12 Store and share

Required material

- Read *Promoting Open Science Through Research Data Management*, (Borghi and Van Gulick 2022)
- Read *Transparent and reproducible social science research*, Chapter 10 "Data Sharing", (Christensen, Freese, and Miguel 2019)
- Read *Datasheets for datasets*, (Gebru et al. 2021)
- Read *Data and its (dis)contents: A survey of dataset development and use in machine learning research*, (Paullada et al. 2021)

Key concepts and skills

- The FAIR principles provide the foundation from which we consider data sharing and storage. These specify that data should be findable, accessible, interoperable, and reusable.
- The most important step is the first one, and that is to get the data off our local computer, and to then make it accessible by others. After that, we build documentation, and datasheets, to make it easier for others to understand and use it. Finally, we ideally enable access without our involvement.
- At the same time as wanting to share our datasets are widely as possible, we must respect those whose information are contained in them. This means, for instance, protecting, to a reasonable extent, and informed by costs and benefits, personally identifying information through selective disclosure, hashing, data simulation, and differential privacy.
- Finally, as our data get larger, approaches that were viable when they were smaller start to break down. We need to consider efficiency with regard to our data, and explore other approaches, formats, and languages.

Key packages and functions

- Base R

- `%%`—"modulo"

- `arrow` (Richardson et al. 2022)
  - `read_parquet()`
  - `write_parquet()`

- `openssl` (Ooms 2021)
  - `md5()`
  - `sha512()`

- `tictoc` (Izrailev 2014)
  - `tic()`
  - `toc()`

12.1 Introduction

After we have put together a dataset it is important to store it appropriately and enable easy retrieval both for ourselves and others. Wicherts, Bakker, and Molenaar (2011) found that a reluctance to share data was associated with research papers that had weaker evidence and more potential errors. While it is certainly possible to be especially concerned about this, and entire careers and disciplines are based on the storage and retrieval of data, to a certain extent, the baseline is not onerous. If we can get our dataset off our own computer, then we are much of the way there. Further confirming that someone else can retrieve it and use it, ideally without our involvement, puts us much further than most. Just achieving that for our data, models, and code, meets the "bronze" standard of Heil et al. (2021).

The FAIR principles are useful when we come to think more formally about data sharing and management. This requires that datasets are (Wilkinson et al. 2016):

1. Findable. There is one, unchanging, identifier for the dataset and the dataset has high-quality descriptions and explanations.
2. Accessible. Standardized approaches can be used to retrieve

the data, and these are open and free, possibly with authentication, and their metadata persist even if the dataset is removed.
3. Interoperable. The dataset and its metadata use a broadly applicable language and vocabulary.
4. Reusable. There are extensive descriptions of the dataset and the usage conditions are made clear along with provenance.

Just because a dataset is FAIR, it is not necessarily an unbiased representation of the world. Further, it is not necessarily fair in the everyday way that word is used i.e. impartial and honest (Lima et al. 2022). FAIR reflects whether a dataset is appropriately available, not whether it is appropriate.

One reason for the rise of data science is that humans are at the heart of it. And often the data that we are interested in directly concern humans. This means that there can be tension between sharing a dataset to facilitate reproducibility and maintaining privacy. Medicine developed approaches to this over a long time. And out of that we have seen Health Insurance Portability and Accountability Act (HIPAA) in the US, and then the more general General Data Protection Regulation (GDPR) in Europe. Our concerns in data science tend to be about personally identifying information (PII). We have a variety of ways to protect especially private information, such as emails and home addresses. For instance, we can hash those variables. Sometimes we may simulate data and distribute that instead of sharing the actual dataset. More recently, approaches based on differential privacy are being implemented. The fundamental problem of data privacy is that increased privacy reduces the usefulness of a dataset. The trade-off means the appropriate decision is nuanced and depends on costs and benefits, and we should be especially concerned about differentiated effects on population minorities.

Finally, in this chapter we consider efficiency. As datasets and code bases get larger it becomes more difficult to deal with them, especially if we want them to be shared. We come to concerns around efficiency, not for its own sake, but to enable us to tell stories that could not otherwise be told. This might mean moving

beyond CSV files to formats with other properties, or even using databases, such as SQL.

12.2 Plan

The storage and retrieval of information is especially connected with libraries. These have existed since antiquity and have well-established protocols for deciding what information to store and what to discard, as well as information retrieval. One of the defining aspects of libraries is deliberate curation and organization. The use of a cataloging system ensures that books on similar topics are located close to each other, and there are typically also deliberate plans for ensuring the collection is up-to-date. This ensures that information storage and retrieval is appropriate and efficient.

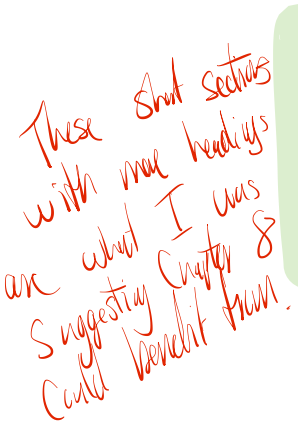
Data science relies heavily on the internet when it comes to storage and retrieval. Vannevar Bush, the twentieth-century engineer, defined a “memex” in 1945 as a device to store books, records, and communications, in a way that supplements memory (Bush 1945). The key to it was the indexing, or linking together, of items. We see this concept echoed just four decades later in the proposal by Tim Berners-Lee for hypertext (Berners-Lee 1989). This led to the World Wide Web and defines the way that resources are identified. They are then transported over the internet, using Hypertext Transfer Protocol (HTTP).

At its most fundamental, the internet is about storing and retrieving data. It is based on making various files on a computer available to others. When we consider the storage and retrieval of our datasets we want to especially contemplate for how long it is important that they are stored and for whom (Michener 2015). For instance, if we want some dataset to be available for a decade, and widely available, then it becomes important to store it in open and persistent formats, such as CSV (Hart et al. 2016). But if we are just using a dataset as part of an intermediate step, and we have the raw data and the scripts to create it, then it might be fine to not

worry too much about such considerations.

Storing raw data is important and there are many cases where raw data have revealed or hinted at fraud (Simonsohn 2013). Shared data also enhances the credibility of our work, by enabling others to verify it, and can lead to the generation of new knowledge as others use it to answer different questions (Christensen, Freese, and Miguel 2019). Christensen et al. (2019) suggests that research that shares its data may be more highly cited.

