P390

Britt Anderson

2025-01-01

Table of contents

Preface **1**

1 Introduction **3**

2 The role of the computer as a tool in psychological research **5**
2.1 Research Project Stages 5
2.1.1 Preliminaries 5
2.1.2 Do the research 6
2.1.3 Analyzing the Research 6
2.1.4 Diseminate the Research 6
2.1.5 Make Your Research Open 7

3 More About Open Research Practices **9**
3.1 Fair Principles 9

4 Lab Notebooks **11**

5 Writing Scientific Documents **13**
5.1 IDEs and Writing up your Research 13
5.1.1 VSCode 13
5.1.2 Some helpful links to get you started 14
5.1.3 Overleaf 14
5.1.4 RStudio 14
5.2 Scientific Publishing Tools 14
5.3 Things You Can Do With Quarto 15
5.3.1 Making a Website with Quarto and Github Pages 15
5.3.2 Journal Article Authoring 15
5.3.3 Presentations 16

6 Terminals **17**
6.0.1 An Historical Aside 18
6.0.2 Why use a terminal? 18
6.0.3 A scripting example 18
6.0.4 Resources on Terminals 20

6.0.5 Terminal Games Again 21
6.0.6 Using the Terminal to Setup a Python Venv and install Jupyterlab 21

7 Coding 23
7.0.1 Languages 23
7.0.2 IDEs 23
7.0.3 Programming Languages 24

8 Programming Experiments 37
8.1 All-in-one options 37
8.2 Python 38
8.2.1 Pygame 38
8.3 Javascript 40

9 In Lab Experiments 41
9.0.1 Online Experiments 41

10 Handling Data 43

11 What are some of the challenges that computational tools pose for reproducibility? 49

12 Knowledge Management (Zettlekasten) 51

13 Reference Management 53
13.1 Using References With Quatro and VS Code 53
13.2 Using References With Overleaf 53
13.3 Poster Presentations ? (MOVE THIS SECTION) 53

14 Finding Citations 55

15 Testing Your Javascript 57
15.1 Run your own server locally 57
15.1.1 XAMPP 57
15.1.2 In Lab Experiments 57
15.1.3 Online Experiments 57

16 Large Language Models 59
16.1 Run locally 59
16.2 How to? 59
16.2.1 Key Features of GPT4ALL 59
16.3 How-to 60
16.4 Using LLMs for Dynamic Research 60
16.5 Why Might You Want To Do This? 61
16.6 Next steps 61

17 Databases and management **63**
17.0.1 Datascience 63
17.0.2 Grant Funding 63

18 Summary **65**

Appendices 67

A Version Control **67**
A.1 Getting Started - Some old videos on git from PSYCH363/310 . 67
A.2 An overview of git and Github 67
A.3 You have choices - other version control system 68
A.4 Git is not Github 68
A.5 Things you should do 68
A.6 But first 68
A.7 Understanding Git and the Workflow in Pictures 69
A.8 Principal Terms to Learn First 69

B Just a place for some notes for installing things on Mac **71**

C Emacs **73**
C.1 init files 73
C.2 pdf-mode 73
C.3 keys and keyboards 73
C.4 orgmode related 74
C.5 emacs 74

D Quarto **75**

E Python **77**

F Virtual Environments **79**
F.1 Methods 79
F.1.1 Creating and Activating the Venv 79
F.1.2 The Terminal is All You Need 80
F.2 Visual Studio Code 80
F.2.1 Quarto Complications 81

Preface

This course is an effort to expose research minded undergraduate students in psychology to the wealth of computational and programming tools that they can use to ease their research tasks, and in some cases do things that they could not do otherwise. It is in written as a series of `qmd` files. This makes it a Quarto book. Quarto can be seen as a successor to RStudio and Rmd files. It is in another in the series of markup languages that allow you to write in plain text, but by following certain conventions, make it easy to convert those plain text files into any number of output formats such as html (for a website) or pdf (for a book or article). I want to emphasize the writing of reproducible code and papers. There are many different systems that can accomplish this goal. Quarto is just one option, but it is one of the easier ones to get started with. In addition, to Quarto, we will explore IDEs via Visual Studio Code, and also some of the basics of programming, bibliographic tools, and auxilliary tools like zettlekasten and databases.

You do not need to read this book in order. At this early draft stage some things are repeated, some are missing, and some are misordered. You may have to jump around to find what you are looking for.

One of the key lessons for learning to use your computer effectively as a research tool is learning how to do things *generally*; rather than mastering one particular variety of software; we want to be able to figure out how to use whatever is the right tool for a particular application. Your needs will change as your science evolves. The tools available will change. You don't want to be still using the software you learned in University twenty years from now. If you are you will be less able to tackle the problems that interest you. Instead of doing the key experiment, you will be figuring out how to do the experiment that is most like, but not quite, the right one, but is as well as you can do with the old software you know how to use.

Chapter 1

Introduction

This is a book being created for PSYCH390 at the the University of Waterloo.

It is very much a work in progress. You can help it get better by providing feedback. Ideally you will do this by raising an issue on the github repo. Even better is if you would fix the book yourself and make a pull request.

Chapter 2

The role of the computer as a tool in psychological research

Your computer can greatly augment your research productivity and quality. There is more to it than just learning to code. At every stage of the research process there are opportunities to facilitate an efficient and effective workflow by using your computer as an auxilliary.

Class Question

Ignore for the moment the specific content of a psychological research project. That is whether it is a study on attention or memory. Consider abstractly what the components of a psychological research project are? Then ask how can computers help you do that stage of research better?

2.1 Research Project Stages

2.1.1 Preliminaries

How do you decide what it is you are going to do? Look up and read the prior literature.

1. How do you find what to read?
2. How do you keep track of what you have read?
3. How do you organize your notes on what you have read?
4. How do you integrate and connect your notes and readings so as to discover new connections or ideas?

5. How to you make sure you can correctly cite the articles you have read when it comes to writing up your research proposal or research result. Some answers to these questions will be discussed in a later chapter.

How do you protect yourself against a change in databases or death of a hard-drive?

2.1.2 Do the research

What skills are needed for this?

1. Program an experiment
2. Language? Python; Javascript; other
    1. How do you make stuff appear on a computer screen?
    2. How do you verify the timing?

    Some thoughts on programming languages will be found in this chapter
3. How do you make sure you can find the version of the experiment if you need to share it or re-use it.
4. There is a new tool in the lab with drivers and hardware. Can you download and install the necessary software?
5. What happens to the data?
    1. How do you store it?
    2. How do you search it?
    3. What if it is big?
    4. What if it needs to be shared?

    Databases are something I hope to document further.

2.1.3 Analyzing the Research

1. How do you organize the data?
2. Working on a big data project?
    1. How do you get the data?
    2. Can you set up a database?
    3. Search it?
3. How do you do statistics?
    1. What is the tool?
    2. Could you write your own analysis?

    Some material on the R programming language, a frequently used programming language for statistical analysis, can be found in this chapter.

2.1.4 Diseminate the Research

1. What is the tool you use for writing your research?
2. What is reproducibility?
3. Does your writing tool support it?
4. How do you make sure others can re-run your analyses?

5. How do you make sure that you can easily adapt your analyses and make sure the right figures are included when you update the data? Or that the p-values in the text are also adapted?
6. How do you change the citation format when you submit to a journal that does not want APA or an edit changes which of your citations is the first (so it changes when you use et al)?

I have a lot to say on writing tools in this chapter.

2.1.5 Make Your Research Open

A selfish reason for learning these tools is that it will make what you do easier and better. But there is another more socially pertinent motivation: it enables the collective scientific enterprise to be better because tools like these facilitate reproducible and open science.

Class Question

- What is "open research"?
- Should research be open?
- How many of your labs are following these practices?

What are Open Science practices?

A recommended reading: 10 strategies for open science

Chapter 3

More About Open Research Practices

Class Question

What are the **Fair** principles for research?

Take a few minutes and see if you can determine what “Fair” stands. Then we can see if we agree. We can then use those principles to evaluate our research tools.

3.1 Fair Principles

- Fair principles
- Fair guiding principles

Chapter 4

Lab Notebooks

I will not actually be spending much time on this, but we could. It is a neglected practice in psychology to keep a lab notebook that is updated. Here are some options if you are interested in exploring this topic.

- Jupyter notebooks in Psychology
- 10 rules for lab notebooks (electronic)
- How to keep a lab notebook
- Pick an electronic lab notebook
- Lab notebook 2023 guide

Chapter 5

Writing Scientific Documents

Writing scientific documents is more than writing papers. It can also include posters, websites, books, and all of these may require images, graphs, references, formuals, and even code. We will take a look here at some of the tools that can help with this, and later we will look in more detail at some of the options.

5.1 IDEs and Writing up your Research

Class Question

What does IDE stand for and what are examples of IDEs?

