# A Reference Manual for the Biostatistics GRA

Towards readable, rigorous, and reproducible results

T. K. Peter

2023-08-16

# Contents

# Chapter 1

# Motivation

The meta-question behind this short book is 'where to begin?'. I started graduate school in August of 2019, coming straight from an undergrad program in math to a doctoral program in biostatistics. In the same week that I started classes, I started my work as a graduate research assistant in the Division of Biostatistics and Computational Biology at the University of Iowa College of Dentistry. It was a steep learning curve. Now heading into my 5th year as a Ph.D. student, I began compiling some notes/tips to share with the other graduate research assistants (GRAs) who joined our team after me. Those notes have become this reference manual.

The 'first principles' chapter is me diving into the philosophy of why I think this kind of training manual is important. Here, I'll give you the TL;DR: Although a significant component of graduate level study is learning how to learn, brand new GRAs also need to know what to Google. GRAs need to struggle with hard concepts and new ideas – the learning is truly in the struggle. At the same time, they need to know where to start. This creates a tension: new GRAs need to engage in independent study work, but they also need resources that outline guiding principles. If GRAs don't know what they're searching for, they won't know if they've found it. Moreover, if novice GRAs *think* they know what they are searching for without any guidance, they could easily come across bad advice (by 'bad', I mean that which leads to suboptimal outcomes with respect to rigor and reproducibility).

With this tension in mind, the goal of this reference manual is to 1) outline some of the main ideas that you will need to be successful, and 2) to provide you with some vocabulary to help you as you learn independently. Many of the chapters here outline some general concepts and then provide links to further reading.

## 1.1 What this book assumes

This is a reference manual, not a formal textbook. As such, my philosophy in writing this is to summarize the most helpful things I have learned in a way that is conducive to quick review. The chapters that are titled after broad areas of research (e.g., 'survival analysis') contain more links than new content. I assume throughout this writing that the readers are either familiar with foundational statistical concepts or are currently enrolled in courses that are teaching those concepts.

## 1.2 Acronymns

Throughout this work, I use the following acronyms:

- **GRA**: Graduate research assistant
- **PI**: Primary investigator
- **CC**: Carbon copy (as in an email)
- **UI**: University of Iowa
- **HPC**: High performance computing
- **SAS**: Statistical Analysis System software
- **GWAS**: Genome-wide association study
- **QC**: quality control

## 1.3 Working with the garage door open

This manual is a work in progress. I am writing this as I work my way through my own program, making notes as I go. This is in the spirit of working with the garage door open, an approach to learning that really resonates with me.

Keep in mind that as I'm writing this, I am also a student myself. I want to make improvements to this manual as I keep learning. If you think of a reference that you'd like to see included, open a GitHub issue or make a pull request to let me know. I would welcome any feedback, and would appreciate any thoughtful critiques.

## 1.4 Acknowledgements

So much of this work comes directly from what I have learned from my dissertation advisor Patrick Breheny and my GRA supervisor X. Jin Xie. Both of these men have had a tremendous influence on my thinking.

I also want to thank the other UI biostats professors who have taught my core courses; our department is stacked with faculty who are dedicated teachers as well as brilliant researchers.

## 1.5 Dedication

This work is dedicated in loving memory of my father, Bill Peter (1955-2022).

# Chapter 2

# First Principles - invisible and integral work

There are professions in which excellence brings invisibility. In such professions, the reward of a job well done is the unnoticed reception of the work into the foundation of the organization which the worker supports. Successes are silent, while failures are deafening.

An illustrative example of this phenomena is observed in custodial work. Custodial staff members know that when their job is done well, no one will notice. However, when their job is undone or done poorly, *everyone* will notice.[1] The work in such professions is both integral and invisible. I argue that the field biostatistics epitomizes this phenomena. We who work as biostatisticians are called upon to work in humility, willing to invest much in the invisible and integral art of communicating the truth in this data-saturated historic moment.

This work is a reference manual, which means most of its contents are not new. Rather, the novelty in what I am providing here is in its form and organization. As a 4th year PhD student, I set out to gather resources together to help the newer GRAs on my team, with the goal of helping them by sharing the links, lectures, and tips that have helped me the most. What I am publishing here is the result of that work. Although this reference manual is intended to serve graduate students and early career professionals in biostatistics, I think many of those working in the fields of statistics, data science, and bioinformatics may resonate with the calling to work in invisible and integral spaces and find this manual relevant to their work.

The objective of this manual is twofold: to provide starting points from which to address common questions of practice, and to summarize the references and

[1] I borrow this example from a phrase often used by my late grandmother (my father's mother), Jean, who was a housekeeper in a hospital.

tools which I have found most helpful in my journey through gradate school and my early years as a biostatistician. Overarching these two goals is a desire to make a contribution toward training biostatisticians whose work is readable, rigorous, and reproducible.

If even one other person is encouraged and sharpened by this work, my goal will be accomplished.

Yours faithfully, TKP

2023

# Chapter 3

# Getting organized: The first 3 days of a new RA job

The first step in starting out as a research assistant is establishing habits of organization. American author O. S. Marden said it this way: "A good system shortens the road to the goal." You will be more productive and produce higher quality work – work that is reproducible and rigorous – if you begin by establishing an organizational system.

I will outline here both general principles and some specific strategies that have proven helpful for my work. These points are organized according to the three major components to your work system that should be established within your first few days on the job:

1. File structure
2. Emails
3. Notes and README files

## 3.1 File structure

'File structure' refers to the system for categorizing your code, notes, reports, and figures for each project. Here are some general principles:

1. One master folder per project

- the 'master folder' structure translates well to starting GitHub repositories [CROSS REF].

2. Subfolders for each file type

- keeping subfolders by file type will help you find what you are looking for. Also, doing it this way will let you script your work from the command line with ease – for example, you can apply a function or use an executable file to work with all the items in a given folder.

3. Names that are readable for both humans and machines

- use names that can be easily ordered by a computer; again, this helps with writing scripts.
- avoid using white spaces, as this is a hassle for machines to read. Choose to work with either 'CamelCase' or 'snake_case', and be consistent.
- use specific keywords in file names: 'tkp_edited_version' is better than 'final_version'
- it's worth saying again: choose a convention and be consistent
- one more time for the folks in the back: choose a convention and **be consistent**.
- for a brief (and entertaining) tutorial on naming things, checkout Jenny Bryan's YouTube video.

4. Example

Here is a step-by-step example of how I use a file structure to organize myself at the start of a new project (if you'd prefer, skip to the video). Suppose I am contacted by Collaborator A for support on a study of a new endodontic treatment…

1. I begin by creating a new folder with a name that fits my established convention. For my new project, I would create a folder `TKP23-A-Endo`. My folders are stored in a shared drive to which other biostatistician have access, so I begin my folder names with my initials (TKP). the '23' tells me that the project started in 2023. The 'A' and 'Endo' keywords tell me who is the PI/what project it is (this is helpful since it is not uncommon for me to have multiple projects from the same lab).

2. The first file I add to my new `TKP23-A-Endo` folder is a README file. This includes notes from the initial meeting/first email from Collaborator A.

3. I open RStudio and go to `File > New Project`. I follow the prompts to create a new project, and the directory (a.k.a the folder) in which this project will live is my new `TKP23-A-Endo` folder. Using an R project is an example of a broader principle of project-oriented workflow.

4. I go to my `TKP23-A-Endo` folder (which now has two things in it: a `.Rproj` file and a `README` file), and I make 4 subfolders: `data`, `scripts`, `reference`, and `reports`. These are the building blocks of my projects.

- I will get data from Collaborator A, and it will go in the data folder. **Nothing else** will go in the data folder, and I will never change any original files in the data folder. If I need to edit the data in some way

(this is esp. relevant when data come in the form of Excel), I make a copy of the files I want to change and then work with those copies.

- `scripts` will be the place where I put all of the code I use for data cleaning and analysis.
- `reference` is where I will put things Collaborator A shares with me that help me understand the research question – this could be articles in endodontics that provide an explanation of the terminology, or a PowerPoint that explains how the data are collected. I would also keep a copy of the IRB documentation in this `reference` folder.
- `reports` is where I will keep my final `.Rmd` file for writing the report which I will share with Collaborator A.

From here, I begin writing scripts.

5. As the project moves along, I may find that I need other subfolders. For instance, a `graphics` subfolder would be useful if I am making a ton of plots for a project. If I am writing a manuscript and there's quite a bit of back-and-forth with reviewers, I would use a `submission` subfolder that contains all my point-to-point responses (CROSS REF) for reviewers.

### 3.1.1 Folder setup video

Here's a short YouTube video that gives an overview of my typical project folder setup.

### 3.1.2 Please use relative file paths

Computers work with files by having a 'pointer' at a specific folder, called a *directory*. Your computer probably has many directories on it – for example, 'Desktop' and 'Downloads' are both directories you have by default in a Mac operating system. When you are doing project-based quantitative work, your scripts (the files with the code in them) will need to 'point' to data that is probably in another place. Instead of copying data into multiple folders, or using functions like `setwd()`, use *relative filepaths*. This can be accomplished using the `..` symbol to reference a *parent directory*. Here is a short video on how I use relative file paths.

