Part 2: “Have a workflow.”

Working with data **workbench** is a place to keep and organize tools, and a **workflow** is how you combine these tools to get things done. This chapter will cover the workbench we use and the three guiding principles of the workflow we recommend.

1. Use free open source software
2. Write code
3. Document everything in plain text

Principle 1: Use open-source software

All of the tools in this book are available open-source and available free of charge. Just as a point of reference, the cost of a subscription to SPSS at the time of this writing is $99.00 per user per month. Stata is $595 per year or $1,595 for a perpetual license. There are educational discounts available, but this cost is not offset by much when you take into account the rising price of tuition.

A more important reason we recommend open source tools are the communities that you’ll get access to when you start using them. By entering the universe of open source software, you get to take advantage of seeing problems solved in the open. You’ll also find people like you, grappling with the same issues, and it’s hard to overstate the benefit of this shared camaraderie.

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

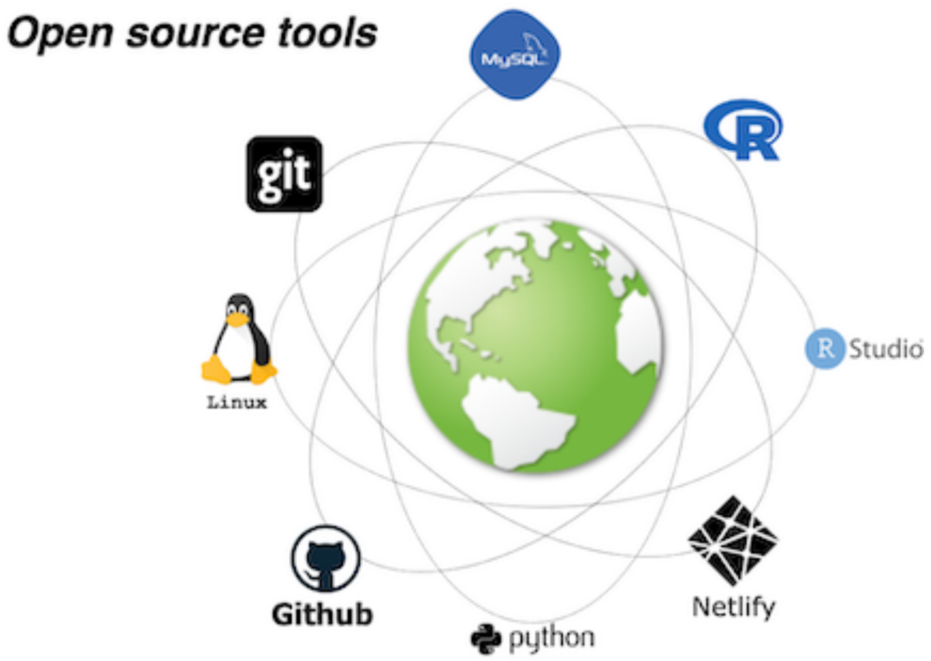
Freedom 0: The freedom to run the program as you wish, for any purpose.