5.1.1 VSCode

There are many powerful IDEs available. It will be useful for you to try more than one. However, at this moment in time the clear winner of the IDE popularity contest is Microsoft's Visual Studio Code.

Since I want you to learn to use your computer as a tool, and to continue to be able to do so when tooling changes, you need to be able to:

1. Locate code/programs on the internet
2. Download and install that program to your computer
3. Locate the help to begin to use the tool without formal instruction.

In-class Exercise

Locate, download, install, and verify you can start VS Code.

5.1.2 Some helpful links to get you started

- Basics Video
- Using VSCode with Python
- Using VSCode with R
- Text Editor Page from Quarto Site
  - What is "pip"? And why will you need to know?

Additional material about VS Code and its use as an IDE will be found in this section

5.1.3 Overleaf

VS Code is far from your only option. For technical writing and collaboration many scientists rely on Overleaf.

Nature will [let you submit your manuscript](https://www.nature.com/srep/author-instructions/submission-guidelines) from an Overleaf template.

You always have choices. You can even [write LaTeX with Quarto](https://github.com/James-Yu/LaTeX-Workshop?tab=readme-ov-file) (see also).

Classroom Discussion

What is single source publishing? How does this idea impact your preference for tools like Overleaf and Quarto?

5.1.4 RStudio

RStudio is a popular IDE for writing R code. R is a programming language particularly useful for statistical analyses. While initially RStudio was only for the R language the number of programming languages it currently supports has grown. It still is not as widely useful as VS Code, and many of RStudio's features are designed for the interactive evaluation of data, and so it has less of the cutting edge features that software developers expect in their code tools. Whether you want to use RStudio or not will depend in part on your planned activity.

5.2 Scientific Publishing Tools

To understand the plethora of contemporary tooling available to facilitate writing you should know generally what is meant by a mark-up language. This will help you evaluate whether it makes sense for you to switch from a word processiing program, like Word for Windows, to a more general tool like VS Code or emacs.

Some of the available options you should be familar with are

- LaTeX
  - LaTeX This was the standard for several decades, and used more by mathematicians than others, but today it seems limited in that it produces mostly pdfs, and many people also want their files to export to html (webpages)
- A guide to LaTex for Word Users (pdf)
- The descendants of markdown
- Quarto/Qmd
  - Quarto (we will skip Rmd and since we are using its successor qmd. Both are related to the people who wrote RStudio,`knitr`, and are now known as Posit).
- Org Mode
  - Org Mode works particularly well with Emacs and allows you to embed code from multiple languages.

5.3 Things You Can Do With Quarto

Here are some of the helper links for things we can do with Quarto:

- journal formats try the Elsevier format for this course. A lot of psych journals are Elsevier owned.
- presentation
- Talking to databases.
- Make a website/blog for your work or lab
- And more …

5.3.1 Making a Website with Quarto and Github Pages

To be included

5.3.2 Journal Article Authoring

Classroom Exercise and HW

Write an article with quarto/qmd and using any of the linked article formats. Each has instructions on their use. Your article should look similar to your overleaf submission. All the components of an article, at least one figure, and at least one reference. Instead of using an image from the web try to include either your R or Python code for the figure you liked.

I find this easier to do from a terminal.

```
#| eval: false

quarto use template quarto-journals/elsevier
```

This will result in several questions for you to answer. When you are done you have starter versions of all the pieces you will need to get going.

5.3.3 Presentations

Presentations are a standard component of professional life. Despite that importance the materials are often developed in a casual way that does not give the best impact to the material. There are many more versatile tools than contemporary Power Point. Though power point can make an effective presentation its default use does not typically achieve that goal. Quarto provides easy access to two other presentation tools that will expand one's recognition of what an effective presentation tool can do.

5.3.3.1 Reveal.JS

Reveal.js is a javascript library for the creating presentations that work as well on the web as they do in person. Quarto also supports Reveal.js.

5.3.3.2 Beamer

Beamer is a package for LaTeX that allows professional quality typesetting. It is particularly relevant for presentations that have a lot of mathematical material, though it is quite capable of supporting any general presentation as well. It is less flexible for the web since it produces PDFs. These can be shared on the web, but they require the intermediate step of downloading them rather than just playing them. Beamer also has a "Poster" variant that can be used to make high quality academic posters.

Overleaf has a beginners tutorial. Quarto also supports Beamer.

5.3.3.3 Academic Posters

Typest

Chapter 6

Terminals

Even before you start coding programs you can use simple coding constructs to efficiently command your computer to do things, script repetitive tasks, or find information hard to locate through a GUI interface. To whet your appetite I have included a short video I made for another course showing some use of the Terminal.

https://vimeo.com/453837142

Class Question

What is a terminal?

To use your terminal you need to do somethings first:

1. Find your terminal. Different operating systems refer to the terminal differently. In Windows the =CMD.exe= [fn:1] command is an approximation to a terminal as is the Power Shell. For OSX you navigate to you applications, find the folder "Utilities" and look in their for the terminal application. For Linux it will depend on which particular flavor you have installed.
2. Learn some terminology While they do not mean exactly the same thing, you will often find the following terms being used relatively interchangeably.
   - terminal
   - shell
   - command line What they have in common is the idea of a text based interface to communicating with the operating system. What this means is that instead of opening a gui (gui: *G* raphical *U* ser *I* nterface) to navigate your file tree you do this with a text based system of commands.

3. Try out some sample commands

```
ls
```

Figure 6.1: Typing this command in your terminal will list the files and directory. What would you have to do to see the hidden files? How would you get more information about this function and how to use it?

6.0.1 An Historical Aside

In the early days of computing people wrote their programs on punch cards. Some see the inspiration as the Jacquard machine. There are still programmers alive who can tell you their horror stories of tripping and falling and scattering their punch cards everywhere. Want to know if your program worked? Take it to the main frame data center, drop it off, and come back the next day to get your print out. Terminals came along as an alternative to communicating with big central processors. There was a screen and keyboard. By typing you could send input to the computer that returned the output to your screen. The terminals we have today are not true terminals, but emulators. Though few people refer to them as such. We emulate this old way of communicating to the processor because it works and is efficient.

6.0.2 Why use a terminal?

You can get more done. You can get it done more quickly. Once you learn to do one hard thing you never have to figure out how to do it again, because you can easily script it. That is why you want to learn to use the terminal

Terminals are ubiquitous. They are low in their resource usage. They permit remote logins without the need for sending graphics back and forth. In fact, the remote computer need not have a system installed for displaying windows or even a physical screen attached (called headless).

Knowing how to use a terminal will allow you to use =ssh= to connect to remote hosts. It will allow you to quickly and efficiently navigate your system, and it will make it easy for you to do things that used to take ages.

6.0.3 A scripting example

Want to convert and compress a large directory of videos as I did for this course. No need to open up each in an application and click a bunch of mouse clicks. Just write a =bash= script to invoke a command line program to do all the work for you. Go get a cup of coffee and come back when the job is done.

```
for i in *.MP4;
do name=`echo "$i" | cut -d'.' -f1`;
echo "$name";
```

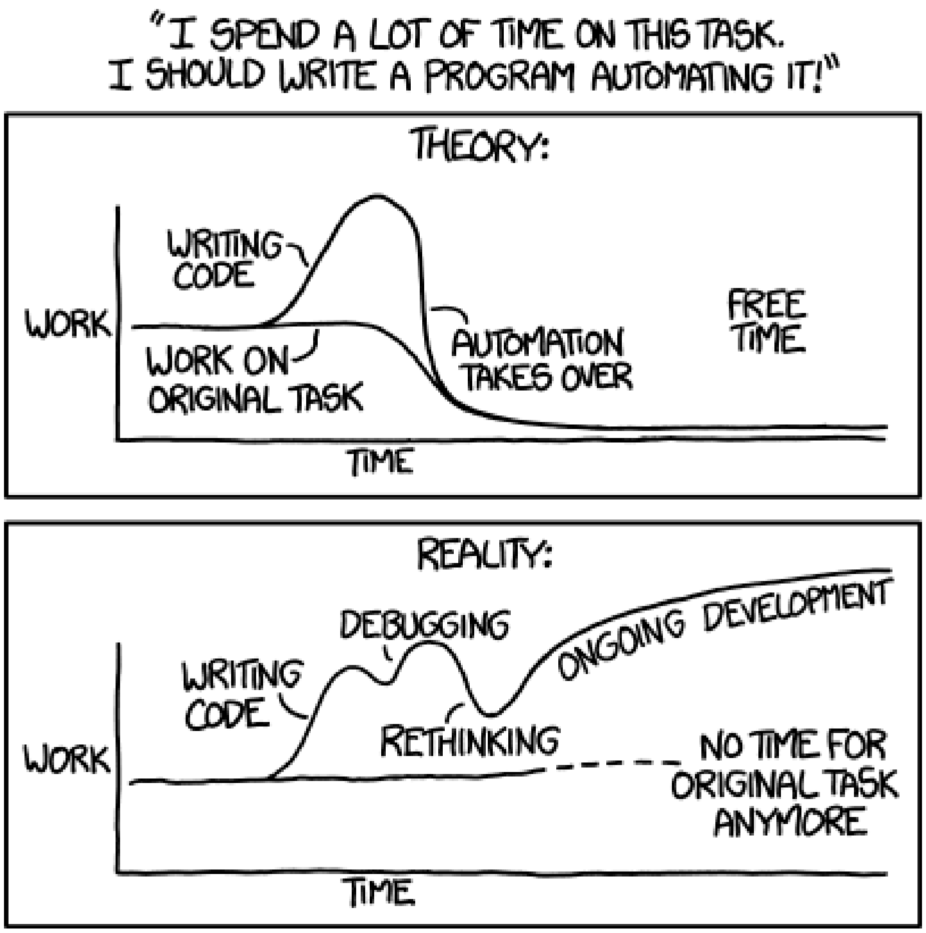


Figure 6.2: How would you pronounce XKCD?

```
  ffmpeg -i "$i" -c:v libx264 -b:v 1.5M -c:a aac -b:a 128k "${name}S.mp4";
  done
```