We invite scrutiny and make it as easy as possible for criticism. We do this even when it is the difficult choice and results in discomfort because we know that is the only way to contribute to the stock of lasting knowledge. For instance, Piller (2022) details potential fabrication in research about Alzheimer's disease, where one of the issues that researchers face when trying to understand whether the results are legitimate is a lack of access to unpublished images.



12.3 Share data

12.3.1 GitHub

The easiest place to get started with storing a dataset is GitHub because that is already built into our workflow. For instance, if we push a dataset to a public repository, then our dataset becomes available. One benefit of this is that if we have set-up our workspace appropriately, then we likely store our raw data, and the tidy data, as well as the scripts that are needed to transform one to the other. We are most of the way to the "bronze" standard of Heil et al. (2021) without changing anything.

As an example of how we have stored some data, we can access "raw_data.csv" from the "starter_folder", (which we recommend using for the papers in Appendix C). We navigate to the file in GitHub ("inputs" -> "data" -> "raw_data.csv"), and then click "Raw" (Figure 12.1).

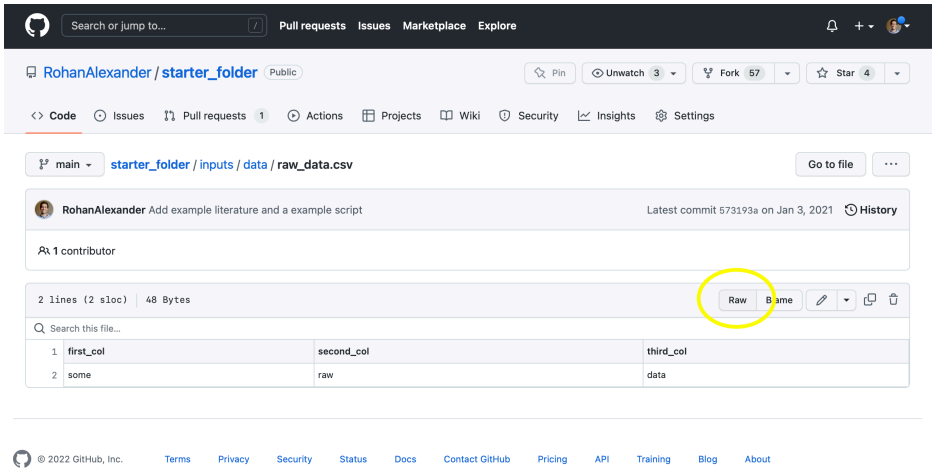


Figure 12.1: Getting the necessary link to be able to read a CSV from a GitHub repository

We can then add that URL as an argument to `read_csv()`.

```
library(tidyverse)

data_location <-
  paste0(
    "https://raw.githubusercontent.com/RohanAlexanc
    "starter_folder/main/inputs/data/raw_data.csv"
  )

starter_data <-
  read_csv(file = data_location)

starter_data
```

```
# A tibble: 1 × 3
  first_col second_col third_col
  <chr>     <chr>      <chr>
1 some      raw        data
```

While we can store and retrieve a dataset easily in this way, it lacks explanation, a formal dictionary, and aspects such as a license that would bring it closer to aligning with the FAIR principles. Another practical concern is that the maximum file size on GitHub is 100MB. And a final concern, for some, is that GitHub is owned by Microsoft, a for-profit US technology firm.

12.3.2 R packages for data

To this point we have largely used R packages for their code, although we have seen a few that were focused on sharing data, for instance, `troopdata` (Flynn 2021) and `babynames` (Wickham 2019) in Chapter 7. We can build an R package for our dataset and then add it to GitHub and potentially eventually CRAN. This will make it easy to store and retrieve because we can obtain the dataset by loading the package. In contrast to the CSV-based approach, it also means a dataset brings its documentation along with it. This will be the first R package that we build. In Chapter 14, we return to R packages and use them to deploy models.

To get started, create a new package ("File" -> "New project" -> "New Directory" -> "R Package"). Give the package a name, such as "favcolordata" and select "Open in new session". Create a new folder called "data". We simulate a dataset of people and their favourite colors to include in our R package.

```
library(tidyverse)

set.seed(853)

color_data <-
  tibble(
    name =
      c(
        "Edward",
        "Helen",
        "Hugo",
        "Ian",
        "Monica",
        "Myles",
        "Patricia",
        "Roger",
        "Rohan",
        "Ruth"
      ),
    fav_color =
      sample(
        x = colors(),
```

```
        size = 10,
        replace = TRUE
      )
  )
```

To this point we have largely been using CSV files for our datasets. To include our data in this R package, we save our dataset in a different format, ".rda", using `save()`.

```
save(color_data, file = "data/color_data.rda")
```

Then we create an R file "data.R" in the "R" folder. This file will only contain documentation using `roxygen2` comments. These start with `#'`, and we follow the documentation for `troopdata` closely.

```
#' Favorite color of various people data
#'
#' @description \code{favcolordata} returns a data
#' of the favorite color of various people.
#'
#' @return Returns a dataframe of the favorite colo
#' of various people.
#'
#' @docType data
#'
#' @usage data(color_data)
#'
#' @format A dataframe of individual-level observat
#' with the following variables:
#'
#' \describe{
#' \item{\code{name}}{A character vector of indivic
#' \item{\code{fav_color}}{A character vector of or
#' of: black, white, rainbow.}
#' }
#'
#' @keywords datasets
#'
#' @source \url{tellingstorieswithdata.com/12-store
#'
```

```
"color_data"
```

Finally, add a README that provides a summary of all of this for someone coming to the project for the first time. Examples of packages with excellent READMEs include `ggplot2`, `pointblank`, `modelsummary`, and `janitor`.

We can now go to the "Build" tab and click "Install and Restart". After this, the package "favcolordata", will be loaded and the data can be accessed locally using "color_data". If we were to push this package to GitHub, then anyone would be able to install the package using `devtools` (Wickham, Hester, and Chang 2020) and use our dataset. Indeed, the following will work.

```
library(devtools)

install_github("RohanAlexander/favcolordata")

library(favcolordata)

color_data
```

This has addressed many of the issues that we faced earlier. For instance, we have included a README and a data dictionary, of sorts, in terms of the descriptions that we added. But if we were to try to put this package onto CRAN, then we might face some issues. For instance, the maximum size of a package is 5MB and we would quickly come up against that. We have also largely forced users to use R. While there are benefits of that, we may like to be more language agnostic (Tierney and Ram 2020), especially if we are concerned about the FAIR principles.