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

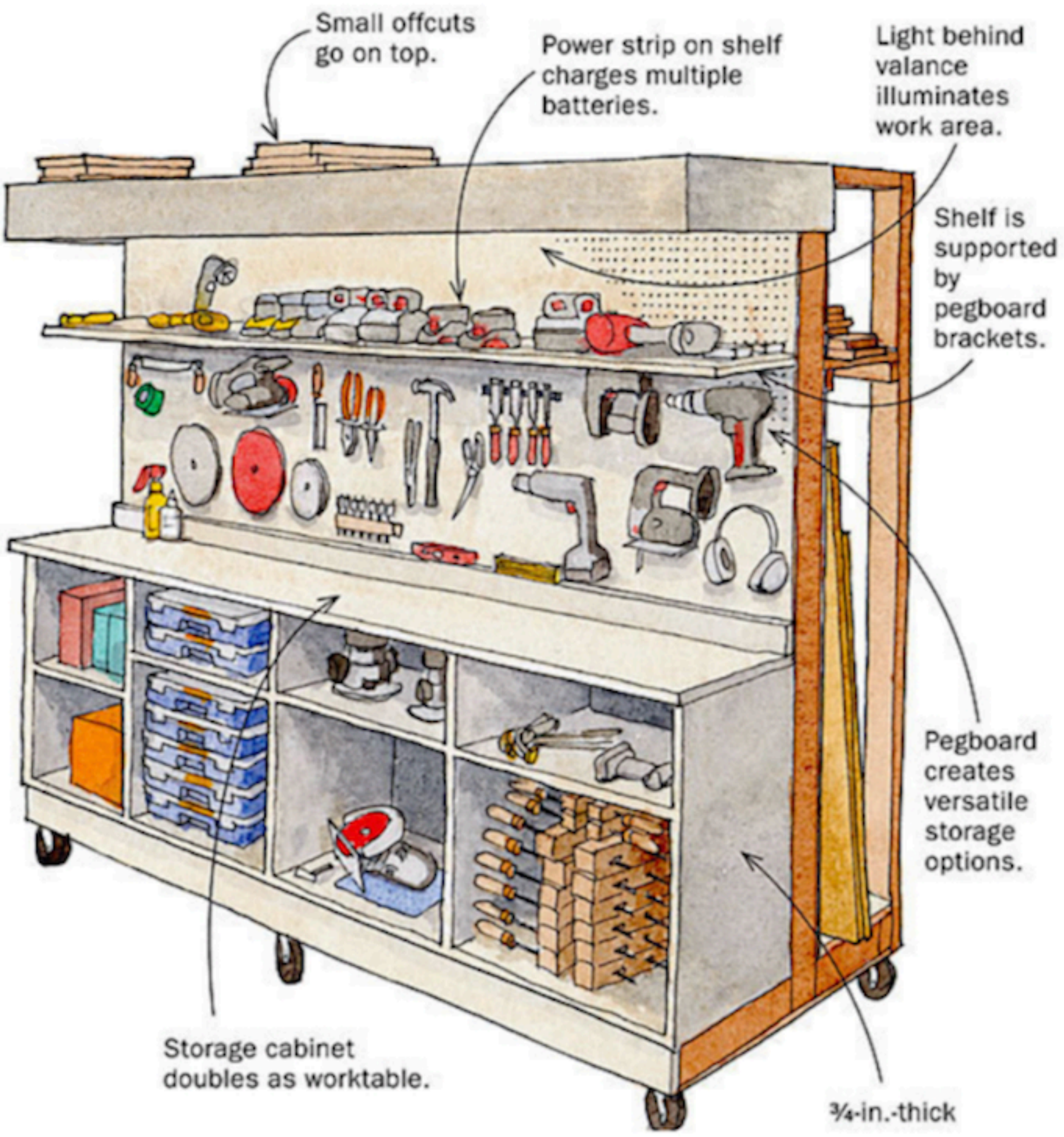


Follow the links below to learn more.