Almost all of this I copied off the Internet where it appeared as an answer to a question from someone else wanting to do essentially the same thing I did. It took me a while to tweak it to my particular use case, but when that was done my problem was solved, /forever./ Every new batch of videos I just put in their own directory and run this script from the command line. Note that this script uses a for loop, this is a very common programming construct.

```
for (i in seq(1,5)) { print(i)}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

This code block demonstrates a simple for loop for the R programming language.

6.0.4 Resources on Terminals

There are a great many resources on how to use the terminal effectively, but don't go out and read them all. One of the skills to learn in learning to use the computer is to develop your own set of links and resources you can go to when need arises. Don't try to learn everything at once. You will get overwhelmed and discouraged. Instead you learn what you need when the need arises. And if you need to know something more than once, then you spend the time to dive deeper. There are a great many things about using the terminal that I do not know, but I know the ones I use often, and I know where to find more when I need to know more. You should do the same. Here are a few online resources to get you started.

1. The command line
2. A Short Series of Terminal Lessons from the UbuntuWiki
3. Some Scripting Basics
4. Another Scripting Introduction

Most of what you want to do at the command line, at least in the beginning, you can do with typing directly into the terminal. But at some point you will want to write a file, a script, that has all the commands typed into it. They you can run that script from the terminal. This blog post has some basic background for how to get started.

Note the use of the term “BASH”. This stands for the Bourne Again Shell. Your terminal can use different shells (and if you are using a mac you are probably using the zsh shell). So while you can use the terms interchangeably most of the time, they don't mean exactly the same thing.

6.0.5 Terminal Games Again

These will only work on linux and OS X. If you are on windows you could enable the linux subsystem for linux or you could learn the powershell equivalents. But I suggest that you use the terminals that are built into VS Code for practice. You will usually see these at the bottom of your screen. If not, there is a menu you can use to open one.

1. ls -la /home/

- What does all this output mean?
- What changes when you leave out the -la?
- What does the hyphen do?

2. Can you find the location of your desktop folder in your terminal?
3. Can you change to that directory? cd
4. Find out where you are? pwd
5. Can you find out who the computer thinks you are, your user name? whoami
6. Find out how much free space you have on your computer disk. df -h
7. How do you get help for most of these commands? Usually command –help or (-h)
8. How do you find the manual? man ls
9. Navigating
   1. Paths: absolute and relative.
   2. What do those "dots" mean?
   3. What do those slashes mean?
   4. Tab is your friend.
   5. Try the up arrow too.
10. File ownership
    1. Make a text file from the command line. touch /home/yourname/Documents/testText.txt
    2. Who owns it?
11. Make a directory mkdir /home/britt/Documents/myFirstDir/ Spaces are the enemy. Never use them, but if you have to, escape () them.

6.0.6 Using the Terminal to Setup a Python Venv and install Jupyterlab

The terminal can also be a very convenient and direct to install programs. You are not clicking buttons in a gui and hoping things work. You have to know exactly what program you want and where you want to put it. As an example we see how we might use the terminal to establish a python virutal environment and use that venv to implement a jupyterlab setup. You will find more on python virtual environments in the appendix.

Our goal is to be able to test and develop the idea of a jupyter lab notebook. But we will need several pieces of code installed. Assuming we have python3 installed we can use the terminal to set up a secure enviroment for creating a virtual environment. Think of a sandbox that let's you play without getting the things you installed for experimentation conflict with your system's tools.

1. Open a terminal
2. Make a directory (`mkdir`)
3. "cd" into your new directory and create the virtual environment

   ```
   python3 -m venv .
   ```

4. Next, "activate" the environment with

   ```
   source ./bin/activate
   ```

5. Now you can install the needed packages with `pip`.

   ```
   pip install numpy matplotlib jupyterlab
   ```

Chapter 7

Coding

For a lot of the above we need to know how to code. In order to write code we need minimally to answer two questions. What language? What tooling?

7.0.1 Languages

What should I consider when selecting a programming language? Will it do what I need it to do now and tomorrow.

- Is SPSS a good language for statistics?
- Is R a good language for statistics?
- Is Python a good language for statistics?
- Is R a good language for coding a web app?
- Is R a good language for coding an in-lab visual experiment?
- Should you use Julia? Common Lisp? Haskell? Lean? OCaml? Rust? Go?

How do you *future proof*? - If languages go in and out of fashion what is it you should really be learning about programming? What are good coding practices?

7.0.2 IDEs

- Who are you writing code for? Human or Machine?
- What is an IDE? What makes for a good IDE?

7.0.2.1 Using an IDE

For this course we will default to VSCode, because it is currently very popular and becoming somewhat of a standard. Everything said above about not getting to attached to the flavor of the month applies to IDEs. Especially since VSCode is a tool tied to Microsoft. However, there is an opensource build of VSCode that you can use instead. You can also use any other tool you want to as long as you can figure out how to make it do the things I will ask you to do. I, for one, live in Emacs.

7.0.2.2 VSCode

https://www.youtube.com/embed/B-s71n0dHUk?si=y3fy80M0mGxGLwr5

- Basics video
- Using VSCode with R
- Using VSCode with Python

Exercise Install VSCode

7.0.2.3 Jupyter Notebooks

What are jupyter notebooks? Are jupyter notebooks ide's? What are their purpose? What languages to they support? MORE TO ADD

7.0.2.4 Beginning with Jupyter and Quarto/VS

Good instructions are on the Quarto website. Note those terminal commands.

You could at this point use the brower interface for editing your ipynb file with `{sh} python3 -m jupyter lab` or you could open a file in VS Code.

Make sure you have the jupyter extension for vscode. Then you will want to create or open a new jupyter file and select the appropriate kernel (initially use python). If you are in the correct directory and had previously created a virtual environment then all should be good to go. And you can *in the terminal* invoke the quarto command to see your document with `quarto preview my-demo-notebook.ipynb`.

7.0.3 Programming Languages

7.0.3.1 Basic Constructs

- Variables Variables are things (lists, values, names, and such) that you want to give a name to. You might do this because you think you are going to want to do something with them later and it might be more convenient to have a short, memorable name for referring to them.
- Functions These have inputs and output. The inputs may be called parameters. Sometimes the outputs are called return values. What the function does is describe the processing of the inputs to the outputs. Think of a factory. It takes in raw materials (inputs) and produces an output. You are describing a factory when you write a function.
- Interpretation and compilation.

7.0.3.2 R

7.0.3.2.1 Installation

- OSX
- Windows
- Linux

Download the CORRECT R

Follow the appropriate instructions for your operating system.

If that went well, you need to make sure you restart VS Code and then click on the little set of squares at the left to install the R extension for VS Code. If you are using VS Code for your IDE as recommended.

If that goes well check out the book on using R that is itself written in Quarto, by the programmer who authored the tidyverse (see below). And make sure you can create a file in VSCode as a quarto document, cut and paste in some R code, and see the whole thing compile to a web page.

7.0.3.2.2 Working Methods - Generally

Have your code open in a window and a terminal open running the R interpreter for testing and checking.

Test your system by, 1. `cd` to the directory where you want to work. 2. open a .R file in your editor. 3. open a terminal that points to the same directory you can move with `cd` and check your location with `pwd`. 4. Verify that you can open R there by typing `r` and then hit enter. If you need to quit you can type `quit()`.

7.0.3.2.3 Packages/Libraries

Many programming languages are very basic in what they offer. In order to do the cool stuff you will need additional collections of code, some of it may even be written in a different programming language, often called packages or libraries that you import, source, or load. `ggplot2` was an example of that for R and `matplotlib` for python.