The definitive guide to including data in R packages is Wickham (2022, chap. 8).

12.3.3 Depositing data

While it is possible that a dataset will be cited if it is available through GitHub or an R package, this becomes more likely if the dataset is deposited somewhere. There are several reasons for

this, but one is that it seems a bit more formal. Another is that it is associated with a DOI. Zenodo and Open Science Framework (OSF) are two that are commonly used. For instance, Carleton (2021) uses Zenodo to share the dataset and analysis supporting Carleton, Campbell, and Collard (2021), and Geuenich et al. (2021b) use Zenodo to share the dataset that underpins Geuenich et al. (2021a). Similarly, Arel-Bundock et al. (2022) use OSF to share code and data.

Another option is to use a dataverse, such as the Harvard Dataverse or the Australian Data Archive. This is a common requirement for journal publications. One nice aspect of this is that we can use `dataverse` (Kuriwaki, Beasley, and Leeper 2022) to retrieve the dataset as part of a reproducible workflow.

In general, these options are free and provide a DOI that can be useful for citation purposes. The use of data deposits such as these is a critical way to offload responsibility for the continued hosting of the dataset (which in this case is a good thing) and prevent the dataset from being lost. It also establishes a single point of truth, which should act to reduce errors (Byrd et al. 2020). Finally, it makes access to the dataset independent of the original researchers, and results in persistent metadata.

12.4 Data documentation

Dataset documentation has long consisted of a data dictionary. This may be just a list of the variables, a few sentences of description, and ideally a source. For instance, the data dictionary of the ACS, which was introduced in Chapter 8, is particularly comprehensive. And OSF provides instructions for how to make a data dictionary.

Datasheets (Gebru et al. 2021) are an increasingly critical aspect of data science because of the contribution they make to documentation. Datasheets are basically nutrition labels for datasets. The process of creating them enables us to think more carefully about what we will feed our model. More importantly,

So good! Love the idea.

they enable others to better understand what we fed our model. One important task is going back and putting together datasheets for datasets that are widely used. For instance, researchers went back and wrote a datasheet for one of the most popular datasets in computer science, and they found that around 30 per cent of the data were duplicated (Bandy and Vincent 2021).

> **Shoulders of giants**
>
> Timnit Gebru is the founder of the Distributed Artificial Intelligence Research Institute (DAIR). After taking a PhD in Computer Science from Stanford University, she joined Microsoft and then Google. One notable paper is Bender et al. (2021), which discussed the dangers of language models being too large. She has made many other substantial contributions to fairness and accountability, especially Buolamwini and Gebru (2018), which demonstrated racial bias in facial analysis algorithms.

Instead of telling us how unhealthy various foods are, a datasheet tells us things like:

- Who put the dataset together?
- Who paid for the dataset to be created?
- How complete is the dataset?
- Which variables are present, and, equally, not present, for particular observations?

Sometimes we have done a lot of work to create a datasheet. In that case, we may like to publish and share it on its own, for instance, Biderman, Bicheno, and Gao (2022) and Bandy and Vincent (2021). But typically a datasheet might live in an appendix to the paper, for instance Zhang et al. (2022), or be included in a file adjacent to the dataset.

As an example, a datasheet for the dataset that underpins Alexander and Hodgetts (2021) is included in Appendix D. The text of the questions directly comes from Gebru et al. (2021). When we create datasheets for a dataset, especially a dataset that we did not put together ourselves, it is possible that the answer to some questions will simply be “Unknown”, but we should do what we

can to minimize that.

The datasheet template created by Gebru et al. (2021) is not the final word. It is possible to improve on it, and add additional detail sometimes. For instance, Miceli, Posada, and Yang (2022) argue for the addition of questions to do with power relations.

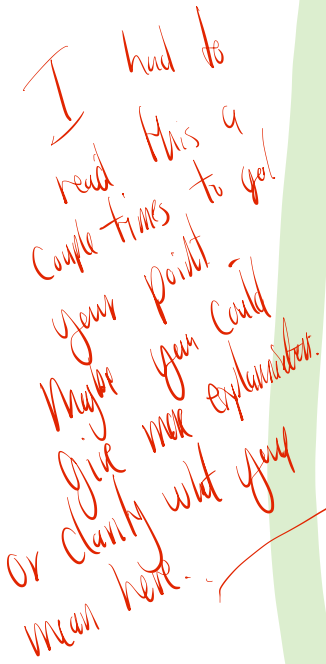
12.5 Personally identifying information

By way of background, Christensen, Freese, and Miguel (2019, 180) define a variable as confidential if the researchers know who is associated with each observation, but the public version of the dataset removes this association. A variable is anonymous if even the researchers do not know.

Personally identifying information (PII) is that which enables us to link an observation in our dataset with an actual person. For instance, email addresses are often PII, as are names and addresses. While some variable may not be PII for many respondents, it could be PII for some. For instance, consider a survey that is representative of the population age distribution. There is not likely to be many respondents aged over 100, and so the variable age may then become PII. The same scenario happens with income, wealth, and many other variables. One response to this is for data to be censored, which was discussed in Chapter 8. For instance, we may record age between 0 and 90, and then group everyone over that into "90+". Another is to construct age-groups: "18-29", "30-44", …. Notice that with both these solutions we have had to trade-off privacy for usefulness. More concerningly, a variable may be PII, not by itself, but when combined with another variable.

Our primary concern should be with ensuring that the privacy of our dataset is appropriate, given the expectations of the reasonable person. This requires weighing costs and benefits. In national security settings there has been considerable concern

about the over-classification of documents (Lin 2014). The reduced circulation of information because of this may result in unrealized benefits. To avoid this in data science, the test of the need to protect a dataset needs to be made by the reasonable person weighing up costs and benefits. It is easy, but wrong, to say that no data should be released unless it is perfectly anonymized, because the fundamental problem of data privacy would mean such data would have little utility. That approach, possibly motivated by the precautionary principle, would be too conservative and could cause considerable loss in terms of unrealized benefits.



