeline. The fourth enrichment feature, resume markers, is illustrated in Figure 4.6. Each is described in more detail below.

Printed data

Printed data (Figure 4.5, column A) can be included in Smallset snapshots for a glimpse of the data and real examples of how it changes between steps.[3] If reading a Smallset Timeline to reproduce a dataset, printed values enable comparisons between the original dataset and reproduced version. Even if preprocessing code appears to run successfully, having the printed values might verify that the code still does what the original author intended it to do. If

[3]When this feature is activated, it may be necessary to truncate long data values to fit in the table cells. The truncation is represented with an ellipsis (i.e., "...").

data are sensitive or not publicly available, the Smallset Timeline can be configured without printed data. Note, though, that omitting the data does not necessarily guarantee data privacy. One can imagine a scenario where identification in a dataset could occur based on the unique combination of preprocessing operations undergone by certain observations.

Missing data tints

The "missing value shadow" [Swayne and Buja, 1998], or "shadow matrix" [Tierney and Cook, 2023], is a visual technique that contrasts light and dark to reveal missing values in a table of data.[4] The Smallset Timeline draws inspiration from this technique, to highlight missing data. However, instead of shadows, tints are used, i.e., *missing data tints* (Figure 4.5, column B). A subtle tint makes the issue noticeable without diverting attention from other visual elements or making it hard to see printed data in table cells. It also has metaphorical value; part of the colour is missing, just like the data itself. If values are imputed, the table cells are not filled with tints in subsequent snapshots.

When this enrichment feature is activated, the colour legend is updated accordingly. The labels of colours with tints are tagged with an asterisk, that points to the note: *A lighter value indicates a missing data value* (e.g., see Figure 6.1). One notable limitation of using missing data tints is that the Smallset Timeline may no longer be colourblind safe, even if the four-colour palette is. For instance, the snapshot in Figure 4.5 (column B, row "on") is not colourblind safe.

Ghost data

When row or column deletions are visualised in a Smallset, the Smallset table shrinks in size. When this happens in a Smallset Timeline, it can become difficult to track individual data points across the entire Smallset Timeline, as they shift in space relative to each other. The *ghost data* enrichment feature (Figure 4.5, column C) provides the option to plot blank (i.e., "ghost") rows and columns where data have been deleted. When ghost data are included, the position of each table cell in a Smallset is maintained throughout the entire Smallset Timeline. In turn, data can be readily traced across snapshots. Ghost data can also serve as a visual reminder in all subsequent snapshots that data have been deleted in a previous step.

[4] A similar visualisation is also offered by the `vis_miss()` command in the `visdat` R package [Tierney, 2017]. It plots an entire tabular dataset (as a grid of boxes) and colour-encodes the values as missing or not.

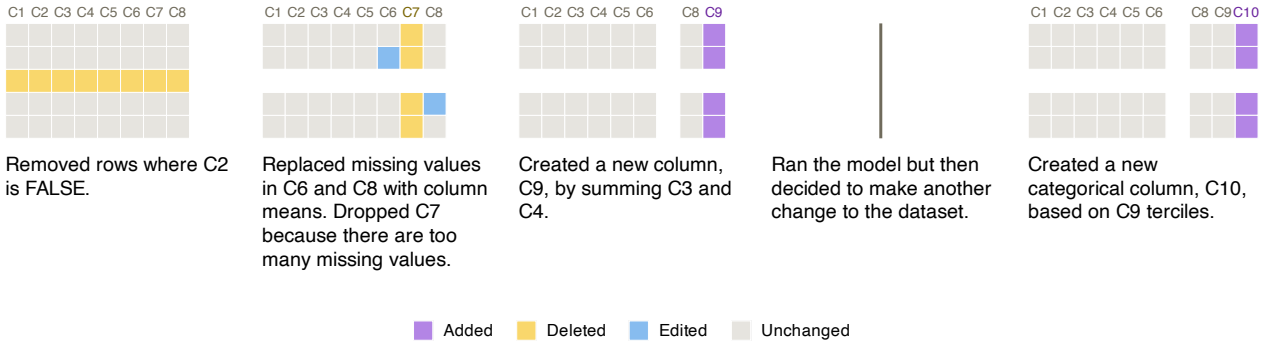


Figure 4.6: A Smallset Timeline (for the s_data example detailed in Appendix A) with a resume marker—the vertical bar after the third snapshot—an enrichment feature discussed in Section 4.2.2.

The default setting in the `smallsets` software (discussed in Chapter 5) is to use ghost data because it seems to make interpretation easier in many cases. However, it is optional, as there are some situations where it might not be helpful or suitable. For example, if many columns are deleted in the first snapshot (e.g., ten columns are deleted), it may be best to deactivate ghost data. Doing so would avoid large blocks of white space in subsequent snapshots and preserve space overall, keeping the figure compact in size.

Resume markers

Resume markers are the fourth enrichment feature. A resume marker is a vertical line placed between two snapshots to denote that preprocessing was stopped to start the final estimation or modelling task but then was subsequently "resumed" to make additional dataset changes. A resume marker can have its own caption, just like a snapshot, to explain why preprocessing resumed. The caption either sits below or beside the resume marker, depending on the layout of the Smallset Timeline. Figure 4.6 is an example Smallset Timeline with a resume marker.

The resume marker enrichment feature is designed to be of use when discussing iterations in more exploratory analyses or unexpected issues that necessitate modifying the initial estimation/modelling plan. In the latter case, this enrichment feature is not meant to condone any type of data dredging. Instead, it is designed to allow Timeline creators to be transparent about unforeseen roadblocks—resulting in multiple attempts at estimation/modelling—and revisions to the original estimation/modelling plan.

4.3 Methods for Smallset selection

When the Smallset was introduced as the first core visual component in Section 4.2, the row selection for a Smallset was treated as given, to focus on the overall design. This section focuses on Smallset selection, and in particular, automated Smallset selection. Manual selection of rows is technically an option, but it places the burden on Timeline creators to find suitable rows. It is also subject to cherry picking, which may result in a misleading visualisation.

The aim in Smallset selection is to find rows containing examples of preprocessing changes. Random sampling is one way to automate Smallset selection, but it may not be effective, especially if the number of Smallset rows is low or certain preprocessing changes occur infrequently. This creates a demand for other automated selection methods that can guarantee rows with preprocessing examples. In turn, two optimisation models were developed for Smallset selection. These models are introduced in Section 4.3.1 and tested in Section 4.3.2.

4.3.1 Two optimisation models

This section details the two optimisation models in Table 4.2, for automated Smallset selection: 1) the *coverage* model and 2) the *coverage + variety* model. The first model centres around the selection criterion *preprocessing coverage*, which tries to ensure that the Smallset contains at least one example for each preprocessing step (i.e., snapshot). The second model centres around two criteria. This includes *preprocessing coverage* (like before) and *visual variety*, which tries to ensure that a variety of changes are shown within and across snapshots. Following is a discussion on the selection criteria, the selection models, and key considerations in model development.

Data representations for the selection criteria

The first selection criterion is *preprocessing coverage*, which means a Smallset has at least one example for each preprocessing step.[5] Incorporating it into a selection model required a new data representation: the *coverage indicator matrix*, denoted by $\mathbf{C}$. Let the original

[5]Here, the terms *step* (as in preprocessing step) and *snapshot* (as in Smallset snapshot) are used interchangeably. In other words, if a Smallset Timeline has four snapshots, there are four preprocessing steps. It is assumed that a step can consist of one or more preprocessing decisions, meaning a single snapshot can show various preprocessing changes resulting from different preprocessing decisions.

Coverage model

$$\max_{\mathbf{z}} \quad 1$$
$$\text{s.t.} \quad \sum_{i=1}^{N} z_i = K$$
$$\sum_{i=1}^{N} z_i c_{ih} > 0, \ \forall h = 1, \ldots, H$$

Coverage + variety model

$$\max_{\mathbf{z}} \quad \mathbf{z}^\top Q \mathbf{z}$$
$$\text{s.t.} \quad \sum_{i=1}^{N} z_i = K$$
$$\sum_{i=1}^{N} z_i c_{ih} > 0, \ \forall h = 1, \ldots, H$$

Table 4.2: Two optimisation models for Smallset selection, discussed in Section 4.3.1.

dataset, $\mathbf{X}$, be an $N \times M$ matrix, where x_{ij} is the data value in the i-th row and j-th column. Data matrix $\mathbf{X}$ goes through $h = 1, \ldots, H$ preprocessing steps, $f_1, \ldots, f_H$, resulting in a processed data matrix after each step, $\hat{\mathbf{X}}_h = f_h \ldots f_1(\mathbf{X})$. The binary *coverage indicator matrix* $\mathbf{C} \in \{0,1\}^{N \times H}$ is N data points by H preprocessing steps. Each element c_{ih} is *1* if and only if the i^{th} row is altered by preprocessing step f_h and *0* otherwise. An example *coverage indicator matrix* is presented in Figure 4.7(A).

The second selection criterion is *visual variety*, which means the Smallset includes rows affected by the steps differently, to show a variety of changes within and across snapshots. The data representation required to incorporate the *visual variety* criterion into a selection model is the *visual appearance matrix*, denoted by $\mathbf{A} \in \mathcal{R}^{N' \times M'}$. It is the size of the original data matrix $\mathbf{X}$ plus any rows and/or columns added in preprocessing. Its elements $a_{ij} \in \{U, A, D, E\}$ (corresponding to *unchanged*, *added*, *deleted*, and *edited*, respectively) encode the last change undergone by a data cell, from the original data matrix $\mathbf{X}$ to the final data matrix $\hat{\mathbf{X}}_H$. An example *visual appearance matrix* is presented in Figure 4.7(B).

Coverage model

The *coverage* model for Smallset selection incorporates the selection criterion *preprocessing coverage* (see Table 4.2). The output is an indicator vector $\mathbf{z} \in \{0,1\}^N$, in which z_i is *1* if row i is selected and *0* otherwise. In the model, there are two constraints. The first is that exactly K rows are selected out of the original N rows, i.e., the Smallset must be of the size specified. The second is *preprocessing coverage*. For each step, the number of rows in the *coverage indicator matrix* that preprocessing step h affects is computed, and this has to be

	Step1	Step2	Step3
1	0	1	1
2	0	1	1
3	1	0	0
4	1	0	0
5	0	1	1
⋮	⋮	⋮	⋮
98	0	1	1
99	1	0	0
100	0	1	1

A) Coverage indicator matrix.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
1	U	U	U	U	U	E	D	E	A
2	U	U	U	U	U	U	D	U	A
3	D	D	D	D	D	D	D	D	D
4	D	D	D	D	D	D	D	D	D
5	U	U	U	U	U	U	D	E	A
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
98	U	U	U	U	U	U	D	U	A
99	D	D	D	D	D	D	D	D	D
100	U	U	U	U	U	U	D	U	A

B) Visual appearance matrix.

Figure 4.7: Example data representations, generated for Smallset selection, for the s_data dataset and three-step preprocessing scenario, detailed in Appendix A. See the discussion on data representations in Section 4.3.1.

greater than zero. The *coverage* model tries to satisfy the constraints without any additional objective (hence, the max 1 term in Table 4.2). This is an integer linear program solved with the Gurobi Optimizer software [Gurobi Optimization, LLC, 2022].

Coverage + variety model

The *coverage + variety* model for Smallset selection incorporates the *preprocessing coverage* and *visual variety* criteria (see Table 4.2). A pre-calculated $N \times N$ distance matrix Q contains the Hamming distance between the vectors of any two rows in the *visual appearance matrix*. That is, $q_{il} = \sum_j d(a_{ij}, a_{lj})$, with distance function $d(\cdot, \cdot)$ being *0* if the two values are the same and *1* otherwise. The objective function $\mathbf{z}^\top Q\mathbf{z}$ computes the total pair-wise Hamming distance among the selected rows. Like in the *coverage* model, there are two constraints: 1) the Smallset must be of size K and 2) there must be *preprocessing coverage*. In this model, though, the objective is to maximise *visual variety*. It is an integer quadratic program also solved with the Gurobi Optimizer software [Gurobi Optimization, LLC, 2022].

Key considerations in model development

Development of the selection models was an iterative process, involving trial and error. Three key moments in model development are discussed below.

- **Snapshots versus data values.** Initially, model development started by considering preprocessing changes at the data value level. The goal was to optimise the selection

of rows with the most values changed (added, deleted, or edited). Imagine, though, that the first preprocessing step includes a row deletion. Here, *all* values in the deleted rows are considered "changed," and those rows would be prioritised in selection. This could result in a Smallset Timeline in which the entire Smallset disappears after the first snapshot. In turn, the focus shifted from the data value level to the snapshot level, eventuating in the selection criterion *preprocessing coverage*.

- **No objective for the *coverage* model.** The *coverage* model has no objective, only two constraints. Before settling on no objective, though, different objectives were tested. For example, one trialled objective was to maximise the number of changes shown, by prioritising rows undergoing changes in multiple steps. Yet, when testing this approach, the solutions were not preferable. For example, in one test, all selected rows were undergoing a change in every step; however, they were all undergoing the exact same set of changes, even though not all changed rows did, in the full dataset. The repetition among rows wasted space and misled. Thus, it seemed preferable to set no objective, only two constraints, and get a mix of rows.

- **Development of *two* models.** One notable shortcoming of the *coverage* model is that, when a snapshot contains multiple types of preprocessing changes, representation of the different changes is not guaranteed. The coverage model is only designed to ensure at least one example change per snapshot. In turn, the *coverage + variety* model was developed. It addresses the shortcoming noted above (with use of the *visual variety* criterion) but at the cost of needing to compute and optimise with a distance matrix Q, that is quadratic in the number of rows. This can result in long runtimes for large datasets. Thus, both models are included as selection algorithms in the `smallsets` software, discussed in Chapter 5.

4.3.2 Comparing selection methods

Each selection method was tested using the s_data example, described in Appendix A. Figure 4.8 presents the test results for selecting $K = 5$ rows. The test shows that random sampling fails to achieve *preprocessing coverage*.[6] Selection with the *coverage* model satisfies

[6]Note that random sampling was used in Figure 4.1, where it happened to satisfy the *preprocessing coverage* criterion, unlike in Figure 4.8. For selection, the seed in R was set to `145` in Figure 4.1 and `99` in Figure 4.8.

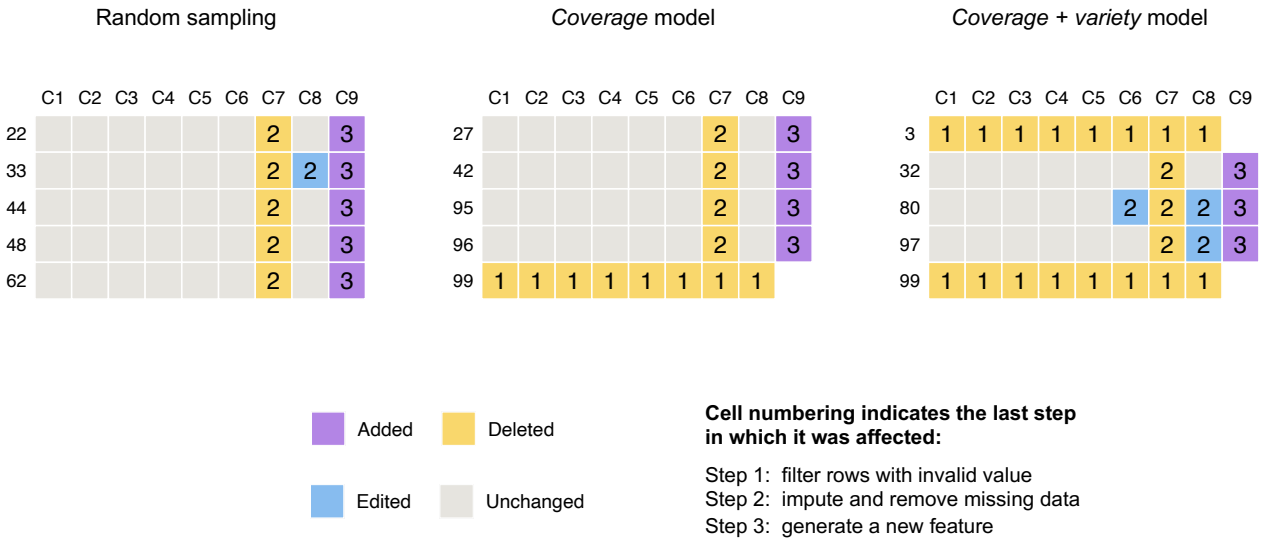
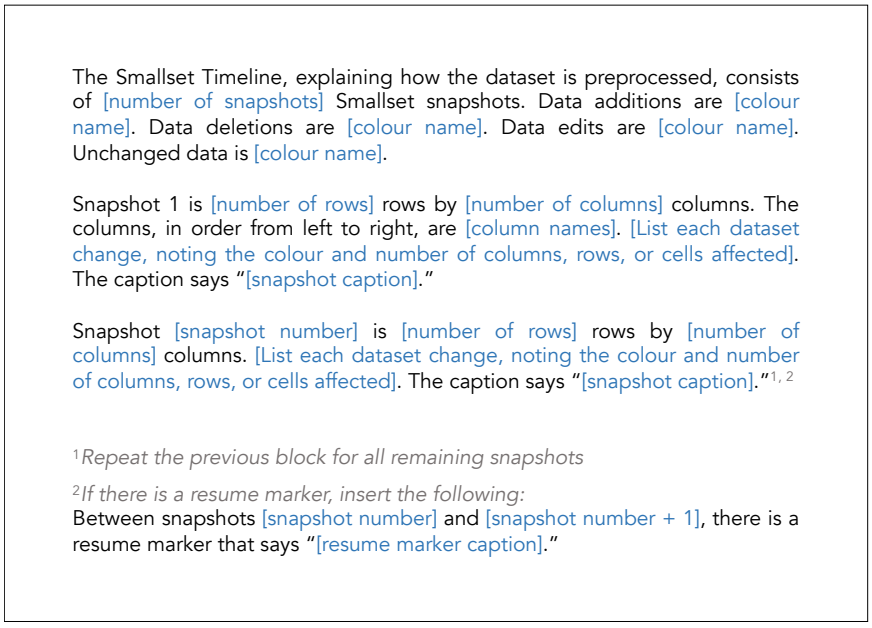


Figure 4.8: Smallsets selected by random sampling (left), the *coverage* model (middle), and the *coverage + variety* model (right) for s_data (see Appendix A). Shown for each selection method is one Smallset snapshot with accumulated changes (indicated by cell color) across the three preprocessing steps (indicated by the numbering of the cell). Row numbers refer to those in the original dataset (see Table A.1).

the *preprocessing coverage* criterion. However, it does not satisfy the *visual variety* criterion; it misses a data edit example in the second step and has visual repetition among the first four rows. Selection with the *coverage + variety* model satisfies *preprocessing coverage* as well as *visual variety*. There are three rows affected differently by the second step. There is also one row with minimal changes, another positive by-product of the *visual variety* criterion and *coverage + variety* model design. In Chapter 6, all three selection methods are used to build example Smallset Timelines.

4.4 Alternative text

One key argument of this thesis is that visualisation is an effective way to make preprocessing information accessible. However, visualisations are not accessible to those with visual impairments, unless there is alternative (alt) text (i.e., a text description of an image's content). Thus, an alt text template was developed for the Smallset Timeline (Figure 4.9). When populated, this alt text template outlines key features of the Smallset Timeline and individual



The Smallset Timeline, explaining how the dataset is preprocessed, consists of [number of snapshots] Smallset snapshots. Data additions are [colour name]. Data deletions are [colour name]. Data edits are [colour name]. Unchanged data is [colour name].

Snapshot 1 is [number of rows] rows by [number of columns] columns. The columns, in order from left to right, are [column names]. [List each dataset change, noting the colour and number of columns, rows, or cells affected]. The caption says "[snapshot caption]."

Snapshot [snapshot number] is [number of rows] rows by [number of columns] columns. [List each dataset change, noting the colour and number of columns, rows, or cells affected]. The caption says "[snapshot caption]."[1, 2]

[1] *Repeat the previous block for all remaining snapshots*

[2] *If there is a resume marker, insert the following:*
Between snapshots [snapshot number] and [snapshot number + 1], there is a resume marker that says "[resume marker caption]."

Figure 4.9: Alt text template for Smallset Timelines. See Section 4.4 for discussion.

snapshots. The `smallsets` software, discussed in Chapter 5, automatically populates this template and prints the text to the screen, at the request of software users. One key limitation of the alt text template is that it is currently only in the English language. Moreover, the template assumes that a Timeline reader knows what a Smallset Timeline is. A Timeline creator may need to add more description regarding the figure layout and Smallset, to assure these aspects of the image are clear to those using the alt text.

4.5 Summary

This chapter proposes a novel visualisation, the Smallset Timeline, for communicating data preprocessing decisions. There are two defined users of the Smallset Timeline: the Timeline creator and Timeline reader. For Timeline creators, the Smallset Timeline is designed to encourage reflection and reflexivity, in regards to data preprocessing decisions. For Timeline readers, the Smallset Timeline is designed to support the replication, comprehension, and evaluation of data preprocessing decisions. The visualisation consists of three core visual components, including the Smallset, snapshots, and captions. The visualisation also offers four optional enrichment features, including printed data, missing data tints, ghost data, and

resume markers. Two optimisation models are available for automated Smallset selection: the *coverage* model and the *coverage + variety* model. There is also support for generating alt text for Smallset Timelines.

In the following chapter, the software tool for building Smallset Timelines, `smallsets`, is introduced. Then, in Chapter 6, `smallsets` is used to build six example Smallset Timelines for real-world datasets, in a series of data preprocessing case studies.

Chapter 5

`smallsets`: Software for Building Smallset Timelines

This chapter presents the `smallsets` R package, an open-source software tool for building Smallset Timelines. It is available on the Comprehensive R Archive Network (CRAN)[1] and licensed under the GNU General Public License v3.0.[2] The `smallsets` codebase is also housed on GitHub in a public repository.[3] Although an R package, `smallsets` can be used to produce Smallset Timelines for data preprocessing decisions encoded in the R *or* Python programming languages in R, R Markdown, Python, or Jupyter Notebook files (extensions `.R`, `.py`, `.Rmd`, or `.ipynb`, respectively).

The chapter begins by outlining the three design goals that shaped `smallsets` software development (Section 5.1). Following, the frontend and backend of `smallsets` are discussed, including the user interface and workflow (Section 5.2) and the package's dependencies and internal structure (Section 5.3). Section 5.4 considers how `smallsets` has evolved over time, using three major `smallsets` software releases as reference points in time. Throughout, the synthetic dataset s_data—first appearing in Chapter 4—is used to illustrate different aspects of the software. Details on s_data can be found in Appendix A.

[1] `https://CRAN.R-project.org/package=smallsets`
[2] `https://cran.r-project.org/web/licenses/GPL-3`
[3] `https://github.com/lydialucchesi/smallsets`

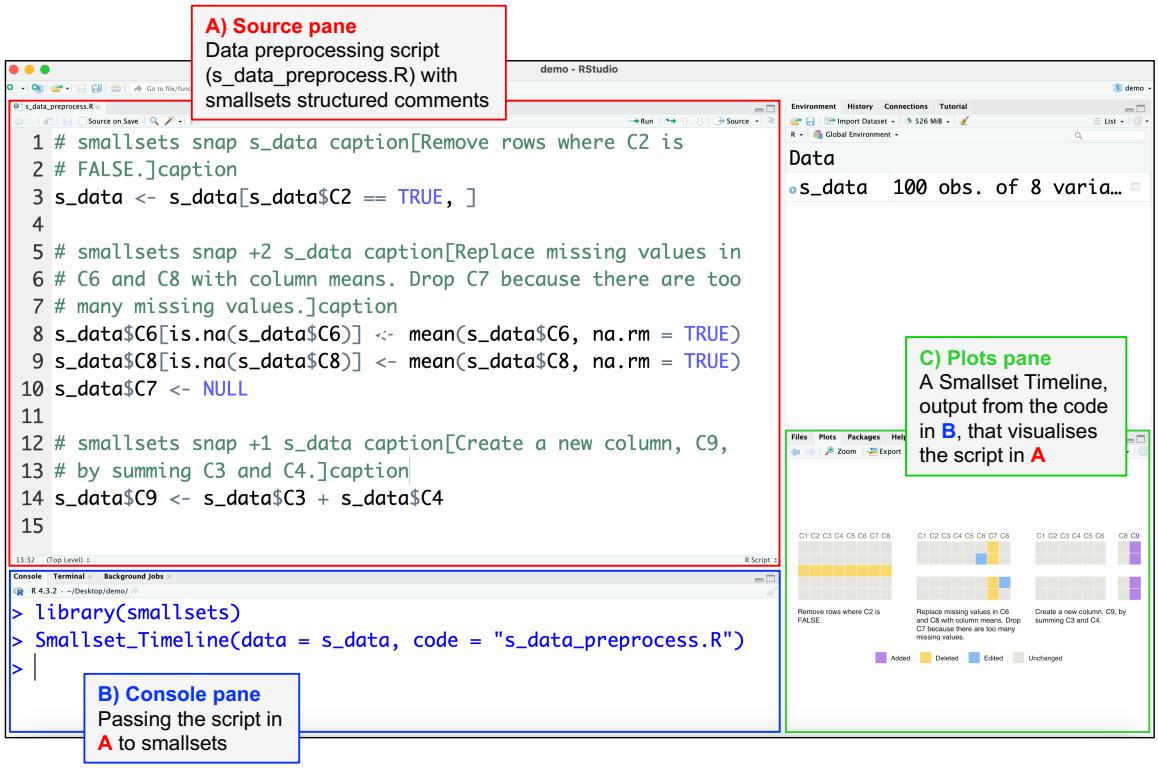


Figure 5.1: Screenshot of a session in RStudio [Posit team, 2023], in which `smallsets` is used to build a Smallset Timeline for the s_data example (see Appendix A). In box A (red), there is a data preprocessing script with `smallsets` structured comments, which is passed to the `Smallset_Timeline()` command in box B (blue), producing the figure in box C (green).

5.1 Design goals

There are three design goals that shaped development of `smallsets`. The first design goal is that any additional work required to build a Smallset Timeline with `smallsets`—on top of what is already required by the data preprocessing stage—should be minimised. The second design goal is that any additional work that *is* required to build a Smallset Timeline with `smallsets` should feel meaningful and productive to complete. The third design goal is that building a Smallset Timeline with `smallsets` should be easily integrable into new *or* existing data preprocessing workflows.

These design goals were grounded in the assumption that both users' time and incentives for producing preprocessing documentation would be low. In other words, the strategy was to

be pragmatic about the `smallsets` deployment context, to preemptively account for possible barriers to uptake. As a result, the design goals centre on making `smallsets` simple and easy to use. These three design goals laid the foundation for `smallsets` software development and informed design choices, especially related to the user interface and workflow, which is discussed next in Section 5.2.

5.2 User interface and workflow

To build a Smallset Timeline with `smallsets`, there are two main steps. First, users must insert structured comments, with snapshot instructions, into their R, R Markdown, Python, or Jupyter Notebook file containing the data preprocessing code (see Section 5.2.1). Next, users must pass their un-preprocessed dataset and commented preprocessing file to the `Smallset_Timeline()` command in `smallsets` (see Section 5.2.2); the output is a Smallset Timeline. Figure 5.1 shows these steps executed in an RStudio session [Posit team, 2023]. Each of the two main steps is detailed below.

5.2.1 Inserting structured comments

The first step in the `smallsets` user workflow is inserting structured comments into the R, R Markdown, Python, or Jupyter Notebook file containing the data preprocessing code. The structured comments contain instructions for `smallsets` on how to build the Smallset Timeline. This is similar to docstrings [Goodger and van Rossum, 2001] in Python or `roxygen2` comments [Wickham et al., 2020] in R, for generating function documentation.

In `smallsets`, there are two types of structured comments—snapshot comments and resume marker comments—which are shown in Figure 5.2. Snapshot comments are required to build a Smallset Timeline. Resume marker comments are only necessary if a Timeline creator wants to include a resume marker enrichment feature in a Smallset Timeline (see Section 4.2.2). The format of snapshot and resume marker comments is the same regardless of whether the preprocessing code is in R or Python. Each comment type is discussed separately below.

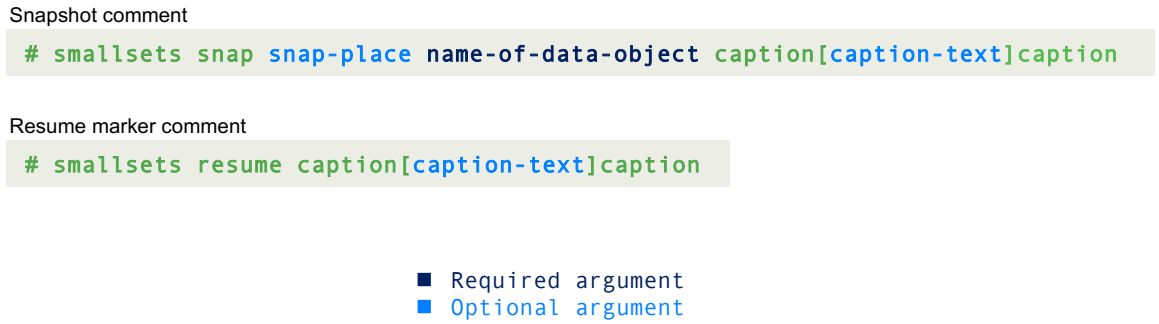


Figure 5.2: Format of snapshot and resume marker structured comments, with colour-coded arguments.