```r
install.packages("tidyverse")
library{tidyverse}
```

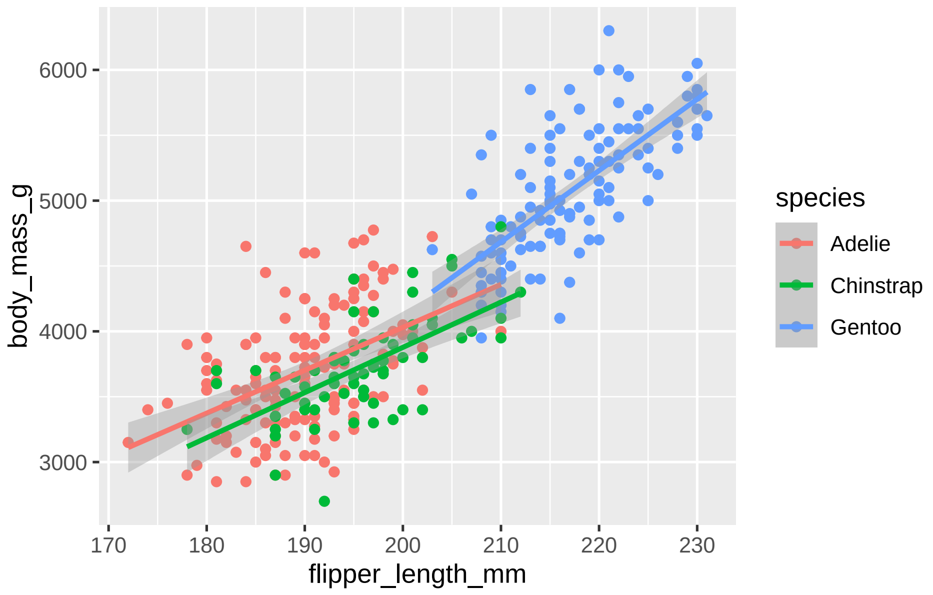
Figure 7.1: This is an example of installing a package in R and then using it. Note the inconsistent use of quotation marks.

The tidyverse is a very popular collection of R code that itself will depend on many additional pieces of R code and other packages, some R and some not. I personally often try to just use “base plot”, but most people seem to prefer the “tidyverse”.

Here is an example of embedding R code in a qmd document. This is very useful when you get to the stage of data analysis or writing your research report. But when you are first exploring the data it is often better to work directly in an R environment and record your code in an .R file to make it easy to replay and edit your work as you go.

```
if(!require(tidyverse)){
    install.packages("tidyverse")
}
if(!require(palmerpenguins)){
    install.packages("palmerpenguins")
}
library(tidyverse)
library(palmerpenguins)

ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)
) +
  geom_point() +
  geom_smooth(method = "lm")
```



7.0.3.2.4 Types

Many programming languages have a notion of a datatype. `1` can be a number, but "1" can also be a character that you are typing. It doesn't make sense to add one, the number, to one, the character, but it might make sense to `1 +`

1 for either. You would expect 2 for numbers and maybe "11" for characters. R in particular has many complicated data types that represent the output of functions and statistical analyses. If things are getting confusing you might want to check on the type of your data.

```
a = 1
typeof(a)
[1] "double"
```

Figure 7.2: Checking the type of a number in R.

Other data structures that you might run across are: - Lists, - Tuples, - Data.Frames

```
tpl = c(1,2)
lst = list("firstName" = 'Britt', "lastName" = 'Anderson')
df = data.frame('fn' = c("bob","jane","griffin"),"gndr" = c('m','f','o'))
df
       fn gndr
1     bob    m
2    jane    f
3 griffin    o
```

Figure 7.3: Each of three datatypes in R are demonstrated. Think of data.frames as a supercharged spreadsheet with columnn names.

7.0.3.2.5 Data.Frames

`Data.Frames` are particularly central to the R analysis model and are a killer feature. They are also moving into the python analysis space. Let's look at the cars data frame that comes as part of the base R installation. You can verify it is installed like this.

```
is.data.frame(cars)
```

```
[1] TRUE
```

```
head(cars)
```

```
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

7.0.3.2.6 Selecting Data

A bit of code that tests for whether something is true or false can be called a *predicates*. These are important when having your code do something conditionally. If this is true then do this otherwise do that. This common construct of "if-then-else" exists in most programming languages.

```r
length(cars$dist[cars$speed > 10 & cars$speed < 20])
[1] 29
```

Figure 7.4: How many cars are there that can go faster than 10, but not more than 20?

How would you do that in Excel?

What is going on here? 1. `cars` is the name of the data frame. 2. We access a *column* of the data frame with the dollar sign notation `cars$dist`. You can see the names of the columns in a data frame with `names(cars)`. 3. We use the square brackets to *index* the column of data we are interested in. Here we do not use specific numbers, but rather we use a *boolean* to compare the values in a column and we only include the *TRUE* values. Here we ask for all the indices where the speed is greater than 10, but less than 20, and we use those indices to get the values for the `dist` column.

Practice

1. Sort (or $_{\text{order}}$) cars by the $_{\text{dist}}$ variable.
2. Find the mean and standard deviation of the speed of the cars.
3. Are there other datasets?

   ~~~
   ```{r}
   library(help="datasets")

   ```
   ~~~

4. Open any of the datasets that catches your eye.
5. What are the column names?
6. How many rows?
7. What is the *comment* designator for R?
8. What is the ending extension of an R script?

7.0.3.2.7 Loops

Tip

Loops are very different between python and R. You use them a lot more in python than R. R often allows you to "map" over data.

`For` loops are the most common looping construct.

```
ml <- seq(1:10)
for  (m in ml) {
  print(m)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Another kind of loop is the `while` loop.

Test your understanding with more practice.

1. Edit the above so that it prints the individual number each time it goes through the loop, and not the whole list.

Another project 1. Create a list of at least 8 individual characters. 2. Make sure they are **not** in alphabetical order 3. Print the letters one at a time. 4. Print the letters sorted alphabetically one at a time, but *do not* overwrite your original list. 5. Print the letters from both lists with a `format` command that says which position the letter is in.

```
myName = "brittAnderson"
myList = unlist(strsplit(myName,""))

for (l in myList){
  print(l)
  }

for (l in myList[order(myList)]){
  print(l)
  }

i = 1
for (n in order(myList)){
  t  <- sprintf("The %.0fth letter of myList is: %s, but is %s in the sorted list.",i,myList[i],m
  print(t)
  i = i+1
  }
```

7.0.3.2.8 Examples of Functions

```
myadd  <- function(x,y) {       (1)
  return(x+y)                   (2)
  }
```

(1) The arrow is used to assign something to a name. Other languages usually use the equals sign. The keyword `function` tells R you are creating a function, and the parentheses following establishes the number of inputs and the nicknames you will use in your function.

(2) `return` specifies what we want to get back from the function.

Now we can use the function we defined with particular inputs.

```
myadd(2,3)
```

```
[1] 5
```

A few years ago I made a video talking about some of these features, but I used a different IDE from VS Code.

Hc : Walking Through the R Topic (i2c4p) from Britt Anderson on Vimeo.

7.0.3.3 Python

Note

If the following doesn't go smoothly for you check out the additional suggestions in the python-venv appendix.

7.0.3.3.1 Installation

Installation Instructions

Install the `mathplotlib` and `numpy` packages.

Class Discussion

What is a "package" when talking about programming languages? What is a "library"? What is an "executable"?

Warning

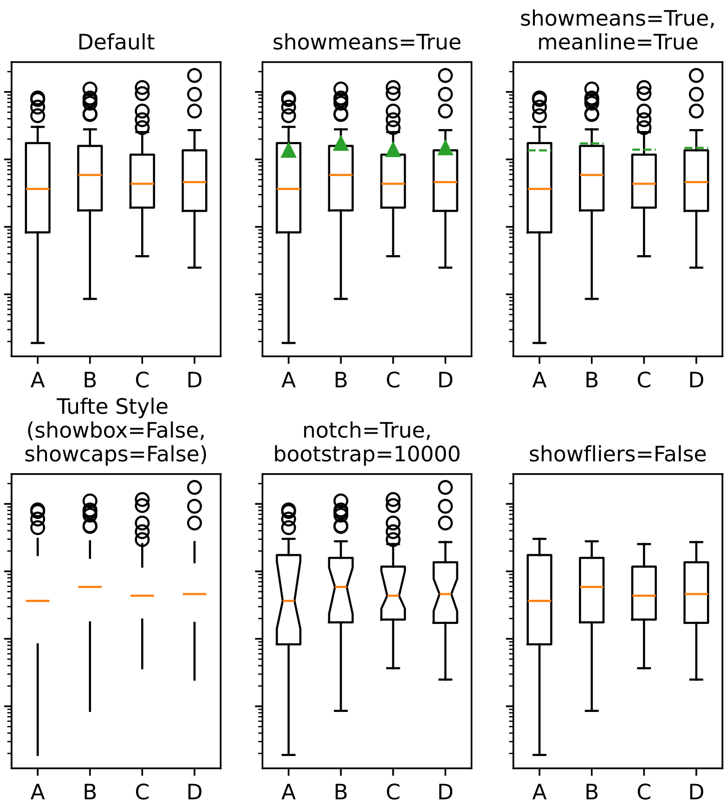
Ask me about virtual environments!

setting up a venv in vscode and python

7.0.3.3.2 Testing

Warning

You may need to install the "reticulate" package for R if you want to run both python and R code in the same document as I am trying to do here.



Warning

Python cares about whitespace. It is strongly recommended that you only use spaces and not tabs when writing python code.