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

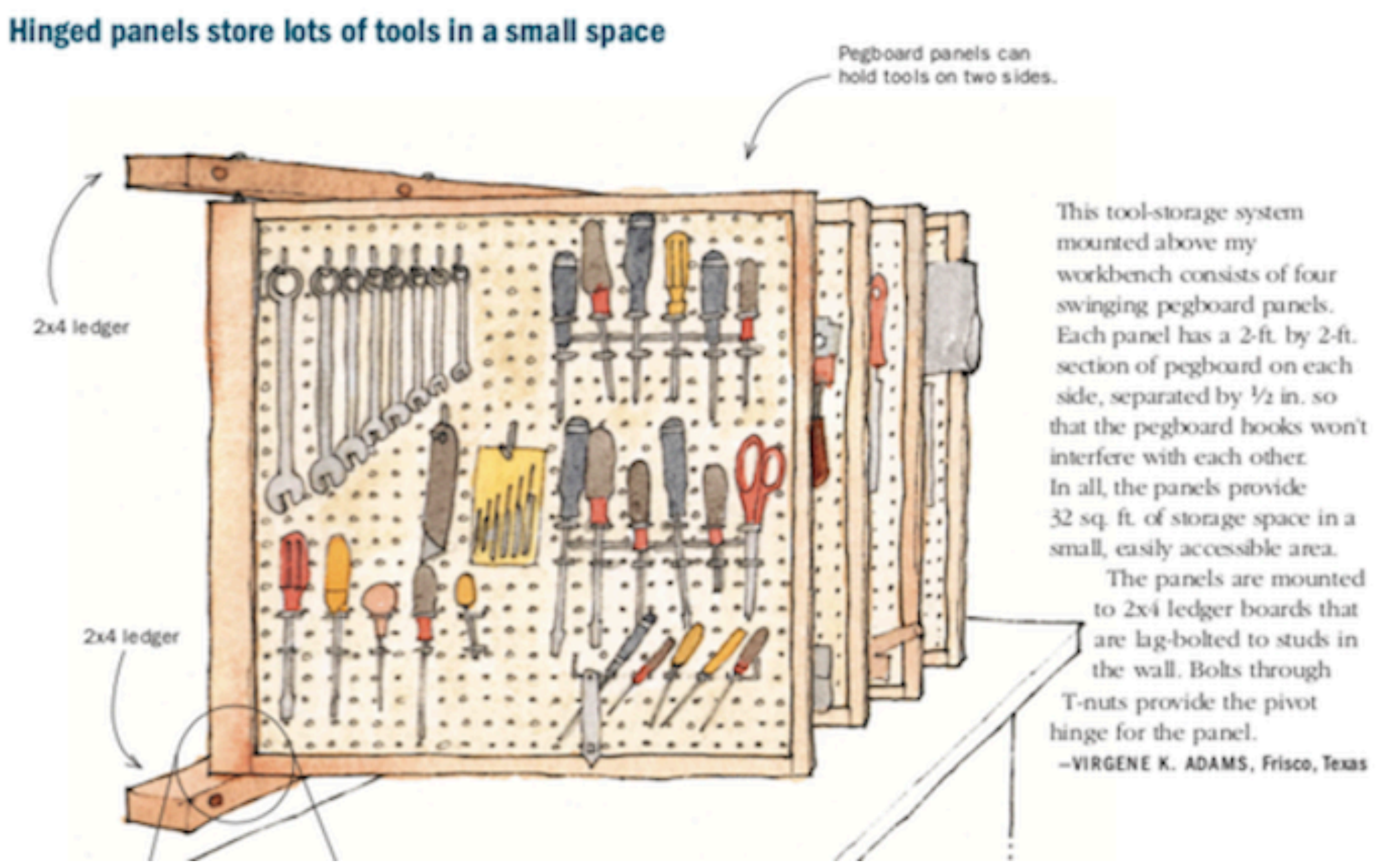
The integrated development environment (aka the workbench)

An integrated development environment (IDE) is an application typically used by programmers to build and test software. We find it helpful to think of a woodworkers workbench as an analogy. The image below is an example of a workbench with a simple rolling cart design (for people with minimal garage space).


Small offcuts go on top.
Power strip on shelf charges multiple batteries.
Light behind valance illuminates work area.
Shelf is supported by pegboard brackets.
Pegboard creates versatile storage options.
Storage cabinet doubles as worktable.
¾-in.-thick

source: https://www.finewoodworking.com/2008/12/11/lighted-storage-cart-for-tools-and-lumber

As we can see, this workbench is efficiently designed to keep essential tools for the job within arms reach, and it uses storage space efficiently. IDE design follows these same principles. However, we also know some models are better than others. For example, consider the design of a different workbench below.