Snapshot comments

Smallset Timelines always consist of, at a minimum, two snapshots. Therefore, users must always include at least two structured snapshot comments in their script of code. A snapshot comment tells the `smallsets` software where to take a snapshot of the Smallset, what data object to take the Smallset snapshot in, and what the snapshot caption should be. All changes made to the dataset between the first and last specified snapshot places will be tracked by `smallsets`. Any code before or after these places will be ignored by `smallsets`. This means that a user can have code related to different stages of the analysis (e.g., importing the data or running a model) within the same script that is passed to `smallsets`.

The format of a snapshot comment is detailed in Figure 5.2. It includes three arguments. The first argument, which is optional, is *snap-place*. The *snap-place* argument tells `smallsets` where to take the snapshot. If unspecified, the snapshot is taken exactly where the snapshot comment is located within the code. Alternatively, a user can specify the snapshot location one of two ways. They can use a plus sign to specify how many lines of code later to take the snapshot; for example, *+3* means take the snapshot after three more lines of code. They can also choose to specify the line of code after which to take the snapshot; for example, *17* means take the snapshot after the seventeenth line of code.

The *snap-place* argument aims to give users substantial flexibility in comment placement within a script of code. Figure 5.3 shows two example preprocessing scripts—Script A and Script B—for the s_data dataset, with inserted snapshot comments. Both Scripts A and B contain the same preprocessing code and produce the same Smallset Timeline (Figure 4.1). However, the comment placement differs between them. In Script A, the comments are

dispersed throughout the preprocessing code. In Script B, the comments are confined to a block at the top of the script, ahead of any code. Across the two scripts, we can see examples of different ways to specify the snapshot place. Script A includes an example of no *snap-place* argument (line 1), which results in a snapshot where the comment is located (this could also be achieved by passing *+0* to *snap-place*).

The second argument in the snapshot comment is *name-of-data-object*. This is a required argument for all snapshot comments. It tells `smallsets` what data object to apply the Smallset snapshot in. The value passed to this argument may change between the snapshot comments in a preprocessing script, if a data practitioner assigns the data object being preprocessed to a new name partway through the preprocessing pipeline.

The third argument is *caption-text*, for specifying the snapshot caption. Although technically optional (`smallsets` will run successfully without it), it is *strongly* recommended to include a caption for each snapshot, as it is one of the three core visual components of the Smallset Timeline (see Section 4.2.1). Regardless, it is always necessary to include the caption brackets, even if no text is included between them. The `smallsets` tool recognises some Markdown syntax within the *caption-text* argument, such as italicising and bolding text with single and double asterisks, respectively (e.g., Figure 6.1).

Resume marker comments

The second type of structured comment is the resume marker comment. It is only necessary if a user wants to include a resume marker in a Smallset Timeline. As previously discussed in Section 4.2.2, the resume marker enrichment feature is a vertical bar placed between two snapshots (see Figure 4.6). It signals that data preprocessing was stopped to move on to the next stage in the data preprocessing pipeline but then resumed to make additional dataset changes.

Within a script, the resume comment must be located between two snapshot comments. In other words, it cannot be the first or last structured comment in a script. Its format is described in Figure 5.2. The format is similar to that of the snapshot comment but includes only one optional argument, *caption-text*, which can be used to create a caption for the resume marker. Unlike the snapshot comment, the resume comment does not include the *snap-place* or *name-of-data-object* arguments. The location of the resume marker is determined by the surrounding snapshot comments, and information about the data object is not needed.

Script A

```
# smallsets snap s_data caption[Remove rows where C2 is FALSE.]caption
s_data <- s_data[s_data$C2 == TRUE, ]

# smallsets snap +2 s_data caption[Replace missing values in C6 and
# C8 with column means. Drop C7 because there are too many missing
# values.]caption
s_data$C6[is.na(s_data$C6)] <- mean(s_data$C6, na.rm = TRUE)
s_data$C8[is.na(s_data$C8)] <- mean(s_data$C8, na.rm = TRUE)
s_data$C7 <- NULL

# smallsets snap +1 s_data caption[Create a new column, C9, by summing
# C3 and C4.]caption
s_data$C9 <- s_data$C3 + s_data$C4

```

Script B

```
# smallsets snap 8 s_data caption[Remove rows where C2 is FALSE.]caption
# smallsets snap 13 s_data caption[Replace missing values in C6 and
# C8 with column means. Drop C7 because there are too many missing
# values.]caption
# smallsets snap 17 s_data caption[Create a new column, C9, by summing
# C3 and C4.]caption

# remove rows where C2 is false
s_data <- s_data[s_data$C2 == TRUE,]

# deal with missing data
s_data$C6[is.na(s_data$C6)] <- mean(s_data$C6, na.rm = TRUE)
s_data$C8[is.na(s_data$C8)] <- mean(s_data$C8, na.rm = TRUE)
s_data$C7 <- NULL

# create a new variable
s_data$C9 <- s_data$C3 + s_data$C4

```

Figure 5.3: Two example R preprocessing scripts (A and B) for s_data, which consist of the same code but illustrate different approaches to inserting structured comments for `smallsets`. Both Scripts A and B produce the same three-snapshot Smallset Timeline (Figure 4.1), despite different comment placement.

5.2.2 Figure production and customisation

The second step in the user workflow, after inserting structured comments in the preprocessing code, is running the main `smallsets` command: `Smallset_Timeline()`. This command has two required arguments: `data` and `code`. Users pass the un-preprocessed dataset to the `data` argument. The `code` argument requires the filename of the script (one of R, R Markdown, Python, or Jupyter Notebook) with inserted structured comments. Listing 5.1 demonstrates the syntax for running this command for the s_data example, where the s_data dataset is stored in a data frame called `s_data` and the preprocessing code is housed the R script *s_data_preprocess.R*. If the file is not located in the working directory, the file path must also be included in the string passed to the `code` argument.

```
# load smallsets
library(smallsets)

# build a Smallset Timeline
Smallset_Timeline(data = s_data, code = "s_data_preprocess.R")
```

Listing 5.1: Example R code for running the main `smallsets` command for s_data, where the dataset is assigned to object `s_data` and the preprocessing code with inserted structured comments is in the file *s_data_preprocess.R*, located in the user's working directory.

Prior to running the `Smallset_Timeline()` command, users must ensure that all packages/libraries required by the preprocessing code are loaded in the current environment, as `smallsets` must be able to re-execute the preprocessing code on the backend when the command is run. Note that, although `smallsets` executes a user's preprocessing code on the backend, the un-preprocessed dataset passed to `data` is not altered in any way by `smallsets`. In other words, the dataset is still un-preprocessed after `Smallset_Timeline()` is run. A more detailed discussion about the backend of `smallsets` can be found in Section 5.3.

The `Smallset_Timeline()` command can be run from within the preprocessing script itself, i.e., the file passed to the `code` argument. This eliminates the need to create additional files in one's data analysis workflow. It is also useful in the case of R Markdown and Jupyter Notebook files, which can mix code, plain text, and figures all in one file. For example, in R Markdown one could create a hidden chunk of code at the top of the file, which produces the Smallset Timeline and assigns it to an object. That object can then be printed anywhere in the file and rendered as a figure in an R Markdown report.

There are also various optional arguments in `Smallset_Timeline()`. They fall under

three main categories: 1) Smallset selection, 2) visual settings, and 3) alternative text (alt text). All optional arguments are listed by category, in Table 5.1. Each optional argument has a default setting for when the argument is not specified by a user. The table includes pointers to example Smallset Timelines within this thesis, where the default setting was changed. Each category of optional arguments is discussed in more detail below.

Smallset selection

The first category of arguments relates to Smallset selection. This includes the size of the Smallset (`rowCount`) and the automated Smallset selection method (`rowSelect`). Within `smallsets`, there are three automated selection methods available: 1) random sampling, 2) the *coverage* model, and 3) the *coverage + variety* model (see Section 4.3). In `smallsets`, the *coverage* and *coverage + variety* models are solved with the Gurobi Optimizer [Gurobi Optimization, LLC, 2022] using the Gurobi's R interface [Gurobi Optimization, LLC, 2021]. Users must obtain a Gurobi license[4] and install Gurobi locally, to use these two automated selection methods.

Other arguments include `rowReturn` and `rowIDs`. When set to `TRUE`, `rowReturn` prints the row names/indices of the Smallset selection, to the console. The `rowIDs` argument accepts a vector of row names/indices, for rows to include in the Smallset. These two arguments can be used in tandem, to avoid having to re-select rows with each run of `Smallset_Timeline()`. For example, if a user chooses the *coverage+variety* model for Smallset selection, they can print the solution with `rowReturn`. Then, they can pass that solution to `rowIDs` in future runs of `Smallset_Timeline()`, to avoid re-selection and minimise overall runtime.

Visual settings

The second category of arguments relates to visual settings for the Smallset Timeline, as there are various ways to customise the figure's appearance. The argument `colours` offers three built-in colour palettes to select from, or users can specify their own custom palette. All three built-in palettes are colourblind safe. One of the palettes also works when converted to grayscale, making it safe for black and white printing. This may be an important feature for Smallset Timelines included as figures in academic publications, which readers may print out in black and white to read.

[4]Gurobi offers free academic licenses.

Optional `Smallset_Timeline()` arguments

Argument	Description	Default	Example
	Smallset selection		
`rowCount`	Number of Smallset rows	Five	Figure 6.2
`rowSelect`	Automated Smallset selection method	Random sampling	Figure 6.1
`rowReturn`	Print row numbers of selected Smallset rows to the console	Do not print	
`rowIDs`	Rows to include in the Smallset	None	
	Visual settings		
`ignoreCols`	Columns to exclude from the Smallset Timeline visualisation	None	Figure 6.3
`colours`	Colour palette (built-in or custom)	First built-in palette	Figure 6.4
`printedData`	Print data values in snapshots	Not printed	Figure 6.9
`truncateData`	Truncate printed data values	No truncation	
`ghostData`	Include blank rows and columns where data have been deleted	Included	Figure 6.3
`missingDataTints`	Use colour tints to highlight missing data values	Not used	Figure 6.4
`align`	Alignment of snapshots	Horizontal	
`font`	Font of captions and labels	Sans	Figure 6.4
`sizing`	Size of caption text, column text, printed data values, legend icons/text, resume markers, and snapshot cells	See `sets_sizing()`	Figure 6.9
`spacing`	Caption space, column name rotation, column name space, and number of Smallset Timeline rows/columns	See `sets_spacing()`	Figure 6.3
`labelling`	Colour of column names and printed data	See `sets_labelling()`	Figure 6.8
	Alt text		
`altText`	Print alt text to console	Not printed	

Table 5.1: Optional arguments in the `Smallset_Timeline()` command, related to Smallset selection, visual settings, and alt text. The **Example** column points to *one* Smallset Timeline presented in this thesis where the default setting was changed (not exhaustive of all examples).

Automated output	Edited output
The Smallset Timeline, explaining how the dataset is preprocessed, consists of 3 Smallset snapshots. Data additions are mediumpurple2. Data deletions are lightgoldenrod2. Data edits are skyblue2. Unchanged data is gray89. Snapshot 1 is 5 rows by 8 columns. The columns, in order from left to right, are C1, C2, C3, C4, C5, C6, C7, and C8. 1 row is lightgoldenrod2. The caption says Remove rows where C2 is FALSE.. Snapshot 2 is 4 rows by 8 columns. Column C7 is lightgoldenrod2. 3 cells are skyblue2. The caption says Replace missing values in C6 and C8 with column means. Drop C7 because there are too many missing values.. Snapshot 3 is 4 rows by 8 columns. Column C9 is mediumpurple2. The caption says Create a new column, C9, by summing C3 and C4..	The Smallset Timeline, explaining how the s_data datast is preprocessed, consists of three Smallset snapshots, plotted in a horizontal line. In the Smallset Timeline, data additions are purple. Data deletions are yellow, and data edits are blue. Unchanged data is light gray. The first Smallset snapshot is five rows by eight columns. The columns, in order from left to right, are C1, C2, C3, C4, C5, C6, C7, and C8. One row in the snapshot is yellow, and the caption says, "Remove rows where C2 is FALSE." The second snapshot is four rows by eight columns. Column C7 is yellow, and three cells are blue. The caption says, "Replace missing values in C6 and C8 with column means. Drop C7 because there are too many missing values." The third snapshot is four rows by eight columns. A new column, C9, is purple. The caption says, "Create a new column, C9, by summing C3 and C4."

Table 5.2: Example alt text for the Smallset Timeline in Figure 4.1. The alt text on the left is automated output from the `smallsets` software. The alt text on the right is a manually edited version of the automated output on the left. This alt text assumes some familiarity with Smallset Timelines and their design.

Several arguments in this category correspond to the enrichment features discussed in Section 4.2.2. This includes printing data in snapshots (`printedData`), highlighting missing values with colour tints (`missingDataTints`), and plotting blank rows and columns where data have been deleted (`ghostData`). The `ignoreCols` argument accepts a character vector of columns to exclude from the Smallset Timeline visualisation. There are also arguments to adjust the snapshot alignment, text font/colour, and element sizing/spacing.

Alt text

The last category includes one argument: `altText`. When set to `TRUE`, `smallsets` produces alt text for a Smallset Timeline. Specifically, `smallsets` populates the alt text template introduced in Section 4.4 and prints it to a user's console. Therefore, a Timeline creator does not need to write alt text from scratch, as `smallsets` provides a first draft. Table 5.2 shows example alt text for the s_data example. The left side of Table 5.2 shows alt text returned

by `smallsets`. The right shows a version edited for improved grammar and clarity.

5.2.3 Resources for users

To teach the user interface and workflow of `smallsets` to users, several resources have been developed. This includes example data and scripts, R help pages, a user guide, a cheatsheet, and the `smallsets` website. Each is discussed briefly below.

Example data and scripts

The synthetic dataset s_data, used throughout this chapter for illustration and detailed in Appendix A, is included in `smallsets`. It is saved in the RData format (extension `.rda`), assigned to the name `s_data`, and lazy-loaded.[5] There are six example preprocessing scripts (for the `s_data` object) included in `smallsets`. Three of the scripts produce the same Smallset Timeline but are of different file types (`.R`, `.Rmd`, and `.py`). The other three demonstrate variations in the use and placement of structured comments.

The example dataset and scripts are used to generate a series of quick start examples, that can be run immediately upon installing `smallsets`. Listing 5.2 shows two quick start examples. Because the example preprocessing scripts are located within the package itself, they must be called with `system.file()` in the `code` argument.[6]

```
library(smallsets)

# quick start example for an R file
Smallset_Timeline(data = s_data,
                  code = system.file("s_data_preprocess.R",
                                     package = "smallsets"))

# quick start example for an R Markdown file
Smallset_Timeline(data = s_data,
                  code = system.file("s_data_preprocess.Rmd",
                                     package = "smallsets"))
```

Listing 5.2: Two quick start examples for `smallsets`, which use example data and preprocessing files included in `smallsets`. The output from each is a Smallset Timeline.

[5] A user does not need to run the `data()` command to load it.

[6] The `system.file()` command [Wickham et al., 2022b] provides the correct pathway to the example preprocessing file, regardless of where the package is installed on a user's machine.

Help pages

Each external (i.e., public-facing) `smallsets` command[7] has an R help page, with documentation. These pages are bundled with the software and accessible in R through the `help()` command or the `?` operator (e.g., `help(Smallset_Timeline)` or `?Smallset_Timeline`). R help pages have seven sections: *Description*, *Usage*, *Arguments*, *Details*, *Value*, *See Also*, and *Examples*. Together, these sections provide a comprehensive overview of a function, from its purpose to its syntax to its output. All help pages for `smallsets` were generated with the `roxygen2` package [Wickham et al., 2020]. Specifically, the documentation was added directly to the function source code files in `roxygen2` comments, which generate the R documentation files (`.Rd`) for the help pages.

User guide

In addition to R help pages, there is a `smallsets` user guide. It is available as a vignette within the `smallsets` installation and can be accessed with `vignette("smallsets")`. It is also available on the `smallsets` website.[8] The user guide is written in R Markdown and rendered as an interactive HTML document. The document is a blend of text, code snippets, and example Smallset Timelines. It contains approximately 1400 words, over 15 example code snippets, and seven Smallset Timelines. The topics covered include supported workflows, structured comments, Smallset selection, and figure customisation, to name a few.

Cheatsheet

Another popular user resource for R packages is cheatsheets [Posit Software, PBC, 2024]. Figure 5.4 is the `smallsets` cheatsheet, available on the `smallsets` website. As per cheatsheet style guidelines,[9] it is a multi-column one-page PDF file. The `smallsets` cheatsheet consists of seven different sections, featuring key components of the software, for quick reference. HTML cheatsheets were recently introduced; future work will involve adapting the PDF in Figure 5.4 to this new format.

[7] This includes `Smallset_Timeline()`, `sets_sizing()`, `sets_spacing()`, and `sets_labelling()`.
[8] `https://lydialucchesi.github.io/smallsets/articles/smallsets.html`
[9] `https://github.com/rstudio/cheatsheets`

Smallset Timelines with smallsets: : **CHEAT SHEET**

smallsets is a tool for visually documenting and communicating data preprocessing decisions. It builds a Smallset Timeline figure [3] based on preprocessing code in an R, R Markdown, Python, or Jupyter Notebook file. Users must first add structured comments, with building instructions, to the preprocessing code.

Structured comments

Structure

`# smallsets snap` *`place data`* `caption[`*`text`*`]caption`

Three place argument options:

1. The line of code after which to take the snapshot (e.g., 7 means take after line 7)
2. A plus sign and the number of lines of code after which to take the snapshot (e.g., +2 means take after the next two lines)
3. No argument means take the snapshot directly after the comment and before the next line of code

Main functions

`Smallset_Timeline(data, code, ...)`
builds a Smallset Timeline

`sets_sizing()`
for adjusting sizing parameters, including column names, caption text, snapshot data, and legend items

`sets_spacing()`
for adjusting spacing parameters, including caption space, column name rotation, and number of figure rows

`sets_labelling()`
for adjusting the colours of the column names and snapshot data

Demo dataset and code

The smallsets package comes with example data and preprocessing code, which are used to illustrate how the package works, such as in the next section.

Synthetic dataset

`s_data` 100 observations and eight variables (C1-C8)

Preprocessing scripts

`s_data_preprocess.R`	preprocessing scenario in R
`s_data_preprocess.Rmd`	preprocessing scenario in R Markdown
`s_data_preprocess.py`	preprocessing scenario in Python
`s_data_preprocess_block.R`	shows alternative comment placement
`s_data_preprocess_4.R`	includes additional snapshot
`s_data_preprocess_resume.R`	includes resume marker

Steps to build a Smallset Timeline

The demo dataset `s_data` and preprocessing code `s_data_preprocess.R` are used to illustrate the process.

Step 1

Add structured comments to the preprocessing code in your script, specifying snapshot points and captions.

File: `s_data_preprocess.R`

```
# smallsets snap s_data caption[Remove rows where C2 is FALSE.]caption
s_data <- s_data[s_data$C2 == TRUE,]

# smallsets snap +2 s_data caption[Replace missing values in C6 and C8 with
# column means. Drop C7 because there are too many missing values.]caption
s_data$C6[is.na(s_data$C6)] <- mean(s_data$C6, na.rm = TRUE)
s_data$C8[is.na(s_data$C8)] <- mean(s_data$C8, na.rm = TRUE)
s_data$C7 <- NULL

# smallsets snap +1 s_data caption[Create a new column, C9, by summing C3 and
# C4.]caption
s_data$C9 <- s_data$C3 + s_data$C4
```

Step 2

Run the main smallsets command to build a Smallset Timeline for your dataset and preprocessing code.

```
Smallset_Timeline(data=s_data, code="s_data_preprocess.R")
```

Smallset selection

To select the small number of rows from the original dataset used in the visualisation, you can use one of three selection methods available in the `Smallset_Timeline()` command.

`rowCount` number of Smallset rows (5-15)

`rowSelect` Smallset row selection method

- `= 1` → *coverage* model (Gurobi required)
- `= 2` → *coverage* + *variety* model (Gurobi required)
- `= NULL` → random sampling

Warning. This method has long runtimes for large datasets. See [2] for workarounds.

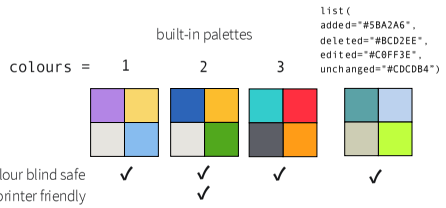
Customisation

To customise the information in a Smallset Timeline and its appearance, you can set different parameters in the `Smallset_Timeline()` command. See [1] for the complete list of parameters.

`printedData = TRUE`
show the data values in Smallset snapshots

`ghostData = TRUE`
plot blank rows/columns after data deletions

`missingDataTints = TRUE`
use colour tints to highlight missing data



References

[1] **CRAN reference manual**
cran.r-project.org/web/packages/smallsets/smallsets.pdf

[2] **smallsets User Guide**
lydialucchesi.github.io/smallsets/articles/smallsets.html
included in the package : `vignette("smallsets")`

[3] **Smallset Timelines: A Visual Representation of Data Preprocessing Decisions**
paper providing a detailed discussion of Smallset Timelines, the Smallset selection optimisation problems, and two case studies with example Smallset Timelines
doi.org/10.1145/3531146.3533175

smallsets v2.0.0 – CRAN.R-project.org/package=smallsets – CC BY SA Lydia R. Lucchesi, Petra M. Kuhnert, Jenny L. Davis, Lexing Xie – December 2023

Figure 5.4: The `smallsets` cheatsheet, available on the `smallsets` website.

Website

There is also a `smallsets` website that centralises the resources discussed above.[10] It is built with `pkgdown` [Wickham et al., 2022a], which turns R package documentation into a website, and hosted through GitHub pages. The website makes information about `smallsets` readily available, without having to install `smallsets` in R. Additionally, it can house resources not permitted in the software installation itself, e.g., the cheatsheet (Figure 5.4).

5.3 Package architecture

Section 5.2 focused on the steps a `smallsets` user would follow to build a Smallset Timeline. This section focuses on what happens on the backend of `smallsets` to produce a Smallset Timeline, based on user inputs. Key software dependencies are first outlined in Section 5.3.1, before the backend process is detailed in Section 5.3.2. Unit testing is covered in Section 5.3.3.

5.3.1 Dependencies

The `smallsets` R package depends on R version ≥3.5.0 [R Core Team, 2023]. It also has ten direct R package dependencies.[11] These packages must be installed alongside `smallsets`, for it to work. They are as follows: `callr` [Csárdi and Chang, 2022], `colorspace` [Zeileis et al., 2020], `flextable` [Gohel and Skintzos, 2023], `ggplot2` [Wickham, 2016], `ggtext` [Wilke and Wiernik, 2022], `knitr` [Xie, 2023], `patchwork` [Pedersen, 2023], `plotrix` [Lemon, 2006], `reticulate` [Ushey et al., 2023], and `rmarkdown` [Allaire et al., 2023]. It is worth noting that each of those ten dependencies has their own set of dependencies, which `smallsets` then indirectly relies on. Figure 5.5 visualises the tree network of direct and indirect package dependencies for `smallsets`.

One dependency *not* visualised in Figure 5.5 is the `gurobi` R package [Gurobi Optimization, LLC, 2021]. This package is required when using either the *coverage* model or *coverage + variety* model for Smallset selection (see Section 4.3.1). The `gurobi` package is not available from a central repository. Rather, it is located in a user's local Gurobi installation and must be installed from there. Thus, `gurobi` is included under the *Suggests* category in the

[10] `https://lydialucchesi.github.io/smallsets`
[11] They are referred to as *Imports* in the package's *DESCRIPTION* metadata file.

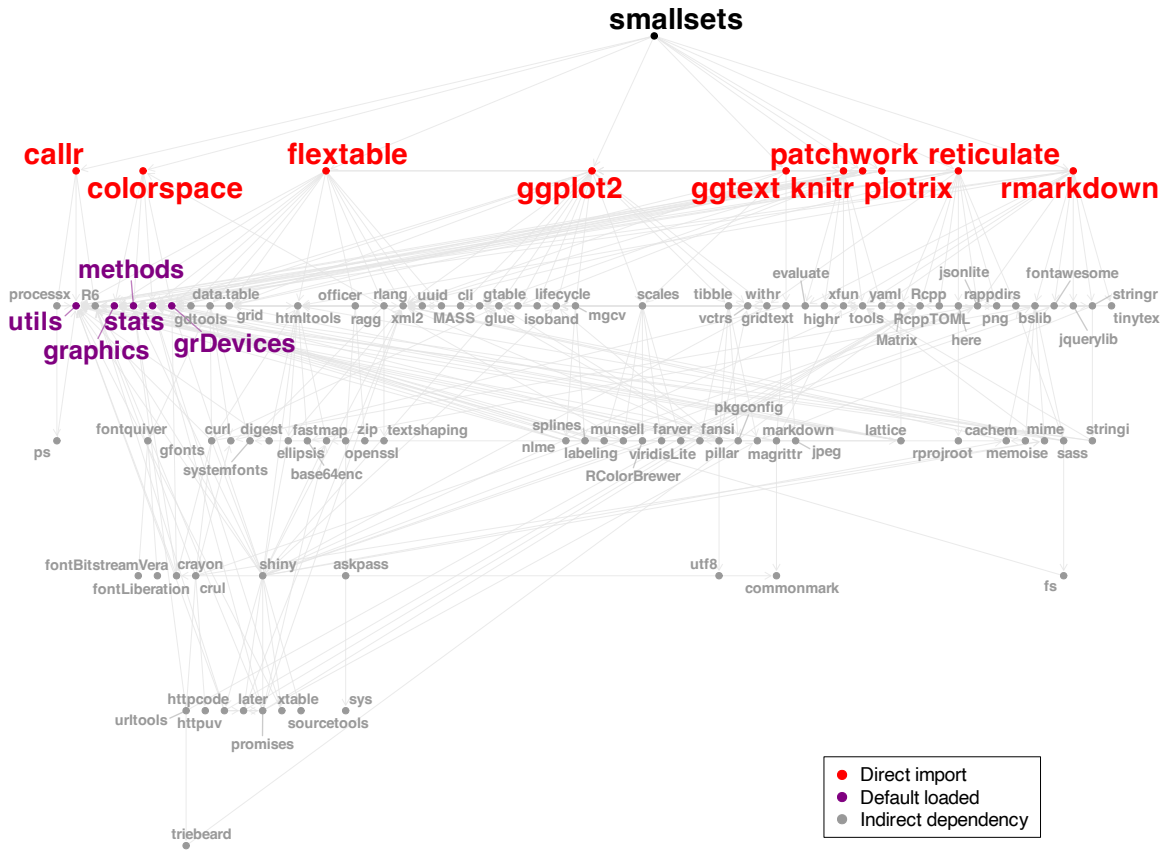


Figure 5.5: Tree network of `smallsets` dependencies, distinguishing between R packages that are direct imports in `smallsets`, default-loaded in an R session, and indirect dependencies of `smallsets`. Graph based on data from a dependency report produced with `pkgnet` [Burns et al., 2021].

package's *DESCRIPTION* metadata file, which means it is only required for optional functionality in `smallsets`. If a user attempts to use one of the optimisation models without the Gurobi R package installed, they are prompted with installation instructions.

5.3.2 Internal structure

As outlined in Section 5.2, `smallsets` users must supply two inputs to the main user-facing command, `Smallset_Timelime()`, to build a Smallset Timeline. This includes an un-preprocessed dataset and preprocessing script with inserted structured comments. Following is an overview of the processing that occurs behind the scenes to transform these user

inputs into a Smallset Timeline. This overview does not detail every aspect of the `smallsets` source code but rather the most important parts. The processing can be divided into four key steps: 1) selecting the Smallset, 2) taking snapshots, 3) detecting data changes, and 4) building the plot. These steps are summarised in Table 5.3 and rely on a series of internal `smallsets` functions, visualised in Figure 5.6.

