Part 2: Have a workflow

NOTE: *This text is an opinionated technical manual for graduate students, researchers, or anyone looking to get started with data analysis, visualization, reporting, dashboards development, or website/blog creation. We primarily recommend performing these tasks with R, RStudio, and Git/Github. We're not saying there aren't other means or tools capable of accomplishing the same activities; these are the tools we've found success with, so they're what we recommend.*

Principle 1: Use open source software

The tools in this book are open source and available free of charge. The main reason we recommend open source tools are the communities that you can get access to when you start using them.

The other reason is philosophical: we all benefit from using these tools and sharing improvements on them together. The 'four freedoms' of open source software captures this sentiment below.

Freedom 0: The freedom to run the program as you wish, for any purpose.

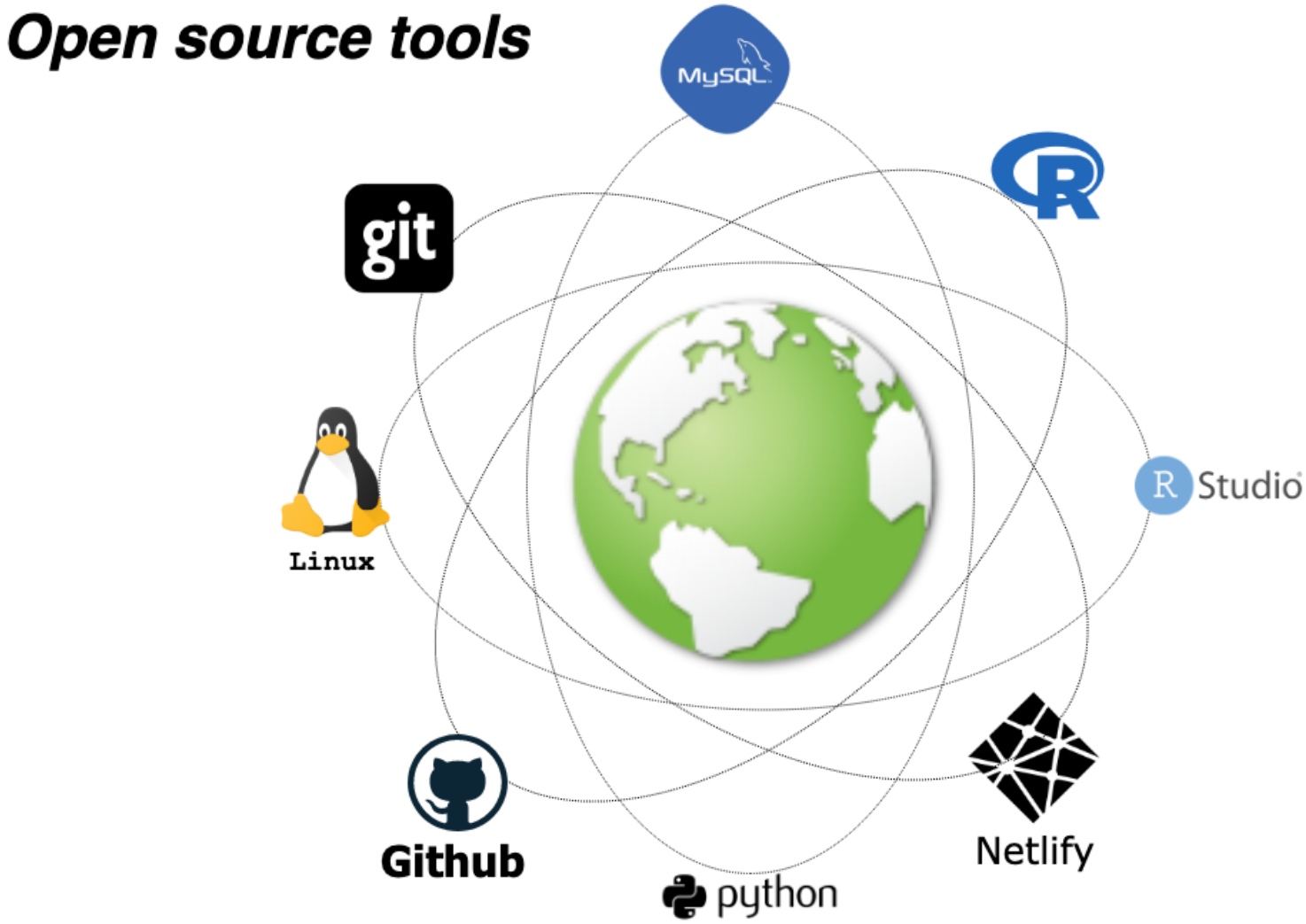
Freedom 1: The freedom to study how the program works, and change it so it does your computing as you wish.