Hinged panels store lots of tools in a small space
Pegboard panels can hold tools on two sides.
2x4 ledger
2x4 ledger
This tool-storage system mounted above my workbench consists of four swinging pegboard panels. Each panel has a 2-ft. by 2-ft. section of pegboard on each side, separated by ½ in. so that the pegboard hooks won't interfere with each other. In all, the panels provide 32 sq. ft. of storage space in a small, easily accessible area.
The panels are mounted to 2x4 ledger boards that are lag-bolted to studs in the wall. Bolts through T-nuts provide the pivot hinge for the panel.
—VIRGENE K. ADAMS, Frisco, Texas

source: https://www.finewoodworking.com/2010/11/12/free-plan-space-saving-tool-rack

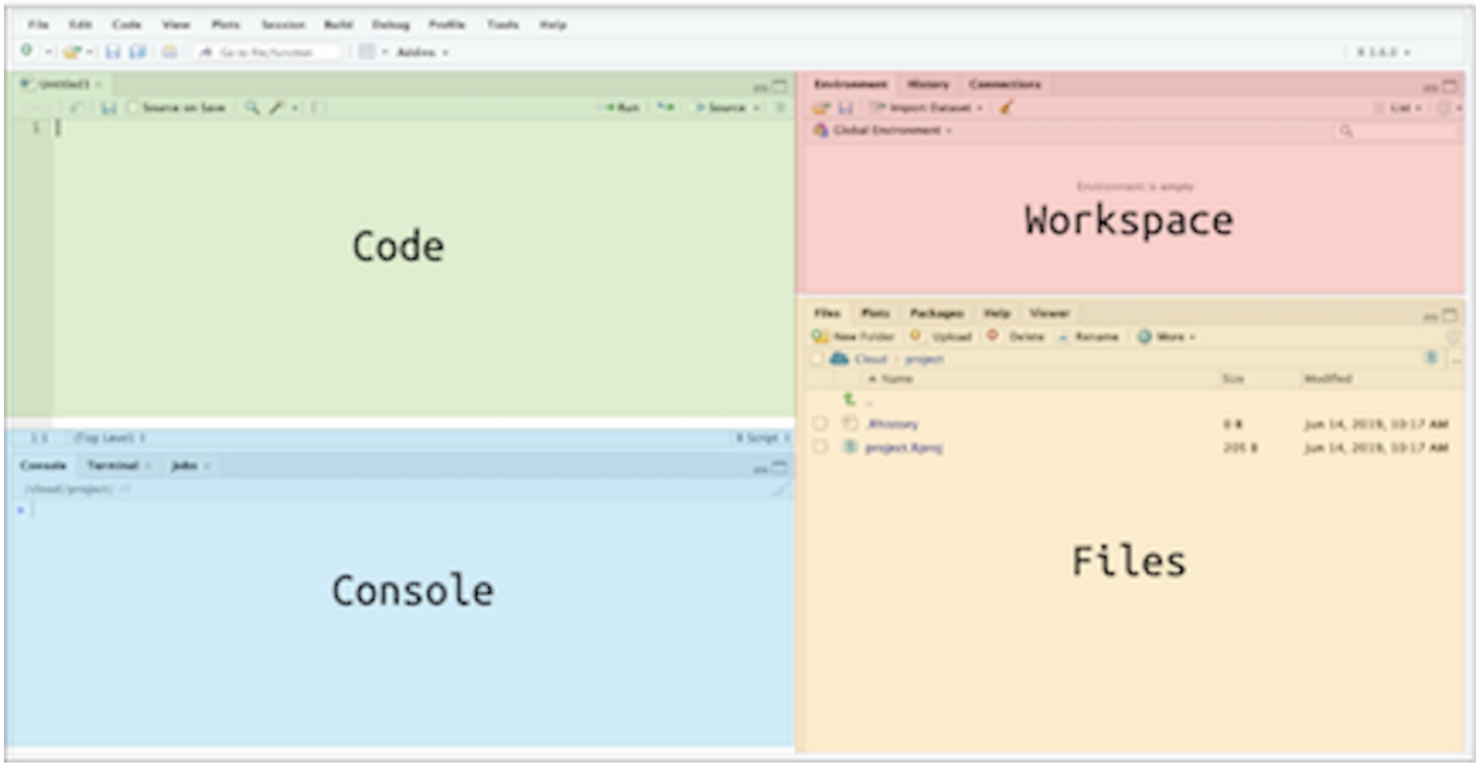
By making the panel positions adjustable, the workbench allows for easier access to more tools. Depending on the job, woodworkers can customize the panel arrangements with "*the simple pin that allows the rack's various faces to swing left/right for access to either side.*."