## 3.2 Emails

Email management is of paramount importance for collaboration. Emails serve 2 purposes:

- Providing a paper trail to document important exchanges of information
- Facilitating conversations that can happen in an asynchronous way

I have found the Inbox Zero method for email management to be a helpful place to begin. This method breaks down emails in to 4 categories: delete, delegate, defer, and do. Practically speaking, I have folders named for each person with whom I correspond. When I receive a new email that needs an answer, this goes in a 'needs follow up' folder. After following up, I move the conversation to the folder with the appropriate name (with group conversations categorized by the PI).

I **do not recommend** using emails to exchange versions of important files. Several tools exist to facilitate collaboration in a more efficient way - Microsoft offers 'OneDrive', Google offers 'GoogleDrive', and of course, there's GitHub. In addition, most organizations have a some kind of shared drive/shared repository. As often as possible, use shared resources for file sharing as opposed to emailing different versions.

### 3.2.1 Organizing your inbox

Here's a short video on how to organize your inbox:

TODO: add video here!

## 3.3 Project notes and README files

You have to keep track of what you are learning. In addition, you need to keep a log of what you have done so that other collaborators/future RAs can look back and figure out what you did. Let's call these two tasks *internal* documentation (keeping track of your project-specific work) and *external* documentation (the project-specific notes you want to leave for others). I recommend writing project notes for internal documentation, and README files for external documentation.

### 3.3.1 Project notes

Each project is a process that unfolds over time. As you learn over the course of a project, you'll need some kind of time-organized system for keeping track of your thoughts and action items. Using Google Drive, I have created a system of weekly project notes to serve this purpose. The setup looks like this:

Each week (typically on Monday morning), I create a new Google doc file named `YYYY_MM_DD1-DD2_notes` (YYYY = year, MM = month, DD1-2 = dates of Monday & Friday). On the first page, I set goals for the week. I check off things that are completed by marking them as done. Colored fonts help me stay organized, especially in seasons where I have several collaborative projects going on concomitantly. I often use blue text to show things that are done, and red text to show deadlines.

At the left panel, you can see that each subsequent page of my document is

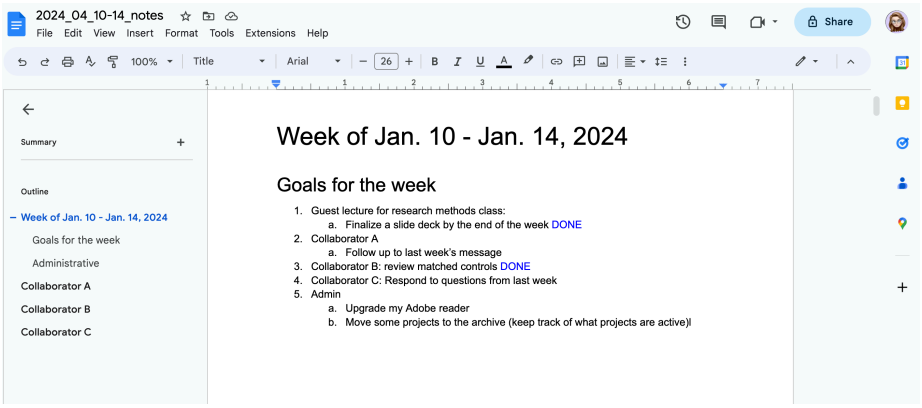



Figure 3.1: Weekly notes template

given a subheading link with the collaborator's name. To navigate to my notes for each collaborator, I can use these subheading links to quickly find what I am looking for. If I click on Collaborator A, I would see this page:

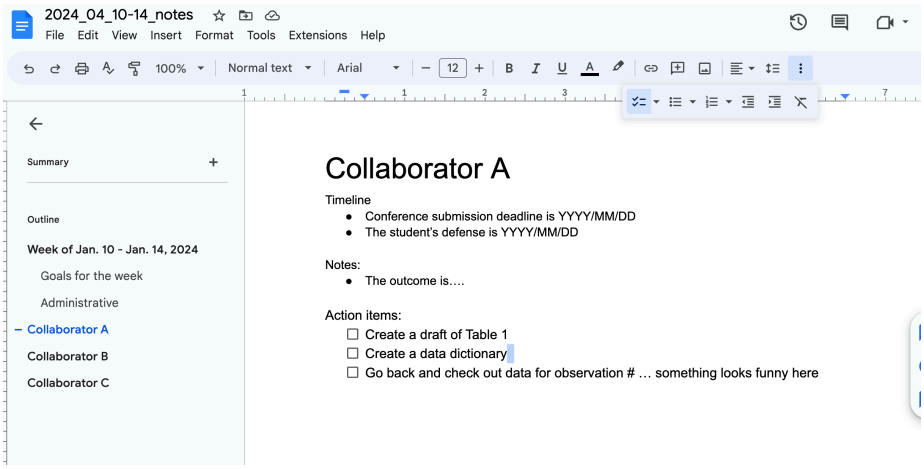



Figure 3.2: Detail notes for a specific collaborator

My notes for each project are typically sorted into times/deadlines, notes, and action items. Whereas my weekly goals are different each week, I probably won't address all the action items for each project in a given week. The notes on this "Collaborator A" page may are updated only as often I am working on this project/in contact with the collaborator.

At the start of the next week, I begin my copying the notes from the previous week (the whole file) and then updating the weekly goals. This is an iterative process, where the 'current' value becomes the 'old' value in the next iteration.

This system of file keeping has allowed me to focus my goals for each week, to measure the progress I am making, and to reference projects that resurface after a long break (which is typical). No one else has access to these notes except me, and I write them in 'shorthand' (using acronyms, etc.) knowing that these notes are just for my personal reference.

Notice that these notes are both 1) linear (chronological) and 2) project-specific. These features differentiate project notes from building a personal knowledge base. Personally, I have found it helpful to keep these two types of notes distinct. My project notes have lots of deadlines, dates, and to-do items – I don't want these things to get mixed up into my personal knowledge base notes.

### 3.3.2 README files

Each project on which I collaborate has a README file, which has the details that a future analyst would need to jump into a project mid-story. Typically, my README files contain:

- The names of the PI/collaborators
- The objective/scientific aim of the project
- The definitions of any project-specific acronyms
- Any notes on data analysis decisions (i.e., dichotomizing a continuous variable, etc.). Often, these notes are lines from an important email that I copy & paste into the README. This keeps a coherent record of the decision points in a project timeline.

For a deeper dive on writing README files, check out this tutorial article.

# Chapter 4

# Taking Notes

One of the most important aspects of studying at the graduate level is taking notes so as to **build a knowledge base**. Unlike taking notes in other contexts (like undergrad), the goal for a graduate student/GRA is not to keep a linear track of what you've learned over a finite period of time. In undergraduate classes, you take notes per lecture, per week, with the objective being a final at the end. You probably never look back to the notes for any particular class once the semester is over. But graduate school note taking has an altogether different goal. Instead of tracking what you are learning in a linear way for a finite time, you are building up what you are learning into a knowledge base to be referenced for the rest of your life.

So, I highly recommend that you take notes in a way that helps you connect concepts across coursework and RA work. During my 3rd year of grad school, my advisor shared some of his note-taking strategies with me. These strategies transformed the way I learn. Here are some of the ideas that led to that transformation.

## 4.1 The Zettlekasten system

One non-linear note taking approach is the Zettelkasten method. The Writing Cooperative has an article that describes the Zettlekasten system of taking notes. The Zettelkasten method allows you to connect your notes to one another, so that you end up with a web of ideas. This web is similar to the way Wikipedia is set up, with pages linked to other pages. Other variants of this kind of note taking include digital gardening and Evergreen notes.

## 4.2 How to connect your notes

So, this raises the question: where do you start in building up a knowledge base like this? I use the free app called Obsidian. Obsidian allows you to store all your notes in Markdown files on your local machine. There are other apps out there, like Roam. Just keep in mind that Roam does not have a free version (as of the time of this writing).

Some people who use this kind of note-taking system have made their notes publicly available. I have learned much from studying these peoples' notes:

- Andy Matuschak
- Maggie Appleton
- Bryan Jenks

# Chapter 5

# Cultivating professionalism

## 5.1 Data management

Many graduate research assistants have data management responsibilities in addition to data analysis. Oracle defines data management as the "…practice of collecting, keeping, and using data securely, efficiently, and cost-effectively." Based on these notes from the University of Mass., here are the general components of data management:

1. Write a data management plan
2. Specify a explicit set of operating procedures
3. Create a data dictionary
4. Determine a data storage location
5. Develop an analysis plan
6. Archive all of your procedures

I expand on each of these items below:

### 5.1.1 Write a data management plan

As soon as you receive a new project, you should establish your organization system [CROSS REF]. After this, your next step should be to spend some time contemplating a data management plan. This plan should describe how you will keep track of all of the data so that you can explain each step of your analysis process.

If collected data comes to you (especially if it comes via an Excel spreadsheet), you should immediately make a copy of this data. **You should never edit the original data you receive** - do not do any data cleaning or re-formatting.

Instead, do all of your work on the copy, so that you can always refer back to the original data.

If you are planning a study with yet-to-be- collected data, spend some time thinking about how the data will get to you.