```
my_name = "britt"                                  ①
my_name_as_list = list(my_name)                    ②
for i, j in enumerate(my_name_as_list):            ③
  print((i, j))
```

① Assignment in python does not use the arrow like R does.
② Strings and lists are different *types* of data. I have to coerce my string to a list.
③ Python has conventions to make it easy to define multiple looping variables at once.

```
(0, 'b')
(1, 'r')
(2, 'i')
(3, 't')
(4, 't')
```

Figure 7.5: What a for loop might look like in python.

7.0.3.3.3 Some programming vocabulary

If you want to watch a video that reviews some of the programming vocabular and examples typed out (using Emacs and not VS Code) the following video may be of use.

Gd : Common programming terms (i2c4p) from Britt Anderson on Vimeo.

An overview of a few basic and fundamental programming terms.

! Important

Python is a dynamically typed language. Python will try to guess the type of your data or change it based on context.

7.0.3.3.4 Assignment and Equality Are Different in Programming

Assignment means you are assigning a name to something. Equality is that you are testing if they are the same. One equals sign does not equal two equals sign. That is `= != ==`. Exclamation points are often used to *negate* something (turn true to false and vice versa).

```
a = 2
print(a == 3)
```

```
False
```

For loop details in python

Python refers to things called “iterables.” To iterate is another way of saying something you can keep doing the same thing over and over to. Imagine a bowl of ice cream. It is “eatable”. You take one spoon, and keep taking spoonfuls until the bowl is empty.

Indexing is how you get the location of an element in a list. Think of the row or column number in a spread sheet program, but indexes can be more powerful, and can be nested easily. In many programming languages, but sadly not all, **indexes start at 0**. Different programming languages will have slightly different syntax.

```
name_dict = {'firstName' : 'Britt', 'lastName' : 'Anderson'}
my_list = list(range(1, 10))                                    (1)
print(name_dict['firstName'])
print(my_list)
print(my_list[0])
print(my_list[-1])
print(my_list[0:4])
```

(1) Notice that here we use `range` where we used `seq` in R. There is usually a way to do something similar in both languages, but they may not have the same name.

```
Britt
[1, 2, 3, 4, 5, 6, 7, 8, 9]
1
9
[1, 2, 3, 4]
```

Figure 7.6: The use of the `-1`as an index is a python trick for getting the last element in a list. Think of the list as a circle. If you count backwards from a list you will get to the beginning eventually (index 0) if you went back one more step (-1) you would circle back to the end of the list. Test what happens when you try -2. Does it keep going? A lot of learning how to program is just doing stuff to see what happens.

A `for` loop in python

```
for ml in my_list:
  print(ml)
```

```
1
2
3
4
5
6
7
8
9
```

```
for i, ml in enumerate(my_list):
  print("The {0}th element was {1}".format(i, ml))
```

```
The 0th element was 1
The 1th element was 2
The 2th element was 3
The 3th element was 4
The 4th element was 5
The 5th element was 6
The 6th element was 7
The 7th element was 8
The 8th element was 9
```

Conditionals in python

```
if 2 == 3:
  print("Wha.....?\n\n")
elif 3 == 2:
  print("Now that is odd")
else:
  print("2 does not equal 3.")
```

```
2 does not equal 3.
```

Note that we use colons and white space to organize code in python, not the {} of R.

! Important

Python uses a different syntax to define functions from R.

```
def my_add(x, y):
  return(x + y)
```

7.0.3.3.5 Popular and Useful Python Libraries

1. Numpy
2. Scipy
3. Matplotlib
4. Pillow
5. Sympy

7.0.3.4 Javascript

7.0.3.4.1 Installation

Since javascript is already included in most browsers you likely have a basic version of javascript installed, and you can get started with almost no additional tools. As a quick illustration open up your favorite browser and look in the settings or options for something like "more Tools". Somewhere in that menu should be a web tools or web developer tools entry. Click it, and you should see something that looks like your VS Code screen with terminal like objects at the bottom. One of those is the *console*. Select it. Then you can create a function and use it with something like:

```javascript
function add(a,b=10){return a + b}
```

A simple javascript function.

7.0.3.4.2 A Better Way to Go

Even though there is a convenience factor in using your browser's console it is a very limited way to code. It is fine for testing out a line or two of javascript, but to actually code something you will want an actual development environment, and we will use VS Code.

To code javascript outside the browser you need a proper environment such as Node.js. There are many ways to install this, but if you are new to javascript using the official installer is probably best. To check if the installation went well you can run `node -v` inside a terminal. You should see a version number. If you are sure you downloaded things correctly you may have to set your path so your terminal can find the installation.

To test your installation you can create a test folder, e.g. call it "hello" and open that folder in VS Code. Then create a new file "app.js". VS Code should give you a lot of help out of the box as you enter:

```javascript
var msg = 'Hello World';
console.log(msg);
```

After saving you should be able to move down into your terminal and run `node app.js` to then see "Hello World" printed.[1]

[1]This example comes from this node tutorial.

Chapter 8

Programming Experiments

One of the most common uses of computers in psychology is for the programming of experiments. Classic experiments often involve a combination of visual stimuli that the participant has to respond to in someway. Typical responses are often button presses or mouse clicks, though many other varieties of stimuli and reports are used. While many programming languages can be used for this task we will begin with one of the most popular, Python. And if time permits we can explore some examples with Javascript as the world of experimental psychology is increasingly complementing the usual lab based testing paradigm with online studies.

8.1 All-in-one options

Excellent tools exist that provide turn-key solutions. A very popular one is Psychopy. This package is available for the popular operating systems and gives a gui like builder that allows you to program visually. It offers the opportunity to output your experiment into a web friendly format that can integrate with an online hosting service for performing experiments. However, my experience is that while such tools make it easy to get started they don't help people develop the general skills that will transfer into programming that distinctive experiment that no one else has done. And the knowledge you gain is often interface specific. It is not general, and so your learning cannot transfer into writing R code or porting your experiment to MATLAB if that becomes the tool a collaborator wants to use. Even if this is the sort of tool you eventually settle on for the very reasonable reasons that your experimental needs are not complex and the quality of the implementation allows you to get work done more easily and on board new investigators more quickly, it will still make you better at planning and handling problems if you have a knowledge of what is going on under the hood.

8.2 Python

The following examples focus on the sort of experiment where you want something to appear on the screen, and you want feedback from the participant. Current computer graphics don't work like they did twenty years ago when we all had heavy CRTs on our desks, but one can still deploy some of the intuitions of that era.

The monitor is a screen and it is displaying a picture. In the background what is to be shown next is in preparation and at some point what is on the display now will be replaced by the next collection in the queue. Even though we no longer literally raster the display from top to bottom and side to side you can still usefully think in terms of *frames*. Thus, almost all your experiments will need the mechanics of a window the participant views and in which you can write things you want them to see.

8.2.1 Pygame

For practice I will use the pygame python library. There are a large number of help and tutorials for this well established library that has been in use for decades. It is well vetted and documented, and relies on a simple direct library for video manipulations.

8.2.1.1 Some Links for Getting Started with Pygame

1. The pygame getting started wiki. Most of my examples will draw from this site.
2. Real python pygame tutorial
3. Game Development Academy

8.2.1.2 Installation

A virtual environment is recommended, but not necessary. If you are using a venv activate it first.

Then

```
pip install pygame                                                  ①

python -m pygame.examples.aliens                                    ②
```

① This line installs the library.
② This launches a test game. If you see the game, you have installed pygame. I find this game plays a little fast on modern hardware.

One of the better ways to get started is to gradually hand in functionality one step at a time until you have the basics of what you need on a trial and then to refine that to loop through the multiple trials you will probably need.

```
import sys,pygame
pygame.init()

mywindow = pygame.display.set_mode([800,800])
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    mywindow.fill((128,15,15))
    pygame.display.flip()

pygame.quit()
sys.exit()
```

Figure 8.1: This python program will, assuming you have installed pygame, launch a window and close it when you click the “x”. Can you change the window size and background color?

Here is an example of a bit of code that shows some elementary use of the mouse, response time, shape and text functionality.