Freedom 2: The freedom to redistribute copies so you can help your neighbor.

Freedom 3: The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes.

We also think it's onerous to require graduate students (and other scientists) to purchase proprietary software licenses to participate in science.

We've displayed some examples of open source tools for data management, statistics, and communication in the image below:



- Git
- Github
- Linux
- MySQL
- Netlify
- Python
- R
- RStudio

Use R, RStudio, Git, and Github

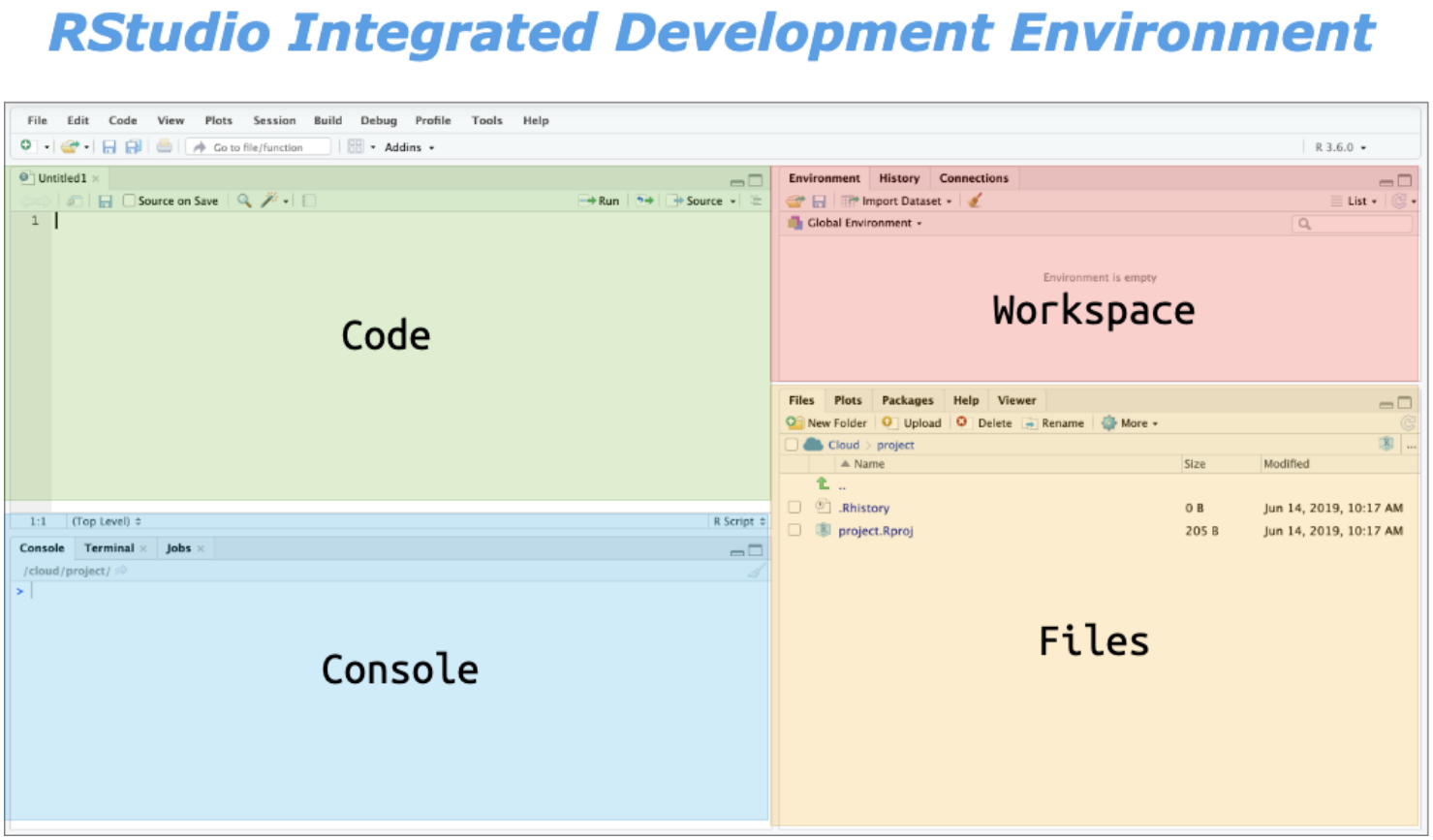
We recommend using R, RStudio, Git, and Github for the majority of your work. You can discover more about these tools here.