- Will the data come from electronic records? If so, save the SQL query/other code used to extract the records.
- Will the data come from a chart review (e.g., a doctor will manually collect data from patient records)? If so, I recommend making a template for the doctor (or whoever is doing the chart review) to fill in.
- Think about each person who will interact with the data, and have a flow-chart in your mind that traces exactly how this data will come to you. This will help you when you write the "Methods" section of your paper/poster, and will keep you from making mistakes in your analysis downstream.

### 5.1.2 Specify a explicit set of operating procedures

If you are in a situation where multiple people will handle the data (as in the chart review scenario), you should go beyond 'having a flow chart in your mind' – in cases like these, write down who will do what with the data at each step of data collection. Send this written document to everyone on the team, and make sure everyone agrees/is on the same page.

If you are in the chart review scenario, instruct the person doing chart review on how to collect data that is in a consistent form. Remember that the computer will recognize these responses as four different values: "Yes", "yes", "y", .

### 5.1.3 Create a data dictionary

Create a spreadsheet with all the variables in the study listed in one column, and their definitions listed in the next column. As you write your scripts for analysis, you may even want to add a third column that lists the name you used in the script for that variable. The US Dept. of Agriculture offers some best practices on organizing a data dictionary.

### 5.1.4 Determine a data storage location

Where you keep your data matters. This is of particular importance when working with data protected by HIPPA (e.g., health records data). As a general principle, don't store data on your personal computer. Store it either online (e.g., a secure OneDrive folder, a GoogleDrive folder, an AWS location) or in an external drive (e.g., a company or university-owned secure shared drive).

### 5.1.5 Develop an analysis plan

At the outset of a project, outline the data analysis tools you think you will use based on the kind of data you expect to receive. Begin with the end in mind. Ask your collaborators these questions:

1. What is your research question?
2. What kind of interpretations/generalizations you want to be able to make at the end of the project?

### 5.1.6 Archive all of your procedures

Document what you have done for data cleaning and analysis. This documentation can include:

1. Well-organized, well-commented code (well-commented does not necessarily mean more comments; in fact comments should not explain what the code is doing).
2. Stick to your file structure; file names themselves can track what you have done.
3. Save a copy of every file you send out for others to read. Even as you update versions of a report, save one copy of each version you have shared.

## 5.2 What to do when ...

Here are some tricky situations that may arise when working in a GRA position, along with some suggestions on how to handle them.

### 5.2.1 The data is not organized in a way that is conducive to analysis

If you've received data from folks with whom you did not consult before data collection, you may find that the data are 'messy' - maybe there are lots of notes in one column, or the data you want is in the column names, or the values within columns are not consistent. There are several resources to help you with processing data in R and/or cleaning data in R. The packages in the tidyverse have been super helpful to me for these tasks. I **highly recommend** that you write functions for your data cleaning/data processing scripts. [CROSS REF]

Before you dig into the computing, though, two more important questions are:

1. **Does the PI know what data you have?** It is quite possible that the PI was not directly involved in data collection. When the PI does not know what data you have, there is room for miscommunication - for example, the names of the variables in the data may not align with what the terminology that the PI is using. You may consider sending the PI

a follow up email that describes the data you have (numbers of rows and columns, or a mock 'Table 1') and ask something like, "I want to confirm that I am looking at the right information as I begin to analyze your data. Is this [the description you've given] aligned with your expectations?"

2. **Does the data contain what you need to answer the PI's research question?** Related to the question above, if the PI is not familiar with the details of the data, it is possible that the data to which you have access is not adequate for addressing the research question. Again, describe the data you have and ask something like, "here is the data to which I have access; with my current understanding of your research question, I think you intend to analyze … Is my understanding correct? If so, some additional information is needed."

### 5.2.2 The data accidentally contain HIPPA protected information

This is one of those things that needs to be addressed early. Suppose that in your first exploration of a data set, you find that there is HIPPA protected (identifiable) data. **As soon as you realize you have this**, send an email to the person who shared the data with you and CC your GRA supervisor. Explain exactly what you have access to; say something like, "When I began to examine these data, I noticed that _____ information is included. This does not look like something to which I need access in order to address the research question. Please advise me on what needs to be done to de-identify these data?"

### 5.2.3 The PI is insistent on presenting results in a way that I believe is sub-optimal

This is a tricky one – talk to your supervisor about this problem and ask for advice. It may be that there is a happy medium; for example, if the PI is insistent on using p-values, it may be possible for you to convince them to present confidence intervals or some measure of effect size to the results as well.

Also, encourage your PI to name the limitation of the analysis choices you've made. For example, if you have done a bunch of multiple tests and have not corrected for multiple comparisons, you should mention that explicitly in the methods section of your work.

### 5.2.4 I realized I made a mistake in my code!

As soon as you've realized you made a mistake, sit down and take a minute to reconstruct when you made the mistake. Write down:

1. When you realized you made the mistake
2. Whose work will be impacted by this mistake

Once you have these things, consider your answer to (2). If the only people impacted by your mistake are other statisticians on your team, then email those people directly and explain what happened and what you are doing to make it right. If collaborators' (e.g., clinicians or administrators) work is impacted by your mistake, email your RA supervisor first and explain what happened. As your supervisor to help you craft an email to the impacted individuals.

## 5.3 Maintaining a CV

A curriculum vitae (CV) is not something you do once. It is something that requires constant maintenance. The tool I've found helpful for both 1) maintaining a CV that both looks professional and 2) is simple to update is the vitae package. The package site has a detailed README file with links to videos on how to set this up. I try to update my CV each time another publication is released.

## 5.4 Responding to reviewers

One more aspect of professionalism that is a part of many research jobs is publishing in academic journals. When you submit a manuscript for publication, the draft you submit is given to peer-reviewers. These reviewers offer anonymous feedback to critique your work. Knowing how to respond to these reviewers is an integral part of the writing process. My research supervisor taught me to use the point-to-point method for responding to reviewers. Templates for point-to-point response are available here and here — check out these templates to get an idea of how to write your responses.

# Chapter 6

# Your computing setup

Your computing setup is an essential part of working as a biostats GRA. It is like organizing a toolbox - you have to choose the tools to work with, invest in those tools, decide how to organize/store them, and then (of course) actually work with them.

Developing a 'toolbox' for quantitative work can have a lot of overhead – a steep learning curve – because systems for working in a computational/quantitative space are highly customizable. There are lots of ways to do things – and in may cases, there are multiple ways to do something well. The sheer number of options can make it hard to know where to start. So, that is the objective of this chapter. For each of the components of my current toolbox, I am sharing *one* way to begin. My goal is to help you get started, trusting that you will gradually customize your setup as you learn more.

I am writing as if I was writing to who I was 5 years ago – a **total** novice. If that's not you, feel free to skip around – I've tried to name the subheadings so that you can grab and go based on what you need from this chapter.

## 6.1 Setting up R and RStudio (now Posit)

### 6.1.1 Download and install R

The tools I use the most are `R`/RStudio - in addition doing statistical analysis, I also use these tools to write documents (like this book!) and create graphics for publications.

To get started with RStudio (now called Posit), you first need to install R. `R` is a programming language, meaning it provides a way to talk to your computer. `R` is specifically designed for statistics. Another great feature of `R` is that it is free – anyone can use it.

In order for your computer to understand `R`, you need to download it and install it on your computer. This is your computer's way of 'learning' a new language. Once you have downloaded R from the link above, you will have a file on your computer that you need to 'unpack' – click on what you've downloaded and follow the prompts to get `R` installed.

### 6.1.2 Download and install RStudio

After your computer is ready to speak `R`, you need to have a way to talk to your computer. Just like when talking to a person, you need to both speak their language *and* have a way to communicate with them. We talk to each other in a number of ways, like using phones and messaging apps – in an analogous way, there are multiple avenues for 'talking' to your computer. One wayo talk `R` with your computer is to use RStudio. RStudio is an application (an 'app') that will let you interact with your computer in ways that make you more productive. Follow the link above to install RStudio for your computer – be sure to click on the download option that matches the kind of operating system your computer has (e.g., Mac OSX, Windows, Linux). Again, you need to install what you download – click on the file that shows up in your downloads folder, and follow the prompts to install RStudio.

If you'd rather watch a video on how to install R and RStudio, check out this 15 minute video on YouTube.

### 6.1.3 Customize RStudio

Once you have RStudio installed, you will see that you have a 'Tools' option at the top of your screen. Follow this menu to Tools > Global Options. This will give you a pane that offers lots of settings. One cool thing to check out is the theme, meaning the color scheme/appearance of your terminal. You can choose a custom font or color palette in the 'Appearances' menu. I always use a dark mode theme (`Tomorrow night` is the theme I am using these days; `Dracula` is cool too).

You will see that in your default RStudio viewer, you have 4 panes: one is the console (probably bottom left); code runs here, but **nothing is saved here** – it is like a place for 'scratch work'. To save your work in a file, you need to open it in File > Open file (this will probably open in the top left). Third, you will have a pane that shows your files (like a 'Finder' window on Mac, or a 'Files' window on Windows) - this pane is usually at the bottom right. At the top right, you will see a pane with a bar at the top listing options like 'Environment', 'History', etc. `R` is an functional programming language (you can get really deep into thinking about this) – for our purposes now, just know that in `R`, the objects you create will show up in this 'Environment' pane.