Randomized response (Greenberg et al. 1969) is a clever way that anonymity can be ensured without much overhead. Each respondent, flips a coin before they answer a question and does not show the researcher the outcome of the coin flip. The respondent is instructed to respond truthfully to the question if the coin lands on heads, but to always give some particular (but still plausible) response if tails. The results of the other options can then be re-weighted to provide an estimate, without a researcher ever knowing the truth about any particular respondent. This is especially used in association with snowball sampling, discussed in Chapter 8. One issue with randomized response is that the resulting dataset can be only used to answer specific questions. This requires careful planning and the dataset will be of less general value.

Zook et al. (2017) recommend considering whether data even need to be gathered in the first place. For instance, if a phone number is not absolutely required then it might be better to not ask for it, rather than need to worry about protecting it before data dissemination.

GDPR and HIPAA are two legal structures that govern data in Europe and the US, respectively. Due to the influence of these regions, they have a significant effect outside those regions also. GDPR concerns data generally, while HIPAA is focused on healthcare. GDPR applies to all personal data, which is defined as:

> …any information relating to an identified or identifiable natural person ("data subject"); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;
>
> Council of European Union (2016), Article 4, "Definitions"

HIPAA refers to the privacy of medical records in the US and codifies the idea that the patient should have access to their medical records, and that only the patient should be able to authorize access to their medical records (Annas 2003). HIPAA only applies to certain entities. This means it sets a standard, but coverage is inconsistent. For instance, a person's social media posts about their health would generally not be subject to it, nor would knowledge of a person's location and how active they are, even though based on that information we may be able to get some idea of their health (Cohen and Mello 2018). Such data are hugely valuable (Ross 2022).

There are a variety of ways of protecting PII, while still sharing some data, that we will now go through. We focus here initially on what we can do when the dataset is considered by itself, which is the main concern. But sometimes the combination of several variables, none of which are PII in and of themselves, can be PII. For instance, age is unlikely PII by itself, but age combined with city, education, and a few other variables could be. One concern is that re-identification could occur by combining datasets and this is a potential role for differential privacy.

12.5.1 Hashing and salting

A cryptographic hash is a one-way transformation, such that the same input always provides the same output, but given the output, it is not reasonably possible to obtain the input. For

instance, a function that doubled its input always gives the same output, for the same input, but is also easy to reverse, so would not work well as a hash. In contrast, the modulo, which for a non-negative number is the remainder after division and can be implemented in R using `%%`, would would be difficult to reverse.

Knuth (1998, 514) relates an interesting etymology for "hash". He first defines "to hash" as relating to chop up or make a mess, and then explaining that hashing relates to scrambling the input and using this partial information to define the output. A collision is when different inputs map to the same output, and one feature of a good hashing algorithm is that collisions are reduced. As mentioned, one simple approach is to rely on the modulo operator. For instance, if we were interested in 10 different groupings for the integers 1 through to 10, then modulo would enable this. A better approach would be for the number of groupings to be a larger number, because this would reduce the number of values with this same hash outcome.

For instance, consider some information that we would like to keep private, such as names and ages of respondents.

```
library(tidyverse)

some_private_information <-
  tibble(
    names = c("Rohan", "Monica", "Edward", "Hugo"),
    ages = c(36, 35, 3, 1)
  )

some_private_information
```

```
# A tibble: 4 × 2
  names   ages
  <chr>  <dbl>
1 Rohan     36
2 Monica    35
3 Edward     3
4 Hugo       1
```

One option for the names would be to use a function that just took the first letter of each name. And one option for the ages would be to convert them to Roman numerals.

```
some_private_information |>
  mutate(
    names = substring(names, 1, 1),
    ages = as.roman(ages)
  )
```

```
# A tibble: 4 × 2
  names ages
  <chr> <roman>
1 R     XXXVI
2 M     XXXV
3 E     III
4 H     I
```

While the approach for the first variable, names, is good because the names cannot be backed out, the issue is that as the dataset grows there are likely to be lots of "collisions"—situations where different inputs, say "Rohan" and "Robert", both get the same output, in this case "R". It is the opposite situation for the approach for the second variable, ages. In this case, there will never be any collisions—"36" will be the only input that ever maps to "XXXVI". However, it is easy to back out the actual data, for anyone who knows roman numerals.

Rather than write our own hash functions, we can use crytographic hash functions such as `md5()` from `openssl` (Ooms 2021).

```
library(openssl)

some_private_information |>
  mutate(
    md5_names = md5(names),
    md5_ages = md5(ages |> as.character())
  )
```

```
# A tibble: 4 × 4
```

```
  names   ages md5_names
md5_ages
  <chr>  <dbl> <hash>
<hash>
1 Rohan     36 02df8936eee3d4d2568857ed530671b2
19ca14e7ea6328a42e0eb13d585e4c22
2 Monica    35 09084cc0cda34fd80bfa3cc0ae8fe3dc
1c383cd30b7c298ab50293adfecb7b18
3 Edward     3 243f63354f4c1cc25d50f6269b844369
eccbc87e4b5ce2fe28308fd9f2a7baf3
4 Hugo       1 1b3840b0b70d91c17e70014c8537dbba
c4ca4238a0b923820dcc509a6f75849b
```

We could share either of these transformed variables and be comfortable that it would be difficult for someone to use only that information to recover the names of our respondents. That is not to say that it is impossible. If we made a mistake, such as accidentally pushing the original dataset to GitHub then they could be recovered. And it is likely that various governments have the ability to reverse the cryptographic hashes used here.

One issue that remains is that anyone can take advantage of the key feature of hashes to back out the input. That feature is that the same input always gets the same output, to test various options for inputs. For instance, they could, themselves try to hash "Rohan", and then noticing that the hash is the same as the one that we published in our dataset, know that data relates to that particular individual. We could try to keep our hashing approach secret, but that is difficult as there are only a few that are widely used. One approach is to add a salt that we keep secret. This slightly changes the input. For instance, we could add the salt `_is_a_person' to all our names and then hash that, although a large random number might be a better option. Provided the salt is not shared, then it would be difficult for most people to reverse our approach in that way.