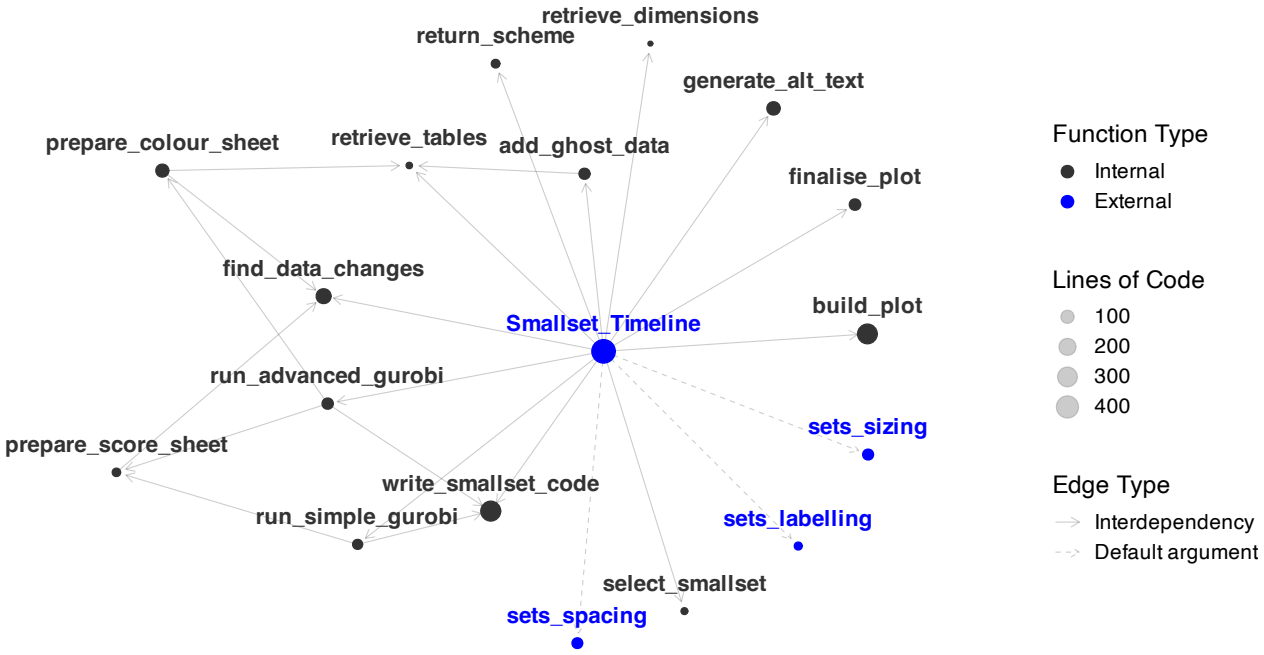


Figure 5.6: Network graph of the `smallsets` functions, showing interdependencies among both internal and external functions. Dashed edges pointing out from the `Smallset_Timeline()` node highlight three external functions serving as default arguments for three of the arguments in `Smallset_Timeline()` (see Table 5.1). The lines of code value does not include lines of `roxygen2` comments for function documentation.

Step 1: Select the Smallset

The first key step is to select a set of Smallset rows, one of three ways, based on the user's method of choice: random sampling, the *coverage* model, or the *coverage + variety* model (see Section 4.3.1). The random sampling approach is straightforward. Selection with an optimisation model is more involved. In Figure 5.6, we can see on the left side of the network a web of internal functions related to the optimisation models. The models are solved with

1. Select the Smallset	2. Take snapshots	3. Detect data changes	4. Build the plot
The Smallset rows are selected with the specified method. If the *coverage* model is specified, a *coverage indicator matrix* is first generated (see Figure 4.7). If the *coverage + variety* model is specified, a *coverage indicator matrix* and *visual appearance matrix* are first generated (see Figure 4.7).	Based on the structured comments, the preprocessing code is supplemented with snapshots of the Smallset rows (data frames are appended to a list object) and turned into a function (see Listing 5.3). The dataset is passed to the function as an argument, and a list of snapshots is returned.	Two adjacent snapshots are compared at a time (i.e., one vs. two, two vs. three, etc.), to search for differences between them. In the first of the two snapshots, data values are labelled as deleted, where applicable. In the second, data values are labelled as added or edited, where applicable.	Each element in the snapshot list is transformed into a snapshot plot. The snapshot captions are added to the plots. These individual snapshot plots are then assembled into a single plot object: the Smallset Timeline. The alt text template is populated with information about the Smallset Timeline.

Table 5.3: The four key backend steps—behind the main `Smallset_Timeline()` command—summarised. See Section 5.3.2 for a full discussion.

either `run_simple_gurobi()` or `run_advanced_gurobi()`, which call other functions to prepare data representations required by the models. For example, `run_simple_gurobi()` calls `prepare_score_sheet()` to prepare the *coverage indicator matrix* required by the *coverage* model (see Figure 4.7). Once selected, the Smallset is represented by a vector of row names, if the preprocessing code is in R, and by a vector of indices, if the preprocessing code is in Python. These values can be used to access the Smallset within the full dataset, where each row is identified by its unique row name or index.

Step 2: Take snapshots

The user's preprocessing script, passed to the `code` argument, is read into R as a character vector. Each line in the file becomes an element in the vector. The elements are strings of R or Python syntax (depending on the user's preprocessing code). For R and Python files (`.R` and `.py`, respectively), the base R command `readLines()` can be used to read in the code. R Markdown and Jupyter Notebook files (`.Rmd` and `.ipynb`, respectively) require some additional formatting during import, which is achieved with the help of functions in `knitr` [Xie, 2023], `callr` [Csárdi and Chang, 2022], and `rmarkdown` [Allaire et al., 2023].

Snapshots are inserted as new elements in the vector. The snapshots are strings of R/Python syntax, appending a Smallset data frame[12] to a list object (see Listing 5.3). Strings

[12] A data frame is a basic two-dimensional data structure in R.

containing the R/Python syntax for writing a function (e.g., "`apply_code <- function() {`") are also inserted at the beginning and end of the vector. The vector is written to a temporary R/Python file (`.R`/`.py`, respectively), at which point the strings become working code again. The function in the temporary file can then be sourced into R,[13] and the `data` object is passed to it, to obtain the list of Smallset snapshots. Listing 5.3 shows the internal snapshot-taking function, for the s_data example.

The function called `write_smallset_code()` is responsible for generating the snapshot-taking function. As depicted in Figure 5.6, `write_smallset_code()` is also called when using an optimisation model for Smallset selection. This is because snapshots of the dataset are required to prepare the input matrices for those models. Therefore, `write_smallset_code()` is designed to take snapshots of either the entire dataset or the Smallset rows, so that it can be used for either task.

```
snapshots <- list()
apply_code <- function(s_data) {
  # smallsets snap s_data caption[Remove rows where C2 is FALSE.]caption
  snapshots[[1]] <- s_data[(row.names(s_data) %in% c("2", "47", "54", "75", "92")),]
  s_data <- s_data[s_data$C2 == TRUE,]

  # smallsets snap +2 s_data caption[Replace missing values in C6 and
  # C8 with column means. Drop C7 because there are too many missing
  # values.]caption
  s_data$C6[is.na(s_data$C6)] <- mean(s_data$C6, na.rm = TRUE)
  s_data$C8[is.na(s_data$C8)] <- mean(s_data$C8, na.rm = TRUE)
  snapshots[[2]] <- s_data[(row.names(s_data) %in% c("2", "47", "54", "75", "92")),]
  s_data$C7 <- NULL

  # smallsets snap +1 s_data caption[Create a new column, C9, by summing
  # C3 and C4.]caption
  s_data$C9 <- s_data$C3 + s_data$C4
  snapshots[[3]] <- s_data[(row.names(s_data) %in% c("2", "47", "54", "75", "92")),]
  return(snapshots)
}
```

Listing 5.3: The internal snapshot-taking function generated by `smallsets` for the s_data example, where *2*, *47*, *54*, *75*, and *92* refer to the Smallset row names. Purple lines of code indicate those added by `smallsets`, to the preprocessing code supplied by the user.

Step 3: Detect data changes

The third step is detecting data changes between snapshots. Two snapshots are compared at a time, starting with the first and second snapshot, then the second and third snapshot,

[13] To source a Python function in R, the `reticulate` package [Ushey et al., 2023] is used.

and so on. Prior to the comparison, the snapshot data frames are converted to `flextable` objects [Gohel and Skintzos, 2023]. The `flextable` object is convenient for this step, as colours can be assigned to data points in a data frame. Specifically, colour information is stored as a separate data frame of hex colour codes within the `flextable`. This `flextable` configuration is used to assign the colours highlighting data changes, to affected data points.

The comparison between two snapshots is based on row and column names. Rows and columns are checked for data additions and deletions. Individual data points are checked for edits.[14] During each comparison, the data frame of hex colour codes is updated, if a difference is detected. Note that—unlike a data addition or a data edit—a data deletion is highlighted in the first of the two compared snapshots, where there is still a row/column in the Smallset *to* highlight with colour.[15]

Step 4: Build the plot

In the fourth step, any column passed to `ignoreCols` (see Table 5.1) is removed from the `flextable` objects. The data frames of hex colour codes are extracted from the `flextable` objects and transformed into the snapshot plots. The snapshot plots are constructed individually as tile plots, using `geom_tile()` from `ggplot2` [Wickham, 2016]. Captions are added to the snapshot plots, using `geom_textbox()` from `ggtext` [Wilke and Wiernik, 2022]. Using `patchwork` [Pedersen, 2023], the individual snapshot plots are assembled into a single plot object: the Smallset Timeline.

5.3.3 Unit testing

The `smallsets` development workflow includes unit testing, to better detect when changes to software dependencies or source code introduce bugs or errors. Thirty-one tests were written for `smallsets` using `testthat` [Wickham, 2011], an R package providing support for unit testing in R software development. The tests compare an output/action, from some part of

[14] Presently, `smallsets` will not detect changes in variable types (e.g., a character variable is changed to a factor variable), if the change in variable type does not change the data values. In future versions of `smallsets`, consideration can be given to showing changes in variable types.

[15] Future versions of `smallsets` could be altered to instead highlight data deletions in the second of the two compared snapshots, by inserting previously deleted rows/columns back into the Smallset table for one additional snapshot, such that they could be visually highlighted with colour. This approach might be more intuitive for Timeline creators and readers. Ultimately, though, the best approach to showing data deletions can be determined by data/feedback collected through a `smallsets` user study.

the `smallsets` source code, with an expected value/outcome. The tests are included in the `smallsets` codebase on GitHub and run automatically each time the package is compiled locally. They are diverse and span all four steps in Table 5.3. For example, one test checks that a data deletion is accurately detected in a snapshot, while another checks that the R Markdown quick start example produces output, without an error. In total, the test coverage is 80.96%. This value was calculated with the `covr` package [Hester, 2023] and means that approximately 80% of the source code is executed across the 31 tests.

5.4 The evolution of `smallsets`

Software development for `smallsets` began in 2020. Since then, `smallsets` has continued to evolve, undergoing significant revision at times, especially in regards to the user interface and workflow. This section discusses these revisions, using three major releases of the `smallsets` software—introduced in Section 5.4.1—as reference points in time. The discussion is not comprehensive of all changes to the `smallsets` source code. Commits to the `smallsets` GitHub repository do, however, provide a comprehensive log of `smallsets` changes.[16]

5.4.1 Development history

To date, there have been three major releases of the `smallsets` software. These releases are summarised in Table 5.4. The first release, v0.0.1.9000, was in November 2022 and is referred to as *R1-V0*. The second release, v1.0.0, was in February 2023 and is referred to as *R2-V1*. The third release, v2.0.0, was in December 2023 and is referred to as *R3-V2*. Each release included substantial revision to the `smallsets` software. *R3-V2* is the version used to build all Smallset Timelines in this thesis.

As indicated in Table 5.4, *R2-V1* and *R3-V2* are published on the Comprehensive R Archive Network (CRAN), a central repository for R packages. To be published on CRAN, a package must adhere to all CRAN policies;[17] compile without any warning messages or errors; and follow best practices for R package development.[18] Submissions must be reviewed and approved by a member of the CRAN team, prior to acceptance to the CRAN repository. Once

[16] `https://github.com/lydialucchesi/smallsets`
[17] `https://cran.r-project.org/web/packages/policies.html`
[18] Best practices for R package development are outlined in Wickham and Bryan [2023].

Label	Version	Date	Code URL
R1-V0	v0.0.1.9000	November 2022	[GitHub]
R2-V1	v1.0.0	February 2023	[CRAN] & [GitHub]
R3-V2	v2.0.0	December 2023	[CRAN] & [GitHub]

Table 5.4: Information about the three major releases of the `smallsets` software (see Section 5.4.1). Section 5.4 uses labels *R1-V0*, *R2-V1*, and *R3-V2* to refer to the releases.

on CRAN, packages are tested on thirteen different *check flavors* (different combinations of R versions, operating systems, compilers, etc.) on a regular basis. In short, R packages must meet and maintain certain standards to be hosted on CRAN, ensuring a certain standard of quality for users and easier installation with the `install.packages()` command.

5.4.2 Streamlining the user interface and workflow

Section 5.2 described the user workflow: 1) insert structured comments and 2) run the `Smallset_Timeline()` command. However, this two-step workflow in *R3-V2* looks significantly different than the workflow in *R1-V0*. With each new release of `smallsets`, the aim has been to improve the user interface and workflow, to make the tool more intuitive and streamlined to use. Following is an outline of how the user interface and workflow have evolved across the three major software releases: *R1-V0*, *R2-V1*, and *R3-V2* (see Table 5.4).

The first release, *R1-V0*, had a four-step workflow that required users to 1) add structured comments, 2) run a command called `prepare_smallset()`, 3) complete an R Markdown caption template, and 4) run a command called `create_timeline()`. As shown in the top script in Figure 5.7, the structured comments in *R1-V0* centred around a series of instructions—*start*, *snap*, and *end*—which were associated with specific snapshot locations. The *start* and *end* comments took the snapshot in place, while *snap* comments always took the snapshot after the subsequent line of code. Users also did not specify the snapshot caption in the structured comments. Instead, users ran `prepare_smallset()`, which generated the snapshots and a custom R Markdown caption template based on the inserted structured comments. Users then filled out the caption template and passed it—along with the other output from `prepare_smallset()`—to the `create_timeline()` command, which assembled the Smallset Timeline.

The main issues with this workflow revolved around the R Markdown caption template.

R1-V0 (November 2022)

```
# start smallset s_data
s_data <- s_data[s_data$C2 == TRUE, ]

s_data$C6[is.na(s_data$C6)] <- mean(s_data$C6, na.rm = TRUE)
# snap s_data
s_data$C8[is.na(s_data$C8)] <- mean(s_data$C8, na.rm = TRUE)
s_data$C7 <- NULL

s_data$C9 <- s_data$C3 + s_data$C4
# end smallset s_data
```

R2-V1 (February 2023)

```
# smallsets start s_data caption[Remove rows where C2
# is FALSE.]caption
s_data <- s_data[s_data$C2 == TRUE, ]

s_data$C6[is.na(s_data$C6)] <- mean(s_data$C6, na.rm = TRUE)
# smallsets snap s_data caption[Replace missing values in C6 and
# C8 with column means. Drop C7 because there are too many
# missing values.]caption
s_data$C8[is.na(s_data$C8)] <- mean(s_data$C8, na.rm = TRUE)
s_data$C7 <- NULL

s_data$C9 <- s_data$C3 + s_data$C4
# smallsets end s_data caption[Create a new column,
# C9, by summing C3 and C4.]caption
```

R3-V2 (December 2023)

```
# smallsets snap s_data caption[Remove rows where C2 is FALSE.]caption
s_data <- s_data[s_data$C2 == TRUE, ]

# smallsets snap +2 s_data caption[Replace missing values in C6 and
# C8 with column means. Drop C7 because there are too many missing
# values.]caption
s_data$C6[is.na(s_data$C6)] <- mean(s_data$C6, na.rm = TRUE)
s_data$C8[is.na(s_data$C8)] <- mean(s_data$C8, na.rm = TRUE)
s_data$C7 <- NULL

# smallsets snap +1 s_data caption[Create a new column, C9, by summing
# C3 and C4.]caption
s_data$C9 <- s_data$C3 + s_data$C4

```

Figure 5.7: An R preprocessing script for s_data shown three times (the code in black is constant), to illustrate how `smallsets` structured comments changed with each major release of the `smallsets` software (*R1-V0*, *R2-V1*, and *R3-V2*). See Section 5.4.2 for a discussion.

Whenever a user changed the number or location of the structured comments, a new caption template had to be generated and filled out. This back and forth between preprocessing script and caption template proved tedious and clumsy. Another problem with the caption template was that `smallsets` had to write a file locally to a user's machine, which is prohibited for R CRAN packages. To resolve these issues, the caption template was dropped from the workflow entirely, and captions were instead integrated into the structured comments. Users could then directly document their code while also scaffolding the Smallset Timeline. This update is visualised in the middle script in Figure 5.7. Once captions were integrated into the structured comments, the commands `prepare_smallset()` and `create_timeline()` were replaced by a single command: `Smallset_Timeline()`. These changes to the user interface and workflow appeared in *R2-V1*.

One aspect of the structured comments that did not change between *R1-V0* and *R2-V1* were the instructions—*start*, *snap*, and *end*—with set snapshot locations. We can see that, in the middle script in Figure 5.7, the second snapshot (line 6) had to be placed in the middle of the preprocessing step that was being described by the caption, to get the snapshot in the preferred location. In other words, the old instruction setup combined with the new caption setup resulted in suboptimal comment/documentation placement within a preprocessing script. Thus, *R3-V2* dropped instructions and added the *snap-place* argument to resolve this issue (see Section 5.2.1). Now, structured comments can be placed in a more intuitive and sensible way throughout a preprocessing script. This most recent update is visualised in the bottom script in Figure 5.7.

5.4.3 Reducing package dependencies

The first `smallsets` software release, *R1-V0*, had more than twice the number of direct dependencies, at 22 packages. Having that many dependencies made `smallsets` more vulnerable to bugs and errors introduced through changes to the dependencies. Thus, a primary focus for the second release, *R2-V1*, was reducing this number to increase the stability of `smallsets`. For *R2-V1*, non-base R sections were re-coded in base R wherever (practically) possible, resulting in substantial dependency reductions. *R2-V1* had seven direct dependencies. This increased to ten in *R3-V2* with the introduction of new `smallsets` features.

5.4.4 Comments on row tracking in R

As noted Section 5.3.2, `smallsets` uses row names in R and indices in Python to track Smallset rows.[19] In *R1-V0*, users could choose whether to have `smallsets` run the preprocessing code (i.e., the internal snapshot-taking function) on the Smallset rows only or the full dataset. Because the Smallset Timeline is designed to demonstrate the preprocessing steps using a "small set" of a data, applying the preprocessing code to just a "small set" of data seemed to hold, conceptually. It was also potentially a way to save users runtime for large datasets.

Yet, running on the Smallset only can lead to inaccurate representations of preprocessing, as certain preprocessing steps may rely on information from the full dataset. For instance, in the s_data example, missing values in column *C6* are imputed with the *C6* mean. If the code is run on the Smallset only, the mean calculation for *C6* will be incorrect. As a result, the Smallset Timeline will be misleading, if data are printed in the snapshots. Imagine a scenario in which data preprocessing steps are dependent on each other; running on the Smallset only could cause a chain reaction of inaccuracies in the visualisation. Thus, that option was removed entirely in subsequent software releases (*R2-V1* and *R3-V2*). Now, the code is always run on the full dataset, with snapshots of the Smallset extracted from it.

It should also be noted that, for `smallsets` to work properly for R preprocessing code, the user's preprocessing code cannot overwrite the row names. That will derail Smallset tracking. This presents some challenges when trying to make `smallsets` functional for preprocessing workflows involving multiple datasets. For instance, the `merge()` command in R automatically rewrites row names. Imagine we have datasets A and B. We want to visualise the preprocessing of A, which involves merging on B (i.e., `merge(A, B)`). Dataset A has row names *1*, *3*, *4*, and *5*. But after the merge, it will have row names *1*, *2*, *3*, and *4*, because `merge()` re-sequences them. If row name *3* is being tracked, then the second row in the dataset will be tracked until the merge happens, at which point the third row in the dataset will be tracked.

This raises the following question: Is there a better way in R to track Smallset rows, than by row names? The possibility of adding a `smallsets` ID column to a user's dataset(s), which could be used for tracking rows, was considered. However, this ID column could interfere with a user's preprocessing code, resulting in inaccurate Smallset Timelines or errors. For

[19] This includes taking Smallset snapshots and detecting data changes between them.

example, say a user has a dataset called *mydata*. In their preprocessing code, they have the following line of R code: `mydata$new <- rowSums(mydata)`. That code creates a column in *mydata* called *new*, by summing across **all** of the values in a row. Therefore, if a numeric `smallsets` ID column has been added to *mydata*, those ID numbers will be included in the row sums for column *new*. If the `smallsets` ID column contained characters instead of numbers (e.g., *"1"* instead of *1*), it would break the code altogether.

Because the R programming language does not have the equivalent of a "primary key" (i.e., a value that uniquely identifies a row and is immutable), this remains an open problem. One possibility—for when overwriting row names is inevitable—is to require users to create their own identification column in the dataset, which `smallsets` uses instead of row names. For cases in which the row names have been rewritten because the dataset has undergone major restructuring (e.g., aggregation across rows), it may make the most sense to re-sample the Smallset at that point in time and make note of this re-sampling somewhere in the Smallset Timeline, for Timeline readers. Exploring these possibilities is future work.

5.5 Summary

To summarise, `smallsets` is an R CRAN package that transforms R or Python preprocessing code—from an R, R Markdown, Python, or Jupyter Notebook file—into a Smallset Timeline. There have been three major releases of the `smallsets` software, each with revisions to the user workflow and interface. The current workflow involves two steps. First, users must add structured comments, with snapshot instructions, to their preprocessing code. Second, users must pass their un-preprocessed dataset and commented preprocessing code as inputs to the `Smallset_Timeline()` command. On the backend, `smallsets` executes a four-step process to build the Smallset Timeline. This includes selecting the Smallset, taking snapshots, detecting data changes, and building the plot.

The next chapter demonstrates the `smallsets` tool in action, in four data preprocessing case studies. The `smallsets` discussion continues in Chapter 7, which presents findings from a focus group study that solicited feedback from data practitioners on the `smallsets` software tool.

Chapter 6

`smallsets` in Action: Preprocessing Case Studies

This chapter presents four data preprocessing case studies. These case studies highlight the importance of communicating data preprocessing decisions in real-world scenarios. They also demonstrate use of the Smallset Timeline and `smallsets` software tool—proposed in Chapters 4 and 5, respectively—as a simple and effective means for doing so. Each case study presents one or more Smallset Timelines, produced with `smallsets`. These Smallset Timelines vary in terms of appearance (e.g., colours, layout, enrichment features, etc.), to illustrate the different customisation options available to `smallsets` users.

The four case studies represent a diverse range of research areas. Section 6.1 features datasets from the NASA Metrics Data Program (MDP), used widely for software defect detection research. Section 6.2 focuses on eBird citizen science data [Sullivan et al., 2009] and its use in ecological modelling. Section 6.3 examines machine learning datasets—retrieved with the `folktables` tool [Ding et al., 2021]—containing American Community Survey (ACS) data. Section 6.4 is on home loan approval data, available through the Home Mortgage Disclosure Act (HMDA) as an open resource for auditing lending institutions. The latter two case studies on ACS and HMDA data include analyses that quantify the downstream effects of data preprocessing decisions on analytical outcomes.

6.1 Case study 1: Predicting bugs with NASA MDP data

In the early 2000s, the NASA Metrics Data Program (MDP) released 13 datasets for software defect detection, which involves developing algorithms to predict bugs in source code. The MDP datasets contained information about software modules in languages like C, C++, and perl.[1] After their release, the datasets became a widely-used resource in the field, with approximately 30% of software defect prediction studies published between 2000-2010 using them [Hall et al., 2011].

Like many real-world datasets, the MDP datasets required data preprocessing. Specifically, there were missing, erroneous, extraneous, and duplicate data to address. In 2011, Gray et al. raised concerns about how these issues were being dealt with in the defect detection literature. In some cases, the concern was that these issues were *not* being dealt with. Over the next several years, researchers reviewed the NASA MDP datasets, their data quality, and related preprocessing practices.

This case study first provides an overview of the MDP preprocessing literature. Then, two example Smallset Timelines are built to visualise the MDP data preprocessing steps recommended in Gray et al. [2011]. The first Smallset Timeline provides a high-level summary of the steps, to support general comprehension and evaluation of them. The second Smallset Timeline provides a more detailed account of the preprocessing steps, to support replication. In accordance with Table 4.1, there are differences in how the steps are presented in the Smallset Timeline, given the different communication goals.

6.1.1 The MDP preprocessing literature

The MDP datasets are used as a case study in large part because of the papers focused specifically on assessing MDP data preprocessing practices [Gray et al., 2011, 2012, Petrić et al., 2016, Shepperd et al., 2013]. To the best of my knowledge, the first of these papers was written by Gray et al. in 2011. In that paper, the main concern was related to the issue of duplicate data occurring in the testing and training sets, which would mean the model was

[1]This included measures like the number of lines of code, comments, and operations; various software metrics, e.g., cyclomatic complexity, which refers to the number of linearly independent paths in a module [McCabe, 1976]; and whether or not the module was defective.

not tested on unseen data only. In five of the 13 MDP datasets, over 20% of the instances are duplicates of other instances. Gray et al. [2011] proposed a five-step data preprocessing approach, which included removing duplicates from the dataset.

In 2012, the same authors published another paper focused on duplicate data, noting that "the impression given from the literature is that many defect prediction researchers using this data have not been aware of this issue" [Gray et al., 2012, p. 557]. They included an experiment with simulated data, showing that the accuracy of a random forest classifier [Breiman, 2001] (a technique used with MDP datasets) increases as the proportion of duplicate data between the testing and training sets increases. They also refined the five-step preprocessing approach from before, advising that duplicates appearing in both the training and testing set be removed from the training set only, to preserve as much data as possible.

Shortly after, Shepperd et al. [2013] published an MDP data quality assessment. It compared the original MDP datasets with alternative versions housed on the PROMISE Software Engineering Repository [Sayyad Shirabad and Menzies, 2005], highlighting differences between them.[2] They also proposed an alternative preprocessing approach to Gray et al. [2012]. They shuffled the order of the preprocessing steps, citing an ordering effect, and included an additional 15 data integrity checks,[3] to the previous three. They noted that "authors (including ourselves) have not been in the habit of providing complete information regarding preprocessing of data" [p. 1213] and called for more comprehensive documentation.

With new integrity checks in place, Ghotra et al. [2015] explored the influential finding in Lessmann et al. [2008] that for ten MDP datasets the prediction performance was largely similar across different classification techniques. Ghotra et al. [2015] first replicated the finding in Lessmann et al. [2008], without any integrity checks applied. The analysis was then repeated but with all 18 integrity checks applied, which resulted in datasets losing between 12.9% and 87.1% of instances. Now there *were* notable differences in prediction performance across classifiers, highlighting the impact of the integrity checks. In 2016, Petrić et al. proposed two additional integrity checks, for a total of 20.

These papers illustrate that data preprocessing represents an important part of working with the MDP datasets, or at least ought to. Section 6.1.2 presents two example Smallset

[2]For example, for the *CM1* dataset, the original version had 161 missing values, while the PROMISE version had zero; yet, for cases with conflicting feature values, the PROMISE version had three, while the original only had two [Shepperd et al., 2013].

[3]The integrity checks are tests of logic confirming the data make sense, given what the different values/-metrics mean.

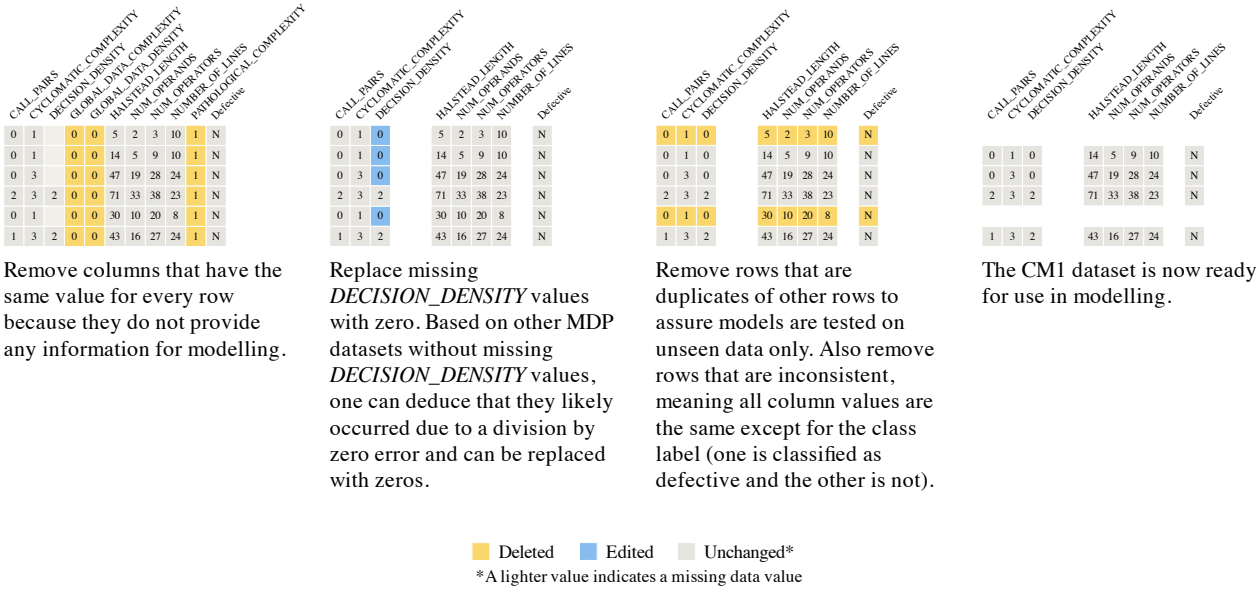


Figure 6.1: Smallset Timeline for MDP CM1 dataset preprocessed according to Gray et al. [2011]. Smallset selected using the *coverage* algorithm. See Section 6.1.2 for a discussion and Appendix B.1 for the preprocessing script and `smallsets` code for this figure.

Timelines that document the preprocessing of MDP dataset *CM1*,[4] using the steps recommended in Gray et al. [2011]. CM1 contains information about modules in the C programming language. It has 505 instances, 40 features, and a column of class labels (defective or not).