### 6.1.4 Setting up your R library

Using `R` requires packages. Most packages that you use will come from one of three places: CRAN, Bioconductor, or someone's GitHub repository.

To download and use a package from CRAN, use

```
install.packages("package_name") # only need to do this once
library("package_name") # this is what you would do each time
```

To download a package from Bioconductor, use

```
install.packages("BiocManager")
BiocManager::install(c("package_name"))
library("package_name")
```

To download a package from a GitHub repository, use

```
install.packages("devtools")
devtools::install_github("user/repo")
```

When you download a package, the package is saved in location that is specified as your *library path*. If you want to know where that is, type

```
.libPaths()
```

```
## [1] "/Library/Frameworks/R.framework/Versions/4.1/Resources/library"
```

### 6.1.5 Tips: things you should never do in R/RStudio

1. In the Environment pane, there is an icon to 'save workspace as...' - **you should never save your R environment and/or your R workspace**. This is bad practice for a number of reasons, including that you do not have control of what is being saved. Better alternative: save only the object(s) you need using `saveRDS()`. You can always re-access these later with `readRDS()`.
2. Going back to the comment on file paths in the organization chapter (CROSS REF), there is a function in `R` called `setwd()` where you can tell the computer where (what folder) to point. When writing a script, **the first line of your script should never be** `setwd("a/directory/that/only/I/have)` – you want your code to be portable, so that other people can run it (and so that you can run it even after you have reorganized your folders!). Better alternative: use relative file paths, like the structure provided in the super handy here R package.

### 6.1.6 Cool keyboard shortcuts in R/RStudio

- to make a chunk of code align (e.g., to fix the indentation): highlight the chunk and then `CNTRL + I` (Windows), `Command + I` (OSX)

- to create a new R script: `CNTRL + Shift + N` or `Command + Shift + N`
- to see all available keyboard shortcuts: `ALT/Option + Shift + K`

## 6.2 Setting up your terminal (i.e., the command line)

There are two ways to interact with your computer: via 'point and click' interfaces (e.g., File Explorer, Finder) and via the **command line**. The command line is much more efficient for many computational tasks – I encourage anyone working in biostatistics to begin learning to use the command line for creating/moving/manipulating files, as this will be good preparation for doing more complicated work (e.g., writing your own `R` packages and so on).

On a Mac, you can open the command line by searching for the **'Terminal'** app (this is installed on every Mac by default). On Windows, the analog is called **'Command Prompt.'** You have a lot of choices in addition to the apps that are on your computer by default. I use the app iTerm2, and I customized the appearance of my terminal window with Oh My ZSH.

As a first step in learning to work with the command line, try to find your files – in the video below, I show how I do this with symbolic links ('symlinks').

Here is a quick video of me doing this.

More details on working with the command line are explained in the online course called "The missing semester of your computer science (CS) education." This free online course was compiled by people at MIT. I have found this to be a really helpful place to begin.

### 6.2.1 Resources for learning the command line

- ARStechnica: command line wizardry, part I

-ARStechnica: command line wizardry, part II

-Linux Command line – official tutorial that starts from the $_{\text{very}}$ beginning

- Quick-start guide to the VIM text editor – VIM is a popular choice for text editing (other kinds of text editors are TextEdit (ships with Mac), NotePad/NotePad ++ (Windows))

## 6.3 Getting started with Git

Git is a free and open source tool for version control. What this means is that Git allows you to avoid the problems that arise when you try to work with multiple copies of code for a project. Have you ever had a folder with files like this: `analysis1.R`, `04-05-2023_analysis.R`, `old_analysis.R`, `final_version.R`,

and `final_final_version.R`? This way of working is not only inefficient, but also inhibits collaboration and makes your work harder to reproduce. None of that is good for science. So, Git offers a way to manage versions, so that you can keep track of the current version of your code and have access to looking back at older versions.

There are a lot of online tutorials to get you started with Git – just Google 'git tutorial for beginners' and several free options will show up. Again, the missing semester of your CS education course has a whole section on learning version control – highly recommend.

Here is a brief overview of a few major vocab words you need to know to use Git (in the order in which you will probably encounter them)[1]:

- **repository ('repo')** (noun): The directory (a.k.a., the folder) that holds all the files for a project
- **local** versus **remote** (adjectives): These terms describe the location of a repo. A local repo exists only on one computer - for instance, a project that I have created on my laptop is a local repo. A remote repo is hosted online (probably on GitHub). Sort of like keeping files in OneDrive or GoogleDrive, remote repositories make it possible for multiple people to access a project from any computer.
- **clone** (verb): The action of making a copy of a repo; e.g., "I will clone Person A's remote repo onto my laptop." The command line code for this is `git clone <repo>`
- **pull** (verb): The action of incorporating all of the changes from a remote repo into a local repo. Example: "My local copy of Repo A is out of sync with the master copy, since Person B made some more changes to our code yesterday. I need to pull those changes from our remote repo into my local repo." Pulling is done with `git pull`.
- **stage** (verb): The action of preparing changes for a commit. This is like looking at your scratch work and deciding which of your ideas you want to keep. This is done with `git add <edited files to keep>`.
- **commit** (noun or verb): A commit (the noun) refers to a documented change to one or more files in a project. To make a commit (verb) is the action of saving a change. In broad, general terms, Git keeps a timeline of all the changes in a project. Once something is committed in Git, the changes that were part of that commit are permanently saved in the project timeline. To make a commit, use `git commit -m "a message describing the changes made"`

[1] **Disclaimer**: my training has consisted of much more math than computer science; what I am presenting here pertaining to version control is a broad and loose conceptual sketch.

- **push** (verb): The action of moving commits you made in your local copy of the repo to the remote copy of the repo. Example: Me and my colleague Y are working on project A together. The master copy of project A is in our remote repo; me and Y have each cloned this repo, so we have local copies on our respective laptops. I have committed some changes in the files on my local copy, so now I need to *push* my commits to the remote repo. After I push, Y will be able to see what I did."
- **branch** (noun): When working on a more complex project, there arises a need to keep two versions of the project: a 'master' version that has everything you want to keep for sure, and a 'development' version where you are debugging/troubleshooting/playing with new ideas. Having multiple branches in your repo allows you to achieve having multiple versions of a project as I've described. The commands that go along with branching are usually `git branch` and `git checkout`.

### 6.3.1 Resources for learning Git

- Quick-start: setting up Git
- Full book on Git - well divided into sections, so you can skip around to the chapters most relevant to what you need

## 6.4 Using high performance computing clusters

In many contemporary applications, the computational work for data analysis needs to be done in a high performance computing cluster. At the University of Iowa, our high performance computing cluster is called Argon. My appreciation for Argon grows the more I work on my dissertation. One resources I have found super helpful in working with high performance computing clusters is the online tutorial written by Profs. Patrick Breheny and Grant Brown. This tutorial outlines how to use the UI Argon cluster. Much of what is here would generalize to other HPC systems as well.

### 6.4.1 Interactive sessions v. submitting jobs

There are two ways to interact with a high performance computing cluster: an *interactive session* and a *job submission*. An interactive session involves you logging into the computer and telling it what to do one step at a time, whereas a job submission means you give the computer a whole set of instructions at once. In an interactive session, you stay logged in and watch the screen for results as they come in. In a job submission, you send you instructions (via a *script*) to the computer and then you are free to log out – the job will run until it either hits and error or finishes the job, and it will notify you in either case (for Argon, you get an email when your job is finished running). From here, you

can see that jobs which you expect to take more than a couple of hours should be done via job submission.

### 6.4.2 Writing your first bash script

For submitting jobs to a computing cluster, you will need to write a *script* – the set of instructions for the computer. Here is a tutorial for how to write a simple `bash` script – this is a great place to start if you are new to script writing.

### 6.4.3 Conda environments v. module configurations

When it comes to using version-specific programs (this comes up a lot when working with `Python`), you will need to change the version of a tool that your computer is 'pointing at' depending on the work you are doing. Example: to use the LDSC command line tool for genetic correlation analysis, you need Python version 2.7 – but Python 3.10 is the most updated release of Python. In such a scenario, you need to tell your computer which version of Python to use. Moreover, you want to create a 'setup' for working with a tool like LDSC, and then every time you need to use this tool, you tell you computer: "load this special configuration of software" and it 'remembers' what configuration that is. Saving a configuration in this way can be done with either `conda` environments or `module` configurations. Conda is a package management system that you can download for your computer and then use the command `conda` to *create custom environments*. Alternatively, some computer systems (like the UI Argon HPC) have pre-installed capability to *configure custom modules*. One thing that gave me issues early on in my dissertation work is that I was trying to create a `conda` environment inside of my module configuration. **Don't mix and match the two approaches** – choose to either create environments or configure modules, and stick with you choice. Your choice here will depend in large part to the type of computer system you are working with, e.g., on UI Argon, it is much simpler to configure modules because that is already pre-installed and set to the default. Checkout the program files and modules section of the HPC tutorial for an example of this syntax.

## 6.5 Advanced R and some Julia

Just for reference, here are some links I have found helpful for learning more advaned R and some Julia – special thanks again to Grant Brown for touching on these ideas in his course on Advanced Computing:

- Advanced R, 2nd edition – again, well-divided into sections, so its simple to navigate
- Efficient R programming – this is really great for people who are learning R with more of a math/stats background than a computing background.