Might be helpful to explain this for a non-specialist reader.

```
some_private_information |>
  mutate(names = paste0(names, "_is_a_person")) |>
  mutate(
    md5_of_salt = md5(names)
```

```
  )
```

```
# A tibble: 4 × 3
  names                ages md5_of_salt
  <chr>               <dbl> <hash>
1 Rohan_is_a_person      36
3ab064d7f746fde604122d072fd4fa97
2 Monica_is_a_person     35
50bb9dfffa926c855b830845ac61b659
3 Edward_is_a_person      3
9845500d4070c0cbba7c6b81ed306027
4 Hugo_is_a_person        1
b9b8c4e9870aca482cf062da4681b232
```

12.5.2 Data simulation

One common approach to deal with the issue of being unable to share the actual data that underpins an analysis, is to use data simulation. We have used data simulation throughout this book toward the start of the workflow to help us to think more deeply about our dataset before we turn to it. We can use data simulation again at the end, to ensure that others cannot think about our actual dataset. The workflow advocated in this book makes this relatively straight-forward.

The approach is to understand the critical features of the dataset and the appropriate distribution. For instance, if our data were the ages of some population, then we may want to use the Poisson distribution and experiment with different parameters for the rate. Having simulated a dataset, we conduct our analysis using this simulated dataset and ensure that the results are broadly similar to when we use the real data. We can then release the simulated dataset along with our code.

For more nuanced situations, Koenecke and Varian (2020) recommend using the synthetic data vault (Patki, Wedge, and Veeramachaneni 2016) and then the use of Generative Adversarial Networks, such as implemented by Athey et al. (2021).

12.5.3 Differential privacy

I struggled a bit with what the takeaway was here.

Differential privacy is a mathematical definition of privacy (Dwork and Roth 2013, 6). It is important to be clear that it is not only one algorithm, it is a definition that many algorithms implement. The main issue it solves is that there are a lot of datasets available. This means there is always the possibility that some combination of them could be combined to identify respondents even if PII were removed from each of these individual datasets. Rather than needing to anticipate how various datasets could be combined to re-identify individuals and adjust variables to remove this possibility, a dataset that is created using a differentially private approach provides assurances that privacy will be maintained.

Shoulders of giants

Cynthia Dwork is Gordon McKay Professor of Computer Science, Harvard University. After taking a PhD in Computer Science from Cornell University, she was a Post-Doctoral Research Fellow at MIT and then worked at IBM, Compaq, and Microsoft Research where she is a Distinguished Scientist. She joined Harvard in 2017. One of her major contributions is differential privacy (Dwork et al. 2006), which has been widely adapted.

To motivate the definition, consider a dataset of responses and PII that only has one person in it. The release of that dataset, as is, would perfectly identify them. At the other end of the scale, consider a dataset that does not contain a particular person. The release of that dataset could never be linked to them because they are not in it. Differential privacy, then, is about the inclusion or exclusion of particular individuals in a dataset. An algorithm is differentially private if the inclusion or exclusion of any particular person in a dataset has at most some given factor of an effect on the probability of some output (Oberski and Kreuter 2020).

More specifically, from Asquith et al. (2022), consider Equation 12.1:

$$\frac{\Pr[M(d) \in S]}{\Pr[M(d') \in S]} \leq e^{\epsilon} \tag{12.1}$$

Here, "M is a differentially private algorithm, d and d' are datasets

that differ only in terms of one row, S is a set of output from the algorithm" and ϵ controls the amount of privacy that is provided to respondents. The fundamental problem of data privacy is that we cannot have completely anonymized data that remains useful (Dwork and Roth 2013, 6). Instead, we must trade-off utility and privacy.

A dataset is differentially private to different levels of privacy, based on how much it changes when one person's results are included or excluded. This is the key parameter, because at the same time as deciding how much of an individual's information we are prepared to give up, or "leak", we are deciding how much random noise to add which will impact our output. The choice of this level is a nuanced one and should involve extensive consideration of the costs of undesired disclosures, compared with the benefits of additional research. For public data that will be released under differential privacy, this *ratio decidendi* must be public because of the costs that are being imposed. Indeed, Tang et al. (2017) argue that even in the case of private companies, such as Apple, users should have a choice about the level of privacy loss.

I found the Latin use to be a bit more formal that your otherwise Conversational style.

A variant of differential privacy has recently been implemented by the US census. This has been shown to not universally protect respondent privacy, and yet it is expected to have a significant effect on redistricting (Kenny et al. 2021) even though redistricting is considered a high priority use case (Hawes 2020). Suriyakumar et al. (2021) found that such a model will be disproportionately affected by large demographic groups. The implementation of differential privacy is expected to result in some publicly available data that are unusable in the social sciences (Ruggles et al. 2019). And even the ACS, introduced in Chapter 8, may be replaced with synthetic content. The issue with this approach is that even if it is fine for what is of interest and known now, because it is model based, it cannot account for the unknowable answers to questions that we do not even think to ask.

The implementation of differential privacy is a costs and benefits

issue (Hotz et al. 2022). Stronger privacy protection fundamentally must mean less information (Bowen 2022, 39). The costs of this have been convincingly shown, while the benefits have not. As always in data science, it is important to consider who this affects. At issue, with all privacy enhancing approaches but especially with hurried implementations of differential privacy, is concerns around our ability to answer questions that we cannot even think to ask today.

12.6 Data efficiency

For the most part, done is better than perfect, and unnecessary optimization is a waste of resources. However, at a certain point, we need to adapt new ways of dealing with data, especially as our datasets start to get larger. Here we discuss iterating through multiple files, and then turn to changing data formats, beginning with Apache Arrow, and finally SQL.

12.6.1 Iteration

There are a number of ways to become more efficient with our data, especially as it becomes larger. The first, and most obvious, is to break larger datasets into smaller pieces. For instance, if we have a dataset for a year, then we could break it into months, or even days. To enable this, we need a way of quickly reading in many different files.