6.1.2 Smallset Timelines for dataset CM1

Gray et al. [2011] propose a five-step data preprocessing approach for the MDP datasets. The steps are as follows: 1) *removal of constant attributes*, 2) *removal of repeated attributes*, 3) *replacement of missing values*, 4) *enforce integrity with domain specific expertise*, and 5) *removal of repeated and inconsistent instances*. I translated these five steps into the R programming language, to build Smallset Timelines for CM1. The code can be found in Appendices B.1 and B.2.[5]

Figure 6.1 contains a Smallset Timeline for CM1, designed to support general comprehension and evaluation of the preprocessing pipeline. It uses a Smallset with six rows and

[4] A copy of the original un-preprocessed CM1 dataset was obtained from Tantithamthavorn [2016].

[5] Note that the code represents an *interpretation* of the text descriptions of the preprocessing steps in Gray et al. [2011].

four snapshots. Only preprocessing steps 1, 3, and 5 are discussed, as steps 2 and 4 do not affect the dataset. The Smallset Timeline outlines key decisions, such as keeping instances with missing values instead of removing them, as is sometimes done (e.g., see Shepperd et al. [2013]). Only eleven data columns are included in the visualisation to keep it compact in size.[6] Note that Figure 6.1 uses a similar amount of space as other figures in this thesis that are not Smallset Timelines (e.g., Figure 3.1) and remains legible.

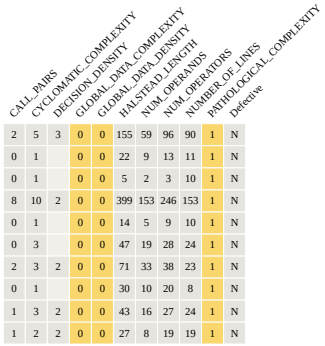
The Smallset Timeline in Figure 6.2 is based on the same preprocessing code as Figure 6.1 but was reconfigured to better support replication efforts. The Smallset size was increased from six rows to ten, to provide more data examples. The number of snapshots was increased from four to six, to separate the process into its component parts. The snapshot captions include specific information, such as the number of rows each operation affects in CM1 and the exact rules for checking data integrity. As a result of these changes, the Smallset Timeline is larger. Thus, if page space is limited, it may be located in an appendix, rather than the main text of a publication.

As indicated previously, the three data integrity checks in step 4 do not affect CM1. While the step was left out of Figure 6.1 for brevity, it is included in Figure 6.2 for clarity. If replicating the preprocessing strategy on another dataset, it would be necessary for accuracy and consistency to conduct the same data checks. Thus, they are detailed in Figure 6.2. It is worth noting that the preprocessing strategy in Shepperd et al. [2013] involves 18 different integrity checks, while Petrić et al. [2016] suggest 20 different checks. Those additional checks *do* result in the loss of instances in CM1. Therefore, for replication, simply saying "the data checks were run" is not enough. Replication requires more specificity.

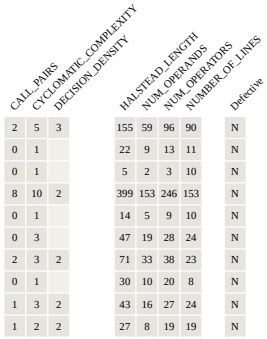
6.2 Case study 2: Inference with eBird citizen science data

This next case study focuses on an application of `smallsets` in the field of ecology: documenting the preprocessing of citizen science data for use in statistical modelling. In particular, the focus is on data from the eBird database, a citizen science program with millions of bird sightings from across the globe [Sullivan et al., 2009]. Citizen scientists upload their bird sightings, by completing an eBird checklist form. The form collects information about every

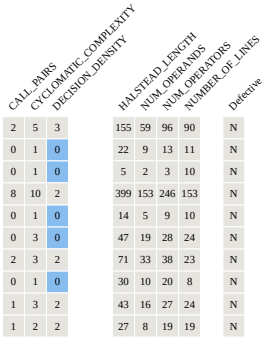
[6] The other 30 columns in the dataset are ignored using `ignoreCols` in `Smallset_Timeline()`.



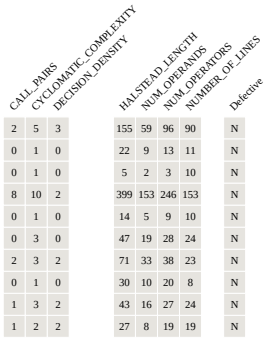
Step 1: All constant attributes are removed from the dataset. These attributes do not offer any useful information when building the classifier. There are three constant attributes in the CM1 dataset: *GLOBAL_DATA_COMPLEXITY*, *GLOBAL_DATA_DENSITY*, and *PATHOLOGICAL_COMPLEXITY*.



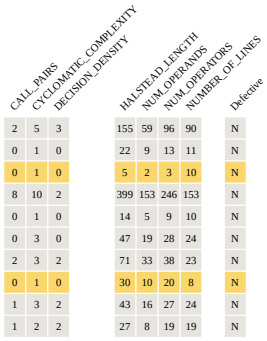
Step 2: Remove repeated attributes. There are none in CM1.



Step 3: The *DECISION_DENSITY* attribute is the only attribute that contains missing values in CM1. This attribute is equal to the *CONDITION_COUNT* divided by the *DECISION_COUNT*. Missing values only occur in *DECISION_DENSITY* when both of these other attributes equal zero. In other MDP datasets, where both equal zero, so does *DECISION_DENSITY*. Thus, the missing values are replaced with zero.
(161 rows affected)

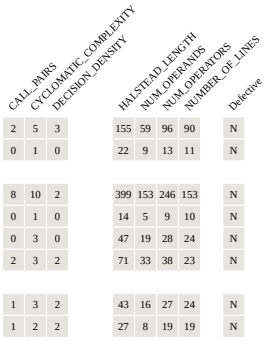


Step 4: All instances which fail one or more of the data integrity checks are removed from the dataset. The integrity checks identify instances that cannot realistically happen. The following integrity checks are used: *NUM_OPERANDS* + *NUM_OPERATORS* = *HALSTEAD_LENGTH*; *CYCLOMATIC_COMPLEXITY* <= *NUM_OPERATORS* + 1; *CALL_PAIRS* <= *NUM_OPERATORS*.
(0 rows affected)



Step 5: Remove repeated and inconsistent cases. Duplicates are dropped.
(49 rows affected)

Inconsistent cases refer to cases in which all values but the class label are equal, and these are dropped as well.
(2 rows affected)



The CM1 dataset is preprocessed and ready for modelling.

Deleted Edited Unchanged*

*A lighter value indicates a missing data value

Figure 6.2: Smallset Timeline for MDP CM1 dataset, for replication. Smallset selected using the *coverage + variety* algorithm. See Section 6.1.2 for a discussion and Appendix B.2 for the preprocessing script and `smallsets` code for this figure.

bird observed during an observation period. As noted on the eBird website,[7] to date the eBird data has been used in over 930 publications.

There are different types of citizen science (e.g., *structured citizen science*, *unstructured citizen science*, and *crowdsourcing*) that exhibit varying levels of control in the data collection process [Welvaert and Caley, 2016]. The eBird program is both structured and unstructured, i.e., semi-structured; reporting is done through checklists, which include observation metadata, but citizens choose when they want to observe and report [Johnston et al., 2021]. Semi-structured modes of data collection come with a cost, including various types of sampling and reporting biases [Welvaert and Caley, 2016]. Therefore, to use eBird data for statistical inference, researchers must account for and address data bias, in part through data preprocessing [Johnston et al., 2021, Strimas-Mackey et al., 2023].

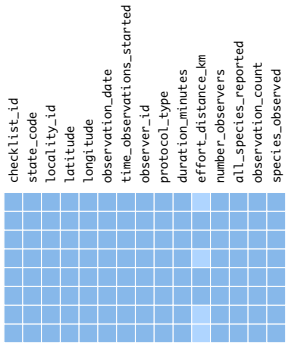
6.2.1 Visualising eBird best practices

Johnston et al. [2021] recommend a series of best practices for using citizen science data. These recommendations are based on an eBird case study that explored the effects of different data preparations on statistical inference. The authors found that the combination of using complete checklists only, spatial subsampling, effort filters,[8] and effort covariates produced the strongest modelling result. As a supplement to the study, Strimas-Mackey et al. [2023] produced the guide "Best Practices for Using eBird Data," which provides a step-by-step implementation of the study's recommendations in the R programming language.

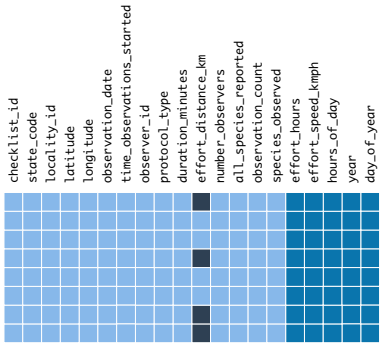
For the preprocessing code in that guide and a sample of eBird data [eBird, 2023], I built a Smallset Timeline, shown in Figure 6.3. In particular, I extracted the R snippets from sections 2.6–2.8 in Strimas-Mackey et al. [2023] and inserted `smallsets` structured comments. The only change made to the code was wrapping one piped `filter()` command in the `tibble` commands `rownames_to_column()` and `column_to_rownames()` [Müller and Wickham, 2023], to preserve row names for `smallsets` data tracking. The extracted code from Strimas-Mackey et al. [2023], with inserted structured comments, can be found in Listing B.5; the added `tibble` commands are highlighted in red. This case study, in particular, demonstrates that `smallsets` can be incorporated into existing data workflows, with minimal overhead.

[7] `https://science.ebird.org/en/research-and-conservation/publications`

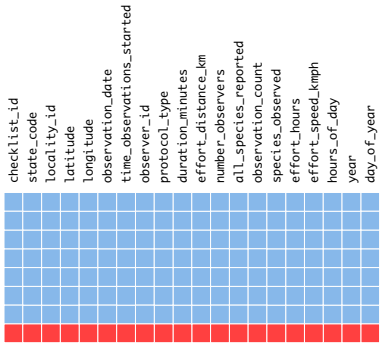
[8] Here, *effort* refers to observation effort, e.g., the length of an observation period, the ground covered and at what speed, and the number of observers.



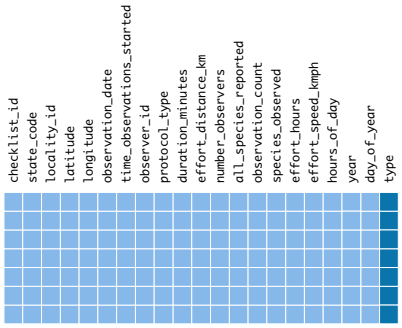
Eight (of >60,000) rows and 15 (of 35) columns from an eBird dataset, used to illustrate the preprocessing steps run prior to encounter modelling.



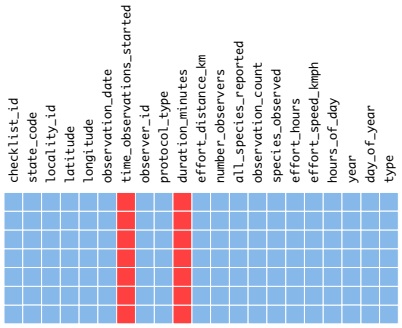
STEP 1:
- Convert observation counts into integers.
- Edit effort distance to be zero for stationary protocols.
- Create effort hours variable, by converting duration to hours.
- Create effort speed variable, by dividing effort distance by effort hours.
- Create hours of day variable (decimal hours since midnight), based on observation start time.
- Create year and day-of-year variables, based on the observation date.



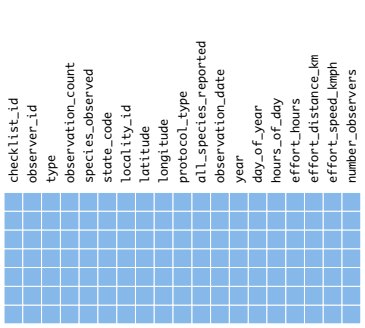
STEP 2:
Filter to the following:
- protocol is stationary or traveling;
- effort hours is <= 6;
- effort distance is <= 10;
- effort speed is <= 100;
- number of observers is <= 10.



STEP 3:
Randomly assign 80% and 20% of the dataset into a train and test set, respectively, and create a new type variable with the assignment "train" or "test."



STEP 4:
Drop 22 columns not needed for modelling, keeping only the 19 columns shown in the next snapshot.



The initial preprocessing of this eBird dataset is complete. Next, a series of land cover covariates will be created from satellite data and merged onto this dataset, based on locality and year. The dataset will then be ready for encounter rate modelling.

Added Deleted Edited Unchanged*
*A lighter value indicates a missing data value

Figure 6.3: Smallset Timeline for the eBird preprocessing steps recommended in Strimas-Mackey et al. [2023] (see Section 6.2.1). Smallset selected with random sampling. Data are not printed in snapshots, as per the eBird terms of use. The preprocessing script and `smallsets` code for this figure are in Appendix B.3.

6.3 Case study 3: The `folktables` data for machine learning

This next case study pivots to the field of machine learning (ML), and in particular, the study of algorithmic fairness. The Adult dataset (also referred to as the Adult Income or Census Income dataset) [Becker and Kohavi, 1996] from the UC Irvine (UCI) Machine Learning Repository is a popular dataset for ML research. It contains 1994 census income data, and the associated estimation task is to predict if an individual earns more than 50,000 dollars per year. The dataset has been especially popular for studying algorithmic fairness in ML, as it contains information on sensitive attributes like race and gender. According to Kelly et al. [2023], the Adult dataset has been formally cited over 250 times, though its use extends far beyond that to classrooms, tutorials, blogs, etc.

Ding et al. [2021] challenge the ML community's continued reliance on the dataset. The authors point to dataset issues, such as the 50,000 dollar threshold for the target variable, which leads to dataset imbalance by race and gender [Ding et al., 2021]. In turn, they develop a new Python package: `folktables`.[9] It generates benchmark datasets from American Community Survey (ACS) data and defines prediction tasks, with adjustable income thresholds and data filtering criteria. This option to adjust preprocessing settings is the focus of this case study. In Section 6.3.1, differences in algorithmic fairness, due to different preprocessing settings, are quantified. In Section 6.3.2, `smallsets` is integrated into a `folktables` workflow in a Jupyter Notebook.

6.3.1 Preprocessing and algorithmic fairness

Previous work on `folktables` preprocessing has quantified differences in fairness metrics across fairness interventions and income thresholds [Ding et al., 2021]. In this analysis of preprocessing effects, the focus is on quantifying differences in fairness metrics across varying data filtering criteria and income thresholds. The analysis uses 2015 ACS income data from California (CA), Connecticut (CT), and Utah (UT), retrieved with the `folktables` tool. First, four different preprocessing settings are generated. Next, logistic regression models are trained and tested on the different data preparations, and fairness metrics for the model

[9] `https://github.com/socialfoundations/folktables`

predictions are calculated. Finally, fairness outcomes are compared, revealing downstream effects from data preprocessing. Each step is discussed below.

Part I: Four preprocessing settings

For this analysis, four preprocessing settings were defined, starting with the `folktables` default setting [Ding et al., 2021], referred to here as *default-50K*. In this setting, an income threshold of $50K is used to generate positive and negative class labels. This is after filtering the dataset to retain an individual's record when they 1) are older than 16 years of age, 2) have a survey weight of at least one, 3) earn more than 100 dollars, and 4) report at least one hour of usual weekly work. The next preprocessing setting, called *default-median*, uses the same set of default data filters but sets the income threshold to the sample median after filtering ($36K, $45K, and $31.1K for CA, CT, and UT, respectively), to generate more balanced prediction tasks.

The last two preprocessing settings aim to be inclusive of all the target population, on the grounds that individuals who did not work and/or reported income losses are still valid instances for prediction. In turn, the last two filters are dropped. This filtering approach is referred to as "validity." The preprocessing setting *validity-match* uses the same threshold as *default-median*, such that the thresholds "match" (e.g., $36K for CA), despite different data filters. The fourth setting, *validity-median*, uses its own sample median after validity filtering is applied ($22.5K, $30.2K, and $23.5K for CA, CT, and UT, respectively). Table 6.1 summarises the four settings. Figure 6.4 is a Smallset Timeline depicting the *validity-median* steps applied to the CA dataset.

Preprocessing setting	Age >16	Survey weight ≥1	Income >100	Hours worked ≥1	Income threshold
default-50K	✓	✓	✓	✓	50K
default-median	✓	✓	✓	✓	median (after filtering)
validity-match	✓	✓			*default-median* value
validity-median	✓	✓			median (after filtering)

Table 6.1: Four different preprocessing settings used in the `folktables` prediction tasks for 2015 ACS income data. Each setting is a unique combination of data filtering criteria and income threshold selection for generating the class labels. See Section 6.3.1.

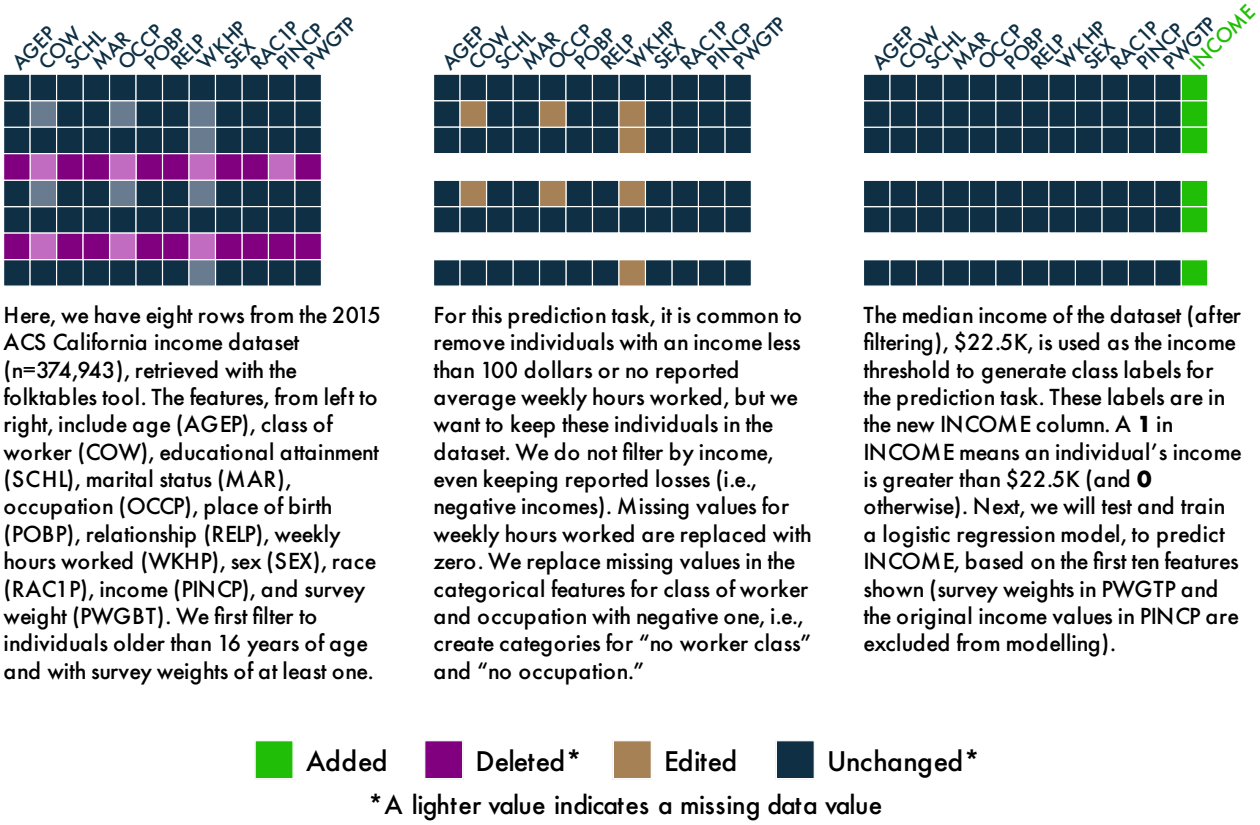


Figure 6.4: Smallset Timeline of ACS California data preprocessed with the *validity-median* setting. Smallset selected with random sampling. The preprocessing script and `smallsets` code for this figure are in Appendix B.4.

Part II: Modelling, prediction, and fairness

For each preprocessing setting in Table 6.1, a logistic regression model was trained and tested on 80% and 20% of the dataset, respectively, using `scikit-learn` default settings [Pedregosa et al., 2011] in Python. The prediction task was to predict if an individual's income is over the income threshold, based on ten features in the dataset.[10] Next, for men and women, differences of equality of opportunity (EO) [Hardt et al., 2016] and statistical parity (SP) [Dwork et al., 2012] were computed from the test set predictions. The EO fairness metric checks that the classifier predicts the positive label with similar accuracy for both groups [Hardt et al., 2016]. The SP fairness metric checks that the classifier predicts similar proportions

[10]These ten features are visualised in Figure 6.4 and correspond with the default feature set for the *ACSIncome* prediction task in Ding et al. [2021].

of the positive label for both groups [Dwork et al., 2012]. Ninety-five percent Newcombe intervals [Newcombe, 1998] were computed for the differences.

Part III: Assessing preprocessing effects

Figure 6.5 presents dataset imbalance by gender[11] as well as the EO and SP differences between men and women, across the four preprocessing settings.

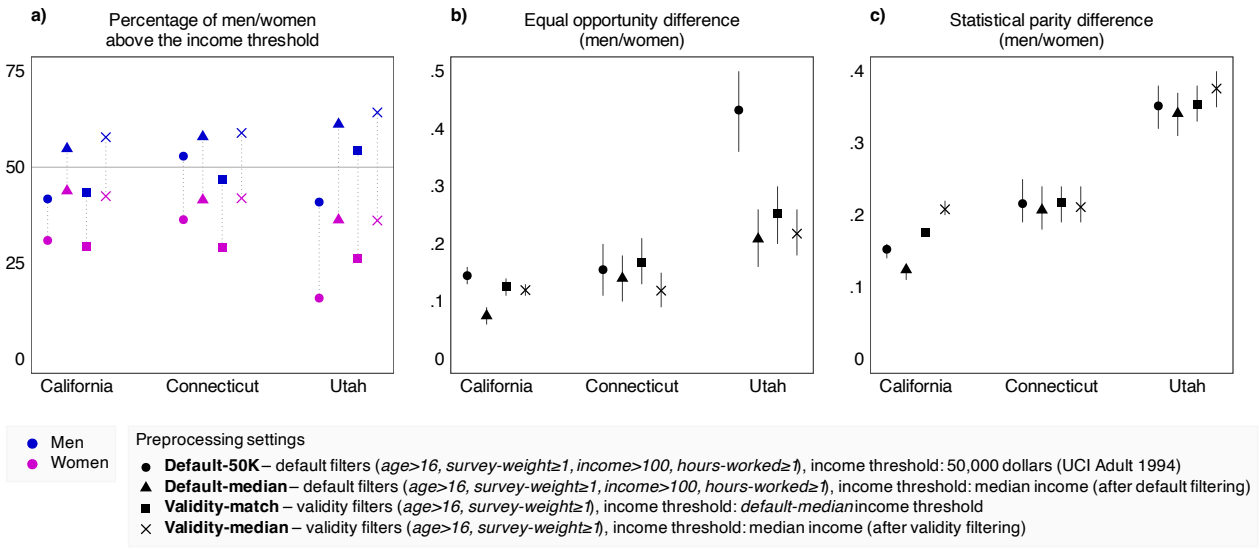


Figure 6.5: The effect of four different preprocessing settings on data and prediction. Plot a) shows dataset imbalance by gender. Plots b) and c) show group fairness measures in predictions from a logistic regression model. Error bars refer to 95% Newcombe intervals [Newcombe, 1998]. See Section 6.3.1 for a complete discussion.

Figure 6.5(a) compares the percentage of men and women above the income threshold, showing that certain thresholds achieve greater balance than others. For example, in the California dataset, the *default-median* setting results in more balance between men and women (aiming for equal splits) than *default-50K*. However, using the same income threshold alone will not guarantee balance or consistency across studies that use different data filters. Comparing *default-median* and *validity-match* for California, which have matching thresholds

[11] In the original dataset, the attribute corresponds with a male/female encoding and does not include nonbinary gender options.

($36K) but different data filters, we see a substantial change in the percentage of women above the income threshold (43.8% and 29.4%, respectively).

Figure 6.5(b) shows a significant difference in the equal opportunity difference between *default-50K* and the other three settings for Utah. For California, there is a significant difference between *default-50K* and *default-median* as well as *default-median* and *validity-match*. Variations among settings in Connecticut are much smaller. Figure 6.5(c) shows that, across the four settings, the statistical parity differences are significantly different for California but not for Connecticut or Utah. The takeaway is twofold. Preprocessing decisions for the `folktables` benchmark datasets can influence fairness outcomes in modelling. Therefore, communicating these decisions is crucial, to ensure valid comparisons across the benchmarking studies that use them.

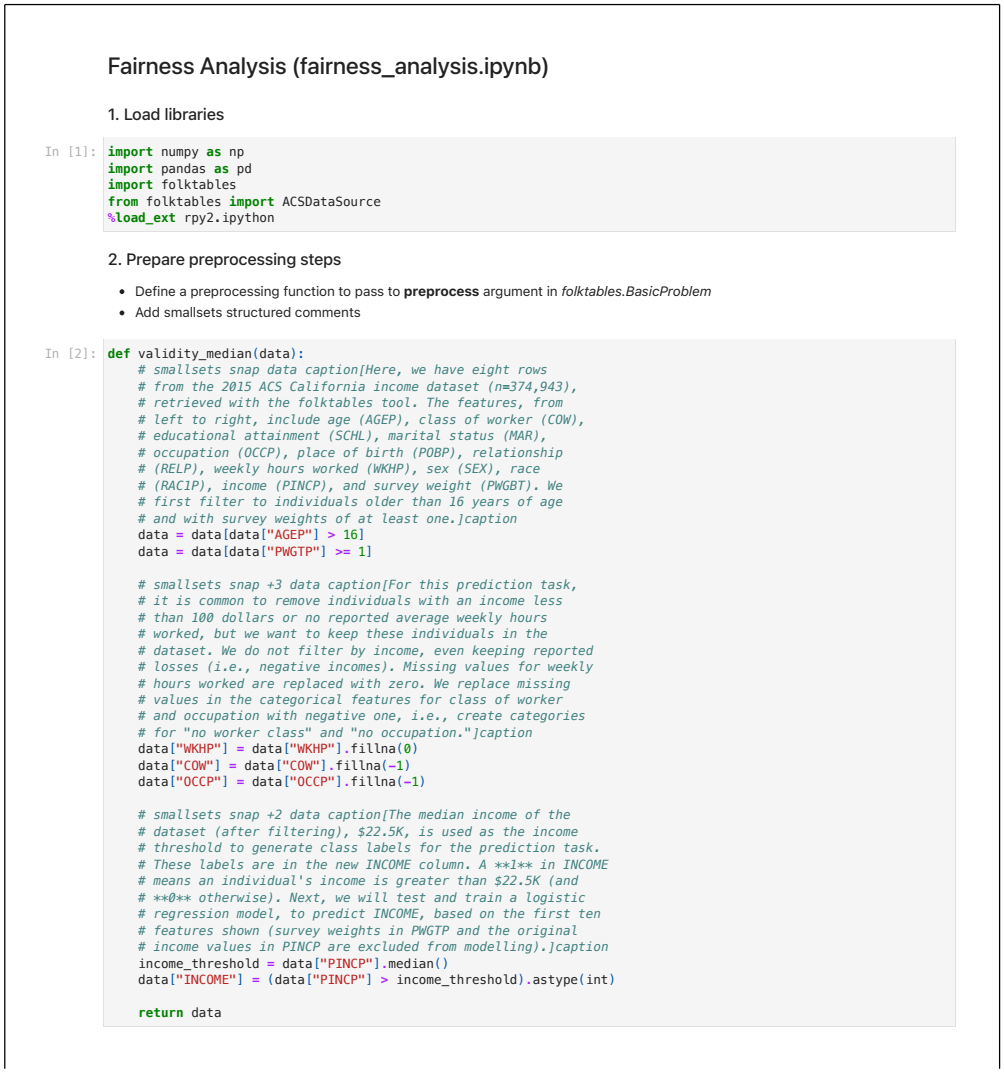
6.3.2 `smallsets` in Jupyter Notebooks

A `smallsets` user can integrate Smallset Timelines directly into a `folktables` [Ding et al., 2021] workflow, housed in a Jupyter Notebook with a Python kernel. Figures 6.6 and 6.7 show a Jupyter Notebook called *fairness_analysis.ipynb* with four cells of code. The first cell imports software packages required for the analysis. The second cell contains a preprocessing function that implements the data filters and income threshold of the *validity_median* preprocessing setting (see Table 6.1). The function is documented with `smallsets` structured comments.

The third cell creates a prediction task using class `folktables.BasicProblem`. In that class, the `validity_median` function is passed to the `preprocess` argument. Additionally, the 2015 ACS California income dataset is downloaded with `folktables`. The dataset is assigned to object `ca_acs_data` and passed to the prediction task, which returns three arrays: preprocessed features, class labels, and group labels for the fairness assessment.

The fourth cell is an Rmagic cell from `rpy2`,[12] which enables execution of R code in the Jupyter Notebook. This cell is used to run `smallsets`. The data object `ca_acs_data` and the Jupyter Notebook itself (*fairness_analysis.ipynb*) are passed to the `data` and `code` arguments, respectively, in the `Smallset_Timeline()` command. This prints a Smallset Timeline for the `validity_median` function (located in the second cell) below the Rmagic cell. From there, the analysis proceeds to building models and measuring fairness in model predictions.