What is R?

R is a free statistical modeling software application and language.

What is RStudio?

RStudio is a free and open source integrated development environment (IDE) for R.



The IDE has four separate panes, each serving a specific function.

- the `Code` pane is where we can document both human readable and computer readable text
- the `Workspace` holds the data, functions, and other data analysis artifacts
- the `Console` displays the results from our written code
- the `Files` gives us a few of happenings outside the RStudio environment (imported raw data, exported results, etc.)

Follow the steps below to install R and RStudio.

1. First, you'll need to download and install R from CRAN.
2. Second, download and install RStudio, the integrated development environment (IDE) for R
3. An alternative to downloading and installing R and RStudio is using RStudio.Cloud which operates entirely in your browser. You'll need to sign up for RStudio.cloud for free using your Google account or email address, but we recommend using a Github account. You can create a Github account here

BONUS: You'll also find a massive network of support on Stackoverflow, RStudio Community, and Google Groups.

What is Git?

Git is a version control system (VCS). VCSs are used to track changes to projects with code. You can read more about Git in this online text.

What is Github?

Github is the web-based hosting service for Git. You should set up a free an account with Github here.

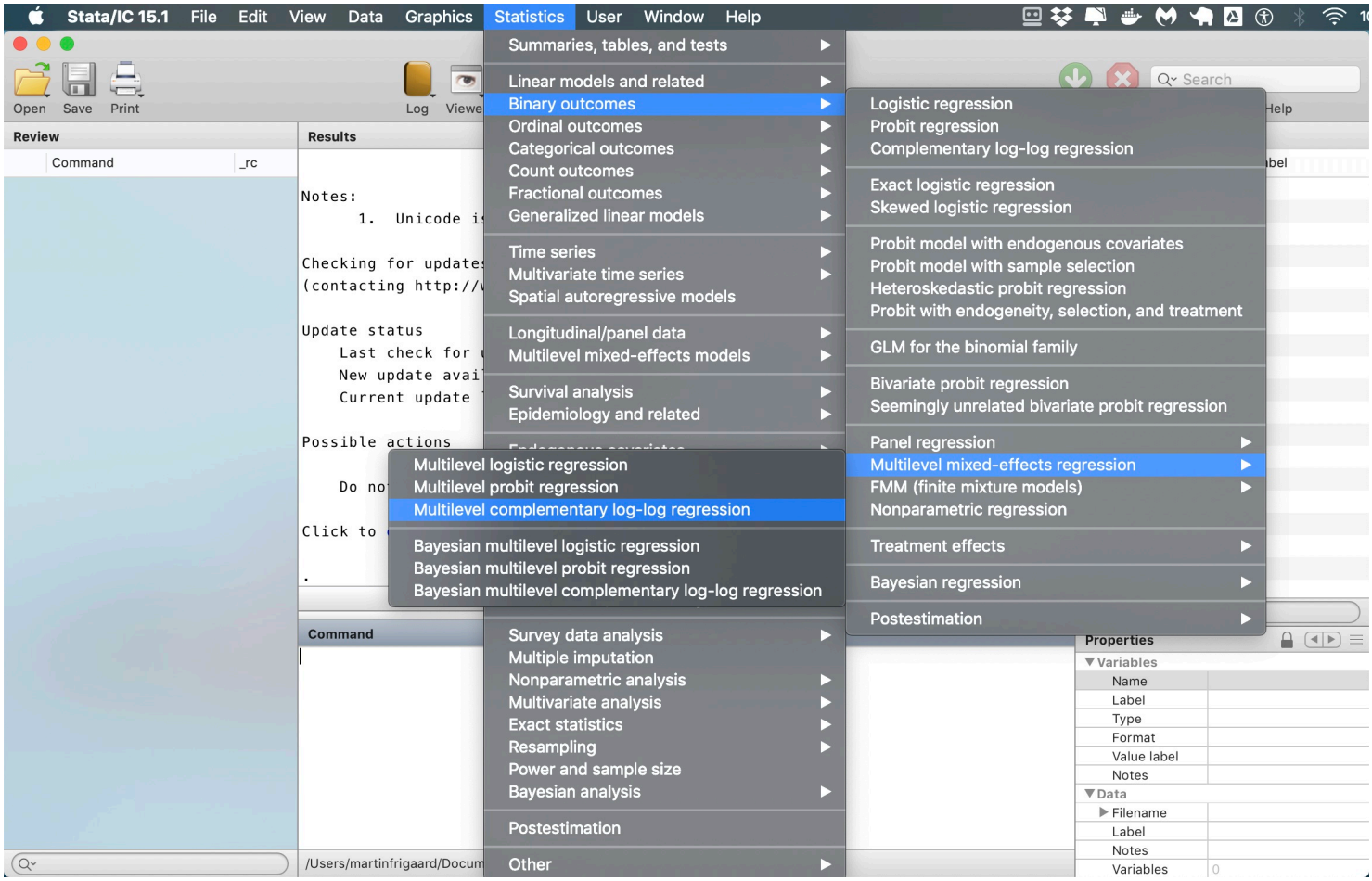
We will cover more on Git/Github in later sections, but for now, know these tools will allow you to keep track of changes to your project over time.