Regarding tips for writing better R code, **there are some real gems here**.

- Free course in Julia – yes, it really is free.

# Chapter 7

# Writing reports

## 7.1 What makes a report 'good'?

The final product for a project is usually a report, rather than a script or a set of unformatted results. Once you have your statistical estimates, consider how you can craft the results into a story. Good research reports effectively communicate stories.

Here are some general features of reports that tell stories:

- Reproducibility: can you reproduce your report in one line of code? If not, this is usually a red flag.
- Easily navigated by audience: don't send raw output to collaborators!
- Outlines context, methods, results, and limitations. You need all four to tell the truth.
- Summarizes information in graphs whenever possible.

Here is a video showing an example of one of my reports.

## 7.2 Tools for writing reports

**R packages to create tables**:

- gtsummary - best overall; best for displaying regression model estimates; 'prettiest' output
- compareGroups - best bivariate testing table (shows ORs)
- arsenal - easiest to customize

**Resources to format reports**:

- kable/kableExtra R packages
- tutorial on LaTeX tables in PDF
- tutorial on tables in HTML

# Chapter 8

# Giving presentations

A presentation is effective when everyone in the audience learns something. With that in mind, craft your presentations with the audience in mind. If you're using PowerPoint/Beamer/some other slide deck tool, use more pictures than words. As Dr. Melissa Marshall says, "Bullets kill" – meaning too many bullet points will kill your presentation.

## 8.1 Tools for giving presentations

- Melissa Marshall's resources at Present Your Science. There's a lot here (including a TED talk) on how to use slideshows to effectively communicate to diverse audiences. 10/10, highly recommend.
- Quarto presentations - unlike Rmd files, you can knit to PowerPoint in Quarto. Also, Quarto is great for presentations where you need to show blocks of code.
- University of Iowa Beamer slides template. This template is UI specific, and the color schemes stay within the guidelines of the UI brand manual.

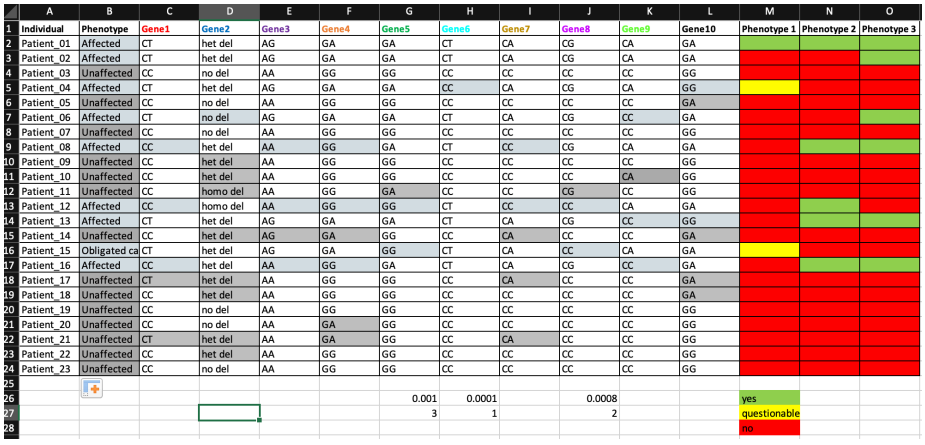
# Chapter 9

# Data processing and preparation

One of the most challenging aspects of a project can be preparing the data for analysis. In real life, our data seldom looks like something from `mtcars`:

```
knitr::kable(
  head(mtcars[, 1:8], 10),
  booktabs = TRUE,
  caption = 'Canonical example of clean data'
)
```

Instead, I often receive data that looks like this:



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Individual | Phenotype | Gene1 | Gene2 | Gene3 | Gene4 | Gene5 | Gene6 | Gene7 | Gene8 | Gene9 | Gene10 | Phenotype 1 | Phenotype 2 | Phenotype 3 |
| 2 | Patient_01 | Affected | CT | het del | AG | GA | GA | CT | CA | CG | CA | GA | | | |
| 3 | Patient_02 | Affected | CT | het del | AG | GA | GA | CT | CA | CG | CA | GA | | | |
| 4 | Patient_03 | Unaffected | CC | no del | AA | GG | GG | CC | CC | CC | CC | GG | | | |
| 5 | Patient_04 | Affected | CT | het del | AG | GA | GA | CC | CA | CG | CA | GG | | | |
| 6 | Patient_05 | Unaffected | CC | no del | AA | GG | GG | CC | CC | CC | CC | GA | | | |
| 7 | Patient_06 | Affected | CT | no del | AG | GA | GA | CT | CA | CG | CC | GA | | | |
| 8 | Patient_07 | Unaffected | CC | no del | AA | GG | GG | CC | CC | CC | CC | GG | | | |
| 9 | Patient_08 | Affected | CC | het del | AA | GG | GA | CT | CC | CG | CA | GA | | | |
| 10 | Patient_09 | Unaffected | CC | het del | AA | GG | GG | CC | CC | CC | CC | GG | | | |
| 11 | Patient_10 | Unaffected | CC | het del | AA | GG | GG | CC | CC | CC | CA | GG | | | |
| 12 | Patient_11 | Unaffected | CC | homo del | AA | GG | GA | CC | CC | CG | CC | GG | | | |
| 13 | Patient_12 | Affected | CC | homo del | AA | GG | GG | CT | CC | CC | CA | GA | | | |
| 14 | Patient_13 | Affected | CT | het del | AG | GA | GA | CT | CA | CG | CC | GG | | | |
| 15 | Patient_14 | Unaffected | CC | het del | AG | GA | GG | CC | CA | CC | CC | GA | | | |
| 16 | Patient_15 | Obligated ca | CT | het del | AG | GA | GG | CT | CA | CC | CA | GA | | | |
| 17 | Patient_16 | Affected | CC | het del | AA | GG | GA | CT | CA | CG | CC | GA | | | |
| 18 | Patient_17 | Unaffected | CT | het del | AA | GG | GG | CC | CA | CC | CC | GA | | | |
| 19 | Patient_18 | Unaffected | CC | het del | AA | GG | GG | CC | CC | CC | CC | GA | | | |
| 20 | Patient_19 | Unaffected | CC | no del | AA | GG | GG | CC | CC | CC | CC | GG | | | |
| 21 | Patient_20 | Unaffected | CC | no del | AA | GA | GG | CC | CC | CC | CC | GG | | | |
| 22 | Patient_21 | Unaffected | CT | het del | AA | GA | GG | CC | CA | CC | CC | GG | | | |
| 23 | Patient_22 | Unaffected | CC | het del | AA | GG | GG | CC | CC | CC | CC | GG | | | |
| 24 | Patient_23 | Unaffected | CC | no del | AA | GG | GG | CC | CC | CC | CC | GG | | | |
| 25 | | | | | | | | | | | | | | | |
| 26 | | | | | | | 0.001 | 0.0001 | | 0.0008 | | | yes | | |
| 27 | | | | | | | 3 | 1 | | 2 | | | questionable | | |
| 28 | | | | | | | | | | | | | no | | |

Figure 9.1: An example adapted from real unprocessed data

Table 9.1: Canonical example of clean data

| | mpg | cyl | disp | hp | drat | wt | qsec | vs |
|---|---|---|---|---|---|---|---|---|
| Mazda RX4 | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.620 | 16.46 | 0 |
| Mazda RX4 Wag | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.875 | 17.02 | 0 |
| Datsun 710 | 22.8 | 4 | 108.0 | 93 | 3.85 | 2.320 | 18.61 | 1 |
| Hornet 4 Drive | 21.4 | 6 | 258.0 | 110 | 3.08 | 3.215 | 19.44 | 1 |
| Hornet Sportabout | 18.7 | 8 | 360.0 | 175 | 3.15 | 3.440 | 17.02 | 0 |
| Valiant | 18.1 | 6 | 225.0 | 105 | 2.76 | 3.460 | 20.22 | 1 |
| Duster 360 | 14.3 | 8 | 360.0 | 245 | 3.21 | 3.570 | 15.84 | 0 |
| Merc 240D | 24.4 | 4 | 146.7 | 62 | 3.69 | 3.190 | 20.00 | 1 |
| Merc 230 | 22.8 | 4 | 140.8 | 95 | 3.92 | 3.150 | 22.90 | 1 |
| Merc 280 | 19.2 | 6 | 167.6 | 123 | 3.92 | 3.440 | 18.30 | 1 |

Instead of Google searching how to handle each step of processing data like this, this chapter will offer a step-by-step outline of the whole game. I will show you what packages and strategies I used to prepare these data for analysis in R.

Aside: when I need to process raw data, I like to begin by sketching the data frame I want to build. Big fan of the white board for tasks like this.