```
from itertools import compress
import sys,pygame
pygame.init()


my_font = pygame.font.Font(None,72)
test_text = my_font.render("Hello P390", 1, (200,10,200))
test_text_pos = test_text.get_rect(centerx = 400)

#Helper functions
def which_buttons (bs):
    return(list(compress(range(len(bs)),bs)))

#button colors
clrs = [(0,0,255),(0,225,0),(225,0,0)]
actv_clr = 0
rslts = {'trial_num': -1,'color' : None ,'time' : None}

mywindow = pygame.display.set_mode([800,800])
running = True
myrt_clock = pygame.time.Clock()
t1 = myrt_clock.tick(60)
anew = pygame.mouse.get_pressed()
trl = -1,
clr = clrs[0]
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    mywindow.fill((128,15,15))
    pygame.draw.circle(mywindow, clrs[actv_clr], (400,400), 75)
    mywindow.blit(test_text, test_text_pos)
    pygame.display.flip()
    aold = pygame.mouse.get_pressed()
    if(any(aold) and (aold != anew)):
        t2 = myrt_clock.tick(60)
        rslts['time'] = t2 - t1
        rslts['trial_num'] = rslts['trial_num'] + 1
        actv_clr = which_buttons(aold)[0]
        rslts['color'] = actv_clr
        anew = aold
        print(rslts)

pygame.quit()
sys.exit()
```

8.2.1.3 Next Steps

You will want to save your data. As a first step see if you can open a file, and the if you can use the dictionary created to add lines to a csv file.

8.3 Javascript

WIP

Chapter 9

In Lab Experiments

- Making stuff appear on monitors.
- What is OpenGL?
- Use pygame to make some simple visual experiments.
- Have a beauty contest the following week to see what people have been able to make?
- Make sure that people are using venv

9.0.1 Online Experiments

Running a server for testing and more? XAMPP Running a lab now seems to require some familiarity with servers. And people who want to write their experiments in javascript often want to try things out first so it seems something like XAMPP might be a good resource. *Exercise* to download and get the XAMPP server running? Need to expand this. An exercise with JavaScript? This site has a simple bit of code for throwing an image on the screen. Then use the XAMPP server to test it? Require changing the image? Animate a button to toggle or get a random image?

Chapter 10

Handling Data

Before we can analyze data, we have to have some data. Broadly speaking our data may come from experiments we run ourselves, or it may come from data that is shared with us. In the first case we have some freedom to choose the format. In the later, we simply have to take what we are given. Therefore, we need to consider both how to save and convert data that we collect from an experiment. And we also need to be broadly familiar with common formats of data storage that others may use.

10.0.0.1 Data Formats

- Plain Text
- CSV
- TSV
- JSON
- Pickle

- Dat (R)

Some advantages and disadvantages of each.

Plain text is easy to do and human readable. It usually will require a unique parser, and forces all items to be text. Thus, things like numbers lose their specificity, and things like lists can become painful to retrieve.

CSV and TSV view all data as rows and a unique character (commas or tabs) separates the cells of the rows. Another character, typically a new line (often seen written as `\n`) is what distinguishes one row from the next. These file types are, largely, human readable if the number of columns and rows are not too large. They are not as space efficient as other formats, but that may be a reasonable tradeoff when the size of the data files is small.

JSON stands for Java Script Object Notation. Although, as the name implies, it had its origin in javascript it is largely language agnostic and is a common format for the storage of data of all types. Its being used as a standard means it has wide support from many programming languages and that means there are good tools for exporting and storing for most programming languages. It can be quite flexible in the type of data stored. It is not friendly for writing by hand, but the use of tools makes it fairly easy to take experimentally collected data created in, say, a python dictionary and convert it to a json format.

```
import random as r
import json as j
a = {'name': 'John Doe', 'age': 24, 'rt': r.random()}
js = j.dumps(a)
js
'{"name": "John Doe", "age": 24, "rt": 0.35413051305980203}'
```

Figure 10.1: The `dumps` command will output the json version as a string.

Note there is a very similar python function without the terminal *s*, `json.dump`, that will save the json data to a file. Some other code is needed to create the file pointer. But once that is done, and the json data saved, you can read it back with `json.load()`

```
import json
a = {'name': 'John Doe', 'age': 24}
js = json.dumps(a)

# Open new json file if not exist it will create
fp = open('test.json', 'a')

# write to json file
fp.write(js)

# close the connection
fp.close()
```

Figure 10.2: A simple example taken from [StackOverflow] shows a working example.

A JSON tutorial can be found here.

Pickling is a data serialization library specific to the python ecosystem. That makes it useful if you are coding your experiment in python, but not in any other language. However, it can be very efficient on space, and there is good support for it in the python language so many experimental coding libraries built around python use pickle for data storage.

This code from a GeeksforGeeks article provides a nice minimal working example.

```
import pickle

def storeData():
    # initializing data to be stored in db
    Omkar = {'key' : 'Omkar', 'name' : 'Omkar Pathak',
    'age' : 21, 'pay' : 40000}
    Jagdish = {'key' : 'Jagdish', 'name' : 'Jagdish Pathak',
    'age' : 50, 'pay' : 50000}

    # database
    db = {}
    db['Omkar'] = Omkar
    db['Jagdish'] = Jagdish

    # Its important to use binary mode
    dbfile = open('examplePickle', 'ab')

    # source, destination
    pickle.dump(db, dbfile)
    dbfile.close()

def loadData():
    # for reading also binary mode is important
    dbfile = open('examplePickle', 'rb')
    db = pickle.load(dbfile)
    for keys in db:
        print(keys, '=>', db[keys])
    dbfile.close()

if __name__ == '__main__':
    storeData()
    loadData()
```

When working in the R environment the `save` and `load` function provide convenient ways to save your data and variables in their current state so that you can make available in your next interactive session what was available in your current one. However, this may not be the most interoperable way to work with your data if you need to share with collaborators who may not use R or if you yourself are open to the possibility of changing your analysis software. R offers two main functions for reading and writing data called `read` and `write`. Each of these has several specialization that target particular file types (e.g. `read.csv`) and also allow a number of customizations (e.g. whether to ignore a number of lines at the top of a file if there is a plain text header). Usually one of these variants will allow you to read your data into R or write data to a file that can be read by others.

10.0.0.2 Considering Pandas and Interoperability with R

Pandas is a major python library and effort to provide cutting edge tools for statistics and analysis to match what is on offer elsewhere. In my experience they are not the first place that statisticians develop their cutting edge tools, R remains most popular, but for all else the choice between the two, R and Python, is really a matter of preference and comfort. Pandas is a very powerful tool and if you are more likely to use python in the future you will want to explore Pandas fully.

Pandas can usually be installed with `pip install pandas`. Because of the size and complexity of python installations a virtual environment is recommended.

Even if you are not thinking of doing your data analysis in python if you are writing your experiments in python you might want to take advantage of some of the pandas tooling to make saving your data convenient. But before we show a pandas method let's see a lower tech, more old school method.

It is possible to manually write a csv file in python using a print command where you explicitly coerce everything to a string and intercalate commas along the way. For example,

```python
a = 1
b = "britt"
c = 2.3
mycolumnheadings = "a," + "b," + "c" + "\n"
mystring = str(a) + "," + b + "," + str(c) + "\n"
print(mycolumnheadings + mystring)
```

```
a,b,c
1,britt,2.3
```

shows how to create a string from one's data. It is complicated and error prone. It may work for a start or for a very small project, but it does not scale well. In this example I am just printing for display, but if this was being used to save experimental data we would want to use a file handler.

10.0.0.2.1 Opening and Closing a File in Python

```python
a = 1
b = "britt"
c = 2.3
mycolumnheadings = "a," + "b," + "c" + "\n"
mystring = str(a) + "," + b + "," + str(c) + "\n"
with open('/Users/britt/gitRepos/p390/p390/myfile.csv','w+') as f:
    f.write(mycolumnheadings)
    f.write(mystring)
```

Now you can see if you can read that csv into R.

```
mydata = as.data.frame(read.csv("/Users/britt/gitRepos/p390/p390/myfile.csv"))
mydata
```

```
  a     b   c
1 1 britt 2.3
```

10.0.0.2.2 A Better Way: Dictionaries

Python dictionaries (which have their counterparts in most programming languages though they may be called something different) are a convenient way to store data where you can give it a meaningful name. For example, in an experiment with numerous trials and conditions you could have a *key* in your dictionary for "trial no." and another for "condition". The *values* for these would then be updated as your experiment unfolded.

A first pass to using dictionaries is to blend this with a python library for CSV files. Then you can create the column headings from your keys and then each trial that you loop through you can use the dictionary to write a new row to your csv file. Some of the functions you might want to use are:

- `csv.DictWriter`
- `writeheader`
- `writerows`

If you set up and define your dictionary at the start of the file you only need to update the changing values as they occur.

Another approach that will start you getting some familiarity with python pandas library is to make to make a distinct dictionary for each of the trials in your experiment. Then you can concatenate these dictionaries to a list and use that list to create a pandas dataframe. That could then be used in analysis python code or for importing into R. Here is a short example to illustrate the principle.