[12] `https://rpy2.github.io`



Fairness Analysis (fairness_analysis.ipynb)

1. Load libraries

In [1]:
```
import numpy as np
import pandas as pd
import folktables
from folktables import ACSDataSource
%load_ext rpy2.ipython
```

2. Prepare preprocessing steps

- Define a preprocessing function to pass to **preprocess** argument in *folktables.BasicProblem*
- Add smallsets structured comments

In [2]:
```
def validity_median(data):
    # smallsets snap data caption[Here, we have eight rows
    # from the 2015 ACS California income dataset (n=374,943),
    # retrieved with the folktables tool. The features, from
    # left to right, include age (AGEP), class of worker (COW),
    # educational attainment (SCHL), marital status (MAR),
    # occupation (OCCP), place of birth (POBP), relationship
    # (RELP), weekly hours worked (WKHP), sex (SEX), race
    # (RAC1P), income (PINCP), and survey weight (PWGBT). We
    # first filter to individuals older than 16 years of age
    # and with survey weights of at least one.]caption
    data = data[data["AGEP"] > 16]
    data = data[data["PWGTP"] >= 1]

    # smallsets snap +3 data caption[For this prediction task,
    # it is common to remove individuals with an income less
    # than 100 dollars or no reported average weekly hours
    # worked, but we want to keep these individuals in the
    # dataset. We do not filter by income, even keeping reported
    # losses (i.e., negative incomes). Missing values for weekly
    # hours worked are replaced with zero. We replace missing
    # values in the categorical features for class of worker
    # and occupation with negative one, i.e., create categories
    # for "no worker class" and "no occupation."]caption
    data["WKHP"] = data["WKHP"].fillna(0)
    data["COW"] = data["COW"].fillna(-1)
    data["OCCP"] = data["OCCP"].fillna(-1)

    # smallsets snap +2 data caption[The median income of the
    # dataset (after filtering), $22.5K, is used as the income
    # threshold to generate class labels for the prediction task.
    # These labels are in the new INCOME column. A **1** in INCOME
    # means an individual's income is greater than $22.5K (and
    # **0** otherwise). Next, we will test and train a logistic
    # regression model, to predict INCOME, based on the first ten
    # features shown (survey weights in PWGTP and the original
    # income values in PINCP are excluded from modelling).]caption
    income_threshold = data["PINCP"].median()
    data["INCOME"] = (data["PINCP"] > income_threshold).astype(int)

    return data
```

Figure 6.6: First half of the Jupyter Notebook *fairness_analysis.ipynb*, for the scenario described in Section 6.3.2, in which `smallsets` is integrated into a `folktables` workflow. The second code cell contains a Python preprocessing function, documented with `smallsets` structured comments. The second half of the Notebook can be found in Figure 6.7, which includes a Smallset Timeline.

3. Set up folktables problem and data

- Pass preprocessing function defined above to **preprocess** argument in *folktables.BasicProblem*

In [3]:
```
# create a new folktables prediction task
ACSIncome = folktables.BasicProblem(
    features = ["AGEP", "COW", "SCHL", "MAR", "OCCP", "POBP", "RELP", "WKHP", "SEX", "RAC1P"],
    target = "INCOME",
    group = "SEX",
    preprocess = validity_median    # use preprocessing function defined above
)

# download folktables data
acs_data = ACSDataSource(survey_year = '2015', horizon = '1-Year', survey = 'person')
ca_acs_data = acs_data.get_data(states = ["CA"], download = True)
ca_acs_data = ca_acs_data[["AGEP", "COW", "SCHL", "MAR", "OCCP", "POBP",
                           "RELP", "WKHP", "SEX", "RAC1P", "PINCP", "PWGTP"]]

# set up prediction task
features, label, group = ACSIncome.df_to_numpy(ca_acs_data)
```

4. Visualise preprocessing with smallsets in an Rmagic cell

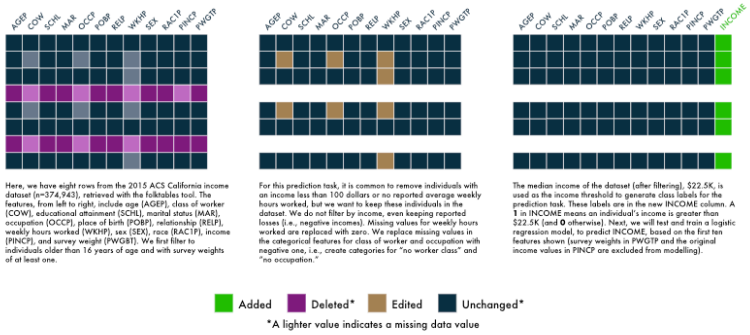
- Input the folktables data into the Rmagic cell with **-i**
- Pass that data and this Juptyer Notebook (fairness_analysis.ipynb) to smallsets

In [5]:
```
%%R -w 800 -h 400 -r 100 -i ca_acs_data

library("smallsets")
set.seed(40)

rownames(ca_acs_data) <- NULL
ca_acs_data[ca_acs_data == "NaN"] <- NA

Smallset_Timeline(data = ca_acs_data,
                  code = "fairness_analysis.ipynb",
                  rowCount = 8,
                  colours = list(unchanged = "#0e2f44", edited = "#A68156",
                                 added = "#20BE06", deleted = "#800080"),
                  missingDataTints = TRUE,
                  font = "Futura",
                  sizing = sets_sizing(captions = 2, columns = 2, legend = 10),
                  spacing = sets_spacing(degree = 45, header = 2, captions = 5, right = 1),
                  labelling = sets_labelling(labelCol = "darker", labelColDif = 0))
```



5. Build model...

6. Assess fairness of model...

Figure 6.7: Second half of the Jupyter Notebook *fairness_analysis.ipynb*, for the scenario described in Section 6.3.2, in which `smallsets` is integrated into a `folktables` workflow. The output of the fourth code cell is a Smallset Timeline, visualising the preprocessing code from the second code cell, which is shown in Figure 6.6 (the first half of the Notebook).

6.4 Case study 4: Home lending audits with HMDA data

The final case study is on home lending data. In 1975, the United States (U.S.) Congress passed the Home Mortgage Disclosure Act (HMDA), mandating that data about home lending be made public. Since then, HMDA data have become a valuable resource to understand the lending market and audit lending bodies for discriminatory practices [McCoy, 2007].[13] It is illegal in the U.S. to deny an applicant a home loan on the basis of race or color, national origin, religion, sex, familial status, or handicap [Fair Housing Act]. Auditing with the use of HMDA data, however, is not a straightforward task. Rather, it requires careful examination of the data and difficult decisions about how to best use it [Avery et al., 2007].

HMDA reporting requirements have evolved over time. For instance, in 1989, reporting requirements broadened to include denied home loan applications, and it became mandatory to report applicants' race, ethnicity, and gender [McCoy, 2007]. Presently, one can visit the HMDA Data Browser website and download an HMDA dataset of their choosing.[14] A standard CSV download will include 99 different columns of data, related to the lending institution, loan, property, geography, applicant, and census track. This case study considers nine of these columns and presents a simple auditing scenario, designed to illustrate some of the preprocessing decisions faced by researchers working with HMDA data.

6.4.1 A missing data dilemma

The auditing scenario involves two (fictitious) researchers, Alice and Bob. Both want to use HMDA data to compare the percentage of home loan denials among those who are Hispanic/Latino White and non-Hispanic/Latino White. Alice and Bob start with the same HMDA dataset for loans in Philadelphia County in 2019. The dataset consists of multiple columns of race and ethnicity information for the applicant, as applicants can specify up to five races and ethnicities.[15] Note that if an applicant does not specify race or ethnicity, a lender can collect it on the basis of visual appearance or surname. The dataset also has

[13] Though, as noted by Poirier [2022], disclosure datasets such as HMDA data are not without their limitations, including conflicts of interest in self-reporting, data dictionaries containing definitions that shift with time and changing political landscapes, and loss of context and nuance in checkbox reporting forms.

[14] `https://ffiec.cfpb.gov/data-browser/`

[15] For simplicity, this audit does not consider the demographic information of co-applicants.

Table 6.2: Percentage of denied loans by group and data availability. See Section 6.4.1.

	White
Non-Hispanic/Latino (n=5213)	5.52
Missing ethnicity data (n=138)	6.52
	Hispanic/Latino
White (n=662)	9.97
Missing race data (n=144)	22.22

Table 6.3: Percentage of denied loans calculated by researchers Alice and Bob. See Section 6.4.1.

	Alice	Bob
Hispanic/Latino White	9.97	12.16
Non-Hispanic/Latino White	5.52	5.55

information on the outcome of the application.

During data preprocessing, Alice and Bob must first collapse the information in the race and ethnicity columns into a single value, for each applicant. Then, they must address the issue of missing data. Some applicants have race data but not ethnicity data, and vice versa. Both start by generating binary variables for race (White or not White) and ethnicity (Hispanic/Latino or not Hispanic/Latino). Afterwards, Alice drops rows with missing race or ethnicity values, while Bob retains them through imputation. Specifically, for Hispanics/Latinos, Bob replaces missing race values with White. For Whites, Bob replaces missing ethnicity values with non-Hispanic/Latino. Bob does this because, in the dataset, 82% of Hispanics/Latinos identify as White and 89% of Whites identify as non-Hispanic/Latino.

However, the missing race data among Hispanics/Latinos does not appear to be "missing at random" [Rubin, 1976]. The percentage of denied loans among those *without* race data is more than double that of those *with* race data (22.22% vs. 9.97%, see Table 6.2). Thus, the decision to drop or impute missing data influences the outcome of the audit.[16] Table 6.3 shows that the calculated percentage of denied loans for Hispanic/Latino Whites differs between Alice and Bob at 9.97% and 12.16%. The Smallset Timelines in Figures 6.8 and 6.9 capture their decisions, respectively. The Smallset for each was selected with the *coverage + variety* model (see Section 4.3.1) on a 5% random sample from the dataset.[17] Bob's decision to impute is emphasised, visually, with the presence of several bright green data cells.

[16]Note that this case study presents a simplified version of an auditing analysis. In practice, analyses often involve a regression model to control for factors like income, loan amount, property type, etc. Each of these additional variables may be subject to preprocessing, with cumulative effects on outputs and conclusions.

[17]The sample size was 550 rows. While the same sample was used for both, the selected rows vary, as the model optimises based on the user's preprocessing steps.

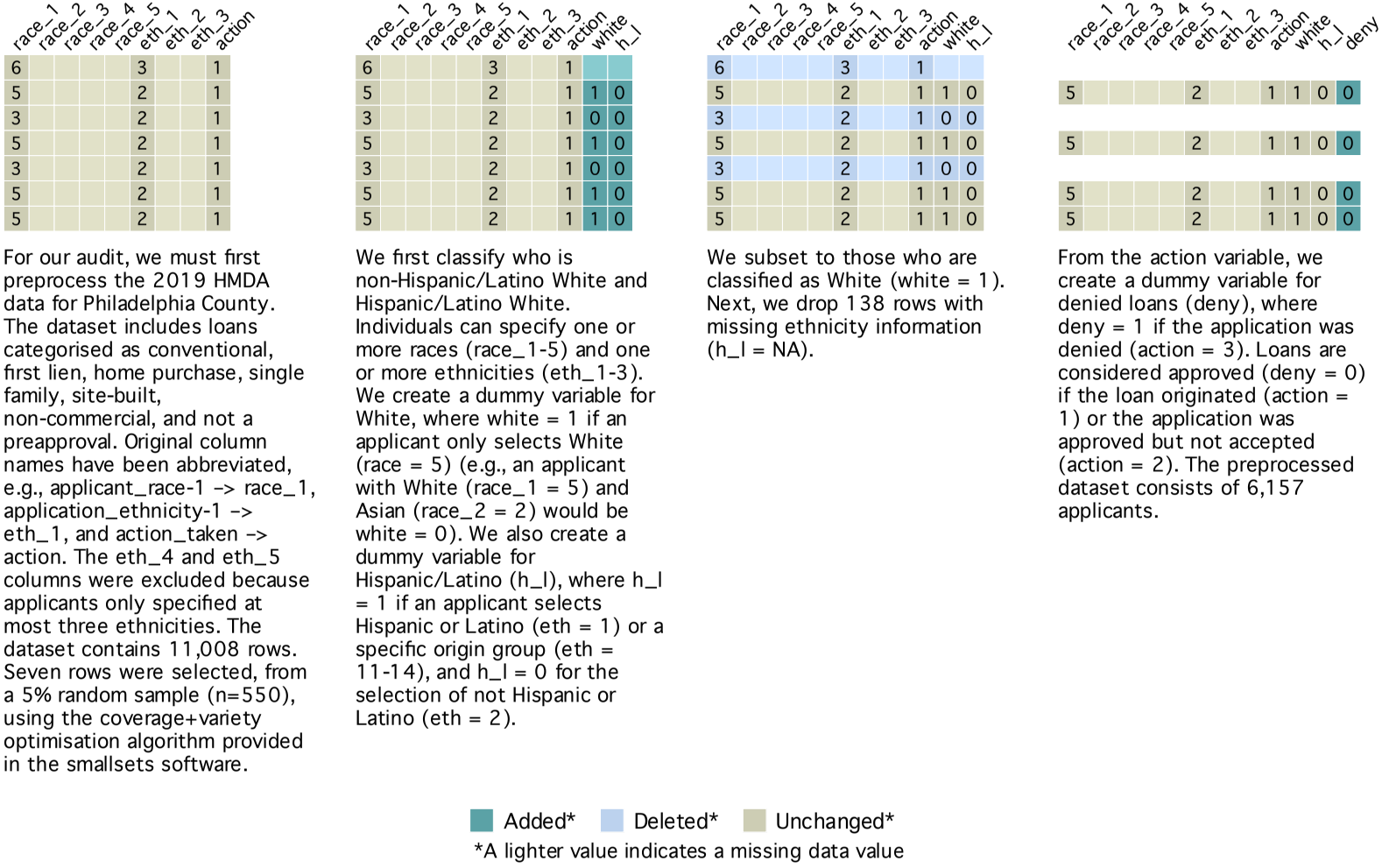


Figure 6.8: Smallset Timeline, created with the `smallsets` software, detailing the preprocessing decisions of researcher Alice in the home loan data case study discussed in Section 6.4.1. The preprocessing script and `smallsets` code for this figure are in Appendix B.5.

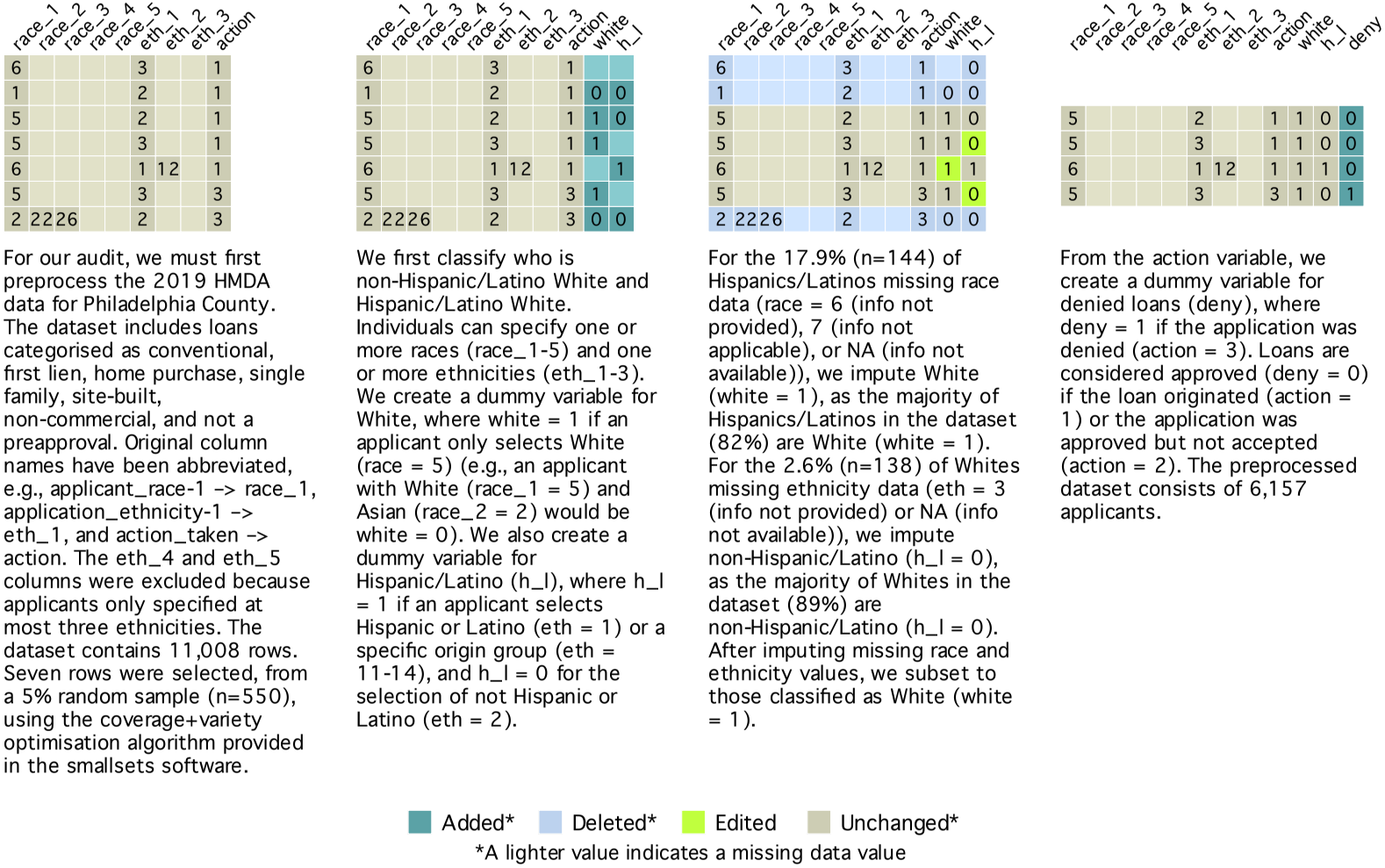


Figure 6.9: Smallset Timeline, created with the `smallsets` software, detailing the preprocessing decisions of researcher Bob in the home loan data case study discussed in Section 6.4.1. The preprocessing script and `smallsets` code for this figure are in Appendix B.6.

Chapter 7

Focus Groups on `smallsets`

This chapter presents the final contribution of this thesis: a focus group study on the `smallsets` software tool, which was proposed in Chapter 5 and demonstrated in Chapter 6. The study served as a first step in formally evaluating `smallsets`, with prospective users. It generated actionable feedback—regarding strengths and weaknesses of the `smallsets` tool—and data on real-world experiences with preprocessing communication. The remainder of this chapter is structured as follows. First, the motivation for this study, including the choice to use focus groups, is detailed (Section 7.1). Next, the methodology for conducting the focus groups and analysing the data is described (Section 7.2). Then, the focus group findings are presented (Section 7.3). Finally, the limitations and key takeaways of the study are discussed (Section 7.4).

7.1 Motivation

After version 1.0.0 of `smallsets` was published (see Table 5.4), attention turned towards evaluating three main aspects of the `smallsets` software, for two main reasons. Following are the three aspects of interest: 1) the `smallsets` deployment context, i.e., data practitioners' experiences with and attitudes towards preprocessing communication; 2) the utility and usability of `smallsets`; and 3) the visualisation produced with `smallsets`: the Smallset Timeline. The first reason for evaluating these three aspects of `smallsets` was to crosscheck

Label	Design Goals
	`smallsets`
s-G1	Any additional work required to build a Smallset Timeline with `smallsets`—on top of what is already required by the data preprocessing stage—should be minimised.
s-G2	Any additional work that is required to build a Smallset Timeline with `smallsets` should feel meaningful and productive to complete.
s-G3	Building a Smallset Timeline with `smallsets` should be easily integrable into new or existing data preprocessing workflows.
	Smallset Timelines
ST-G1	For data producers, creating the Smallset Timeline should encourage reflection and reflexivity on data preprocessing decisions.
ST-G2	For data producers/consumers, reading the Smallset Timeline should support the replicability and reproducibility of data preprocessing decisions.
ST-G3	For data consumers, reading the Smallset Timeline should support the comprehension and evaluation of data preprocessing decisions.

Table 7.1: The three design goals for the `smallsets` software and the Smallset Timeline visualisation, first presented in Section 5.1 and Section 4.1, respectively.

the design goals and assumptions that shaped development of `smallsets` (and Smallset Timelines). These design goals are listed in Table 7.1. The second reason was to inform future software development efforts, such that they align with the needs and preferences of data practitioners.

For this evaluation, the focus group method was selected. Focus groups provide an efficient and meaningful way to collect qualitative data on peoples' experiences, attitudes, opinions, and preferences [Krueger and Casey, 2000]. In the context of this study, the focus group format provided a formal means of connecting directly with prospective `smallsets` users, i.e., data practitioners who preprocess data. Focus group questions were designed to initiate discussion on the three areas of interest: 1) deployment context, 2) utility and usability, and 3) output (Smallset Timelines). Moreover, the data were analysed to check underlying design

goals and assumptions and to inform future work. As indicated previously, focus groups were a first step in the evaluation process. Chapter 8 includes a discussion on future work involving other modes of evaluation, such as user studies and semi-structured interviews.

7.2 Methods

This section describes the methodology used for the focus group study. The methodology follows the recommendations and techniques proposed in Krueger and Casey [2000] and is presented in five parts. This includes question development, recruitment and participants, focus group procedures, audio transcription, and analysis of transcripts. The study protocol was approved by the Science and Medical Delegated Ethical Review Committee at the Australian National University (ANU). Reciprocal approval was received from the Commonwealth Scientific and Industrial Research Organisation (CSIRO).

7.2.1 Question development

The first step was preparing a list of questions to ask participants, during the focus group. The purpose of these questions was to initiate discussion within a focus group, on the topics of interest, and to assure some degree of consistency across focus groups, in terms of the topics covered. Based on recommendations in Krueger and Casey [2000], the questions were designed to be open-ended, conversational, and jargon-free and to elicit descriptions of specific examples and experiences. Table 7.2 shows the question list prepared for this study. It consists of seven questions (Q1-Q7), designed to begin with broad attention to preprocessing and then narrow to focus on `smallsets`. The questions also ask participants to consider preprocessing communication from the perspective of both data producer and consumer.

The reason for starting with broad questions was twofold. It could introduce the general topic area and capture data on the `smallsets` deployment context. Then, in shifting the focus to `smallsets`, data could be collected on the tool's utility/usability and its output the Smallset Timeline. To collect data on these different aspects of `smallsets`, participants were asked to assume the perspective of data producer and data consumer. In Table 7.2, Q2 and Q5 pertain to data producers and are, respectively, about preprocessing communication broadly and `smallsets` specifically. Q3 and Q6 pertain to data consumers and follow the same broad to specific format.

	Question	Est. time (min.)
	Welcome and introduction to the focus group study	2
Q1.	What kind of data preprocessing do you do in your work?	6
Q2.	Do you ever need to communicate your preprocessing decisions? If so, tell me more about this.	8
Q3.	Are there instances in which you need to learn about another person's preprocessing work? If so, what is that learning process like?	8
	`smallsets` presentation	6
Q4.	What are your initial impressions of the `smallsets` tool?	7
Q5.	Imagine you are asked to use `smallsets` in your work. What challenges come to mind?	10
Q6.	Imagine a data analyst includes a Smallset Timeline as a figure in a report. If you were reading this report, what might your response to that figure be?	10
Q7.	Do you have any final comments or thoughts you'd like to share?	3

Table 7.2: Focus group question outline.

7.2.2 Recruitment and participants

Thirteen participants were recruited through my professional network. Data practitioners were invited by email to participate. Invitations were sent to 18 data practitioners. Fifteen data practitioners accepted the invitation to participate; in the end, two were unable to participate. Three data practitioners did not respond to the invitation. To be eligible for the study, participants had to have data preprocessing experience and be fluent in the English language. All invitees were sent a participant information sheet, which outlined the project objective and study protocol. It also provided information about potential risks, voluntary participation, withdrawal, confidentiality, etc. Those who decided to participate provided written consent ahead of the focus group.

Recruited participants included a statistician, research scientist, software developer, epi-

demiologist, software engineer, medical student, research fellow, and PhD students. Participants had diverse disciplinary backgrounds and research interests, including machine learning, the environment, digital humanities, and public health, to name a few. Participants were located in Australia and the United States. The groupings were largely defined by participants' availability to attend focus groups at certain times. There were focus groups where all, some, or none of the participants knew each other. Three focus groups had three participants. One focus group had four participants.

7.2.3 Focus group procedures

Four focus groups were conducted over the video conferencing platform Zoom, over the course of two weeks in August 2023. Each focus group lasted approximately one hour and followed the structure outlined in Table 7.2. Throughout the focus groups, probing questions were used in moderation, as per the recommendation of Krueger and Casey [2000]; this is to leave the majority of the time for discussion amongst participants. Following are two examples of probing questions from the data. *What do you think the importance of that particular piece of information is? Can you explain more what you mean by that?*

As noted in Table 7.2, a short `smallsets` presentation was given partway through the focus group. The purpose of the presentation was to provide a basic overview of `smallsets`. In other words, for this study, participants were not expected to have a comprehensive or nuanced understanding of `smallsets`. Though, some participants did have prior exposure to `smallsets` from research seminars and informal research discussions, and two participants had previously tried using `smallsets`, of their own volition.

The `smallsets` presentation included a slideshow and live software demonstration. First, the slideshow introduced the Smallset Timeline, the `smallsets` structured comments, and the `Smallset_Timeline()` command. The s_data example—detailed in Appendix A—was used for illustration. Participants were shown the example Smallset Timeline in Figure 4.1, followed by the `smallsets` structured comments and code to produce it. Then, the live software demonstration in RStudio reproduced Figure 4.1 with `smallsets`, and participants saw something similar to the RStudio screenshot in Figure 5.1.[1] Several of the optional arguments

[1]Note, however, that the study occurred before the most recent release of the `smallsets` software, version 2.0.0 (see Table 5.4). Focus group participants were instead presented the previous release of `smallsets`, version 1.0.0. The key difference between versions 1.0.0 and 2.0.0 was the switch from snapshot instructions to snapshot location arguments in the `smallsets` structured comments. This switch is illustrated in Figure 5.7.

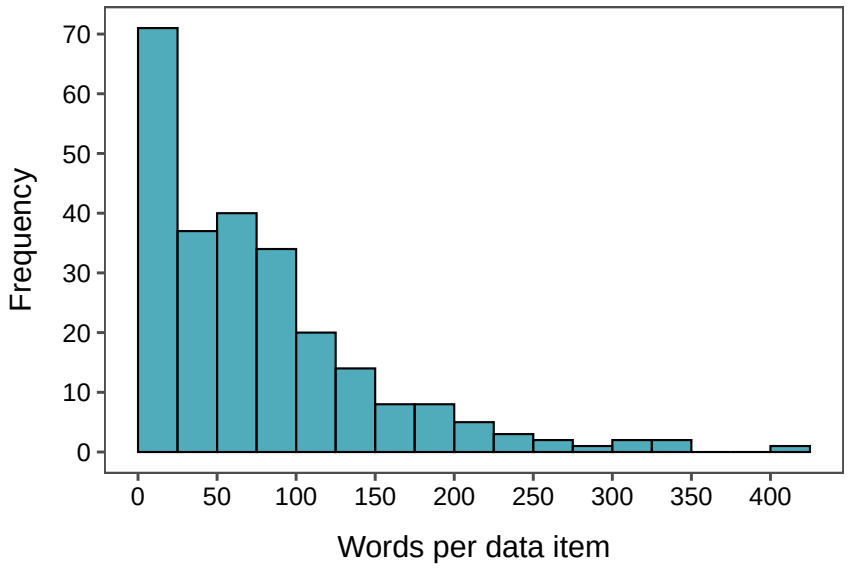


Figure 7.1: Frequency of different word counts of transcript data items, where a data item refers to each time a participant took a turn speaking, not including the moderator.

for `Smallset_Timeline()` were also featured—including `rowCount`, `colours`, `printedData`, `missingDataTints`, and `font` (see Table 5.1)—to show customising a Smallset Timeline.

7.2.4 Audio transcription

After the focus groups, the first step was transcribing the audio of each focus group recording. The video conferencing platform Zoom, used to host and record the focus groups, automatically generated an audio transcript for each recording. These auto-generated transcripts, however, contained errors that had to be manually identified and corrected. They also required anonymisation and significant formatting. Preparing the transcripts (verbatim) required carefully listening to each focus group several times, resulting in familiarisation with the data. Each participant was sent the de-identified transcript for their focus group, to review and check for accuracy, if they preferred to do so.

Following are several summary statistics for the transcripts. Note that each turn a participant took to speak was considered one data item, and the moderator's turns were not counted as data items. In total, there were 248 data items, which ranged in length from one word to 408 words. Figure 7.1 is a histogram visualising the frequency of different word

lengths across the data items. The average length of a data item was 75 words; the median length was 56 words. All participants spoke at least 12 times; the median was 18 times.