These examples illustrate how differences in the design of a workbench can have a meaningful impact on levels of productivity. Well-designed workbenches give us access to more tools without making these tools more difficult to find.

Our workbench

We recommend choosing a workbench that minimizes the number of additional applications you'll need to have open to get work done. We've found we can use RStudio for ~90% of our daily work (*they're not paying us to say this*). RStudio gives us access to all the tools we need in the same place.

RStudio Integrated Development Environment



RStudio has four primary panes, each serving a specific function.

- the **Code** or **Source** pane is where we can document both human-readable and computer-readable text
- the **Workspace** holds the data, functions, and other data analysis objects
- the **Console** displays the results from our written code (and allows us to enter commands directly)
- the **Files** gives us access to the happenings outside the RStudio environment (imported raw data, exported results, etc.)

Just like any workbench, we need to fill RStudio with tools we need for the job (and RStudio plays well with many open-source software tools!). For now, we are just going to focus on R and Git.

What is R?

R is a free statistical modeling software application and language. If you are using the desktop application, follow the links below to install R.

Installing R & 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

Use RStudio in the browser

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.

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 their online text here.

What is Github?

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

What do Git/Github do?

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.

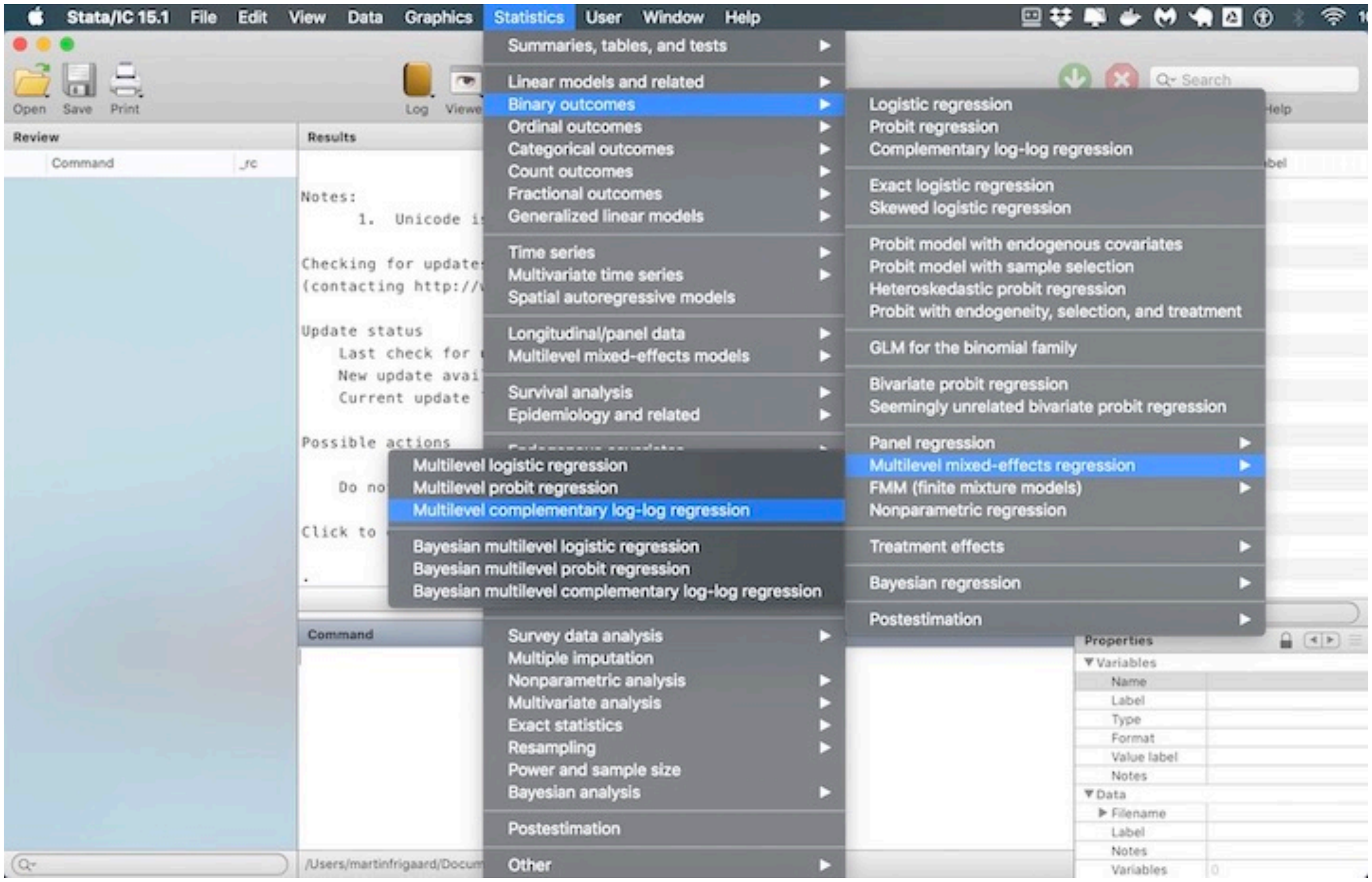
Note: *You should explore different IDE's on your own– you'll see there are many options, both paid and unpaid. We're confident you'll see RStudio is well suited to handle more than most of the things you'll want to accomplish.*

Open-source software bonus: *As mentioned previously, you'll also find a massive network of support on Stackoverflow, RStudio Community, and Google Groups.*

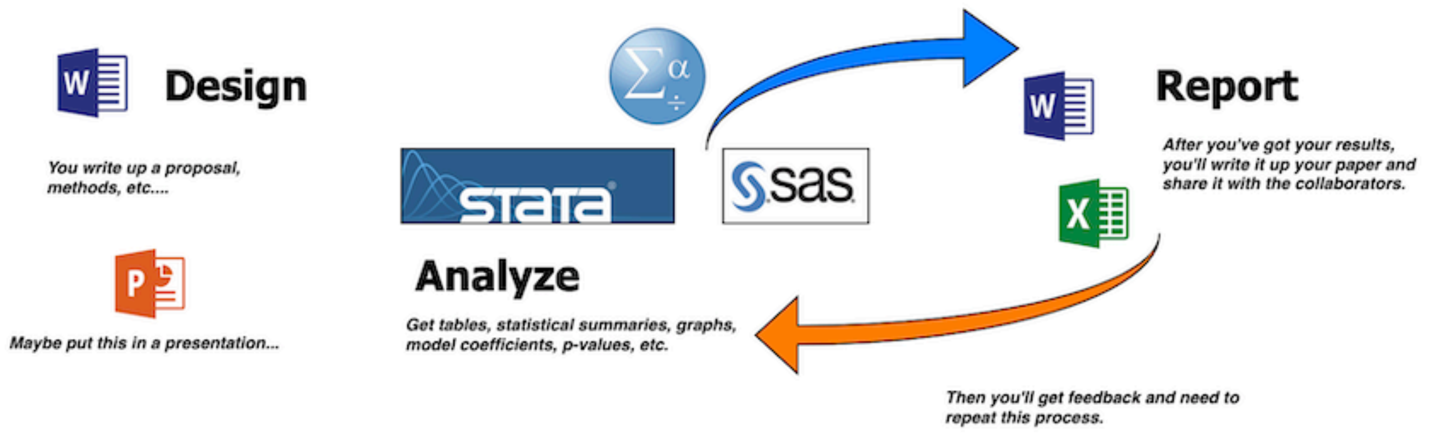
Principle 2: Write code

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, or 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 past 20+ years.