The need to read in multiple files and combine them into the one tibble is a surprisingly common task. For instance, it may be that the data for a year, are saved into individual CSV files for each month. We can use `purrr` (Henry and Wickham 2020) and `fs` (Hester, Wickham, and Csárdi 2021) to do this painlessly. To illustrate this situation we will simulate data from the exponential distribution using `rexp()`. Such data may reflect, say, comments on a social media platform, where the vast majority of comments are made by a tiny minority of users. We will use `dir_create()` from `fs` to create a folder, simulate monthly data and save it. We will then illustrate reading it in.

```
library(tidyverse)
library(fs)

dir_create(path = "user_data")

set.seed(853)

simulate_and_save_data <- function(month) {
  number_of_observations <- 1000
  file_name <- paste0("user_data/", month, ".csv")
  user_comments <-
    tibble(
      user = c(1:number_of_observations),
      month = rep(x = month, times = number_of_obse
      comments = rexp(n = number_of_observations, r
    )
  write_csv(
    x = user_comments,
    file = file_name
  )
}

walk(month.name |> tolower(), simulate_and_save_dat
```

Having created our dataset with each month saved to a different CSV, we can now read it in. There are a variety of ways to do this. The first step is that we need to get a list of all the CSV files in the directory. We use the "glob" argument here to specify that we are interested only in the ".csv" files, and that could change to whatever files it is that we are interested in.

```
library(fs)

files_of_interest <-
  dir_ls(path = "user_data/", glob = "*.csv")

files_of_interest
```

```
 [1] "april.csv"     "august.csv"
"december.csv"  "february.csv"
 [5] "january.csv"   "july.csv"      "june.csv"
```

```
"march.csv"
 [9] "may.csv"       "november.csv"  "october.csv"
"september.csv"
```

We can pass this list to `read_csv()` and it will read them in and combine them.

```
year_of_data <-
  read_csv(
    files_of_interest,
    show_col_types = FALSE
  )

year_of_data
```

```
# A tibble: 12,000 × 3
    user month comments
   <dbl> <chr>    <dbl>
 1     1 april        0
 2     2 april        2
 3     3 april        2
 4     4 april        5
 5     5 april        1
 6     6 april        3
 7     7 april        2
 8     8 april        1
 9     9 april        4
10    10 april        3
# … with 11,990 more rows
```

This works well when we have CSV files, but we might not always have CSV files and so will need another way, and can use `map_dfr()` to do this. One nice aspect of this approach is that we can include the name of the file alongside the observation using ".id". Here we specify that we would like that column to be called "file", but it could be anything.

```
library(purrr)

year_of_data_using_purrr <-
  files_of_interest |>
```

```
  map_dfr(read_csv, .id = "file")
```

```
# A tibble: 12,000 × 4
   file       user month comments
   <chr>     <dbl> <chr>    <dbl>
 1 april.csv     1 april        0
 2 april.csv     2 april        2
 3 april.csv     3 april        2
 4 april.csv     4 april        5
 5 april.csv     5 april        1
 6 april.csv     6 april        3
 7 april.csv     7 april        2
 8 april.csv     8 april        1
 9 april.csv     9 april        4
10 april.csv    10 april        3
# … with 11,990 more rows
```

12.6.2 Apache Arrow

While the use of CSVs is great because they are widely used and have little overhead, they are also quite minimal. This can lead to issues, especially in terms of class. There are various alternatives, including Apache Arrow, which is a columnar data framework. There is an R implementation of Apache Arrow, `arrow` (Richardson et al. 2022). This has the advantage of requiring very little change from us while delivering significant benefits. This makes two, distinct, file formats available to us: ".arrow", and ".parquet".

We use ".arrow" when we are actively using the data, for instance, as we clean, prepare, and model. And we use ".parquet" for data storage, for instance, saving a copy of the original data, and making the final dataset available to others. This is because ".parquet" is focused on size, while ".arrow" is focused on efficiency.

For long-term storage, we might replace ".csv" files with ".parquet" files. We can do this by replacing `write_csv()` with `write_parquet()`, and `read_csv()` with `read_parquet()`.

```
library(arrow)
library(fs)
library(tidyverse)

number_of_draws <- 1000000

# Homage to: https://www.rand.org/pubs/monograph_re
a_million_random_digits <-
  tibble(
    numbers = runif(n = number_of_draws),
    letters = sample(x = letters, size = number_of_
    states = sample(x = state.name, size = number_d
  )

write_csv(
  x = a_million_random_digits,
  file = "a_million_random_digits.csv"
)

write_parquet(
  x = a_million_random_digits,
  sink = "a_million_random_digits.parquet"
)

file_size("a_million_random_digits.csv")
```

```
29.3M
```

```
file_size("a_million_random_digits.parquet")
```

```
8.17M
```

We get significant file size improvements when we compare the size of the same datasets saved in each format, especially as they get larger (Table 12.1).

Table 12.1: Comparing the file sizes of '.csv' and '.parquet', as the file size increases, for CSV and Parquet

Number of observations	CSV file size	Parquet file size

10,000	300.11K	99.57K
100,000	2.93M	1016.46K
1,000,000	29.29M	8.15M
10,000,000	292.88M	79.11M
100,000,000	2.86G	788.82M

Having considered ".parquet", and explained the benefits compared with CSVs for large data that we are storing, we now consider ".arrow" and the benefit of this file type for large data that we are actively using. Again, these benefits will be most notable for larger datasets, and so we will consider the ProPublica US Open Payments Data, from the Centers for Medicare & Medicaid Services, which is 6.66 GB and available here. It is available as a CSV file, and so we will compare reading in the data and creating a summary of the average total amount of payment on the basis of state using `read_csv()`, with the same task using `read_csv_arrow()`. We find a considerable speed-up when using `read_csv_arrow()` (Table 12.2).

```
library(arrow)
library(tidyverse)
library(tictoc)

tic.clearlog()

tic("CSV - Everything")
tic("CSV - Reading")
open_payments_data_csv <-
  read_csv(
    "OP_DTL_GNRL_PGYR2016_P01172018.csv",
    col_types =
      cols(
        "Teaching_Hospital_ID" = col_double(),
        "Physician_Profile_ID" = col_double(),
        "Applicable_Manufacturer_or_Applicable_GPO_
        "Total_Amount_of_Payment_USDollars" = col_d
        "Date_of_Payment" = col_date(format = "%m/%
        "Number_of_Payments_Included_in_Total_Amour
```

```
        "Record_ID" = col_double(),
        "Program_Year" = col_double(),
        "Payment_Publication_Date" = col_date(forma
        .default = col_character()
      )
  )

toc(log = TRUE, quiet = TRUE)

tic("CSV - Manipulate and summarise")
summary_spend_by_state_csv <-
  open_payments_data_csv |>
  rename(
    state = Recipient_State,
    total_payment_USD = Total_Amount_of_Payment_USD
  ) |>
  filter(state %in% c("CA", "OR", "WA")) |>
  mutate(total_payment_USD_thousands = total_paymer
  group_by(state) |>
  summarise(average_payment = mean(total_payment_US

summary_spend_by_state_csv

toc(log = TRUE, quiet = TRUE)
toc(log = TRUE, quiet = TRUE)

tic("arrow - Everything")
tic("arrow - Reading")
open_payments_data_arrow <-
  read_csv_arrow(
    "OP_DTL_GNRL_PGYR2016_P01172018.csv",
    as_data_frame = FALSE
  )
toc(log = TRUE, quiet = TRUE)

tic("arrow - Manipulate and summarise")
summary_spend_by_state_arrow <-
  open_payments_data_arrow |>
  rename(
    state = Recipient_State,
    total_payment_USD = Total_Amount_of_Payment_USD
  ) |>
```

```
  filter(state %in% c("CA", "OR", "WA")) |>
  mutate(total_payment_USD_thousands = total_paymer
  group_by(state) |>
  summarise(average_payment = mean(total_payment_US
  collect()

summary_spend_by_state_arrow

toc(log = TRUE, quiet = TRUE)
toc(log = TRUE, quiet = TRUE)
```