Note: *You should explore different IDE's on your own– you'll see there are many options, both paid and unpaid. We're confident you'll see RStudio is well suited to handle >90% of the things you'll want to accomplish.*

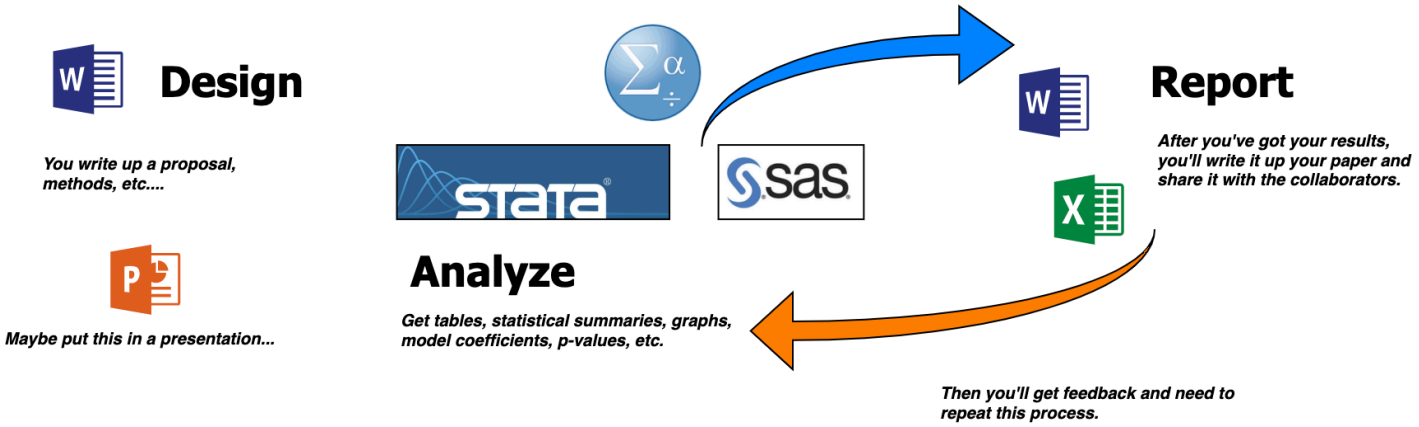
Principle 2: Code not clicks

Usually, people interact with their computers using point-and-click graphical user interfaces or GUIs (pronounced 'gooey'). GUIs are quick and easy to learn because their design environment usually mimics an actual physical space (i.e., desktops, folders, documents). GUIs are a mostly positive development because designing software with a more user-centered design is one of the main reasons technology adoption has been on the rise for the 20+ years.

Examples of user-centered-design GUIs include statistical programs like Stata, SPSS, and SAS. These programs give the user the ability to click through a predetermined list of statistical tests and operations using their mouse or track-pad.



However, we think there are times when we should resist the temptation to abstract away some of life's complexity, and data analysis is one of them. Using applications like these encourage a copy + paste workflow like the one below.