User-centered-designed software includes most of the point-and-click operating systems and applications. These programs give the user the ability to click through a predetermined list of options and procedures using their mouse or track-pad.


Stata/IC 15.1 File Edit View Data Graphics Statistics User Window Help
Summaries, tables, and tests
Linear models and related
Binary outcomes
Ordinal outcomes
Categorical outcomes
Count outcomes
Fractional outcomes
Generalized linear models
Time series
Multivariate time series
Spatial autoregressive models
Longitudinal/panel data
Multilevel mixed-effects models
Survival analysis
Epidemiology and related
Survey data analysis
Multiple imputation
Nonparametric analysis
Multivariate analysis
Exact statistics
Resampling
Power and sample size
Bayesian analysis
Postestimation
Other
Logistic regression
Probit regression
Complementary log-log regression
Exact logistic regression
Skewed logistic regression
Probit model with endogenous covariates
Probit model with sample selection
Heteroskedastic probit regression
Probit with endogeneity, selection, and treatment
GLM for the binomial family
Bivariate probit regression
Seemingly unrelated bivariate probit regression
Panel regression
Multilevel mixed-effects regression
FMM (finite mixture models)
Nonparametric regression
Treatment effects
Bayesian regression
Postestimation
Multilevel logistic regression
Multilevel probit regression
Multilevel complementary log-log regression
Bayesian multilevel logistic regression
Bayesian multilevel probit regression
Bayesian multilevel complementary log-log regression

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


Design
You write up a proposal, methods, etc....
Maybe put this in a presentation...
Analyze
Get tables, statistical summaries, graphs, model coefficients, p-values, etc.
Report
After you've got your results, you'll write it up your paper and share it with the collaborators.
Then you'll get feedback and need to repeat this process.

As you can see from the image, you'll be required to go back through the same elaborate workflow whenever you receive feedback or input to your project. Each time will take 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 outlined in Chapter 1:

1. Develop code in the open
2. Publish data and code open source
3. Post preprints of your 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 "*writing code*" wherever possible. We are aware that 'writing code' 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 using a computer.

Software as a tool, not a solution

We think of data science software as the tools that help us gain a deeper understanding of the world. We don't think of software as an alternative to thinking or expect a software tool to do our thinking for us. We also prefer tools that we can reuse on future projects.

As we stated in chapter 1, the scientific method is a process. Software is a tool to help move that process along faster. Tools that improve our understanding shorten the distance between questions and answers, but doesn't leave out any crucial details.

For this reason, we don't recommend relying too heavily on point-and-click proprietary software applications (SPSS, Stata, SAS, etc.). These GUI's make it hard to keep track of what you've clicked on, the order of you clicked on them in, and can oversimplify or obfuscate what's going on.

What do we mean by 'a workflow'?

A workflow is a set of steps that can be used repeatedly to answer any question you might encounter in your work.

The quote below 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.

> ***Question:*** *"I'm wondering how you, as an educator and statistician, would like to see statistical and data literacy change in general for a general population?"*
>
> ***Answer:*** *"… 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*

As the quote above illustrates, ***how you got an answer is just as relevant as the answer you got***. The tools we provide in this text give you a start-to-finish chain of documentation from question to solutions.

Data science jobs need a particular set of tools, and a workbench to organize these tools. To manage your data science projects, you'll need a workflow that gives you the ability to 1) document your intentions and, 2) write code that can translate your plans into something a computer can execute. These two points bring us to our next topic: plain text. As you'll discover, plain text files are a great way to accomplish these tasks.