Table 12.2: Comparing the speed of reading and manipulating using 'read_csv()' and 'read_csv_arrow()'

Which function	Task	Time (seconds)
CSV	Reading	265
CSV	Manipulate and summarise	3
CSV	Everything	269
arrow	Reading	43
arrow	Manipulate and summarise	8
arrow	Everything	51

Crane (2022) provides further information about specific tasks and Navarro (2022) provides helpful examples of implementation.

12.6.3 SQL

Structured Query Language (SQL) ("see-quell" or "S.Q.L.") is used with relational databases. A relational database is a collection of at least one table, and a table is just some data organized into rows and columns. If there is more than one table in the database, then there should be some column that links them. An example is the `AustralianPoliticians` datasets that were used in Chapter 3. Using SQL feels a bit like HTML/CSS in terms of being halfway between markup and programming. One fun aspect is that, by convention, commands are written in upper case. Another is that line spaces mean nothing: include them or do not, but always end

a SQL command in a semicolon;

SQL was developed in the 1970s at IBM. While it may be true that the SQL is never as good as the original, SQL is an especially popular way of working with data. There are many "flavors" of SQL, including both closed and open options. Here we introduce SQLite, which is open source, and pre-installed on Macs. Windows users can install it from here.

Advanced SQL users do a lot with it alone, but even just having a working knowledge of SQL increases the number of datasets that we can access. A working knowledge of SQL is especially useful for our efficiency because a large number of datasets are stored on SQL servers, and being able to get data from them ourselves is handy.

We could use SQL within RStudio, especially drawing on `DBI` (R Special Interest Group on Databases (R-SIG-DB), Wickham, and Müller 2022). Although given the demand for SQL skills, independent of demand for R skills, it may be a better idea, from a career perspective to have a working knowledge of it that is independent of RStudio. We can consider many SQL commands as straightforward variants of the `dplyr` verbs that we have used throughout this book. Indeed, if we wanted to stay within R, then `dbplyr` (Wickham, Girlich, and Ruiz 2022) would explicitly allow us to use `dplyr` functions and would then automatically translate them into SQL. Having used `mutate()`, `filter()` and `left_join()` in the `tidyverse` means that many of the core SQL commands will be familiar. That means that the main difficulty will be getting on top of the order of operations because SQL can be pedantic.

To get started with SQL, download DB Browser for SQLite (DB4S), which is free and open-source, and open it (Figure 12.2).

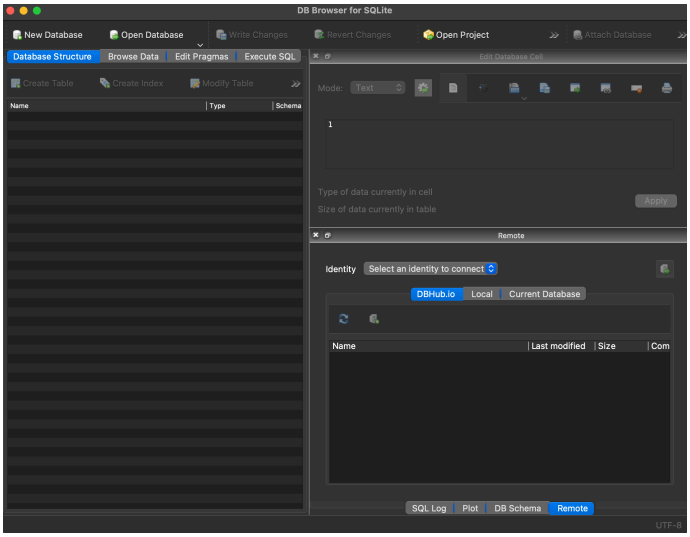


Figure 12.2: Opening DB Browser for SQLite

Download “AustralianPoliticians.db” here and then open it with “Open Database” and navigate to where you downloaded the database.

There are three key SQL commands that we now cover: `SELECT`, `FROM`, and `WHERE`. `SELECT` allows us to specify particular columns of the data, and we can consider `SELECT` in a similar way to `select()`. In the same way that we need to specify a dataset with `select()` and did that using a pipe operator, we specify a dataset with `FROM`. For instance, we could open “Execute SQL”, and then type the following, and click “Execute”.

```
SELECT
    surname
FROM
    politicians;
```

The result is that we obtain the column of surnames. We could select multiple columns by separating them with commas, or all of them by using an asterisk, although this is not best practice because if the dataset were to change without us knowing then our result would differ.

```
SELECT
    uniqueID
    surname
```

```
FROM
    politicians;
```

```
SELECT
    *
FROM
    politicians;
```

And, finally, if there were repeated rows, then we could just look at the unique ones using `DISTINCT`, in a similar way to `distinct()`.

```
SELECT
    DISTINCT surname
FROM
    politicians;
```

So far we have used `SELECT` along with `FROM`. The third command that is commonly used is `WHERE`, and this will allow us to focus on particular rows, in a similar way to `filter()`.

```
SELECT
    uniqueID,
    surname,
    firstName
FROM
    politicians
WHERE
    firstName = "Myles";
```