As you can see from the image when you receive feedback or input to your research, you'll be required to go back through the same elaborate procedure (with each time taking just as long as the first).

We recommend an alternative to the copy + paste workflow based on the activities of a modern scientist from Jeff Leek we listed in Chapter 1:

1. Develop code in the open
2. Publish data and code open source
3. Post preprints of their academic work
4. Submit and review for traditional journals
5. Blog or use social media to critique published work

Two things should stand out from the list above: first, modern science is mostly writing. Second, some of that writing is code (i.e., programming). It's, for this reason, we recommend adopting a workflow based on, "code, don't click" wherever possible.

NOTE: A 'code, don't click' workflow means being able to type, which might be daunting for people who struggle on a keyboard. We recommend practicing this skill (there are plenty of great apps out there to help!) because typing is an unavoidable necessity for doing research.

Software isn't a solution

An extension of the "*write, don't click*" philosophy is to view software applications as tools for improving the scientific process, not oversimplifying or obfuscating it.

As we stated in chapter 1, science is a process. An iterative, problem-solving process with specific steps:

1) make an observation
2) ask a question
3) develop a testable explanation
4) predict with the new explanation
5) test that prediction
6) review the result, take notes
7) make new predictions

Software that improves scientific discoveries shortens the distance between these necessary steps without leaving out the crucial details. For this reason, we recommend avoiding point-and-click environments in most proprietary software applications (SPSS, Stata, SAS, etc.). It's hard to keep track of everything you click on (or the order of you clicked on them in) inside GUIs like these, and this makes it hard to know what step you're currently doing.

Furthermore, most GUIs come with a limited collection of possible operations. The designer of the software decided on all of the icons, drop-down lists, and options (i.e., not you). Inevitably, there will be something you'll need to do the GUI won't offer.

Lastly, although these applications give you the ability to write code, these are typically not seen as programming languages. The esoteric structure and syntax for the code is a direct result of the language being dependent on its parent application (*the reason the applications cost money is because they're trying to attract users who don't want to program*).

What do we mean by workflow?

> *"I've come to realize that a lot of people don't even know what they did. People don't have a workflow, they have a bunch of numbers, and they start screwing around with the numbers and putting calculations in different places on their spreadsheet, and then at the end, they pull a number out and write it down and type it into their report."* - *Andrew Gelman*

The quote above is from an interview with Andrew Gelman, a statistician from Cornell, who is an author on the excellent blog Statistical Modeling, Causal Inference, and Social Science. I've included it here to emphasize the need for having **any workflow**. Your workflow is your chosen software tools you use (or the '*how*' you got your results). As the quote above shows, this is just as relevant as the results you get, which means you should strive to have a start-to-finish chain of documentation for your analytic or scientific process.

Data analysis and research are complicated, and in order organize that complexity, we need to have a workflow that gives us the ability to 1) document our thoughts and intentions clearly, and 2) write code that translates those thoughts and intentions into something a computer can execute. Plain text files are a great way to accomplish these tasks.

Principle 3: Work in plain text

In the classic text The Pragmatic Programmer, authors Hunt and Thomas advise *'Keep[ing] Knowledge in Plain Text'*. This sentiment has been repeated here, here, and here.

We recommend you keep track of your changes, notes, and any pertinent documentation about your project in plain text `README` files. The reasons for this will become more apparent as we move through the example, but I wanted to outline a few here:

- plain text lasts forever (files written 40 years ago are still readable today)
- plain text can be *converted* to any other kind of document
- plain text is text searchable (ctrl+F or cmd+F allows us to find keywords or phrases)

These all sound great, but you might still be wondering what makes a file 'plain text,' so we'll define this below. This chapter will also cover why you might want to consider switching over to a plain text editor if you're currently using something like, Google Docs, Apple Papers, or Microsoft Word.

Wait–why would I change what I'm doing if it works?

We get it–change is difficult, and if you have a working ecosystem of software that keeps you productive, don't abandon it. However, you should be aware of these technologies and recognize that people using them will be adapting *their* workflows to collaborate with you.