## 9.1 Libraries

After a sketch, I start with choosing some tools for working with color-coded Excel data. Beside each library I load, I'll comment the reason I find it useful:

```
# for data processing
library(readxl) # read in .xlsx documents
library(dplyr) # transform & clean data (with 'pivot_' functions)
library(tidyr) # transform & clean data
library(tidyxl) # read in .xlsx *formatting*, including color codes

# for visualization
library(ggplot2) # drawing plots
library(knitr) # creating tables (has 'kable()' function)
library(kableExtra) # making tables pretty
library(viridis) # choose color palettes
```

## 9.2 Reading in the data

Here, I read in the data multiple ways to obtain different information. The first line creates `raw`, which reads in the data without any formatting information. This will make the phenotype columns have all `NA` values. I need the other `raw_`

objects to get that information into a workable form.

```
raw <- read_excel(path = "data/unprocessed_data_example.xlsx")
raw_cells <- xlsx_cells(path = "data/unprocessed_data_example.xlsx")
raw_formats <- xlsx_formats(path = "data/unprocessed_data_example.xlsx") # has color codes
raw_id <- raw_cells[raw_cells$col == 1, c("address", "row", "col", "character")]
```

## 9.3 Work with color codes

Next, I need to fill in the phenotype information that is color-coded. The colors will be included in `raw_formats` as HEX codes (without the leading hashtag).

```
colors <- raw_formats$local$fill$patternFill$fgColor$rgb
# NB: colors are given here in order of appearance on the sheet (reading top to bottom)
```

To help me work with the color codes, I wrote a function called `cells_by_color`. **As soon as you catch yourself being tempted to copy and paste a chunk of code several times, it is time to write a function!**. This is the *most important message* of the tutorial. Data cleaning is a place where there is no room to be sloppy.

```
#' a function to return the cells of a given color
#' @param formats An object returned from tidyxl::xlsx_formats
#' @param cells An object returned from tidyxl::xlsx_cells
#' @param color The hex color WITHOUT the leading "#" mark
#' @return a tibble with the addresses of the cells in that color
#'
cells_by_color <- function(formats, cells, color){
  colors <- formats$local$fill$patternFill$fgColor$rgb

  cells[cells$local_format_id %in% which(colors == color),
                      c("address", "row", "col")]

}
```

Now, to see my function in action:

```
# NB: column M in raw data = phenotypeotype 1
# column N in raw data = phenotypeotype 2
# column O in raw data = phenotypeotype 3

green <- cells_by_color(formats = raw_formats,
                        cells = raw_cells,
                        color = "FF92D050") |>
  mutate(phenotype = case_when(
    substr(address, 1, 1) == "M" ~ "phenotype1",
    substr(address, 1, 1) == "N" ~ "phenotype2",
```

```
    substr(address, 1, 1) == "O" ~ "phenotype3"
  ),
  color = "green")

red <- cells_by_color(formats = raw_formats,
                      cells = raw_cells,
                      color = "FFFF0000") |>
  mutate(phenotype = case_when(
    substr(address, 1, 1) == "M" ~ "phenotype1",
    substr(address, 1, 1) == "N" ~ "phenotype2",
    substr(address, 1, 1) == "O" ~ "phenotype3"
  ),
  color = "red")


yellow <- cells_by_color(formats = raw_formats,
                         cells = raw_cells,
                         color = "FFFFFF00") |>
  mutate(phenotype = case_when(
    substr(address, 1, 1) == "M" ~ "phenotype1",
    substr(address, 1, 1) == "N" ~ "phenotype2",
    substr(address, 1, 1) == "O" ~ "phenotype3"
  ),
  color = "yellow")
```

With the color codes labeled, I am ready to create a data frame with both genotype and phenotype information:

(As of 2023/06/19, this chapter is still in progress… will add more below).

```
phen1 <- bind_rows(
  red |> filter(phenotype == "phenotype1"),
  green |> filter(phenotype == "phenotype1"),
  yellow |> filter(phenotype == "phenotype1")
) |>
  mutate(phen1 = case_when(
    color == "green" ~ 1,
    color == "red" ~ 0,
    color == "yellow" ~ 0.5
  )) |>
  right_join(raw_id, by = "row") |>
  select(c("character", "phen1")) |>
  # drop cells with no data (left over from header row)
  drop_na(phen1)
```

```
phen2 <- bind_rows(
  red |> filter(phenotype == "phenotype2"),
  green |> filter(phenotype == "phenotype2")
) |>
  mutate(phen2 = case_when(
    color == "green" ~ 1,
    color == "red" ~ 0
  )) |>
  right_join(raw_id, by = "row") |>
  select(c("character", "phen2")) |>
  # drop cells with no data (left over from header row)
  drop_na(phen2)


phen3 <- bind_rows(
  red |> filter(phenotype == "phenotype3"),
  green |> filter(phenotype == "phenotype3")
) |>
  mutate(phen3 = case_when(
    color == "green" ~ 1,
    color == "red" ~ 0
  )) |>
  right_join(raw_id, by = "row") |>
  select(c("character", "phen3")) |>
  # drop cells with no data (left over from header row)
  drop_na(phen3)
```

## 9.4 Final steps

I will write one more function to help me with the last step:

```
#' a function to retrieve the Nth value in an strsplit() list
#' @param x The character to split
#' @param split The split
#' @param which The numeric value indicating which item to retrieve
split_which <- function(x, split, which){
  split_x <- strsplit(x, split) |> lapply(function(l){l[which]})
  return(unlist(split_x))
}
```

Finally, I have:

```
phen <- full_join(phen1, phen2) |>
  full_join(phen3) |>
  rename(individual = character) |>
```

```
  filter(individual != "Individual") |>
  # need numeric ID for pedigreemm functions
  mutate(id = split_which(individual, "_", 2))
```

```
## Joining with `by = join_by(character)`
## Joining with `by = join_by(character)`
```

```
wide <- left_join(phen, raw,
                  by = c("individual" = "Individual")) |>
  # analysis decision: make the obligated carrier unaffected, as per discussion
  #   with clinicians
  mutate(phenotype = if_else(Phenotype == "Obligated carrier (unaffected?)",
                             "Unaffected",
                             Phenotype),
         id = as.numeric(id)) |>
  select(individual, id, phenotype, starts_with('Gene'), paste0("phen", 1:3)) |>
  arrange(id)

# snapshot of final data
knitr::kable(wide[,c(1:6,14:16)],
       booktabs = TRUE,
       caption = 'Final product - ready to analyze/graph')
```

Table 9.2: Final product - ready to analyze/graph

| individual | id | phenotype | Gene1 | Gene2 | Gene3 | phen1 | phen2 | phen3 |
|---|---|---|---|---|---|---|---|---|
| Patient_01 | 1 | Affected | CT | het del | AG | 1.0 | 1 | 1 |
| Patient_02 | 2 | Affected | CT | het del | AG | 0.0 | 0 | 1 |
| Patient_03 | 3 | Unaffected | CC | no del | AA | 0.0 | 0 | 0 |
| Patient_04 | 4 | Affected | CT | het del | AG | 0.5 | 0 | 0 |
| Patient_05 | 5 | Unaffected | CC | no del | AA | 0.0 | 0 | 0 |
| Patient_06 | 6 | Affected | CT | no del | AG | 0.0 | 0 | 1 |
| Patient_07 | 7 | Unaffected | CC | no del | AA | 0.0 | 0 | 0 |
| Patient_08 | 8 | Affected | CC | het del | AA | 0.0 | 1 | 1 |
| Patient_09 | 9 | Unaffected | CC | het del | AA | 0.0 | 0 | 0 |
| Patient_10 | 10 | Unaffected | CC | het del | AA | 0.0 | 0 | 0 |
| Patient_11 | 11 | Unaffected | CC | homo del | AA | 0.0 | 0 | 0 |
| Patient_12 | 12 | Affected | CC | homo del | AA | 0.0 | 1 | 0 |
| Patient_13 | 13 | Affected | CT | het del | AG | 0.0 | 1 | 1 |
| Patient_14 | 14 | Unaffected | CC | het del | AG | 0.0 | 0 | 0 |
| Patient_15 | 15 | Unaffected | CT | het del | AG | 0.5 | 0 | 0 |
| Patient_16 | 16 | Affected | CC | het del | AA | 0.0 | 1 | 1 |
| Patient_17 | 17 | Unaffected | CT | het del | AA | 0.0 | 0 | 0 |
| Patient_18 | 18 | Unaffected | CC | het del | AA | 0.0 | 0 | 0 |
| Patient_19 | 19 | Unaffected | CC | no del | AA | 0.0 | 0 | 0 |
| Patient_20 | 20 | Unaffected | CC | no del | AA | 0.0 | 0 | 0 |
| Patient_21 | 21 | Unaffected | CT | het del | AA | 0.0 | 0 | 0 |
| Patient_22 | 22 | Unaffected | CC | het del | AA | 0.0 | 0 | 0 |
| Patient_23 | 23 | Unaffected | CC | no del | AA | 0.0 | 0 | 0 |

# Chapter 10

# Power and sample size calculations

Power and sample size calculations are a common request for a GRA. In order to estimate the power or calculate the necessary sample size, you will need some measure of effect size. Often, the PI may not have an effect size in mind – in this case, offer some example scenarios and gauge the PI's reaction. Examples:

- "If you saw that a subject being treated in group B had a 2 mm increase in the outcome, would you be impressed? Would that be a 'big' or notable change?"
- "How much of an impact would you need to see in order to catch you attention in an abstract? A 50% increase? Double the outcome?"