```
import random as r
import pandas as pd

mylistofkeys=["name","v1","v2","v3","v4"]

mydictlist = []
for i in range(5):
    mydictlist.append({key:value for key,value in zip(["name","v1","v2","v3","v4"],["britt"] + [r

df = pd.DataFrame(mydictlist)

df.loc[:,'v3'].mean()
```

```
np.float64(0.6827185107425899)
```

```
myfilename = "mypandastest.csv"
```

```
df.to_csv(myfilename)
```

You can save pandas dataframes as a pickle file, but then you will not be able to easily read it into R. CSV remains the easiest and most direct way to do this. It is probably all you need for most use cases. Though there are other options.

```
rdf = read.csv("mypandastest.csv")
mean(rdf$v3)
```

```
[1] 0.6827185
```

10.0.0.3 Some Analysis Tools

- Pandas TBD
- Some more R TBD

Chapter 11

What are some of the challenges that computational tools pose for reproducibility?

Classroom Exercise

Read and discuss: A toolkit for transparency ....

Chapter 12

Knowledge Management (Zettlekasten)

1. Org-roam
2. Dendron - works with VS Code
3. Obsidian

Chapter 13

Reference Management

- What is Crossref?
- What is a doi?
- What is an orcid?
- What is zotero?
- What is csl?

Classroom Exercise

Locate and download the csl file for the current APA style and also for one at least one other non-apa style.

13.1 Using References With Quatro and VS Code

13.2 Using References With Overleaf

VS Code is not your only option. Overleaf also use ".bib" files to hold references, though some of the other syntax is different.

Here are some links to get you started with references in overleaf. 1. Linking Overleaf and Zotero 2. [Connecting Zotero and Quarto](https://quarto.org/docs/visual-editor/technical.html#citations-from-zotero)

13.3 Poster Presentations ? (MOVE THIS SECTION)

[Scientific Posters with Quarto](https://github.com/quarto-ext/typsttemplates/tree/main/poster)

LaTeX and Knitr in Overleaf][knitr]] with overleaf. A scientific poster with Overleaf?

Chapter 14

Finding Citations

- Semantic Scholar
- OpenAlex
- Google Scholar
- Consensus
- Undermind

Chapter 15

Testing Your Javascript

15.1 Run your own server locally

Running a lab now seems to require some familiarity with servers. And people who want to write their experiments in javascript often want to try things out first so it seems something like XAMMP is a good tool to know about.

15.1.1 XAMPP

XAMMP

15.1.1.1 Downloading

Download link

15.1.2 In Lab Experiments

- Making stuff appear on monitors.
- What is OpenGL?
- Use pygame to make some simple visual experiments.
- Have a beauty contest the following week to see what people have been able to make?
- Make sure that people are using venv

15.1.3 Online Experiments

An exercise with JavaScript? This site has a simple bit of code for throwing an image on the screen.

Can you use the XAMPP server to test it? Require changing the image? Animate a button to toggle or get a random image? WIP HERE

Chapter 16

Large Language Models

16.1 Run locally

Why? Own your own data.

Security and Privacy.

Research applications you may not want participants' data fed to openAI or other proprietary vendors.

16.2 How to?

There are many methods to running LLMs locally on your own hardware. I have chosen GPT4All. It can run under OSX, Windows, and Ubuntu (linux).

16.2.1 Key Features of GPT4ALL

According to the website https://getstream.io/blog/best-local-llm-tools/:

GPT4All can run LLMs on major consumer hardware such as Mac M-Series chips, AMD and NVIDIA GPUs. The following are its key features.

- Privacy First: Keep private and sensitive chat information and prompts only on your machine.
- No Internet Required: It works completely offline.
- Models Exploration: This feature allows developers to browse and download different kinds of LLMs to experiment with. You can select about 1000 open-source language models from popular options like LLama, Mistral, and more.

- Local Documents: You can let your local LLM access your sensitive data with local documents like .pdf and .txt without data leaving your device and without a network.
- Customization options: It provides several chatbot adjustment options like temperature, batch size, context length, etc.
- Enterprise Edition: GPT4ALL provides an enterprise package with security, support, and per-device licenses to bring local AI to businesses.

It is also a very popular LLM for self-hosting (second only to Ollama, which you should also try out).

16.3 How-to

From the GPT4all download page download the version for your operating system.

Follow the download instructions appropriate to your system and pay particular attention to any warnings and make sure that you have sufficient freespace. This is a big application and the model parameters are also quite large.

After you have successfully downloaded and installed the program checkout the quickstart guide.

1. Start Chatting
2. Install a Model
3. I tried "Llama 3.2 1b Instruct"
4. Load the model
5. Start chatting

16.4 Using LLMs for Dynamic Research

What are the mechanics we need for this? We need our model to accept input and generate output and we need a method to transfer the elements of the conversation back and forth. The first stage is to think of the components we have used so far and how we might stitch them together for this task.

It is also a useful transitional step to work incrementally. First, maybe try to set up an interface to allow someone to talk to the model via a web interface. Then you might be able to figure out how to use GET and POST from PHP to communicate between two web servers. There are definitely better ways to do that, but it is a fairly direct starting point. If you want an additional challenge perhaps try to use the tool: curl.

Classroom Exercise

Download and implement the above.
Start the GPT4All Server
See if you can get your GPT4All model to talk to another group's model.

16.5 Why Might You Want To Do This?

Durably reducing conspiracy beliefs through dialogues with AI

https://www.youtube.com/watch?v=qD1fnYDTbHM

This LinkedIn post talks about ChatGPT integration. What would you have to change to get it working with your local server?

Class Question

What is a RAG in this context?

16.6 Next steps

Recently (Duan, Li, and Cai 2024) has published an R package to make all of this easier. The github repository has the library, a "read-me" and installation instructions. Our *class activity* is to see if anyone can get this up and working by next week.

Chapter 17

Databases and management

- What is it
- A book on DuckDB - database stuff
- MariaDB This SQL like, and open source. Might be easier to get started with and still be SQL enough to give them some professional benefits. I was thinking we could get some data online, often they come as CSV's and read it into the database? This is one example how. A blog that compares MariaDB to SQL. A quickie tutorial.
- Why would I want to use a relational database over a csv file (or R data frame or similar)? This could be a class exercise and discussion.
- *Exercise* Download MariaDB

17.0.1 Datascience

- one person's roadmap for non-cs grads

17.0.2 Grant Funding

At the moment I am not sure if will have time for this. But thinking of having the students review the peer review manual for NSERC Discovery Grants. Then have them each write a minimal proposal. Assign the proposals to members of the class, and then hold our own reviewers meeting to decide which projects to fund. Top grants get performed for experiments? Get extra-credit points?

Chapter 18

Summary

In summary, I have not gotten to writing a summary yet.

Appendix A

Version Control

A.1 Getting Started - Some old videos on git from PSYCH363/310

A few years ago when I was teaching a more elementary version of some of this material (PSYCH363) I included some videos on git and using Github. Some components of these videos may be a bit out of date, and a few things about the interface have changed. Still they may be a reasonable starting point for those of you who feel you need a refresher on version control. Note that for PSYCH390 I am assuming you can either already use version control or figure it out on your own. I will give some class time for getting things working, but mostly you will have to pursue this on your own. The notes here are intended to be prompts and guides, but not to be comprehensive.

You will notice that in the videos I talk a lot about Linux. We are not using that for this course so you can ignore those references. However, I do demonstrate the use in the *terminal*. That is available to all of you regardless of operating system. You will just have to find your own OS version of it.

Frequenly you will see this referred to as a command line tool ("cli"). It is now easier to get this this working on Windows and OSX than it used to be. For OSX you can also check out **Homebrew**. For the adventurous Windows user you can look at WSL2.

A.2 An overview of git and Github

https://vimeo.com/456349738

A.3 You have choices - other version control system

1. Mercurial
2. Darcs
3. CVS
4. Subversion
5. Pijul

Each has their own fans. CVS and Subversion are more legacy options, but you will still see them occasionally. Darcs is more of an experiment than a broadly used system. Mercurial used to be the cool kid, but now seems eclipsed by *Pijul*. That is the one for experimental users.

A.4 Git is not Github

Git is the version control software. Github is a very popular place to host your publically accessible git repository, but it is far from your only option. You can host elsewhere.

1. OSF.io For scientists OSF.io seeks to make itself a way to host scientific projects and their data. Trivia question? Do I have any repositories on Osf.io?
2. Sourceforge An oldie, but still used.
3. Bitbucket
4. Gitlab The university provides you with a gitlab account: https://git.uwaterloo.ca
5. Codeberg If you believe in freedom and neck beards.

A.5 Things you should do

1. Install git
2. Get an account on github
3. Fork the course repository
4. Clone the course repository to your laptop
5. Set up my version as an additional **remote**.
6. See if you can make a "pull request" to me. You may find that you have to set up an **ssh key** in order to efficiently pull and push to Github. Github has very clear instructions on how to go about doing this.

A.6 But first

If you have not used git before you will have to configure git. It needs to know who you are and how to reach you at least.

You can execute commands like this in your terminal:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

More on these configuration options can be found in a nice online book.

A.7 Understanding Git and the Workflow in Pictures

When starting to use git I found it very confusing to tell what was where and what direction things were going when I pushed and pulled. I found pictures helpful adjuncts to the prose descriptions. A few helpful illustrations to the distinction of clones, forks, pull, push, and pull request can be found here.

I also made a little video to try and illustrate some of these concepts a few years ago. Read for the gist and not for the specific examples.

https://vimeo.com/456349595

A.8 Principal Terms to Learn First

- **Fork:** A copy of one github repository to another **github** repository.
- **Clone:** A copy of one **git** repository to another **git** repository. The first repository might be hosted on github, but the second one, the *cloned* one exists on a local machine. In your case this is probably your laptop.
- **Remote:** This is a repository that you are following. You will typically *pull* from these, but your *push* permissions may be limited depending on the distinctions between forks and clones, and whether you own the remote or someone else does. You can have more than one remote.
- **Pull:** the transfer of information and changes **from** one repository incorporated into another. This is how you get the new information from a remote transported to a local repository that you control.
- **Push:** this is the transfer of information **to** a repository you control (or have permissions to push to) from another repository that you control. This is often from your local laptop version to the hosted repository (your fork) on github.
- **Pull Request:** When you have information or changes that you think would be helpful to a remote you do **not** have push permissions for then you can request that the owner of that repository pull in your changes. This is a formal process called a pull request. It is primarily a github concept and not a git concept.
- **Branch:** within a repository the development of the code may be proceeding in a few different directions at the same time. The principal production branch is conventionally called **master**. And the principal repository

that is the main, shared one is called **origin**. We will not be working with branches in our course, but those terms do show up in commands.

All of these "basics" are covered in detail in the book Pro Git (available on line).

Appendix B

Just a place for some notes for installing things on Mac

Appendix C

Emacs

I used homebrew awhile ago for this and forgot the details.

C.1 init files

Seems to live at `~/.emacs.d/init.el`

Added these lines and restarted to get the basic package set up working.