7.2.5 Analysis of transcripts

The data were analysed by focus group question, for Q2-Q6. For each question, all relevant data items were compiled, across transcripts. This included data items occurring later in the focus group—after a subsequent question had been asked—but containing relevant content. For instance, after Q5 was asked, P9 prefaced a comment as follows: *"This doesn't respond exactly to the question you asked ..., but I just did want to add something to what [P10] said before."* Because P9 was referring to what P10 said in Q4, that data item and the related data items that followed were included in the Q4 analysis. Once consolidated, the data items for a question were assessed for similarity in content, clustered accordingly, and summarised. As suggested in Krueger and Casey [2000], special attention was paid to comments about specific experiences or a matter raised by many different participants, especially across groups.

7.3 Focus group findings

In this section, findings from the focus group study are presented, by focus group question. This includes Q2-Q6, listed in Table 7.2. First, findings for Q2 and Q3, which focused on preprocessing communication broadly, are presented (Sections 7.3.1 and 7.3.2). Then, findings for Q4-Q6, which focused on `smallsets` specifically, are presented (Sections 7.3.3 to 7.3.5). New ideas for `smallsets`—that were proposed and discussed by participants—are also highlighted (Section 7.3.6). Throughout, references are made to the `smallsets` and Smallset Timeline design goals, using the labels listed in Table 7.1 (e.g., label *s-G1* refers to `smallsets` design goal one). Participants are referred to by participant IDs (P1-P13).

7.3.1 Preprocessing communication: Data producers

The first key focus group question, Q2, was as follows: *Do you ever need to communicate your preprocessing decisions? If so, tell me more about this.* It is a broad question, and participants' responses reflected this. Participants described varying amounts, types, and modes of communication. Three notable aspects of Q2 are discussed below.

Internal team-based communication

Many participants discussed internal team-based communication. For some, this appeared to be the first type of preprocessing communication that came to mind, when asked Q2. Participants talked about group projects, in which preprocessing details had to be discussed, for a variety of reasons. This included checking decisions with a supervisor, deliberating the best approach, handing over a prepared dataset, or ensuring consistency across team members' work. Moreover, participants noted several different mediums for internally communicating decisions, including documentation in code, conversations, and messaging.

Assumptions in preprocessing

Several participants mentioned the need to make assumptions while preprocessing and the uncertainty this introduces to an analysis. One participant described needing to hastily remember past assumptions, to verify them with a principal investigator, prior to an imminent paper submission. Another participant mentioned disclosing preprocessing assumptions, when communicating results, and advising caution in the use of results, given the assumptions. Another participant commented on the high prevalence of assumptions made during preprocessing work. However, they felt it was often not feasible or appropriate, when writing an academic article, to explain *all* of the assumptions made. Rather, they noted that sharing the preprocessing code could suffice.

Varying amounts of external communication

In terms of external (or formal) communication, participants described communicating varying amounts of preprocessing information, ranging from none to some. Table 7.3 includes five example data items on external communication, that highlight the spectrum of practices. The items in Table 7.3 are ordered, approximately, from the least to the most communication. Across these example data items, we can see a range of communication practices. Moreover, we can see reasons, provided by participants, for why they communicate as little or as much as they do. Reasons include assumptions about the impact (or lack thereof) of data preprocessing decisions, an external lack of interest in preprocessing, the information needs of an audience, and research norms/expectations.

Example data items from Q2 Do you ever need to communicate your preprocessing decisions? If so, tell me more about this.
(Discussing a group project) "It's interesting because we are just writing up a paper about this one, and I don't think we actually included how we preprocess the data. I think we just jump straight into 'This is the data. These are the models that we used.' And I think we didn't actually bother. I think it's almost like kind of taken for granted that like some preprocessing has occurred but it's not significant. Do you know what I mean? Like it shouldn't be impacting the results." (P2)
"So, I would say no. ... Nobody ever seems to ask for it. Yet sometimes it can be a massive amount of code. ... there are times ... I will post it on GitHub, but I don't think anybody looks at it. The only person who looks at it is me for next time I'm doing something similar, but nobody has ever asked me about it or for it." (P7)
"So, if it's ... an academic article or something, we might talk a little bit about the major things that we've done to prepare the data for the X-Y-Z. But if it's a report that we're giving to government or some other sort of report that we've been doing, we may not even discuss it, not because we don't want to, but it's not the right audience for that sort of discussion." (P6)
(Responding to P12's comment in the next row) "...I'll note that research-wise, at least in my side of like computer science and machine learning, [they] are very relaxed in terms of documenting preprocessing. It's usually just a very smaller side and like an appendix section and usually not super detailed either, like they'll just say some type of normalisation was done." (P11)
"Yeah, I have to. ... with text data, ... I guess preprocessing is a lot of cleaning of the data. And because I'm doing this in the context of writing publications, it's often you do need to detail the preprocessing steps in quite a bit of detail generally speaking is the expectation, in either an appendix or in the main body of an academic paper." (P12)

Table 7.3: Five example data items from Q2 (see Table 7.2), that highlight a range of practices for communicating data preprocessing decisions. Items are ordered, approximately, from the least to the most communication. Blue text provides context, helpful for understanding the data item. See Section 7.3.1 for a discussion.

7.3.2 Preprocessing communication: Data consumers

The next question, Q3, was as follows: *Are there instances in which you need to learn about another person's preprocessing work? If so, what is that learning process like?* In the data, there were two main situations, in which participants had to learn about another data practitioner's preprocessing work. The first was in the use of a dataset, already preprocessed by another data practitioner; participants had to know how it was preprocessed, to use it correctly. The second was in the recreation of a (preprocessed) dataset, for a benchmarking task. Some participants detailed situations in which the preprocessing information they needed was not publicly available. In turn, they had to track it down.

For a couple participants, this involved writing to paper authors privately, to request more dataset information. Regarding an experience trying to uncover preprocessing details for a benchmarking task, P1 said, *"...there was a lot of missing preprocessing details that you have to kind of find out for yourself."* P5 noted that, when working with datasets from organisations, proximity could be beneficial in tracking down information: *"If I know the data custodian well and they have sort of good access to the data and that sort of stuff, I can often get some kind of answer."* It is worth noting that efforts to track down information were not always successful, resulting in a standstill or data practitioners proceeding—as best they could—without it.

7.3.3 Impressions of `smallsets`

After the moderator's presentation on `smallsets`, participants were asked Q4: *What are your initial impressions of the `smallsets` tool?* Broadly, participants talked about the use of visualisation, the `smallsets` structured comments, and/or envisioned use cases of `smallsets`. Each topic is discussed below.

Visualisation

Many participants had a positive response to the use of visualisation. Responses ranged from general enthusiasm for visualisation to specific comments about the Smallset Timeline, e.g., *"I think the colour boxes are very clean"* (P1). An appreciation for visualisation was expressed from both the perspective of data producer and data consumer. Speaking as a data producer, P5 noted that they will sometimes visualise a subset of their tabular data during

preprocessing, *"...to sense check a few things,"* and that `smallsets` seems like *"...a nice way of actually formalising that process."* Speaking as a data consumer, P9 said, *"...I think visualising it is much appreciated, relative to sort of having to go back through someone's script and trying to figure out whether they've ... commented what decisions they've made or whether you just have to go through their code and figure something out."*

Structured comments

There was also interest in the `smallsets` structured comments. Several participants expected that adding them to code might result in better documentation within their code. P8 said the following about having to insert structured comments, to get snapshots that break preprocessing down into steps: *"...I think that forces a linear, logical way of wrangling data that then ... will ... I would say most likely lead to more linear, logical comments and documentation on those steps that are going on."* One participant did wonder if the insertion of `smallsets` structured comments could be automated, for lengthy preprocessing pipelines with many steps. However, generally speaking, the `smallsets` structured comments were seen as having value and not simply a means to an end. This finding provides some indication that *s-G2* and *ST-G1* are being met (see Table 7.1).

Envisioned use cases

Q4 also generated data on envisioned use cases of `smallsets`. Notably, participants envisioned using `smallsets` as a communication tool *and* a real-time preprocessing aid. For example, referring to `smallsets`, P4 said, *"...I would be primarily thinking about this as first for myself and then for other people."* As a preprocessing aid, participants discussed using `smallsets` to identify and diagnose problems in preprocessing and to understand how the code they run changes their dataset. As a communication tool, participants envisioned using Smallset Timelines in publications and discussions with a supervisor. Envisioned use cases, though, were sometimes followed by a catch. Specifically, participants noted that `smallsets` seemed useful, but there was a current limitation of the tool that would impede uptake of it into their work. These catches are discussed next in Section 7.3.4.

7.3.4 Uptake: Challenges and concerns

The next question, Q5, was as follows: *Imagine you are asked to use* `smallsets` *in your work. What challenges come to mind?* Q5 aimed to gather information on barriers, that would impede the uptake of `smallsets` into data workflows. Once identified, these barriers can be addressed in future work, where possible, to better achieve *s-G3* (see Table 7.1). The Q5 findings are discussed below.

Limitations in `smallsets` functionality

Version 1.0.0 of `smallsets` had several key limitations in functionality. It was functional for one tabular dataset at a time, only. All preprocessing code had to be contained in a single script. Furthermore, complex operations overwriting row names in R were not supported (see Section 5.4.4).[2] Each of these limitations proved to be a barrier to uptake, for one or more participants. Specifically, for participants working in the areas of digital humanities and natural language processing, the primary issue is that `smallsets` cannot yet handle unstructured text data. For both participants who had previously tried using `smallsets` in their work, the requirement that all code be contained in a single script was the barrier to uptake; their preprocessing code existed across multiple scripts. Some participants described working with multiple datasets at once and using more complex operations like aggregations and joins as well as chained and piped operations, which `smallsets` currently has no/limited capacity to visualise.[3]

Time

One concern, raised by participants, was the time required to incorporate a new documentation tool, such as `smallsets`, into a data workflow. For example, P10 said, *"...if I have to clean up the data at all to use* `smallsets`*, then that seems like that would be an upfront—it would probably be worth it in the end—but the upfront cost to clean it up and use it, I could see in the moment possibly not wanting to deal with it, if it has that at all."* In other words, if integration of the tool into their workflow felt troublesome, it may deter them from using it. P10's statement resonated with another participant, who talked about wanting to produce

[2]These three limitations also exist in version 2.0.0 of `smallsets`.

[3]For example, `smallsets` can take snapshots of a data object before and after a piped operation, but it cannot capture intermediate states within a piped operation (see Listing B.5, as an example).

documentation but having limited time to do so. This concern about time—and potential hassles associated with using `smallsets` in one's work—underscores the necessity of *s-G1*, *s-G2*, and *s-G3* and the importance of achieving them (see Table 7.1).

Miscellaneous concerns

One participant noted that the Smallset Timeline's horizontal layout could pose problems in two-column articles, in which figures are limited to half the page width. Another concern was that adding `smallsets` structured comments, alongside other comments in the code, could lead to a cluttered preprocessing script. There would be too many comments overall, likely with redundant content. Both concerns have since been addressed, in the most recent release of the `smallsets` software, version 2.0.0 (see Table 5.4). Firstly, there is now an option to arrange Smallset Timelines vertically (see Table 5.1); snapshots are plotted top to bottom, and captions are located to the right of snapshots. Secondly, to avoid comment clutter, users can place a block of `smallsets` structured comments at the top of a script, separate from the code and other comments (see Figure 5.3). Additional concerns about the Smallset are discussed in Section 7.3.5.

7.3.5 Reactions to Smallset Timelines

The next question, Q6, aimed to collect data on the Smallset Timeline. Participants were asked to assume the perspective of data consumer: *Imagine a data analyst includes a Smallset Timeline as a figure in a report. If you were reading this report, what might your response to that figure be?* Discussion for Q6 centred around two main topics: accessibility and the Smallset. Each is detailed below.

Accessibility

Participants discussed how the Smallset Timeline could make preprocessing information more accessible and the benefits from this. One participant commented that by presenting the process in steps, the Smallset Timeline could make it easier to comprehend the extent of the preprocessing work. Another participant thought that having the visualisation would enable them to ask more insightful questions about a data producer's preprocessing decisions. P6 noted that, when included as a figure in a publication, the Smallset Timeline *"encodes"*

dataset and preprocessing information into a paper, so there is some long-term access to it. This is regardless of what happens to the dataset after the paper is published.

Several participants discussed the relationship between accessibility and trust. Specifically, the Smallset Timeline was viewed as a way for data producers to be candid about dataset changes, with positive outcomes. For example, P7 said that if a data producer included a Smallset Timeline in a paper, it would *"...increase my confidence that they knew what they were doing."* This comment resonated with other participants in the group. They appreciated the idea of data producers showing the steps—in snapshots with tracked changes—as opposed to telling them in purely text-based descriptions. P8 said, *"You wouldn't just have to like take people's word for it on how they wrangled. It would actually just, you know, be there to see."*

The Smallset

Upon seeing a Smallset Timeline, some participants expected to have questions about the Smallset.[4] To start, how was the Smallset selected? P9 said, *"I think it'd be good if it was included in a paper to include a description of how the actual Smallset was selected."* The concern was that a data producer could strategically select a set of rows that only tells a partial (and biased) story about preprocessing. Other participants expected they would have questions about the Smallset's representativeness; these questions stem, in part, from interests in the broad-scale effects of preprocessing decisions. However, the Smallset is not designed to be a representative sample of the full dataset (see Section 4.2.1). Thus, these questions about representativeness underscore the importance of the previous point, about communicating the Smallset selection method. A clear explanation of the Smallset selection method can help to minimise confusion about the meaning of a Smallset.

7.3.6 New information and features

Throughout the focus groups, participants discussed new ideas for `smallsets`. There were pieces of preprocessing information, not presently captured in the Smallset Timeline, that participants imagined could be useful to have, in understanding preprocessing. This included compute time, memory use, variable types, and column classes. This also included summary

[4]The Smallset is discussed in depth in Section 4.2.1. Automated Smallset selection methods are presented in Section 4.3.

statistics about the dataset, which could be presented at different points in a Smallset Timeline, such as before, during, and/or after preprocessing. Adding more description about the decision-making process and the option to document and compare alternative decisions was also mentioned.

Some participants wanted features that would enable them to assess and communicate the broader effects of their preprocessing decisions on the dataset and overall analytical objective. Participants discussed increasing the granularity of the colour legend and assigning colours to specific preprocessing tasks. One participant imagined how to visualise parallel processing and considered the visualisation of multiple Smallsets simultaneously. Developing an interactive version of the Smallset Timeline was also discussed. Interactive elements included hovering over the visualisation to reveal additional dataset information and the option to link snapshots to segments of code (e.g., users click a snapshot to see the related preprocessing code).

7.4 Discussion

To summarise, this chapter presented a focus group study on the `smallsets` software tool. The study involved four focus groups, with a total of 13 data practitioners with preprocessing experience. The focus groups generated qualitative data on three aspects of `smallsets`, including its deployment context, utility/usability, and output (the Smallset Timeline). These data provide insight into how prospective users view `smallsets` and where to direct future software development efforts, for `smallsets`. This chapter ends with a discussion on the limitations of the study (Section 7.4.1) and the key takeaways from it (Section 7.4.2).

7.4.1 Limitations

This focus group study is subject to several limitations. Participants were recruited through one person's professional network, and the moderator was not an independent third-party. Both of these factors could have influenced the level of comfort that participants felt to express negative opinions about `smallsets`. Note that during the welcome/introduction (see Table 7.2), the focus group was framed in part as a request for help, to make `smallsets` a better tool for data practitioners, at large. Moreover, as indicated in Section 7.3, participants did express concerns, identify problems with uptake, and share ideas to enhance `smallsets`.

Regardless, the lack of a independent third-party moderator is a factor to consider, when considering the focus group findings.

Another limitation is that participants did not use or test `smallsets` themselves, during the focus group.[5] Rather, feedback was largely based on what participants had heard about `smallsets` from the moderator and seen in a short demonstration of the tool. If trying the tool themselves, participants may have identified additional issues or concerns, not obvious from the `smallsets` presentation. Future work can involve user studies, in which participants interact with `smallsets`.

This focus group study is also subject to recognised limitations of the focus group method, including the effects of group dynamics [Krueger and Casey, 2000]. For example, some participants may have hesitated to express dissenting opinions or share difficult experiences. Future work can involve semi-structured (one-on-one) interviews with data practitioners, to learn more about experiences with preprocessing communication.

Finally, participation in the study was limited to data practitioners, and in particular, those with data preprocessing experience. However, the Smallset Timeline is also designed for data consumers who are not data practitioners and/or who do not have firsthand data preprocessing experience. For this segment of data consumers, the Smallset Timeline is designed to support comprehension and evaluation of data practitioners' preprocessing decisions (see Table 4.1). Future evaluations can be more inclusive of all intended Smallset Timeline users.

7.4.2 Key takeaways

Following are several key takeaways from this focus group study on `smallsets`.

- Many participants described needing to communicate and discuss data preprocessing decisions internally with team members, collaborators, and supervisors. The means of internal communication varied, from documenting code to having conversations.

- The amount of external (formal) preprocessing communication practised by participants ranged from none to some. Furthermore, participants gave different reasons for why they communicate as little or as much as they do.

[5] Two of the participants had previously tried using `smallsets`, of their own volition. However, the majority of participants had never tried the tool.

- Speaking as data consumers, participants described challenging situations trying to track down preprocessing details—that were not publicly available—to either understand or reproduce a preprocessed dataset, for use in a new analysis.
- Many participants had a positive response to the `smallsets` structured comments and saw this aspect of `smallsets` as valuable in and of itself. Moreover, there was significant interest in using `smallsets` not only as a communication tool but also as a real-time preprocessing aid.
- Participants pointed to current limitations of `smallsets` that would impede uptake. This included the lack of functionality for non-tabular data, multiple preprocessing scripts, and more complex operations like data joins. The time required to learn and use `smallsets` could also be a barrier to uptake.
- Many participants liked the use of visualisation, to communicate preprocessing decisions. Participants weighed the upsides and downsides of using a Smallset, to explain a preprocessing pipeline. On the one hand, it makes the information accessible. On the other hand, it does not capture the broader effects of data preprocessing decisions.

Acknowledgement. *Future improvements to* `smallsets` *will be in large part thanks to the input and feedback from the 13 focus groups participants. I am grateful to all participants for sharing their time, experiences, opinions, and thoughts, to help the project progress.*

Chapter 8

Conclusion

This chapter summarises the contributions of the thesis (Section 8.1) and discusses future work (Section 8.2).

8.1 Summary

Data preprocessing is a crucial part of many data analyses. Moreover, it often requires data practitioners to make difficult decisions about how to resolve dataset issues and prepare their dataset for the subsequent estimation or modelling task. Yet, data preprocessing is often overlooked in research dissemination. This discrepancy—in how data analytics is conducted in private versus how it is presented in public—limits the ability of data consumers to interpret, replicate, and evaluate research findings from quantitative analyses. In turn, this thesis focuses on the communication of data preprocessing decisions and makes several contributions in this area.

This thesis presents a novel visualisation of data preprocessing decisions called the Small-set Timeline. The visualisation is designed to be static and compact, for simplicity, practicality, and accessibility. It can be included in academic publications, blog posts, README files, etc., to communicate preprocessing choices. It can also be used to answer the preprocessing questions and prompts in the data provenance tools proposed for machine learning research, such as datasheets [Gebru et al., 2021] and model cards [Mitchell et al., 2019]. For example,

33. **Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)?** If so, please provide a description. If not, you may skip the remaining questions in this section.

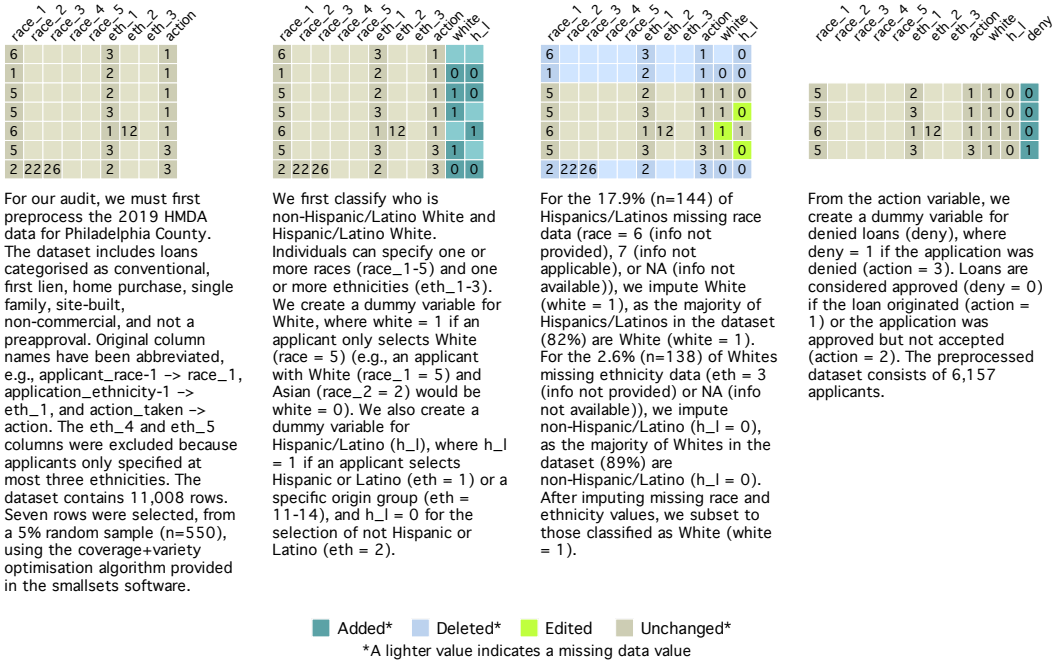


Figure 8.1: Example of question 33 from a datasheet [Gebru et al., 2021] being answered with a Smallset Timeline, built with `smallsets`. The Smallset Timeline used as an example above is from case study 4, on Home Mortgage Disclosure Act (HMDA) data, in Section 6.4.

a Smallset Timeline could be used to answer question 33 in a datasheet (see Figure 8.1). This recommendation is supported by findings from the focus group study presented in Chapter 7, in which some participants expressed a preference for visual explanations of preprocessing steps, as opposed to purely text-based descriptions.

To make building Smallset Timelines simple and easy, this thesis presents the open-source software tool `smallsets`. It is an R CRAN package that transforms R or Python preprocessing code in an R, R Markdown, Python, or Jupyter Notebook file into a Smallset Timeline. Users add structured comments with snapshot instructions to their preprocessing code and run the `Smallset_Timeline()` command. Within that command, there are options

to customise different aspects of the visualisation. Several resources have been developed for `smallsets` users, including a user guide, cheatsheet, and website.

To demonstrate use of the `smallsets` tool, this thesis presents four data preprocessing case studies. The case studies resulted in a total of six example Smallset Timelines visualising real-world preprocessing scenarios (Figures 6.1 to 6.4, 6.8 and 6.9). From machine learning benchmark datasets to citizen science data for statistical modelling, these case studies are diverse. However, all four case studies underscore the importance of communicating data preprocessing decisions. Two of the four case studies include quantifying the effects of preprocessing decisions on analytical outcomes.

Finally, this thesis presents a focus group study. The study was designed to gather feedback on `smallsets` from prospective users. The focus group findings detail strengths and weaknesses of the `smallsets` tool and how it can be improved to best meet the needs of data practitioners in their everyday work. The findings also offer insights into experiences with and attitudes towards preprocessing communication.

8.2 Future work

There are several different areas of future work, related to Smallset Timelines, `smallsets`, and data preprocessing, at large. The items below are ordered accordingly.

- **Redesign of Smallset Timelines for other data types.** Presently, the Smallset Timeline is designed for tabular data. Future work could involve redesigning the visualisation for other types of data, such as text and image data. This would involve rethinking the configuration of the Smallset and the categories of changes, highlighted in Smallset snapshots.

- **Statistical summaries and diagrams with Smallset Timelines.** Another area of future work, related to Smallset Timelines, is the incorporation of statistical diagrams and summaries into the visualisation, to communicate broad effects from preprocessing. For example, if a snapshot visualises the transformation of a variable, could there be a histogram above the snapshot showing the distribution of the variable before and after the transformation? Or could a simple table with statistical summaries be included below each snapshot caption? It may be that statistical diagrams and summaries

should be included separately, however, to preserve the simplicity and compact size of the Smallset Timeline.

- **Expansion of the `smallsets` software.** In the focus group study presented in Chapter 7, participants noted several key hurdles to the uptake of `smallsets`. This included that `smallsets` cannot currently handle multiple preprocessing scripts and more complex operations such as merges and aggregations. Addressing these hurdles will be a top priority of future `smallsets` work. Another important area of future work is providing full support for the creation of non-English Smallset Timelines. Currently, users can write snapshot captions in other languages, but the labels for the colour legend are hard-coded in English, as is the automated alt text template (Figure 4.9).

  Section 4.3.1 of the thesis presented two optimisation models for Smallset selection. Future work could also involve the development of additional automated Smallset selection methods. One focus, in particular, could be on developing fully open-source methods that do not require a `smallsets` user to obtain a Gurobi license. Methods that perform column selection in addition to row selection could also be useful.

- **User studies for the `smallsets` software.** In the focus group study presented in Chapter 7, the majority of participants were providing feedback on `smallsets` based on what they had heard and seen in presentations/demonstrations of the tool. Only two participants had tried using the tool. Future work could involve a user study, in which users engage with the software tool themselves and provide feedback based on that experience. This could result in improvements to the usability of `smallsets` and to the software's help documentation.

- **Assistance for `smallsets` caption writing.** The main task of `smallsets` users is to write snapshot captions for the Smallset Timeline. Table 4.1 and Section 4.2.1 provide some guidance on caption-writing, but future work could further explore what preprocessing content, in particular, data consumers find most useful, in different settings. Those content needs could be translated into a series of caption-writing guidelines and prompts for Timeline creators.

- **Development of a reflexivity guide for data preprocessing.** In Chapter 4, one of the design goals of the Smallset Timeline is to encourage the practice of reflexivity

(Table 4.1), which aligns with a recent push to incorporate reflexivity into quantitative research [D'Ignazio and Klein, 2020, Miceli et al., 2021, Tanweer et al., 2021]. Reflexivity is a well-established concept and practice in qualitative research but remains largely unfamiliar to many quantitative researchers. One area of future work is writing a reflexivity guide for data preprocessing, specifically. The guide could introduce the concept of reflexivity and include questions that prompt reflexive practices. It could be included as a vignette within the `smallsets` software, so that it could be easily accessed through the `vignette()` command in R.

- **Qualitative research on data preprocessing practices.** Another area of future work is conducting ethnographic research on data preprocessing. It could involve extended observation of data practitioners as they preprocess data, as well as interviews with data practitioners on the topic of their data preprocessing work. It could be conducted in different research sectors, including academia, industry, government, etc. This type of research could provide further insight into the nature of data preprocessing work and the ways in which data practitioners make data preprocessing decisions.

8.3 Concluding note

As mentioned at the very beginning of this thesis, data preprocessing typically does not have a reputation as being the exciting or interesting part of data analytics. However, the hope is that this thesis sheds light on the importance, challenge, and impact of data preprocessing work. Moreover, the hope is that `smallsets` can serve as a highly practical tool, that equips data producers to be transparent about critical data preprocessing decisions.