All the usual logical operators are fine with `WHERE`, such as “=”, “!=”, “>”, “<”, “>=”, and “<=”. We could combine conditions using `AND` and `OR`.

```
SELECT
    uniqueID,
    surname,
    firstName
FROM
    politicians
```

```
WHERE
    firstName = "Myles"
    OR firstName = "Ruth";
```

If we have a query that gave a lot of results, then we could limit the number of them with `LIMIT`.

```
SELECT
    uniqueID,
    surname,
    firstName
FROM
    politicians
WHERE
    firstName = "Robert"       LIMIT 5;
```

And we could specify the order of the results with `ORDER`.

```
SELECT
    uniqueID,
    surname,
    firstName
FROM
    politicians
WHERE
    firstName = "Robert"
ORDER BY
    surname DESC;
```

See the rows that are pretty close to a criteria:

```
SELECT
    uniqueID,
    surname,
    firstName
FROM
    politicians
WHERE
    firstName LIKE "Ma__";
```

The “_” above is a wildcard that matches to any character. So this

provides results that include “Mary”, and “Mark”. `LIKE` is not case-sensitive: “Ma__” and “ma__” both return the same results.

Focusing on missing data is possible using “NULL” or “NOT NULL”.

```
SELECT
    uniqueID,
    surname,
    firstName,
    comment
FROM
    politicians
WHERE
    comment     IS NULL;
```

There is an underlying ordering build into number, date and text fields that means we can use `BETWEEN` on all those, not just numeric. For instance, we could look for all surnames that starts with a letter between X and Z (not including Z).

```
SELECT
    uniqueID,
    surname,
    firstName
FROM
    politicians
WHERE
    surname     BETWEEN "X" AND "Z";
```

Using `WHERE` with a numeric variable means that `BETWEEN` is inclusive, compared with the example with letters which is not.

```
SELECT
    uniqueID,
    surname,
    firstName,
    birthYear
FROM
    politicians
WHERE
    birthYear     BETWEEN 1980 AND 1990;
```

In addition to providing us with dataset observations that match what we asked for, we can modify the dataset. For instance, we could edit a value using `UPDATE` and `SET`.

```
UPDATE
    politicians
SET
    surname = "Alexanderrrrrrrr"
WHERE
    uniqueID = "Alexander1951";
```

We can integrate if-else logic with `CASE` and `ELSE`. For instance, where we add a column called “wasTreasurer”, which is “Yes” in the case of “Josh Frydenberg”, and “No” in the case of “Kevin Rudd”, anad “Unsure” for all other cases.

```
SELECT
    uniqueID,
    surname,
    firstName,
    birthYear,
    CASE
        WHEN uniqueID = "Frydenberg1971" THEN "Yes"
        WHEN surname = "Rudd" THEN "No"
        ELSE "Unsure"
    END AS "wasTreasurer"
FROM
    politicians;
```

We can create summary statistics using commands such as `COUNT`, `SUM`, `MAX`, `MIN`, `AVG`, and `ROUND` in the place of `summarize()`. `COUNT` counts the number of rows that are not empty for some column by passing the column name, and this is similarly how `MIN`, etc, work.

```
SELECT
    COUNT(uniqueID)
FROM
    politicians;
```

```
SELECT
    MIN(birthYear)
FROM
    politicians;
```

We can get results on the basis of different groups in our dataset using `GROUP BY`, in a similar manner to `group_by` in R.

```
SELECT
    COUNT(uniqueID)
FROM
    politicians
GROUP BY
    gender;
```

And finally, we can combine two tables using `LEFT JOIN`. We need to be careful to specify the matching columns using dot notation.

```
SELECT
    politicians.uniqueID,
    politicians.firstName,
    politicians.surname,
    party.partySimplifiedName
FROM
    politicians
LEFT JOIN
    party
        ON politicians.uniqueID = party.uniqueID;
```

As SQL is not our focus we have only provided a brief overview of some essential commands. From a career perspective it is important to develop a comfort with SQL. It is so integrated into data science that it would be "difficult to get too far without it" (Robinson and Nolis 2020, 8) and that "almost any" data science interview will include questions about SQL (Robinson and Nolis 2020, 110).

12.7 Exercises and tutorial

Exercises

1. *(Plan)* Consider the following scenario: *You work for a large news media company and focus on subscriber management. Over the course of a year most subscribers will never post a comment beneath a news article, but a few post an awful lot.* Please sketch what that dataset could look like and then sketch a graph that you could build to show all observations.
2. *(Simulate)* Please further consider the scenario described and simulate the situation.
3. *(Acquire)* Please describe one possible source of such a dataset.
4. *(Explore)* Please use `ggplot2` to build the graph that you sketched.
5. *(Communicate)* Please write two paragraphs about what you did.
6. Following Wilkinson et al. (2016), which of the following are FAIR principles (please select all that apply)?
    a. Findable.
    b. Approachable.
    c. Interoperable.
    d. Reusable.
    e. Integrated.
    f. Fungible.
    g. Reduced.
    h. Accessible.

7. Please create an R package for a simulated dataset, push it to GitHub, and submit the link.
8. Please simulate some data, add it to a GitHub repository and then submit the link.
9. According to Gebru et al. (2021), a datasheet should document a dataset's (please select all that apply):
    a. composition.
    b. recommended uses.
    c. motivation.
    d. collection process.

10. Do you think that a person's name is PII?
    a. Yes.
    b. No.

11. Under what circumstances do you think income is PII (please write a paragraph or two)?
12. Using `openssl::md5()` what is the hash of "Rohan" (pick one)?
    a. 243f63354f4c1cc25d50f6269b844369
    b. 09084cc0cda34fd80bfa3cc0ae8fe3dc
    c. 02df8936eee3d4d2568857ed530671b2
    d. 1b3840b0b70d91c17e70014c8537dbba

Tutorial

Please identify a dataset you consider interesting and important, that does not have a datasheet (Gebru et al. 2021). As a reminder, datasheets accompany datasets and document "motivation, composition, collection process, recommended uses," among other aspects. Please put together a datasheet for this dataset. You are welcome to use the template here as a starting point. The datasheet should be completely contained in its own GitHub repository. Please submit a PDF.

Paper

At about this point the *Dysart* Paper in Appendix C would be appropriate.