```
(require 'package)
(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)
(package-initialize)
(require 'use-package)
(setq use-package-always-ensure t)
```

C.2 pdf-mode

Do a `package-list-packages` to get a refesh and install (mark with an "i") the `pdf-tools` package from melpa. Then you "x" to execute the installation. Afterwards you need to run `M-x pdf-tools-install`. This can take awhile.

C.3 keys and keyboards

Several emacs shortcuts conflict with defaults on mac. You can change these in the system utilities for keyboard shortcuts. Just search them out as they arise and turn off the mac ones. The ctrl key is ^ in their printed list.

C.4 orgmode related

1. for exporting html needed to install package `htmlize`.

C.5 emacs

There is a quarto editing mode which I installed through the packaging tools.

Appendix D

Quarto

Find the download here. The demo needs `matplotlib` and `plotly`. So, you need python which I got from brew. `brew install python-matplotlib` will pull in a lot of other libraries as well. There are some tricky steps here. Plotly is not easily gotten via homebrew, and I did not want to mix up my system by also using pip now that I was working hard under homebrew. This brings up the idea of virtual environments. That is generally a good idea, especially with python in my opinion. While doing some searching I came across `pipx` that seems to try and do both. Allow you to install via homebrew (the pipx) and then use pipx to create virtual environments that can isolate the libraries, but expose the binaries.

```
brew install pipx
pipx ensurepath
sudo pipx ensurepath --global # optional to allow pipx actions with --global argument
brew update && brew upgrade pipx.
```

Then, because I could not get plotly in homebrew, I created a **venv**.

```
python3 -m venv /Users/britt/p390venv
cd p390venv
source bin/activate
pip install plotly
```

I needed to end up also doing pip install for matplotlib and jupyter from within the venv. Then I simply cut and pasted the demo document and followed their recommendation.

The file I tested as "test.qmd" is …

[]: < include ../../../p390venv/test-quarto/test.qmd >

And the command was `quarto render test.qmd --to html`.

And if you `brew install texlive` then export to pdf works well too.

Appendix E

Python

Appendix F

Virtual Environments

The advantage of a virtual environment is the ability to isolate libraries you may install for a particular project from interfering with other projects or previously installed libraries. The disadvantage is that they do involve some additional steps and complexity and you may end up installing multiple versions of the same library, which can be an issue if hard disk space on your computer is limited.

F.1 Methods

There is more than one tool for creating a python virutal environment, but one of them is built into the standard python installation: *venv*, and so for simplicity this is the one that I recommend starting with.

F.1.1 Creating and Activating the Venv

If you are in a console you can simply run these commands in the terminal.

```
python -m venv <dir-where-you-want-venv>
cd <dir-where-you-want-venv>
source ./bin/activate
```

Note

Depending on the specifics of your operating system and python installation you may find you need to use `python3` and `pip3` commands instead of plain `python` and `pip`.

If you are using a Windows device you will have to find your equivalent of a linux terminal. This can be using WSL or something like MinGW.

Once you have activated your `venv` you will see a change in the terminal display that indicates the `venv` is active. Now when you run things like `pip install <something>` it will install that library to a sandbox that is only accessible when using python from within that virtual environment. You should note that this notion of a sandbox or virtual environment is also available for some other programming languages.

To deactivate the environment you run ,

```
deactivate
```

from within the venv. Look for the change in the terminal display. Note how activating the environment shows up in the terminal prompt.

```
britt@arts-220486 ~ % source p390venv/bin/activate
(p390venv) britt@arts-220486 ~ % deactivate
britt@arts-220486 ~ %
```

F.1.2 The Terminal is All You Need

If you are a minimalist or doing something simple you can invoke the python interpreter in your venv and complete your task there with no other tools, and using only the libraries required for your particular, limited task. However, if you use some companion tooling things may get more complicated and trickier to debug.

F.2 Visual Studio Code

VS Code tries to make using a venv easier, but this also means there is some indirection that may make it harder to puzzle out errors. The basic work flow is to make sure that VS code is working in the folder you want your venv to be in. If not, use the `open folder` menu of VS Code to do so. Then, you can use the `Command Palette` (found under the *View* menu tab) to locate the `Python Create Environment` command. If you have never created an environment before it will do so. This can take some time so be patient and make sure this process completes. If there was a virtual environment previously constructed VS Code will probably be able to figure that out and ask you if you want to re-use it. Usually you will, but if not you can select to destroy it and create a new one. You will probably also need to affilate a particular python version to association to your environment. VS Code can often make a reasonable suggestion and you can also use the command `Python Select Interpreter` if necessary.

After you have done that you will need to move into the affiliated terminal window of your VS Code environment to install the necessary libraries. You will need to be deliberate and make sure that you are in the correct terminal. VS Code may have several of these open for different purposes if you are working on a complicated project.

Note that while it may be convenient to use the VS Code terminals you do not have to. If you have a terminal program on your computer accessible to you outside of VS Code you can use VS Code as the editor and then move to your other terminal for activating and implementing and compiling the python project.

F.2.1 Quarto Complications

When you embed a python code block in your .qmd document you will want to compile it. Quarto is assembling a document. It will need more than just the libraries necessary for compiling the python code. It also needs the library for the textual and graphical representation of the output that you will insert into your document. Specifically, it needs the `jupyter` python library. This means that whatever the python is doing, even if no additional libraries are needed, you will have to have done a `pip install jupyter` in your venv *and* you will need to have the venv active for the location where the qmd is stored and compiled.

It is worth knowing that although VS Code offers the convenience of buttons to invoke the quarto process it is *not* necessary. You can invoke all the quarto commands through a terminal directly. For instance `quarto preview <file-name>.qmd` will try to compile your file and will also usually launch your browser so you can see the html result. It will then watch the source file and update the html whenever it detects you have changed the source. This can be quite convenient for editing.

Duan, Xufeng, Shixuan Li, and Zhenguang G. Cai. 2024. "MacBehaviour: An r Package for Behavioural Experimentation on Large Language Models." *Behavior Research Methods* 57 (1). https://doi.org/10.3758/s13428-024-02524-y.