Bibliography

JJ Allaire, Yihui Xie, Christophe Dervieux, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. *rmarkdown: Dynamic Documents for R*, 2023. R package version 2.25. (cited on pages 69 and 72)

Matthew Arnold, Rachel KE Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, K Natesan Ramamurthy, Alexandra Olteanu, David Piorkowski, et al. FactSheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM Journal of Research and Development*, 63(4/5):6–1, 2019. (cited on page 28)

Australian Bureau of Statistics. Data processing. 2023. URL `https://www.abs.gov.au/websitedbs/D3310114.nsf/home/Basic+Survey+Design+-+Data+Processing`. Accessed: June 6, 2023. (cited on page 35)

Robert Avery, Kenneth Brevoort, and Glenn Canner. Opportunities and issues in using HMDA data. *Journal of Real Estate Research*, 29(4):351–380, 2007. (cited on pages 25 and 96)

Louis Bavoil, Steven P Callahan, Patricia J Crossno, Juliana Freire, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. VisTrails: enabling interactive multiple-view visualizations. In *VIS 05. IEEE Visualization, 2005.*, pages 135–142, 2005. (cited on page 29)

Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. URL `https://doi.org/10.24432/C5XW20`. (cited on page 89)

Colin Begg, Mildred Cho, Susan Eastwood, Richard Horton, David Moher, Ingram Olkin, Roy Pitkin, Drummond Rennie, Kenneth F. Schulz, David Simel, and Donna F. Stroup. Improving the quality of reporting of randomized controlled trials: The CONSORT statement. *JAMA*, 276(8):637–639, 1996. (cited on page 30)

Emily M Bender and Batya Friedman. Data statements for natural language processing:

Toward mitigating system bias and enabling better science. *Transactions of the Association for Computational Linguistics*, 6:587–604, 2018. (cited on page 28)

Avinash Bhat, Austin Coursey, Grace Hu, Sixian Li, Nadia Nahar, Shurui Zhou, Christian Kästner, and Jin L.C. Guo. Aspirations and practice of ML model documentation: Moving the needle with nudging and traceability. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2023. (cited on page 28)

Alexander W. Blocker and Xiao-Li Meng. The potential and perils of preprocessing: Building new foundations. *Bernoulli*, 19(4):1176 – 1211, 2013. (cited on pages 26 and 35)

Christian Bors, Theresia Gschwandtner, and Silvia Miksch. Capturing and visualizing provenance from data wrangling. *IEEE Computer Graphics and Applications*, 39(6):61–75, 2019. (cited on page 29)

Geoffrey Boulton, Philip Campbell, Brian Collins, Peter Elias, Wendy Hall, Graeme Laurie, Onora O'Neill, Michael Rawlins, Janet Thornton, Patrick Vallance, et al. Science as an open enterprise. *The Royal Society*, 2012. (cited on pages 18 and 19)

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. (cited on page 83)

Brian Burns, James Lamb, and Jay Qi. *pkgnet: Get Network Representation of an R Package*, 2021. R package version 0.4.2. (cited on pages 10 and 70)

Scott Chamberlain and Kyle Voytovich. *charlatan: Make Fake Data*, 2020. R package version 0.4.0. (cited on page 136)

Gábor Csárdi and Winston Chang. *callr: Call R from R*, 2022. R package version 3.7.3. (cited on pages 69 and 72)

Alim Dayim. *consort: Create Consort Diagram*, 2023. R package version 1.2.1. (cited on page 30)

Matthew J. Denny and Arthur Spirling. Text preprocessing for unsupervised learning: Why it matters, when it misleads, and what to do about it. *Political Analysis*, 26(2):168–189, 2018. (cited on pages 17, 24, 25, 26, 27, and 35)

Matthew J. Denny and Arthur Spirling. *preText: Diagnostics to Assess the Effects of Text Preprocessing Decisions*, 2021. R package version 0.7.2. (cited on page 26)

Catherine D'Ignazio and Lauren F Klein. *Data Feminism.* MIT press, 2020. (cited on pages 39 and 121)

Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring Adult: New datasets for fair machine learning. *Advances in Neural Information Processing Systems*, 34:6478–6490, 2021. (cited on pages 17, 27, 81, 89, 90, 91, and 93)

Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 214–226, 2012. (cited on pages 91 and 92)

eBird. eBird: An online database of bird distribution and abundance [web application]. Cornell Lab of Ornithology, Ithaca, New York, 2023. URL `http://www.ebird.org`. (cited on page 87)

Madeleine Clare Elish and danah boyd. Situating methods in the magic of big data and AI. *Communication monographs*, 85(1):57–80, 2018. (cited on page 39)

Fair Housing Act. *42 U.S.C. 3601-3619*, 1968. URL `https://www.law.cornell.edu/uscode/text/42/chapter-45/subchapter-I`. (cited on page 96)

A. Famili, Wei-Min Shen, Richard Weber, and Evangelos Simoudis. Data preprocessing and intelligent data analysis. *Intelligent Data Analysis*, 1(1):3–23, 1997. (cited on pages 24, 32, and 33)

Paul Fitzpatrick, Edwin de Jonge, and Gregory R. Warnes. *daff: Diff, Patch and Merge for Data.frames*, 2023. R package version 1.0.1. (cited on page 43)

Sorelle A. Friedler, Carlos Scheidegger, Suresh Venkatasubramanian, Sonam Choudhary, Evan P. Hamilton, and Derek Roth. A comparative study of fairness-enhancing interventions in machine learning. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 329–338. Association for Computing Machinery, 2019. (cited on pages 17 and 27)

Salvador García, Julián Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining*. Springer, 2015. (cited on pages 24 and 32)

Brodie Gaslam. *diffobj: Diffs for R Objects*, 2021. R package version 0.3.5. (cited on page 43)

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021. (cited on pages 12, 19, 28, 117, and 118)

Baljinder Ghotra, Shane McIntosh, and Ahmed E Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 789–800, 2015. (cited on pages 17, 27, and 83)

Lisa Gitelman, editor. *"Raw Data" Is an Oxymoron*. MIT press, 2013. (cited on page 34)

David Gohel and Panagiotis Skintzos. *flextable: Functions for Tabular Reporting*, 2023. R package version 0.9.2. (cited on pages 69 and 74)

David Goodger and Guido van Rossum. Pep 257 – docstring conventions, 2001. URL `https://www.python.org/dev/peps/pep-0257/`. Accessed: May 20, 2024. (cited on page 58)

David Gray, David Bowes, Neil Davey, Yi Sun, and Bruce Christianson. The misuse of the NASA Metrics Data Program data sets for automated software defect prediction. In *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, pages 96–103, 2011. (cited on pages 11, 25, 82, 83, and 84)

David Gray, David Bowes, Neil Davey, Yi Sun, and Bruce Christianson. Reflections on the NASA MDP data sets. *IET Software*, 6(6):549–558, 2012. (cited on pages 25, 82, and 83)

Gurobi Optimization, LLC. *gurobi: Gurobi Optimizer 9.1 Interface*, 2021. R package version 9.1-2. (cited on pages 63 and 69)

Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, 2022. URL `https://www.gurobi.com`. (cited on pages 51 and 63)

Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2011. (cited on page 82)

Moritz Hardt, Eric Price, and Nathan Srebro. Equality of opportunity in supervised learning. *Advances in Neural Information Processing Systems*, 29, 2016. (cited on page 91)

Jim Hester. *covr: Test Coverage for Packages*, 2023. R package version 3.6.4. (cited on page 75)

Sarah Holland, Ahmed Hosny, Sarah Newman, Joshua Joseph, and Kasia Chmielinski. The Dataset Nutrition Label: A framework to drive higher data quality standards. *arXiv preprint arXiv:1805.03677*, 2018. (cited on pages 19 and 28)

Alison Johnston, Wesley M Hochachka, Matthew E Strimas-Mackey, Viviana Ruiz Gutierrez, Orin J Robinson, Eliot T Miller, Tom Auer, Steve T Kelling, and Daniel Fink. Analytical guidelines to increase the value of community science data: An example using eBird data to estimate species distributions. *Diversity and Distributions*, 27(7):1265–1277, 2021. (cited on pages 17, 25, 27, and 87)

Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. Association for Computing Machinery, 2011. (cited on page 29)

Sayash Kapoor, Emily M. Cantrell, Kenny Peng, Thanh Hien Pham, Christopher A. Bail, Odd Erik Gundersen, Jake M. Hofman, Jessica Hullman, Michael A. Lones, Momin M. Malik, Priyanka Nanayakkara, Russell A. Poldrack, Inioluwa Deborah Raji, Michael Roberts, Matthew J. Salganik, Marta Serra-Garcia, Brandon M. Stewart, Gilles Vandewiele, and Arvind Narayanan. REFORMS: Consensus-based recommendations for machine-learning-based science. *Science Advances*, 10(18):eadk3452, 2024. (cited on page 28)

Stephen Kasica, Charles Berret, and Tamara Munzner. Table scraps: An actionable framework for multi-table data wrangling from an artifact study of computational journalism. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):957–966, 2021. (cited on pages 35 and 43)

Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. The UCI Machine Learning Repository, 2023. URL `http://archive.ics.uci.edu/ml`. (cited on page 89)

Meraj Khan, Larry Xu, Arnab Nandi, and Joseph M Hellerstein. Data tweening: incremental visualization of data transforms. *Proceedings of the VLDB Endowment*, 10(6):661–672, 2017. (cited on page 29)

Richard A. Krueger and Mary Anne Casey. *Focus Groups: A Practical Guide for Applied Research*. SAGE Publications, 2000. (cited on pages 101, 102, 104, 106, and 115)

Sam Lau, Sean Kross, Eugene Wu, and Philip J. Guo. Teaching data science by visualizing data table transformations: Pandas tutor for Python, tidy data tutor for R, and SQL tutor. In *DataEd '23: Proceedings of the 2nd International Workshop on Data Systems Education: Bridging education practice with education research*, pages 50–55. Association for Computing Machinery, 2023. (cited on page 42)

Erin Leahey. The role of status in evaluating research: The case of data editing. *Social Science Research*, 33(3):521–537, 2004. (cited on pages 18 and 35)

Erin Leahey. Overseeing research practice: The case of data editing. *Science, Technology, & Human Values*, 33(5):605–630, 2008. (cited on page 18)

Jim Lemon. Plotrix: a package in the red light district of R. *R-News*, 6(4):8–12, 2006. (cited on page 69)

Barbara Lerner. *provSummarizeR: Summarizes Provenance Related to Inputs and Outputs of a Script or Console Commands*, 2022. R package version 1.5.1. (cited on page 29)

Barbara Lerner, Emery Boose, Orenna Brand, Aaron M. Ellison, Elizabeth Fong, Matthew Lau, Khanh Ngo, Thomas Pasquier, Luis A. Perez, Margo Seltzer, Rose Sheehan, and Joseph Wonsil. Making provenance work for you. *The R Journal*, 14:141–159, 2023. (cited on page 28)

Barbara S. Lerner and Emery R. Boose. RDataTracker: Collecting provenance in an interactive scripting environment. In *Proceedings of TAPP 2014*, 2014. (cited on pages 29 and 30)

Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008. (cited on page 83)

Roderick J. Little, Ralph D'Agostino, Michael L. Cohen, Kay Dickersin, Scott S. Emerson, John T. Farrar, Constantine Frangakis, Joseph W. Hogan, Geert Molenberghs, Susan A. Murphy, James D. Neaton, Andrea Rotnitzky, Daniel Scharfstein, Weichung J. Shih, Jay P. Siegel, and Hal Stern. The prevention and treatment of missing data in clinical trials. *New England Journal of Medicine*, 367(14):1355–1360, 2012. (cited on page 17)

Karen Lumsden, Jan Bradford, and Jackie Goode. Introduction: The reflexive turn and the social sciences. In *Reflexivity*, pages 1–24. Routledge, 2019. (cited on page 39)

Natasha S. Mauthner and Andrea Doucet. Reflexive accounts and accounts of reflexivity in qualitative data analysis. *Sociology*, 37(3):413–431, 2003. (cited on page 39)

T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4): 308–320, 1976. (cited on page 82)

Patricia McCoy. The Home Mortgage Disclosure Act: A synopsis and recent legislative history. *Journal of Real Estate Research*, 29(4):381–398, 2007. (cited on page 96)

Xiao-Li Meng. Enhancing (publications on) data quality: Deeper data minding and fuller data confession. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 2021. (cited on page 18)

Milagros Miceli, Tianling Yang, Laurens Naudts, Martin Schuessler, Diana Serbanescu, and Alex Hanna. Documenting computer vision datasets: An invitation to reflexive data practices. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 161–172, 2021. (cited on pages 18, 39, and 121)

Alan Mislove, Sune Lehmann, Yong-Yeol Ahn, Jukka-Pekka Onnela, and J. Rosenquist. Understanding the demographics of Twitter users. *Proceedings of the International AAAI Conference on Web and Social Media*, 5(1):554–557, 2021. (cited on pages 18 and 33)

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for

model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019. (cited on pages 19, 28, and 117)

David Moher, Kenneth F Schulz, and Douglas G Altman. The CONSORT statement: revised recommendations for improving the quality of reports of parallel-group randomised trials. *The Lancet*, 357(9263):1191–1194, 2001. (cited on page 30)

Kirill Müller and Hadley Wickham. *tibble: Simple Data Frames*, 2023. R package version 3.2.1. (cited on page 87)

Robert G Newcombe. Two-sided confidence intervals for the single proportion: comparison of seven methods. *Statistics in Medicine*, 17(8):857–872, 1998. (cited on pages 11 and 92)

Christina Niederer, Holger Stitz, Reem Hourieh, Florian Grassinger, Wolfgang Aigner, and Marc Streit. TACO: visualizing changes in tables over time. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):677–686, 2017. (cited on page 43)

Orestis Papakyriakopoulos, Anna Seo Gyeong Choi, William Thong, Dora Zhao, Jerone Andrews, Rebecca Bourke, Alice Xiang, and Allison Koenecke. Augmented datasheets for speech datasets and ethical decision-making. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pages 881–904. Association for Computing Machinery, 2023. (cited on page 28)

Samir Passi and Steven Jackson. Data vision: Learning to see through algorithmic abstraction. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 2436–2447, 2017. (cited on page 18)

Thomas Lin Pedersen. *patchwork: The Composer of Plots*, 2023. R package version 1.1.3. (cited on pages 69 and 74)

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. (cited on page 91)

Jean Petrić, David Bowes, Tracy Hall, Bruce Christianson, and Nathan Baddoo. The jinx on the NASA software defect data sets. In *Proceedings of the 20th International Conference on*

Evaluation and Assessment in Software Engineering, pages 1–5, 2016. (cited on pages 25, 82, 83, and 85)

Jean-Christophe Plantin. Data cleaners for pristine datasets: Visibility and invisibility of data processors in social science. *Science, Technology, & Human Values*, 44(1):52–73, 2019. (cited on page 35)

Lindsay Poirier. Accountable data: The politics and pragmatics of disclosure datasets. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, page 1446–1456. Association for Computing Machinery, 2022. (cited on page 96)

Posit Software, PBC. Data transformation with dplyr :: Cheatsheet, 2023. URL `https://rstudio.github.io/cheatsheets/html/data-transformation.html`. Accessed: October 3, 2023. (cited on pages 42 and 43)

Posit Software, PBC. Posit cheatsheets, 2024. URL `https://posit.co/resources/cheatsheets/`. Accessed: January 4, 2024. (cited on page 67)

Posit team. *RStudio: Integrated Development Environment for R*. Posit Software, PBC, Boston, MA, 2023. URL `http://www.posit.co/`. (cited on pages 10, 57, and 58)

Xiaoying Pu, Sean Kross, Jake M Hofman, and Daniel G Goldstein. Datamations: Animated explanations of data analysis pipelines. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2021. (cited on page 29)

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2023. URL `https://www.R-project.org/`. (cited on page 69)

Alexander Radovic, Mike Williams, David Rousseau, Michael Kagan, Daniele Bonacorsi, Alexander Himmel, Adam Aurisano, Kazuhiro Terao, and Taritree Wongjirad. Machine learning at the energy and intensity frontiers of particle physics. *Nature*, 560(7716):41–48, 2018. (cited on page 34)

Kay A Robbins, Jonathan Touryan, Tim Mullen, Christian Kothe, and Nima Bigdely-Shamlo. How sensitive are EEG results to preprocessing methods: A benchmarking study. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28(5):1081–1090, 2020. (cited on pages 17, 27, and 35)

Negar Rostamzadeh, Diana Mincu, Subhrajit Roy, Andrew Smart, Lauren Wilcox, Mahima Pushkarna, Jessica Schrouff, Razvan Amironesei, Nyalleng Moorosi, and Katherine Heller. Healthsheet: Development of a transparency artifact for health datasets. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1943–1961. Association for Computing Machinery, 2022. (cited on page 28)

Donald B Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976. (cited on page 97)

Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. "Everyone wants to do the model work, not the data work": Data cascades in high-stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2021. (cited on pages 16 and 18)

Abhraneel Sarma, Alex Kale, Michael Jongho Moon, Nathan Taback, Fanny Chevalier, Jessica Hullman, and Matthew Kay. Multiverse: Multiplexing alternative data analyses in R notebooks. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2023. (cited on page 26)

J. Sayyad Shirabad and T.J. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005. URL `http://promise.site.uottawa.ca/SERepository`. (cited on page 83)

Kenneth F. Schulz, Douglas G. Altman, David Moher, and the CONSORT Group. CONSORT 2010 statement: updated guidelines for reporting parallel group randomised trials. *BMC Medicine*, 8(1):18, 2010. (cited on page 30)

Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. Data quality: Some comments on the NASA software defect datasets. *IEEE Transactions on Software Engineering*, 39(9):1208–1215, 2013. (cited on pages 25, 82, 83, and 85)

Steven D. Shirk, Donald G. McLaren, Jessica S. Bloomfield, Alex Powers, Alec Duffy, Meghan B. Mitchell, Ali Ezzati, Brandon A. Ally, and Alireza Atri. Inter-rater reliability of preprocessing EEG data: Impact of subjective artifact removal on associative memory task ERP results. *Frontiers in Neuroscience*, 11, 2017. (cited on pages 17, 25, and 27)

Sara Steegen, Francis Tuerlinckx, Andrew Gelman, and Wolf Vanpaemel. Increasing transparency through a multiverse analysis. *Perspectives on Psychological Science*, 11(5):702–712, 2016. (cited on pages 17, 18, 26, 27, and 35)

Matthew Strimas-Mackey, Wesley M. Hochachka, Viviana Ruiz-Gutierrez, Orin J. Robinson, Eliot T. Miller, Tom Auer, Steve Kelling, Daniel Fink, and Alison Johnston. *Best Practices for Using eBird Data*. Cornell Lab of Ornithology, Ithaca, New York, 2023. URL `https://ebird.github.io/ebird-best-practices/`. Version 2.0. (cited on pages 11, 15, 25, 87, 88, and 143)

Brian L. Sullivan, Christopher L. Wood, Marshall J. Iliff, Rick E. Bonney, Daniel Fink, and Steve Kelling. eBird: A citizen-based bird observation network in the biological sciences. *Biological Conservation*, 142(10):2282–2292, 2009. (cited on pages 81 and 85)

Deborah F Swayne and Andreas Buja. Missing data in interactive high-dimensional data visualization. *Computational Statistics*, 13(1):15–26, 1998. (cited on page 47)

Cong Tang, Keith Ross, Nitesh Saxena, and Ruichuan Chen. What's in a name: A study of names, gender inference, and gender behavior in Facebook. In *International Conference on Database Systems for Advanced Applications*, pages 344–356. Springer, 2011. (cited on pages 18 and 33)

Chakkrit Tantithamthavorn. NASADefectDataset, 2016. URL `https://github.com/klainfo/NASADefectDataset`. Accessed: November 23, 2023. (cited on page 84)

Anissa Tanweer, Emily Kalah Gade, P.M. Krafft, and Sarah Dreier. Why the data revolution needs qualitative thinking. *Harvard Data Science Review*, 3(3), 2021. (cited on pages 39 and 121)

Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2023. R package version 4.1.23. (cited on page 34)

Nicholas Tierney. visdat: Visualising whole data frames. *JOSS*, 2(16):355, 2017. (cited on pages 43 and 47)

Nicholas Tierney and Dianne Cook. Expanding tidy data principles to facilitate missing data exploration, visualization and assessment of imputations. *Journal of Statistical Software*, 105(7):1–31, 2023. (cited on page 47)

Kevin Ushey, JJ Allaire, and Yuan Tang. *reticulate: Interface to 'Python'*, 2023. R package version 1.31. (cited on pages 69 and 73)

Joaquin Vanschoren and Serena Yeung. Announcing the NeurIPS 2021 datasets and benchmarks track. *NeurIPS Blog*, 2021. URL `https://blog.neurips.cc/2021/04/07/announcing-the-neurips-2021-datasets-and-benchmarks-track/`. (cited on page 19)

Lars Vilhuber. Report by the AEA data editor. *AEA Papers and Proceedings*, 111:808–17, 2021. (cited on page 19)

Shirly Wang, Matthew B. A. McDermott, Geeticka Chauhan, Marzyeh Ghassemi, Michael C. Hughes, and Tristan Naumann. MIMIC-Extract: A data extraction, preprocessing, and representation pipeline for MIMIC-III. In *Proceedings of the ACM Conference on Health, Inference, and Learning*, pages 222–235. Association for Computing Machinery, 2020. (cited on page 25)

Zezhong Wang, Jacob Ritchie, Jingtao Zhou, Fanny Chevalier, and Benjamin Bach. Data comics for reporting controlled user studies in human-computer interaction. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):967–977, 2021. (cited on page 29)

Marijke Welvaert and Peter Caley. Citizen surveillance for environmental monitoring: combining the efforts of citizen science and crowdsourcing in a quantitative data framework. *SpringerPlus*, 5(1):1890, 2016. (cited on page 87)

James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. The What-If Tool: Interactive probing of machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):56–65, 2020. (cited on page 26)

Hadley Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. (cited on page 74)

Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. (cited on pages 69 and 74)

Hadley Wickham and Jennifer Bryan. *R Packages*. O'Reilly Media, Inc., 2023. (cited on page 75)

Hadley Wickham, Peter Danenberg, Gábor Csárdi, and Manuel Eugster. *roxygen2: In-Line Documentation for R*, 2020. R package version 7.1.1. (cited on pages 58 and 67)

Hadley Wickham, Jay Hesselberth, and Maëlle Salmon. *pkgdown: Make Static HTML Documentation for a Package*, 2022a. R package version 2.0.7. (cited on page 69)

Hadley Wickham, Jim Hester, Winston Chang, and Jennifer Bryan. *devtools: Tools to Make Developing R Packages Easier*, 2022b. R package version 2.4.5. (cited on page 66)

Claus O. Wilke and Brenton M. Wiernik. *ggtext: Improved Text Rendering Support for 'ggplot2'*, 2022. R package version 0.1.2. (cited on pages 69 and 74)

Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1):160018, 2016. (cited on page 19)

Yihui Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2023. R package version 1.45. (cited on pages 69 and 72)

Ke Yang, Biao Huang, Julia Stoyanovich, and Sebastian Schelter. Fairness-aware instrumentation of preprocessing pipelines for machine learning. In *Workshop on Human-In-the-Loop Data Analytics (HILDA'20)*, 2020. (cited on pages 26 and 30)

Nicholas E Young, Ryan S Anderson, Stephen M Chignell, Anthony G Vorster, Rick Lawrence, and Paul H Evangelista. A survival guide to Landsat preprocessing. *Ecology*, 98(4):920–932, 2017. (cited on pages 23, 25, and 35)

Achim Zeileis, Jason C. Fisher, Kurt Hornik, Ross Ihaka, Claire D. McWhite, Paul Murrell, Reto Stauffer, and Claus O. Wilke. colorspace: A toolbox for manipulating and assessing colors and palettes. *Journal of Statistical Software*, 96(1):1–49, 2020. (cited on page 69)

Zhenpeng Zhao, Rachael Marr, and Niklas Elmqvist. Data comics: Sequential art for data-driven storytelling. 2015. Human-Computer Interaction Lab (HCIL) Technical Report, University of Maryland. (cited on page 29)

Yeyi Zhu, Ladia M. Hernandez, Peter Mueller, Yongquan Dong, and Michele R. Forman. Data acquisition and preprocessing in studies on humans: What is not taught in statistics classes? *The American Statistician*, 67(4):235–241, 2013. (cited on pages 24 and 33)

Appendix A: The s_data dataset

A synthetic dataset—named *s_data*[1]—and preprocessing scenario were generated for use in Chapters 4 and 5, to explain Smallset Timelines and `smallsets`, respectively. Appendix A.1 details data generation, and Appendix A.2 details data preprocessing.

A.1 Data generation

The s_data dataset was generated in R, using functions and methods from the `charlatan` package [Chamberlain and Voytovich, 2020] (see Listing A.1). It contains 100 observations and eight variables, *C1-C8*. The dataset is simple and nondescript for two reasons: 1) to minimise confusion over data content, when explaining Smallset Timelines and `smallsets`, and 2) to introduce Smallset Timelines and `smallsets` without association to one particular domain. The full dataset is provided in Table A.1.

```
library(charlatan)

set.seed(7)

NA_vector <- MissingDataProvider$new()

s_data <- data.frame(
  C1 = ch_integer(n = 100, min = 1, max = 5),
  C2 = sample(
    c(TRUE, FALSE),
    size = 100,
    replace = TRUE,
    prob = c(.9, .1)
  ),
  C3 = ch_integer(n = 100, min = 20, max = 40),
  C4 = ch_integer(n = 100, min = 100, max = 200),
  C5 = round(ch_norm(
    n = 100, mean = 5, sd = 1
  ), 2),
  C6 = round(NA_vector$make_missing(ch_norm(
    n = 100, mean = 10, sd = 3
  )), 2),
  C7 = round(NA_vector$make_missing(ch_norm(
    n = 100, mean = 0, sd = 1
  )), 2),
  C8 = round(NA_vector$make_missing(ch_norm(
    n = 100, mean = 0, sd = 1
  )), 2)
)
```

Listing A.1: R code for generating the s_data dataset.

[1]The *s* stands for *synthetic*.

Table A.1: The s_data dataset, printed in full, rows 1-100 and columns C1-C8.

	C1	C2	C3	C4	C5	C6	C7	C8
1	2	TRUE	33	199	5.44	NA	0.99	NA
2	3	TRUE	31	161	4.84	6.74	1.24	1.22
3	4	FALSE	33	188	5.97	9.35	-0.04	0.59
4	2	FALSE	24	129	4.33	7.8	NA	NA
5	2	TRUE	32	115	6.64	10.64	NA	NA
6	3	TRUE	22	101	3.93	7.18	-1.12	-0.63
7	3	FALSE	38	131	4.99	8.14	-0.84	-2.12
8	2	TRUE	30	183	3.13	10.73	NA	0.25
9	4	TRUE	34	159	5.02	11.12	NA	0.13
10	3	TRUE	36	157	4.75	11.88	NA	-0.98
11	4	TRUE	20	138	4.08	6.78	-0.21	-0.51
12	2	TRUE	40	156	2.83	7.87	NA	0.94
13	3	TRUE	38	132	4.35	12.72	NA	-1.56
14	5	TRUE	32	187	5.53	9.04	1	-1.09
15	4	TRUE	30	100	4.72	13.5	-0.77	-0.94
16	3	TRUE	25	127	5.05	8.13	NA	-1.83
17	2	TRUE	29	179	4.61	13.74	NA	0.75
18	2	TRUE	26	101	4.58	14.32	NA	0.1
19	4	TRUE	35	191	3.84	12.29	-1.22	1.14
20	3	TRUE	25	195	6.74	11.68	-1.82	NA
21	4	FALSE	21	109	4.75	14.8	-0.4	-0.45
22	1	TRUE	33	191	4.03	9.7	NA	-0.83
23	3	TRUE	26	148	6.11	5.65	1.42	NA
24	4	TRUE	22	183	5.97	9.23	0.56	-0.58
25	1	TRUE	37	174	6.26	3.9	NA	-0.98
26	3	TRUE	29	108	6.78	11.76	NA	NA
27	5	TRUE	40	189	2.32	10.48	-0.24	0.09
28	5	TRUE	37	111	5.66	10.34	1.09	-1.56
29	1	TRUE	31	135	6.18	11.72	NA	-1.62
30	5	TRUE	34	131	5.39	7.1	-2.18	0.11
31	5	TRUE	24	176	3.81	10.95	0.88	-1.92
32	1	TRUE	23	191	4.64	9.48	NA	-0.76
33	2	TRUE	33	141	3.53	7.26	NA	NA
34	4	TRUE	27	125	3.91	4.46	-0.69	1
35	5	TRUE	39	118	4.34	12.01	NA	1.05
36	2	TRUE	36	133	7.26	9.67	0.28	-0.97
37	2	TRUE	35	198	5.09	14.7	NA	NA
38	2	TRUE	33	152	5.77	7.47	0.73	1.18
39	3	TRUE	40	158	4.39	13.29	NA	0.71
40	2	TRUE	21	191	5.15	10.9	-0.65	0.65
41	5	TRUE	39	173	4.41	7.67	-0.56	1.5
42	3	TRUE	22	196	4.04	9.88	NA	1.15
43	5	TRUE	22	178	4.42	13.33	NA	NA
44	1	TRUE	22	120	5.66	11.61	NA	-1.45
45	4	TRUE	21	184	4.71	8.91	0.06	0.53
46	2	TRUE	23	142	5.9	9.68	-0.15	-1.77
47	2	TRUE	27	129	5.2	NA	NA	-1.35
48	5	TRUE	34	157	4.95	12.43	-1.26	1
49	2	TRUE	37	111	4.09	4.95	-0.68	NA
50	5	TRUE	27	168	3.86	12	-0.16	-0.6

Continued in the next column.

Continued from the previous column.

	C1	C2	C3	C4	C5	C6	C7	C8
51	1	TRUE	33	197	5.99	7.62	NA	1.27
52	1	TRUE	21	154	5.11	14.58	NA	2.62
53	4	FALSE	35	191	6.15	5.54	NA	-1.43
54	4	FALSE	36	114	4.09	7.99	1.54	-0.81
55	1	FALSE	36	102	3.9	15.03	NA	NA
56	3	TRUE	22	175	3.37	11.84	NA	NA
57	5	TRUE	33	128	3.38	9.3	-0.74	NA
58	4	TRUE	36	147	4.16	11.82	NA	0.38
59	4	TRUE	24	173	6.09	6.52	-0.3	0.78
60	2	TRUE	25	103	6.24	6.51	-1.44	-0.72
61	1	TRUE	33	133	5.17	7.39	NA	1.17
62	1	TRUE	33	160	5.12	9.22	1.56	0.99
63	1	TRUE	40	164	5.56	8.52	NA	2.09
64	4	TRUE	29	124	5.49	6.08	NA	-0.22
65	3	TRUE	39	117	4.93	16.07	-0.04	-1.07
66	2	FALSE	29	176	3.65	7.73	-0.69	0.26
67	1	TRUE	29	119	5.8	9.28	-0.13	0.96
68	2	TRUE	20	160	4.68	NA	0.16	0.53
69	1	TRUE	22	184	4.64	7.16	NA	-0.82
70	1	TRUE	40	176	4.49	6.6	1.05	0.37
71	3	TRUE	28	130	3.12	7.35	NA	-2.27
72	1	TRUE	34	176	4.05	11.55	-0.1	2.57
73	4	TRUE	40	177	7.27	8.99	2.5	-0.59
74	3	FALSE	20	112	5.2	9.93	-0.78	-0.16
75	4	TRUE	22	199	2.69	9.69	0.76	NA
76	5	TRUE	24	174	4.94	10.9	NA	-1.53
77	4	TRUE	26	142	5.06	12.42	NA	0.04
78	1	TRUE	30	143	5.71	9.89	-0.89	-1.1
79	2	TRUE	29	130	4.41	5.63	-0.61	0.15
80	3	TRUE	22	176	5.3	NA	-0.21	NA
81	2	TRUE	40	106	5.64	9.49	1.47	-2.03
82	5	TRUE	34	160	7.11	13.83	NA	-1.1
83	2	TRUE	20	192	5.92	9.23	1.82	NA
84	5	TRUE	28	151	3.43	9.5	NA	-2.15
85	3	TRUE	38	190	5.99	8.72	-0.54	0.39
86	2	TRUE	37	142	5.48	9.83	NA	-0.18
87	4	TRUE	29	103	4.7	5.66	-0.86	-1.54
88	2	TRUE	35	108	5.82	15.65	-0.4	-1.9
89	2	TRUE	26	141	4	10.35	NA	-1.52
90	5	TRUE	29	106	5.55	11.26	0.08	-1.26
91	1	TRUE	38	130	5.28	14.73	NA	NA
92	5	TRUE	20	196	5.92	9.25	NA	0.02
93	1	TRUE	28	141	3.27	9.83	0.36	NA
94	4	TRUE	26	120	3.52	9.93	-0.52	0
95	3	TRUE	23	122	7.23	16.99	NA	0.97
96	3	TRUE	36	118	5.39	9.7	1.62	-0.97
97	5	TRUE	25	156	5.66	10.5	0.45	NA
98	2	TRUE	23	197	4.72	9.19	NA	0
99	4	FALSE	24	191	5.28	12.53	-0.57	0.96
100	5	TRUE	40	134	2.73	14.23	0.32	-0.79

A.2 Data preprocessing

A simple three-step preprocessing scenario was generated for s_data (Table A.2). These steps were crafted to efficiently illustrate different features of Smallset Timelines and `smallsets`. For example, all three types of data changes in the Smallset Timeline colour legend—addition, deletion, and edit—occur in the steps. To illustrate the resume marker enrichment feature in Section 4.2.2, the preprocessing scenario is extended by one step (Table A.2). Listing A.2 shows the steps coded in R, with the fourth step commented out at the bottom.