Your calculations for power/sample size could be done several ways in `R`: the `pwr` and `pwr2` packages can be a good place to start. Suppose you are planning an ANOVA study with these specifications:

- desired power - 85%
- outcome - 4 level ordered factor which I will treat as numeric (0 - 3)
- main predictor - 3 level factor (treatment A, treatment B, positive control (C))
- desired alpha = 0.05

Suppose your PIs don't know what effect size they are looking for, but they do offer some preliminary data from a similar study. Begin with that preliminary data:

```
library(dplyr)
prelim <- matrix(data = c("A", 1.94, 1.20,
                          "A", 2.75, 0.39,
                          "B", 0.15, 0.55,
```

```
                              "B", 0.33, 0.74),
                    nrow = 4,
                    byrow = T)
prelim <- prelim |>
  as.data.frame() |>
  mutate(across(.cols = 2:3, .fns = as.numeric)) |>
  # collapse isthmus and canal measures, since our study is primarily
  # aimed at comparing treatments A and B with standard of care
  group_by(V1) |>
  # average the mean and sd of each A/B group
  summarise(mean = mean(V2),
            sd = mean(V3))

names(prelim) <- c("measure", "mean", "sd")
```

Next, you can break down the types of variation:

```
# estimate the between group variance to be the variance in mean
btw_group_var <- var(prelim$mean)

# estimate the common variance to be the average of the reported variances
within_group_var <- mean((prelim$sd)^2)
```

After taking these steps, I recommend calculating power using multiple tools and comparing the results you see:

```
# using 'stats' package - this ships with R
power.anova.test(groups = 3,
                 n = NULL,
                 between.var = btw_group_var,
                 within.var = within_group_var,
                 sig.level = 0.05,
                 power = 0.90)
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##           groups = 3
##                n = 2.799745
##     between.var = 2.215512
##      within.var = 0.524025
##       sig.level = 0.05
##           power = 0.9
##
## NOTE: n is number in each group
```

```r
# using pwr2
library(pwr2)
# using 'pwr2' package
ss.1way(k = 3,
        alpha = 0.05,
        beta = 0.1,
        delta = abs(diff(prelim$mean)),
        sigma = mean(prelim$sd),
        B = 100)
```

```
##
##      Balanced one-way analysis of variance sample size adjustment
##
##               k = 3
##       sig.level = 0.05
##           power = 0.9
##               n = 5
##
## NOTE: n is number in each group, total sample = 15
```

The first method estimated that 3 observations would be needed in each group, while the second method estimated this value to be 5. In this case, 5 was a feasible number of observations to achieve, so I reported that value to the PIs. This higher value will also help maintain the level of power desired if I have to pivot to a non-parametric test like the Kruskal-Wallis.

## 10.1 SAS

Note that SAS offers the `PROC POWER` procedure for doing power and sample size calculations. I have often used both R and SAS functions for the same calculation and compared the results. Typically, SAS has much more in-depth documentation – if you are wondering about the theoretical details for a calculation, SAS documentation is a helpful resource.

# Chapter 11

# Survival (time-to-event) analysis

Survival analysis is one of the problem-solving settings that separates biostatistics from pure statistics. Survival analysis research questions ask, "How long until (some event) happens?". This is a messy question, especially in biomedical research contexts where the investigators do not observe what happens to every observational unit (e.g, every patient) in the study. A patient may come to the clinic once, receive treatment, and then never come back to the clinic again. Whatever outcome the investigators wanted to observe remains unknown for that patient – we would say the outcome for such a patient is *censored*.

The most-used methods in survival analysis are the *log-rank test* and the *Cox proportional hazards model*. The log-rank test is a conceptual analog to a Cochran-Mantel-Haenzel test in categorical data analysis. When working with a survival outcome and a single categorical predictor, the log-rank test can be used to test the null hypothesis: "there is no difference in survival between the groups being compared." The Cox proportional hazards model is a multivariate regression approach in which the exponentiated coefficients are the *hazard ratios*.

[ADD A WORD BANK HERE]

## 11.1 Assumptions/diagnostics

[ADD INFO HERE]

## 11.2 Examples:

- Clinical trials: patients battling a chronic condition are randomized to either new drug B or the standard of care drug A. Patients have follow-up

visits once per month for 1 year to monitor the time until their symptoms flare up again (e.g., time to relapse or time to recurrence).

- Dentistry: electronic dental records are used to assess the time until re-intervention for crown margin repairs (CMRs). CMRs are compared to assess which materials last longer: glass ionomer, resin-modified glass ionomer, resin-based composite, and amalgam.
- Industry/manufacturing: Suppose there are four machines on a factor floor, two from brand A and two from brand B. The time until next malfunction could be used to compare the two brands of machines.

## 11.3 R code tips

Packages to know about:

- `survival`: a must-have for survival analysis
- `ggsurvfit`: more options for for graphics

## 11.4 References

- Emily Zaboor's survival analysis tutorial in R. Good quick-reference for code and plots.
- Patrick Breheny's publically available course notes for survival analysis. Lots of examples here, goes in depth in the theory.

# Chapter 12

# Survey data analysis

One of my more recent projects has involved working with the UI Public Policy Center to study data from the Pregnancy Risk Assessment and Monitoring System (PRAMS). PRAMS contains survey data from participants across the US and its territories. The data I have are a subset of PRAMS that describes participants from specific years who used Medicaid during their pregnancies.

I will give an overview of the tools and tips I have found helpful as I have worked with survey data

## 12.1 Tools for analyzing survey data

- The survey R package is the classic tool for this type of analysis.
- The srvyr R package offers `tidyverse` syntax of the `survey` package functions - this was so helpful for subsetting and transforming the data to fit my research objective.
- The svydiags R package offers several functions for model diagnostics with survey data

## 12.2 Tips

1. Whenever you subset the data from a survey, you must always consider the survey weights! In R, create a `svy.design` object *before* you subset.
2. The `gtsummary` package has functions that are designed to present results from a survey - take advantage of this when making tables
3. Acronymn to know: "FPC" = **f**inite **p**opulation **c**orrection. This is a common variable included in survey data – if you have this, you should use it when you create your 'design' object in `R`.

## 12.3 Quick demo

Suppose our research question pertains to the academic performance index (api) of elementary schools. Using the `apipop` data from the `survey` package, we will investigate this question.

```
library(survey)
library(srvyr)

data(api) # data from survey package

# first, create a 'design' object
api_design <- apistrat |>
  as_survey_design(ids = 0, # if there was a clustering variable, it'd go here
                   strata = stype, # stratification variable
                   fpc = fpc, # finite population correction
                   weights = pw # survey weights
                   )
```

After creating a 'design' object, I can filter down to a subset of interest. Supposing I am interested in schools from LA county, I can examine the api of students in LA county alone:

```
# look at LA county only
la <- api_design |>
  filter(cname == "Los Angeles")

# summary table & bivariate tests
library(gtsummary)
interest_vars <- c("pcttest", "both", "meals", "ell", "yr.rnd", "mobility",
                   "avg.ed", "full","enroll", "api.stu")
tbl_svysummary(la,
               # include only select variables in the table
               include = all_of(interest_vars),
               by = "both",
               type = list(all_dichotomous() ~ "categorical")) |>
  # formatting
  bold_labels() |>
  # tests
  add_p(test = list(all_categorical() ~ "svy.chisq.test",
                    all_continuous() ~ "svy.t.test"))
```

| **Characteristic** | **No**, N = 620 | **Yes**, N = 753 | **p-value** |
|---|---|---|---|
| ___pcttest___ | 99 (97, 99) | 100 (99, 100) | 0.10 |
| ___meals___ | 72 (28, 93) | 60 (35, 82) | 0.8 |
| ___ell___ | 23 (10, 34) | 20 (7, 46) | 0.6 |
| ___yr.rnd___ | | | 0.067 |
| No | 585 (94%) | 576 (77%) | |
| Yes | 35 (5.7%) | 177 (23%) | |
| ___mobility___ | 16 (12, 19) | 19 (13, 23) | 0.11 |
| ___avg.ed___ | 2.52 (2.20, 3.27) | 2.74 (2.26, 3.35) | 0.6 |
| ___full___ | 76 (52, 85) | 74 (67, 87) | 0.2 |
| ___enroll___ | 446 (413, 936) | 401 (288, 673) | 0.056 |
| ___api.stu___ | 401 (325, 628) | 338 (257, 571) | 0.072 |

```
  # NB: survey-specific tests available; these tests incorporate/adjust for
  #   survey weights
```

I can also summarize a multivariate model:

```
lhs <- paste(interest_vars, collapse="+") # lhs = left hand side (of equation)
rhs <- paste0("api99~", lhs)

fit <- lm(as.formula(rhs),
          data = apistrat)

tbl_regression(fit) |> bold_labels()
```

| **Characteristic** | **Beta** | **95% CI** | **p-value** |
|---|---|---|---|
| ___pcttest___ | 1.0 | -1.3, 3.3 | 0.4 |
| ___both___ | | | |
| No | — | — | |
| Yes | -3.2 | -21, 15 | 0.7 |
| ___meals___ | -1.6 | -2.2, -0.98 | <0.001 |
| ___ell___ | 0.13 | -0.51, 0.77 | 0.7 |
| ___yr.rnd___ | | | |
| No | — | — | |
| Yes | 2.6 | -27, 32 | 0.9 |
| ___mobility___ | -0.05 | -0.76, 0.66 | 0.9 |
| ___avg.ed___ | 87 | 65, 109 | <0.001 |
| ___full___ | 1.5 | 0.81, 2.2 | <0.001 |
| ___enroll___ | -0.07 | -0.13, 0.00 | 0.042 |
| ___api.stu___ | 0.04 | -0.03, 0.11 | 0.3 |

# Chapter 13

# Longitudinal data analysis

## 13.1 Tools for longitudinal data analysis

- lme4 R package - fits all your basic linear mixed models
- nlme R package - offers functionality for nonlinear mixed effect models

# Chapter 14

# Microbiome data analysis

Microbiome data analysis requires working with two separate subtypes of data: the taxonomic data and the sample (meta)data. Taxonomic data is *hierarchical*, meaning that the data are structured in tiers. These tiers are typically the Operational Taxonomic Units (OTUs). The sample data is the data describing the observations (people or subjects) in the data, and often includes demographic/clinical features.