The problems with the copy+paste (Word, SPSS, Stata, Excel, etc.) workflow have been discussed elsewhere (see Baumer et al. for an in-depth summary), but we will briefly summarize,

1. It's not reproducible
2. It's not logical or necessarily honest to separate computation from the analysis or presentation
3. It's error prone

What *isn't* plain text

Non-plain text files are usually called binary (i.e., files with binary-level compatibility) need special software to run on your computer. The language below is a handy way to think about these files:

> *"Binary files are computer-readable but not human-readable"*

What *is* plain text

So if binary files aren't plain text, what is a plain text file? The language from the Wikipedia description is helpful here:

> *When opened in a text editor, plain text files display computer and human-readable content.*

And here is the most crucial distinction–**human-readable vs. computer-readable**. I'll be sure to point out which files are binary and which are plain text as we go through the example, but generally speaking, a plain text file can be opened using a text editor. Examples of text editors include Atom, Sublime Text, and Notepad++

Markdown & Rmarkdown

A common type of plain text file is a markdown file, or `.md` file. Markdown has a straightforward syntax that's easy for both humans and computers to read, and it allows for some formatting options to aid with communication (see Markdown Syntax Documentation on John Gruber's site).

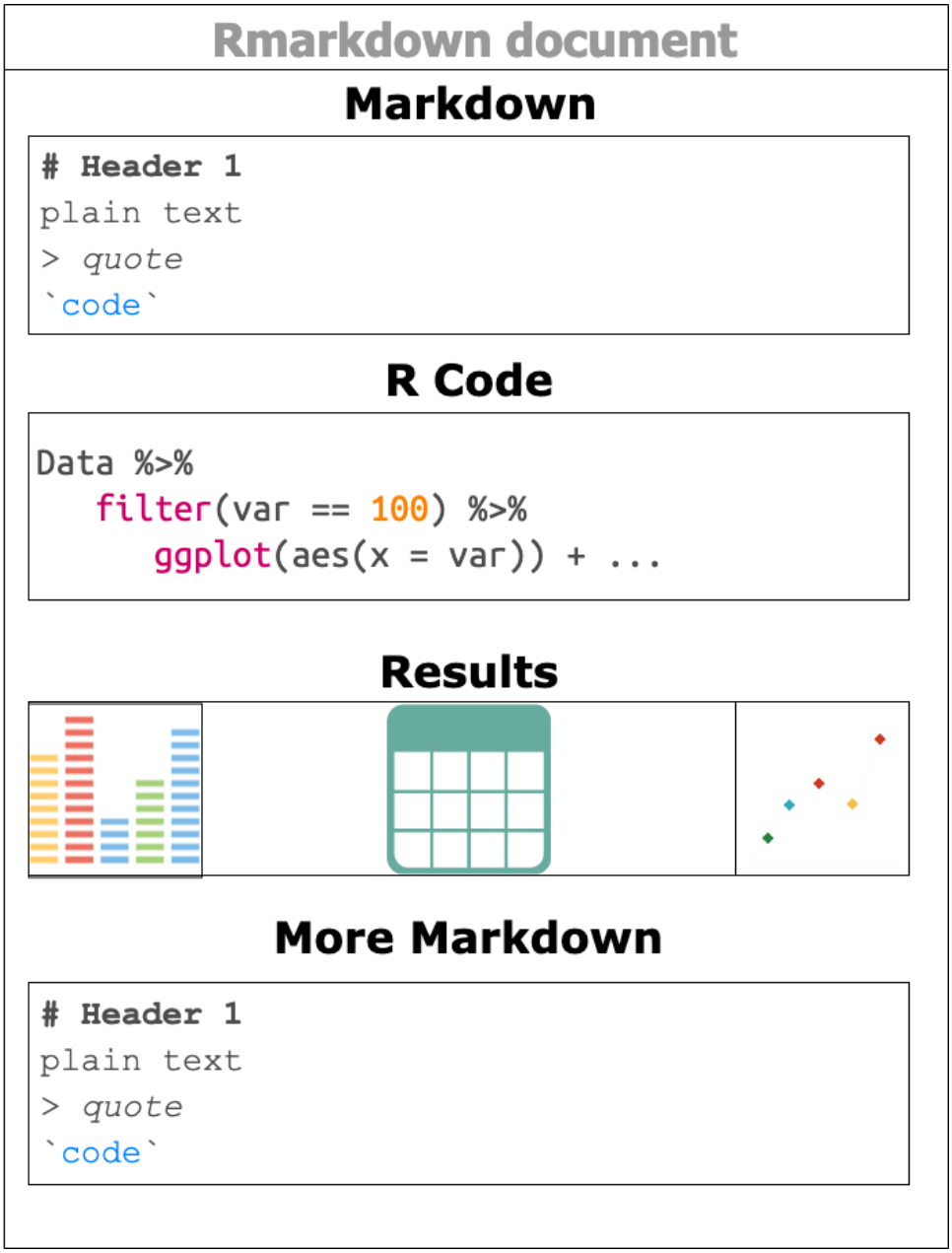
RStudio has an extension of markdown, RMarkdown. Using RMarkdown in RStudio allows for a genuinely reproducible workflow because you're able to write your thoughts, code, display results, and then share everything in multiple outputs.