Preprocessing step	**Operation(s)**
1. Filtering data	Drop observations where C2 is *FALSE*.
2. Dealing with missing data	In C6 and C8, impute the variable mean. Drop C7.
3. Generating a new variable	Sum C3 and C4 to create C9.
Extension for the resume marker	
4. Generating a new variable	Split C9 into terciles to create C10.

Table A.2: Three-step preprocessing scenario for s_data, with a fourth-step extension to illustrate the resume marker enrichment feature. Listing A.2 is the implementation in R.

```r
# step 1: filtering data
s_data <- s_data[s_data$C2 == TRUE,]

# step 2: dealing with missing data
s_data$C6[is.na(s_data$C6)] <- mean(s_data$C6, na.rm = TRUE)
s_data$C8[is.na(s_data$C8)] <- mean(s_data$C8, na.rm = TRUE)
s_data$C7 <- NULL

# step 3: generating a new variable
s_data$C9 <- s_data$C3 + s_data$C4


# # extension for resume marker
# # step 4: generating a new variable
# t <- quantile(s_data$C9, c(0:3 / 3))
# s_data$C10 = with(s_data,
#                   cut(
#                     C9,
#                     t,
#                     include.lowest = T,
#                     labels = c("Low",
#                                "Med",
#                                "High")
#                   ))
```

Listing A.2: R code for preprocessing the s_data dataset.

Appendix B: Scripts for Smallset Timelines

B.1 Materials for Figure 6.1

```
# smallsets snap 15 cm1 caption[Remove columns that have the same value for
# every row because they do not provide any information for modelling.]caption

# smallsets snap 26 cm1 caption[Replace missing *DECISION_DENSITY* values with zero.
# Based on other MDP datasets without missing *DECISION_DENSITY* values, one can deduce that
# they likely occurred due to a division by zero error and can be replaced with zeros.]caption

# smallsets snap 34 cm1 caption[Remove rows that are duplicates of other rows to
# assure models are tested on unseen data only. Also remove rows that are inconsistent, meaning
# all column values are the same except for the class label (one is classified as defective and
# the other is not).]caption

# smallsets snap 42 cm1 caption[The CM1 dataset is now ready for use in modelling.]caption

# step 1: remove constant attributes
for (c in colnames(cm1)) {
  if (length(unique(cm1[, c])) == 1) {
    cm1[, c] <- NULL
  }
}

# step 2: remove repeated attributes
cm1 <- cm1[!duplicated(as.list(cm1))]

# step 3: replace missing values
cm1$DECISION_DENSITY[is.na(cm1$DECISION_DENSITY)] <- 0

# step 4: run integrity checks
cm1 <- subset(cm1, HALSTEAD_LENGTH == NUM_OPERANDS + NUM_OPERATORS)
cm1 <- subset(cm1, CYCLOMATIC_COMPLEXITY <= NUM_OPERATORS + 1)
cm1 <- subset(cm1, CALL_PAIRS <= NUM_OPERATORS)

# step 5: remove duplicates and inconsistent cases
# only keep first instance of a duplicate
d <- cm1[duplicated(cm1), ]
cm1 <- cm1[!rownames(cm1) %in% rownames(d), ]

# remove all inconsistent cases
l <- which(colnames(cm1) == "Defective")
i <- cm1[duplicated(cm1[-l]), -l]
cases <- cm1[do.call(paste0, cm1[-l]) %in% do.call(paste0, i), ]
cm1 <- cm1[!rownames(cm1) %in% rownames(cases), ]
```

Listing B.1: R preprocessing script (*preprocess_mdp_1.R*) for Figure 6.1, passed to `code` argument in Listing B.2.

```
# load smallsets
library(smallsets)

# load dataset
# data frame of NASA CM1 data
# 505 rows x 41 columns
cm1 <- readRDS(file = "CM1.rds")

# create a vector of column names to ignore
ignore <- colnames(cm1)
ignore <- ignore[!ignore %in% c(
  "CALL_PAIRS",
  "CYCLOMATIC_COMPLEXITY",
  "DECISION_DENSITY",
  "GLOBAL_DATA_COMPLEXITY",
  "GLOBAL_DATA_DENSITY",
  "HALSTEAD_LENGTH",
  "NUM_OPERATORS",
  "NUMBER_OF_LINES",
  "PATHOLOGICAL_COMPLEXITY",
  "Defective"
)]

# build a Smallset Timeline
Smallset_Timeline(
  data = cm1,
  code = "preprocess_mdp_1.R",    # see Listing B.1
  rowCount = 6,
  rowSelect = 1,
  ignoreCols = ignore,
  printedData = TRUE,
  missingDataTints = TRUE,
  font = "Times",
  sizing = sets_sizing(
    columns = 2,
    captions = 4,
    data = 2,
    legend = 15,
    icons = 1
  ),
  spacing = sets_spacing(
    degree = 45,
    header = 7,
    right = 4,
    captions = 9
  ),
  labelling = sets_labelling(labelColDif = 1)
)
```

Listing B.2: The `smallsets` code for Figure 6.1.

B.2 Materials for Figure 6.2

```
# smallsets snap cm1 caption[**Step 1:** All constant attributes are removed from the dataset.
# These attributes do not offer any useful information when building the classifier. There are
# three constant attributes in the CM1 dataset: *GLOBAL_DATA_COMPLEXITY*, *GLOBAL_DATA_DENSITY*,
# and *PATHOLOGICAL_COMPLEXITY*.]caption
for (c in colnames(cm1)) {
  if (length(unique(cm1[, c])) == 1) {
    cm1[, c] <- NULL
  }
}

# smallsets snap +1 cm1 caption[**Step 2:** Remove repeated attributes. There are none in
# CM1.]caption
cm1 <- cm1[!duplicated(as.list(cm1))]

# smallsets snap +1 cm1 caption[**Step 3:** The *DECISION_DENSITY* attribute is the only
# attribute that contains missing values in CM1. This attribute is equal to the *CONDITION_COUNT*
# divided by the *DECISION_COUNT*. Missing values only occur in *DECISION_DENSITY* when
# both of these other attributes equal zero. In other MDP datasets, where both equal zero,
# so does *DECISION_DENSITY*. Thus, the missing values are replaced with zero. <br>
# (161 rows affected)]caption
cm1$DECISION_DENSITY[is.na(cm1$DECISION_DENSITY)] <- 0

# smallsets snap cm1 caption[**Step 4:** All instances which fail one or more of the data
# integrity checks are removed from the dataset. The integrity checks identify instances
# that cannot realistically happen. The following integrity checks are used: <br>
# *NUM_OPERANDS* + *NUM_OPERATORS* = *HALSTEAD_LENGTH*; <br>
# *CYCLOMATIC_COMPLEXITY* <= *NUM_OPERATORS* + 1; <br>
# *CALL_PAIRS* <= *NUM_OPERATORS*. <br>
# (0 rows affected)]caption
cm1 <- subset(cm1, HALSTEAD_LENGTH == NUM_OPERANDS + NUM_OPERATORS)
cm1 <- subset(cm1, CYCLOMATIC_COMPLEXITY <= NUM_OPERATORS + 1)
cm1 <- subset(cm1, CALL_PAIRS <= NUM_OPERATORS)

# smallsets snap cm1 caption[**Step 5:** Remove repeated and inconsistent cases. Duplicates are
# dropped. <br> (49 rows affected) <br> <br> Inconsistent cases refer to cases in which all values
# but the class label are equal, and these are dropped as well. <br> (2 rows affected)]caption

# only keep first instance of a duplicate
d <- cm1[duplicated(cm1), ]
cm1 <- cm1[!rownames(cm1) %in% rownames(d), ]

# remove all inconsistent cases
l <- which(colnames(cm1) == "Defective")
i <- cm1[duplicated(cm1[-l]), -l]
cases <- cm1[do.call(paste0, cm1[-l]) %in% do.call(paste0, i), ]
cm1 <- cm1[!rownames(cm1) %in% rownames(cases), ]

# smallsets snap cm1 caption[The CM1 dataset is preprocessed and ready for modelling.]caption
```

Listing B.3: R preprocessing script (*preprocess_mdp_2.R*) for Figure 6.2, passed to `code` argument in Listing B.4.

```
# load smallsets
library(smallsets)

# load dataset
# data frame of NASA CM1 data
# 505 rows x 41 columns
cm1 <- readRDS(file = "CM1.rds")

# create a vector of column names to ignore
ignore <- colnames(cm1)
ignore <- ignore[!ignore %in% c(
  "CALL_PAIRS",
  "CYCLOMATIC_COMPLEXITY",
  "DECISION_DENSITY",
  "GLOBAL_DATA_COMPLEXITY",
  "GLOBAL_DATA_DENSITY",
  "HALSTEAD_LENGTH",
  "NUM_OPERATORS",
  "NUMBER_OF_LINES",
  "PATHOLOGICAL_COMPLEXITY",
  "Defective"
)]

# build a Smallset Timeline
Smallset_Timeline(
  data = cm1,
  code = "preprocess_mdp_2.R",    # see Listing B.3
  rowCount = 10,
  rowSelect = 2,
  ignoreCols = ignore,
  printedData = TRUE,
  missingDataTints = TRUE,
  font = "Times",
  sizing = sets_sizing(
    columns = 3.5,
    captions = 5.5,
    data = 3,
    legend = 20,
    icons = 1.5
  ),
  spacing = sets_spacing(
    degree = 45,
    header = 5,
    right = 4.5,
    captions = 9,
    rows = 2
  ),
  labelling = sets_labelling(labelColDif = 1)
)
```

Listing B.4: The `smallsets` code for Figure 6.2.

B.3 Materials for Figure 6.3

```
# smallsets snap zf caption[Eight (of >60,000) rows and nine (of 35) columns from an eBird dataset,
# used to illustrate the preprocessing steps run prior to encounter modelling.]caption
time_to_decimal <- function(x) {
  x <- hms(x, quiet = TRUE)
  hour(x) + minute(x) / 60 + second(x) / 3600
}

# smallsets snap +18 zf caption[**STEP 1:**<br>
# -- Convert observation counts into integers.<br>
# -- Edit effort distance to be zero for stationary protocols.<br>
# -- Create effort hours variable, by converting duration to hours.<br>
# -- Create effort speed variable, by dividing effort distance by effort hours.<br>
# -- Create hours of day variable (decimal hours since midnight), based on observation start time.<br>
# -- Create year and day-of-year variables, based on the observation date.]caption
zf <- zf %>%
  mutate(
    observation_count = as.integer(observation_count),
    effort_distance_km = if_else(protocol_type == "Stationary", 0, effort_distance_km),
    effort_hours = duration_minutes / 60,
    effort_speed_kmph = effort_distance_km / effort_hours,
    hours_of_day = time_to_decimal(time_observations_started),
    year = year(observation_date),
    day_of_year = yday(observation_date)
)

# smallsets snap zf caption[**STEP 2:** <br> Filter to the following: <br>
# -- protocol is stationary or traveling; <br>
# -- effort hours is <= 6; <br>
# -- effort distance is <= 10; <br>
# -- effort speed is <= 100; <br>
# -- number of observers is <= 10.]caption
zf_filtered <- zf %>%
  rownames_to_column() %>%
  filter(protocol_type %in% c("Stationary", "Traveling"),
         effort_hours <= 6,
         effort_distance_km <= 10,
         effort_speed_kmph <= 100,
         number_observers <= 10) %>%
  column_to_rownames()

# smallsets snap +2 zf_split caption[**STEP 3:** <br> Randomly assign 80% and 20% of the dataset into
# a train and test set, respectively, and create a new type variable with the assignment "train" or
# "test."]caption
zf_split <- zf_filtered %>% mutate(type = if_else(runif(nrow(.)) <= 0.8, "train", "test"))

# smallsets snap zf_split caption[**STEP 4:** <br> Drop 22 columns not needed for modelling, keeping
# only the 19 columns shown in the next snapshot.]caption
checklists <- zf_split %>% select(checklist_id, observer_id, type,
                                  observation_count, species_observed, state_code,
                                  locality_id, latitude, longitude, protocol_type,
                                  all_species_reported, observation_date, year,
                                  day_of_year, hours_of_day, effort_hours, effort_distance_km,
                                  effort_speed_kmph, number_observers)

# smallsets snap checklists caption[The initial preprocessing of this eBird dataset is complete. Next, a
# series of land cover covariates will be created from satellite data and merged onto this dataset, based
# on locality and year. The dataset will then be ready for encounter rate modelling.]caption
```

Listing B.5: R preprocessing code (*preprocess_ebird.R*) **copied from 2.6-2.8 in Strimas-Mackey et al. [2023]** (except for the red code, added to preserve row names) and supplemented with structured comments for Figure 6.3. Passed to `code` in Listing B.6.

```
# load packages
library(tidyverse)
library(smallsets)

# load dataset
# data frame of zero-filled detection/non-detection eBird data
ebird <- readRDS(file = "ebird.rds")

# create a vector of column names to ignore
ignore <- colnames(ebird)[!colnames(ebird) %in% c(
  "checklist_id",
  "observer_id",
  "locality_id",
  "time_observations_started",
  "observation_count",
  "species_observed",
  "state_code",
  "locality_id",
  "latitude",
  "longitude",
  "protocol_type",
  "all_species_reported",
  "observation_date",
  "duration_minutes",
  "effort_distance_km",
  "number_observers"
)]

# build a Smallset Timeline
Smallset_Timeline(
  data = ebird,
  code = "preprocess_ebird.R",    # see Listing B.5
  rowCount = 8,
  rowIDs = c(    # selected through random sampling
    "1364",
    "1376",
    "1682",
    "1830",
    "32400",
    "36377",
    "39594",
    "51926"
  ),
  ignore = ignore,
  colours = list(
    unchanged = "#87b8ea",
    edited = "#2E4053",
    added = "#0a75ad",
    deleted = "#ff4040"
  ),
  ghostData = F,
  missingDataTints = T,
  font = "Ayuthaya",
  sizing = sets_sizing(
    captions = 2,
    columns = 1.5,
    legend = 10
  ),
  spacing = sets_spacing(
    degree = 90,
    header = 7,
    right = 0,
    captions = 10,
    row = 2
  ),
  labelling = sets_labelling(labelColDif = 1)
)
```

Listing B.6: The `smallsets` code for Figure 6.3.

B.4 Materials for Figure 6.4

```
def validity_median(data):
    # smallsets snap data caption[Here, we have eight rows
    # from the 2015 ACS California income dataset (n=374,943),
    # retrieved with the folktables tool. The features, from
    # left to right, include age (AGEP), class of worker (COW),
    # educational attainment (SCHL), marital status (MAR),
    # occupation (OCCP), place of birth (POBP), relationship
    # (RELP), weekly hours worked (WKHP), sex (SEX), race
    # (RAC1P), income (PINCP), and survey weight (PWGBT). We
    # first filter to individuals older than 16 years of age
    # and with survey weights of at least one.]caption
    data = data[data["AGEP"] > 16]
    data = data[data["PWGTP"] >= 1]

    # smallsets snap +3 data caption[For this prediction task,
    # it is common to remove individuals with an income less
    # than 100 dollars or no reported average weekly hours
    # worked, but we want to keep these individuals in the
    # dataset. We do not filter by income, even keeping reported
    # losses (i.e., negative incomes). Missing values for weekly
    # hours worked are replaced with zero. We replace missing
    # values in the categorical features for class of worker
    # and occupation with negative one, i.e., create categories
    # for "no worker class" and "no occupation."]caption
    data["WKHP"] = data["WKHP"].fillna(0)
    data["COW"] = data["COW"].fillna(-1)
    data["OCCP"] = data["OCCP"].fillna(-1)

    # smallsets snap +2 data caption[The median income of the
    # dataset (after filtering), $22.5K, is used as the income
    # threshold to generate class labels for the prediction task.
    # These labels are in the new INCOME column. A **1** in INCOME
    # means an individual's income is greater than $22.5K (and
    # **0** otherwise). Next, we will test and train a logistic
    # regression model, to predict INCOME, based on the first ten
    # features shown (survey weights in PWGTP and the original
    # income values in PINCP are excluded from modelling).]caption
    income_threshold = data["PINCP"].median()
    data["INCOME"] = (data["PINCP"] > income_threshold).astype(int)

    return data
```

Listing B.7: Python preprocessing script (*preprocess_folktables.py*) for Figure 6.4, passed to `code` argument in Listing B.8.

```
# load packages
library(reticulate)
library(smallsets)

# load Python environment
use_condaenv("r-reticulate")

# load dataset
# data frame of California American Community Survey (ACS) data from folktables
# 374,943 rows x 12 (of 284) columns
ca_acs <- readRDS("ca_acs.Rds")

# build a Smallset Timeline
Smallset_Timeline(
  data = ca_acs,
  code = "preprocess_folktables.py",     # see Listing B.7
  rowCount = 8,
  rowIDs = c(    # selected through random sampling
    78493,
    157606,
    246297,
    266760,
    277902,
    311180,
    334372,
    347840),
  colours = list(
    unchanged = "#0e2f44",
    edited = "#A68156",
    added = "#20BE06",
    deleted = "#800080"
  ),
  missingDataTints = TRUE,
  font = "Futura",
  sizing = sets_sizing(
    captions = 5,
    columns = 4,
    legend = 19
  ),
  spacing = sets_spacing(
    degree = 45,
    header = 2.4,
    captions = 14,
    right = 2
  ),
  labelling = sets_labelling(labelCol = "darker", labelColDif = 0)
)
```

Listing B.8: The `smallsets` code for Figure 6.4.

B.5 Materials for Figure 6.8

```
# smallsets snap hmda caption[For our audit, we must first preprocess the 2019 HMDA
# data for Philadelphia County. The dataset includes loans categorised as conventional,
# first lien, home purchase, single family, site-built, non-commercial, and not a
# preapproval. Original column names have been abbreviated, e.g., applicant_race-1 -->
# race_1, application_ethnicity-1 --> eth_1, and action_taken --> action. The eth_4 and
# eth_5 columns were excluded because applicants only specified at most three ethnicities.
# The dataset contains 11,008 rows. Seven rows were selected, from a 5% random sample (n=550),
# using the coverage+variety optimisation algorithm provided in the smallsets software.]caption

# smallsets snap +24 hmda caption[We first classify who is non-Hispanic/Latino White
# and Hispanic/Latino White. Individuals can specify one or more races (race_1-5) and
# one or more ethnicities (eth_1-3). We create a dummy variable for White, where
# white = 1 if an applicant only selects White (race = 5) (e.g., an applicant with White
#(race_1 = 5) and Asian (race_2 = 2) would be white = 0). We also create a dummy
# variable for Hispanic/Latino (h_l), where h_l = 1 if an applicant selects
# Hispanic or Latino (eth = 1) or a specific origin group (eth = 11-14),
# and h_l = 0 for the selection of not Hispanic or Latino (eth = 2).]caption
hmda$white <- NA
for (i in 1:nrow(hmda)) {
  if (3 %in% hmda[i, 1:5]) {
    hmda$white[i] <- 0
  } else if (1 %in% hmda[i, 1:5]) {
    hmda$white[i] <- 0
  } else if (sum(c(4, 41, 42, 43, 44) %in% hmda[i, 1:5]) > 0) {
    hmda$white[i] <- 0
  } else if (sum(c(2, 21, 22, 23, 24, 25, 26, 27) %in% hmda[i, 1:5]) > 0) {
    hmda$white[i] <- 0
  } else if (5 %in% hmda[i, 1:5]) {
    hmda$white[i] <- 1
  }
}

hmda$h_l <- NA
for (i in 1:nrow(hmda)) {
  if (2 %in% hmda[i, 6:8]) {
    hmda$h_l[i] <- 0
  }
  if (sum(c(1, 11, 12, 13, 14) %in% hmda[i, 6:8]) > 0) {
    hmda$h_l[i] <- 1
  }
}

# smallsets snap hmda caption[We subset to those who are classified as White (white = 1).
# Next, we drop 138 rows with missing ethnicity information (h_l = NA).]caption
hmda <- subset(hmda, white == 1)
hmda <- subset(hmda, !is.na(h_l))

# smallsets snap +1 hmda caption[From the action variable, we create a dummy variable for
# denied loans (deny), where deny = 1 if the application was denied (action = 3). Loans
# are considered approved (deny = 0) if the loan originated (action = 1) or the application
# was approved but not accepted (action = 2). The preprocessed dataset consists of
# 6,157 applicants.]caption
hmda$deny <- ifelse(hmda$action %in% c(1, 2), 0, 1)
```

Listing B.9: R preprocessing script (*preprocess_hmda_A.R*) for Figure 6.8, passed to `code` argument in Listing B.10.

```
# load smallsets
library(smallsets)

# load dataset
# data frame of HMDA data (2019 Philadelphia County)
# 11,008 rows x 9 (of 99) columns, with abbreviated names
# following filters were applied before subsetting/renaming columns and saving R object (.rds):
  # -- derived_loan_product_type == "Conventional:First Lien"
  # -- derived_dwelling_category == "Single Family (1-4 Units):Site-Built"
  # -- loan_purpose == "1"
  # -- business_or_commercial_purpose == 2
  # -- action_taken %in% c(1, 2, 3)
hmda <- readRDS(file = "HMDA.rds")

# build a Smallset Timeline
Smallset_Timeline(
  data = hmda,
  code = "preprocess_hmda_A.R",    # see Listing B.9
  rowCount = 7,
# rowSelect = 2,    # run on a 5% random sample, result passed to rowIDs
  rowIDs = c("24", "34", "242", "513", "599", "5306", "28436"),
  colours = list(
    unchanged = "#CDCDB4",
    edited = "#C0FF3E",
    added = "#5BA2A6",
    deleted = "#BCD2EE"
  ),
  printedData = T,
  missingDataTints = T,
  font = "Geneva",
  sizing = sets_sizing(
    columns = 3,
    captions = 3,
    data = 3,
    legend = 10,
    icons = 1
  ),
  spacing = sets_spacing(
    degree = 45,
    header = 2,
    right = 1,
    captions = 18
  ),
  labelling = sets_labelling(labelColDif = 1)
)
```

Listing B.10: The `smallsets` code for Figure 6.8.

B.6 Materials for Figure 6.9

```
# smallsets snap hmda caption[For our audit, we must first preprocess the 2019 HMDA data for Philadelphia County.
# The dataset includes loans categorised as conventional, first lien, home purchase, single family, site-built,
# non-commercial, and not a preapproval. Original column names have been abbreviated, e.g., applicant_race-1 -->
# race_1, application_ethnicity-1 --> eth_1, and action_taken --> action. The eth_4 and eth_5 columns were excluded
# because applicants only specified at most three ethnicities. The dataset contains 11,008 rows. Seven rows were
# selected, from a 5% random sample (n=550), using the coverage+variety optimisation algorithm provided in the
# smallsets software. ]caption

# smallsets snap +24 hmda caption[We first classify who is non-Hispanic/Latino White and Hispanic/Latino White.
# Individuals can specify one or more races (race_1-5) and one or more ethnicities (eth_1-3). We create a dummy
# variable for White, where white = 1 if an applicant only selects White (race = 5) (e.g., an applicant with
# White (race_1 = 5) and Asian (race_2 = 2) would be white = 0). We also create a dummy variable for Hispanic/Latino
# (h_l), where h_l = 1 if an applicant selects Hispanic or Latino (eth = 1) or a specific origin group (eth =
# 11-14), and h_l = 0 for the selection of not Hispanic or Latino (eth = 2).]caption
hmda$white <- NA
for (i in 1:nrow(hmda)) {
  if (3 %in% hmda[i, 1:5]) {
    hmda$white[i] <- 0
  } else if (1 %in% hmda[i, 1:5]) {
    hmda$white[i] <- 0
  } else if (sum(c(4, 41, 42, 43, 44) %in% hmda[i, 1:5]) > 0) {
    hmda$white[i] <- 0
  } else if (sum(c(2, 21, 22, 23, 24, 25, 26, 27) %in% hmda[i, 1:5]) > 0) {
    hmda$white[i] <- 0
  } else if (5 %in% hmda[i, 1:5]) {
    hmda$white[i] <- 1
  }
}

hmda$h_l <- NA
for (i in 1:nrow(hmda)) {
  if (2 %in% hmda[i, 6:8]) {
    hmda$h_l[i] <- 0
  }
  if (sum(c(1, 11, 12, 13, 14) %in% hmda[i, 6:8]) > 0) {
    hmda$h_l[i] <- 1
  }
}

# smallsets snap +6 hmda caption[For the 17.9% (n=144) of Hispanics/Latinos missing race data (race = 6 (info
# not provided), 7 (info not applicable), or NA (info not available)), we impute White (white = 1), as the
# majority of Hispanics/Latinos in the dataset (82%) are White (white = 1). For the 2.6% (n=138) of Whites missing
# ethnicity data (eth = 3 (info not provided) or NA (info not available)), we impute non-Hispanic/Latino (h_l = 0),
# as the majority of Whites in the dataset (89%) are non-Hispanic/Latino (h_l = 0). After imputing missing race and
# ethnicity values, we subset to those classified as White (white = 1).]caption
for (i in 1:nrow(hmda)) {
  if ((sum(c(1, 11, 12, 13, 14) %in% hmda[i, 6:8]) > 0) & is.na(hmda$white[i])) {
    hmda$white[i] <- 1
  }
}
hmda$h_l <- ifelse(is.na(hmda$h_l), 0, hmda$h_l)
hmda <- subset(hmda, white == 1)

# smallsets snap +1 hmda caption[From the action variable, we create a dummy variable for denied loans (deny), where
# deny = 1 if the application was denied (action = 3). Loans are considered approved (deny = 0) if the loan originated
# (action = 1) or the application was approved but not accepted (action = 2). The preprocessed dataset consists of
# 6,157 applicants.]caption
hmda$deny <- ifelse(hmda$action %in% c(1, 2), 0, 1)
```

Listing B.11: R preprocessing script (*preprocess_hmda_B.R*) for Figure 6.9, passed to `code` argument in Listing B.12.

```
# load smallsets
library(smallsets)

# load dataset
# data frame of HMDA data (2019 Philadelphia County)
# 11,008 rows x 9 (of 99) columns, with abbreviated names
# following filters were applied before subsetting/renaming columns and saving R object (.rds):
  # -- derived_loan_product_type == "Conventional:First Lien"
  # -- derived_dwelling_category == "Single Family (1-4 Units):Site-Built"
  # -- loan_purpose == "1"
  # -- business_or_commercial_purpose == 2
  # -- action_taken %in% c(1, 2, 3)
hmda <- readRDS(file = "HMDA.rds")

# build a Smallset Timeline
Smallset_Timeline(
  data = hmda,
  code = "preprocess_hmda_B.R",    # see Listing B.11
  rowCount = 7,
# rowSelect = 2,    # run on a 5% random sample, result passed to rowIDs
  rowIDs = c("24", "7285", "28436", "33169", "49433", "53709", "63553"),
  colours = list(
    added = "#5BA2A6",
    deleted = "#BCD2EE",
    edited = "#C0FF3E",
    unchanged = "#CDCDB4"
  ),
  printedData = T,
  missingDataTints = T,
  font = "Geneva",
  sizing = sets_sizing(
    columns = 3,
    captions = 3,
    data = 3,
    legend = 15,
    icons = 1
  ),
  spacing = sets_spacing(
    degree = 45,
    header = 2,
    right = 1,
    captions = 15
  ),
  labelling = sets_labelling(labelColDif = 1)
)
```

Listing B.12: The `smallsets` code for Figure 6.9.