## 14.1 Tools for working with microbiome data

- metacoder and taxa are `R` packages built on previous packages vegan and phyloseq. All of these links include tutorials for working with taxonomic data.

# Chapter 15

# Genetics (GWAS) data analysis

The rate at which new tools for GWAS data analysis are created is mind-boggling. It seems like a new paper in this area comes out each week – which makes it a challenge to summarize the tools available for working in this area. With this in mind, consider the set of tools and tips provided here as select suggestions from my own experience rather than an exhaustive list.

## 15.1 R packages

- bigsnpr - a relatively new package that offers functions for manipulating PLINK data files, doing QC and relatedness/imputation, pruning/clumping and correlation analysis, and calculation of polygenic scores. Functions for basic marginal association analysis are also available.
- snpStats - an older, widely-used Bioconductor package that has functions for doing QC, imputation, and marginal association analysis. Note: this package is not as memory efficient as `bigsnpr`.
- ncvreg - a newer package that implements penalized regression models for high dimensional data. TL;DR: this package will allow you to analyze SNPs (or whatever the features of the data are) *jointly*, taking into account the relationships between them. This is in stark contrast to the marginal (one-feature-at-a-time) approaches that are widely used. A note for the stats nerds: unlike the popular glmnet package, `ncvreg` offers non-convex penalties in addition to the lasso.
- penalizedLMM - an in-development package for joint analysis of features in contexts where the data are structured (e.g., admixed populations, family-

based GWAS data, etc.). This package has contributions from several of Patrick Breheny's doc students (including me).

- qqman - a package for creating QQ plots and Manhattan plots from GWAS analysis results.
- CMplot - a package for creating 'fancy' QQ plots and Manhattan plots, including those that are circular. This package offers many customization options for annotation these plots.
- PLACO: this isn't a 'package' *per se*, but this method for assessing pleiotropy between traits is implemented in `R`.

## 15.2 Command-line tools

- PLINK
- LDSC

# Chapter 16

# Geographic (GIS) data analysis

In a couple of my projects, I have had the opportunity to visualize data using maps at the county level. Here is an example of the sort of maps I have made, using some simulated data.

Suppose that a PI wants to study access to care through teledentistry. The specific research question is something like, "in a population of patients treated at my dental clinic from January 2021 - May 2023, what were the patterns in the relationships between 1) the distance traveled to the clinic, 2) the mode of the intake exam (in-person or virtual), and 3) treatment completion within 6 months (yes or no). Suppose further that I (the analyst) have access to the county in which each person lives, in addition to the clinical information relevant to our research in the electronic dental records.

The first thing I would do is look online to find publicly available data with Federal Information Processing Standard (FIPS) codes for Iowa at the county level. I will use the R package `sf` to handle GIS data, which often comes in `.shp` or `GeoJSON` formats. Data in these sorts of files have what we need to draw a map of Iowa with the counties demarcated on the map.

```
library(sf) # package for GIS data
```

```
## Linking to GEOS 3.10.2, GDAL 3.4.2, PROJ 8.2.1; sf_use_s2() is TRUE
```

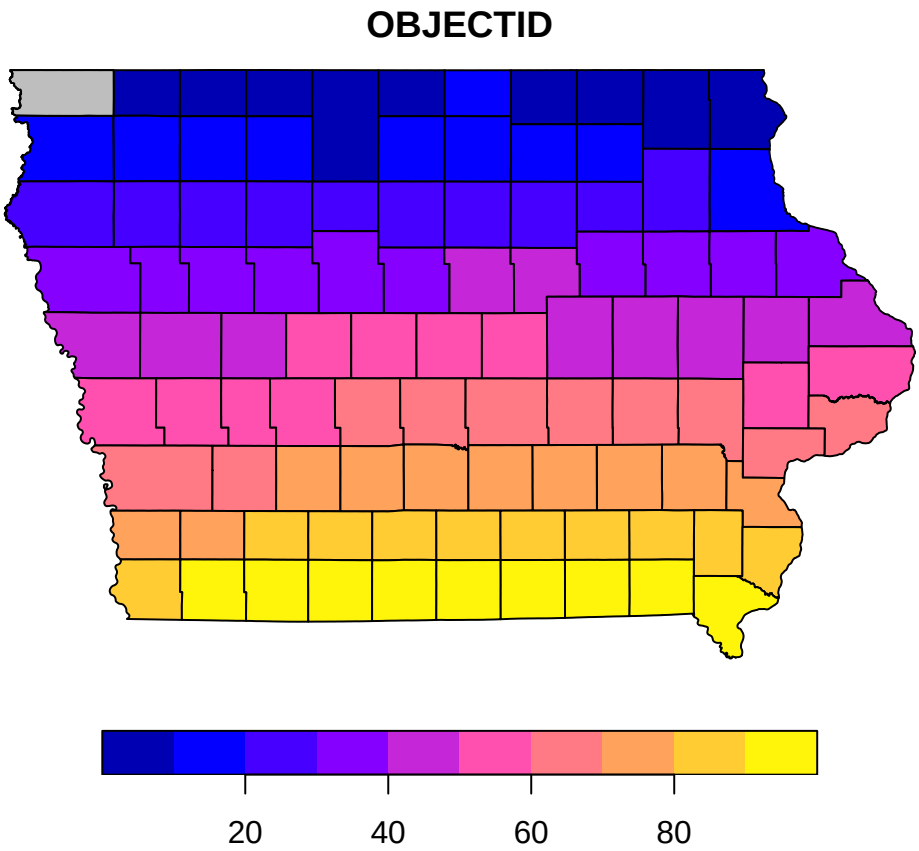
```
ia <- st_read("data/Iowa_County_Boundaries.geojson")
```

```
## Reading layer `IowaCounties' from data source
##   `/Users/tabithapeter/Desktop/train_gra/data/Iowa_County_Boundaries.geojson'
##   using driver `GeoJSON'
## Simple feature collection with 99 features and 9 fields
```

```
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -96.63944 ymin: 40.37566 xmax: -90.1401 ymax: 43.50109
## Geodetic CRS:  WGS 84
```

```
ia_geom <- st_geometry(ia)


# check - a test plot to make sure the map looks right.
par(mar = c(0,0,1,0))
plot(ia[1], reset = FALSE) # reset = FALSE: we want to add to a plot with a legend
plot(ia[1,1], col = 'grey', add = TRUE)
```




This sample map is the correct shape – here, the 99 counties are colored according to their index (1-99). We want to make a map where the colors correspond to the number of patients representing each county.

Supposing one has access to electronic dental records, the data for such a study may look like this:

Table 16.1: Simulated data from electronic dental records

| id | year | complete | county | mode |
|---|---|---|---|---|
| 1 | 2021 | 0 | Howard | virtual |
| 2 | 2021 | 1 | Clay | in-person |
| 3 | 2021 | 0 | Palo Alto | virtual |
| 4 | 2022 | 0 | Buchanan | virtual |
| 5 | 2022 | 0 | O'Brien | virtual |
| 6 | 2023 | 1 | Floyd | virtual |

```
# simulate data
library(dplyr)
key <- read.csv("data/ia_counties.csv") |>
  # narrow down to only IA (for sake of example)
  filter(State == "IA") # gives me county names



# TD = teledentistry
set.seed(52242)
td <- data.frame(
  id = 1:1000,
  year = sample(2021:2023, 1000, replace = TRUE),
  complete = sample(0:1, 1000, replace = TRUE),
  county = sample(key$NAME, 1000, replace = TRUE),
  mode = sample(c("virtual", "in-person"), 1000, replace = TRUE)
)

library(knitr)
head(td) |>
  kable(caption = "Simulated data from electronic dental records")
```

Now, I am ready to create a map that will communicate to my collaborators where our patients are driving from to receive their treatment.

```
# determine how many patients in each county
td_summarize <- td |>
    group_by(county) |>
    summarise(N = n()) %>%
  ungroup()

# add fips codes (from GeoJSON file)
td_summarize <- right_join(td_summarize, ia,
            by = c("county" = "CountyName"))
```

```
# create sf object (for drawing a map)
map <- td_summarize %>%
  st_as_sf()

# draw the map
library(ggplot2)
library(viridis)
ggplot() +
  geom_sf(data = map,
          aes(fill = N)) +
  scale_fill_viridis() +
  labs(title = "Map of IA Residents by County", fill = "Number of \npatients") +
  theme_bw()
```