Words for humans to read (prose, expectations, predictions...)

Code that translates our thoughts into something the computer can read

Imported data, summary tables, graphs, model results, etc.

Output to interpret, explain, makes sense of, generally ponder over, etc.



I recommend reading up on R and RMarkdown because of how many different outputs this combination can be used to produce (.pdf, .docx, and .html). Consult the R Markdown: The Definitive Guide for more information. The image below is an output from an `.Rmd` document in RStudio.

Rmarkdown Document

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

Load the `tidyverse`.

```
## ● __  _     __    .    ○          ●  .
##  / /_(_)__/ /_ ___  _____ _______ ___
```

```
## / __/ / _  / // / |/ / -_) __(_-</ -_)
## \__/_/\_,_/\_, /|___/\__/_/ /___/\__/
##      ●  . /___/      ○      .       ●
```

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

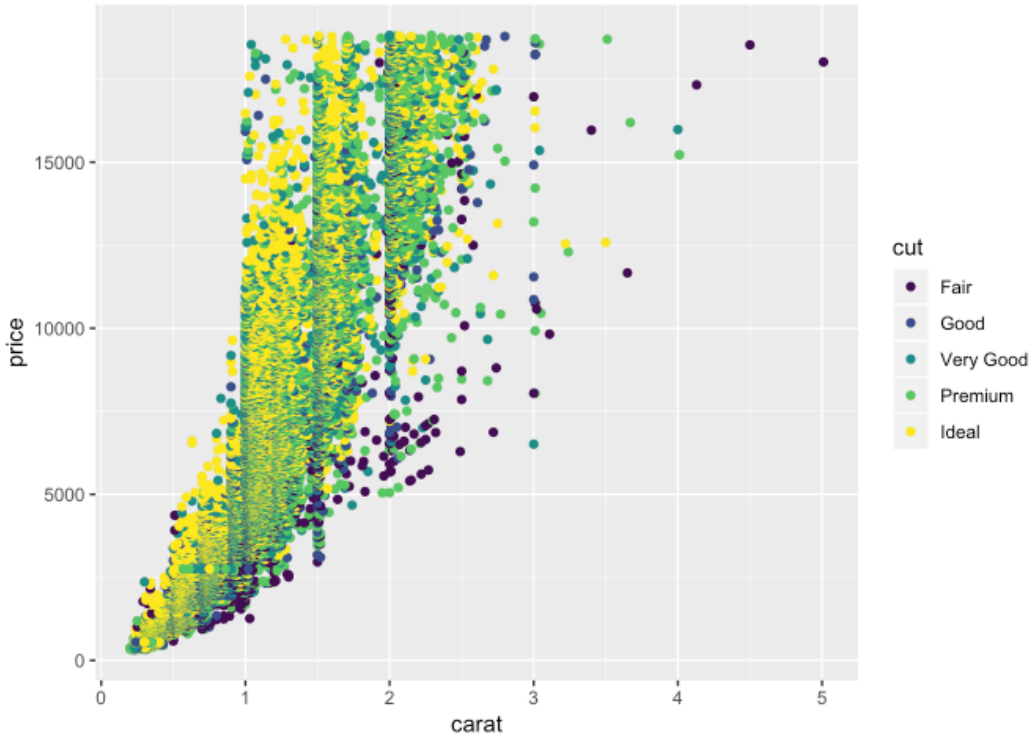
```
ggplot2::diamonds %>% glimpse(78)
```

```
## Observations: 53,940
## Variables: 10
## $ carat   <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0.23,…
## $ cut     <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Good, …
## $ color   <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, J, J,…
## $ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1, VS…
## $ depth   <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 59.4,…
## $ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62,…
## $ price   <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339, 340,…
## $ x       <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, 4.00,…
## $ y       <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, 4.05,…
## $ z       <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, 2.39,…
```

Including Plots

You can also embed plots, for example:

```
diamonds %>%
    ggplot(aes(x = carat,
               y = price,
               color = cut)) + geom_point()
```



Don’t just take our word–listen to the pro

Hadley Wickham, Chief Scientist at RStudio and primary developer of the `tidyverse`, made this point in an excellent talk aptly titled, You can’t do data science in a GUI

> *The gooey is the easiest type of approach where you point and click, and everything is laid out in front of you. All of the options are laid out in front of you, which is great because you can see everything you can do. But it’s also terrible because you have constraints–you can only do what the inventors of (SAS or Excel) wanted. Whereas with R–or other programming languages–is the opposite. All you get is this blinking cursor, and it’s just telling you can do literally anything, but it’s not gonna give you much…*
>
> *So I think an important thing about programming languages–like R or Python–is they give you a language to express your ideas, they give you very few constraints, which makes life tough for your learning or doing data science things occasionally, but the payoff for investing in a programming language is you get this whole this new language, and what you can express with them.*

You should write code because it makes you think explicitly about what you want to do with your computer. Writing out instructions for how to use a GUI is possible, but it amounts to a bunch of pictures with text saying * “click here, then click here ”*. Learning to code well is like learning to write well–the better you get, the more precise your intentions become to* both* your computer and everyone else reading your code.

Python vs. RStudio

Python is a great language, and it can do a broader range of computational tasks than R. I would never tell a researcher or scientist that Python is something they shouldn’t learn (the benefits of being multilingual extend beyond just spoken languages, too).

We recommend R/RStudio because we wrote this book for people researching graduate school. Most graduate students have a research question or general curiosity that they will turn into a data set that needs to be analyzed. Thus, the entry point for our audience into data science is *with data they need to analyze*, and this is what R was built to do.

Additional reasons for using R/RStudio

Below are a few more reasons you should consider using R/RStudio in case you’re still on the fence.

The time you’ll save (eventually)

We recommend R/RStudio because of the time saved by switching between software applications. For example, when I was in graduate school, I had to have *a minimum of five applications open* to do my daily work of data analysis (MS Word to write, MS Excel to create tables, Stata for statistics, the browser for internet research, and Adobe for reading .PDFs).

Five different GUIs, each with their design characteristics, and each costing me valuable neurons every time I had to switch between them (read more about attentional residue in the footnotes). With R/RStudio, I cut this number to two (RStudio and the browser).

> *’The only factor becoming scarce in a world of abundance is human attention*" – *Kevin Kelly in Wired*

RStudio gives you a better mental model for data analysis

The third reason is the design of the IDE itself. RStudio is a complementary cognitive artifact, something described in this article from David Krakauer,

> *"[complementary cognitive artifacts are] certainly amplifiers, but in many cases they're much, much more. They're also teachers and coaches…Expert users of the abacus are not users of the physical abacus—they use a mental model in their brain. And expert users of slide rules can cast the ruler aside having internalized its mechanics. Cartographers memorize maps, and Edwin Hutchins has shown us how expert navigators form near symbiotic relationships with their analog instruments."*

These are in contrast to competitive cognitive artifacts, which is what a GUI does.

> *"In the case of competitive artifacts, when we are deprived of their use, we are no better than when we started. They're not coaches and teachers—they are serfs."*

RStudio does not remove the complexity of doing data analysis, writing blog posts, building applications, debugging code, etc. Instead, it creates an environment where you can do each of these tasks without having them abstracted away from you into drop-down menus, dialogue boxes, and point-and-click options.

There have been considerable efforts from the scientists at RStudio to create an environment and ecosystem of tools (called `packages`) to make data analysis less painful (and even fun). We're confident you'll find it helps you think about the inputs and outputs of your work in productive and creative ways.

FOOTNOTES

- The Ford Foundation report, "Roads and Bridges", outlines some other reason you should be using open source software.
- Read these articles on attentional residue and multitasking (then try to stop doing it).
  - Why is it so hard to do my work? The challenge of attention residue when switching between work tasks - ScienceDirect
  - Information, Attention, and Decision Making
  - Causes, effects, and practicalities of everyday multitasking
- Other examples of IDEs are DataGrip for relational data, Spyder IDE for Python.