Principle 3: Document everything in plain text

In 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 your files, notes, and any pertinent documentation about your project in plain text 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 searchable (`ctrl+F` or `cmd+F` allows us to find keywords or phrases)

We'll also cover why you might want switch over to a plain text editor if you're currently using 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.

We covered the problems with a copy+paste workflow previously, but there are additional reasons to avoid this toolset:

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

If we've sold you on using this flexible and adaptable tool, but you're still wondering what makes a file 'plain text,' we'll cover that next. But first, we need to talk about what *isn't* a plain text file.

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.*

The last bit is the most crucial distinction–**human-readable vs. computer-readable**. In this manual, we'll point out which files are binary and which are plain text. Generally speaking, plain text files can be opened using a text editor or viewed with a command-line tool. Examples of text editors include Atom, Sublime Text, and Notepad++.

Markdown & Rmarkdown

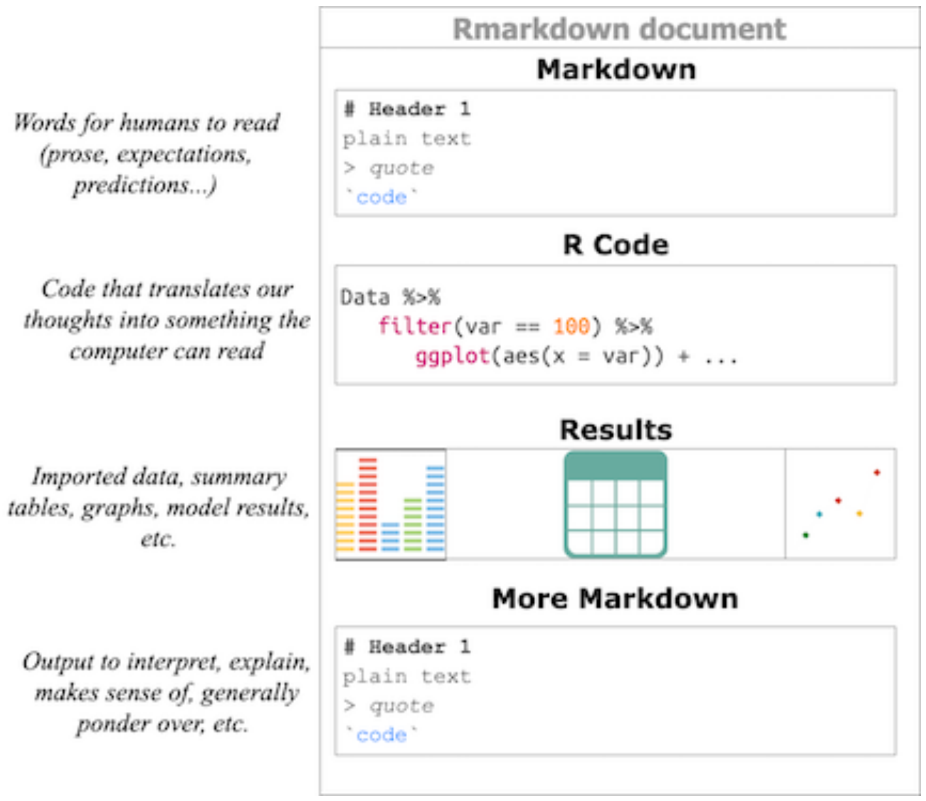
Markdown files (`.md`) are common type of plain text files. Markdown is a ‘lightweight markup language,’ which means it’s easy for humans to read, and computers can convert it to HyperText Markup Language (HTML). Markdown allows for some formatting options to aid with communication (see below)

```
<!-- comments -->

normal text

*italic*

**bold**

> quote

`code`

# h1
## h2
### h3
#### h4
##### h5
###### h6
```

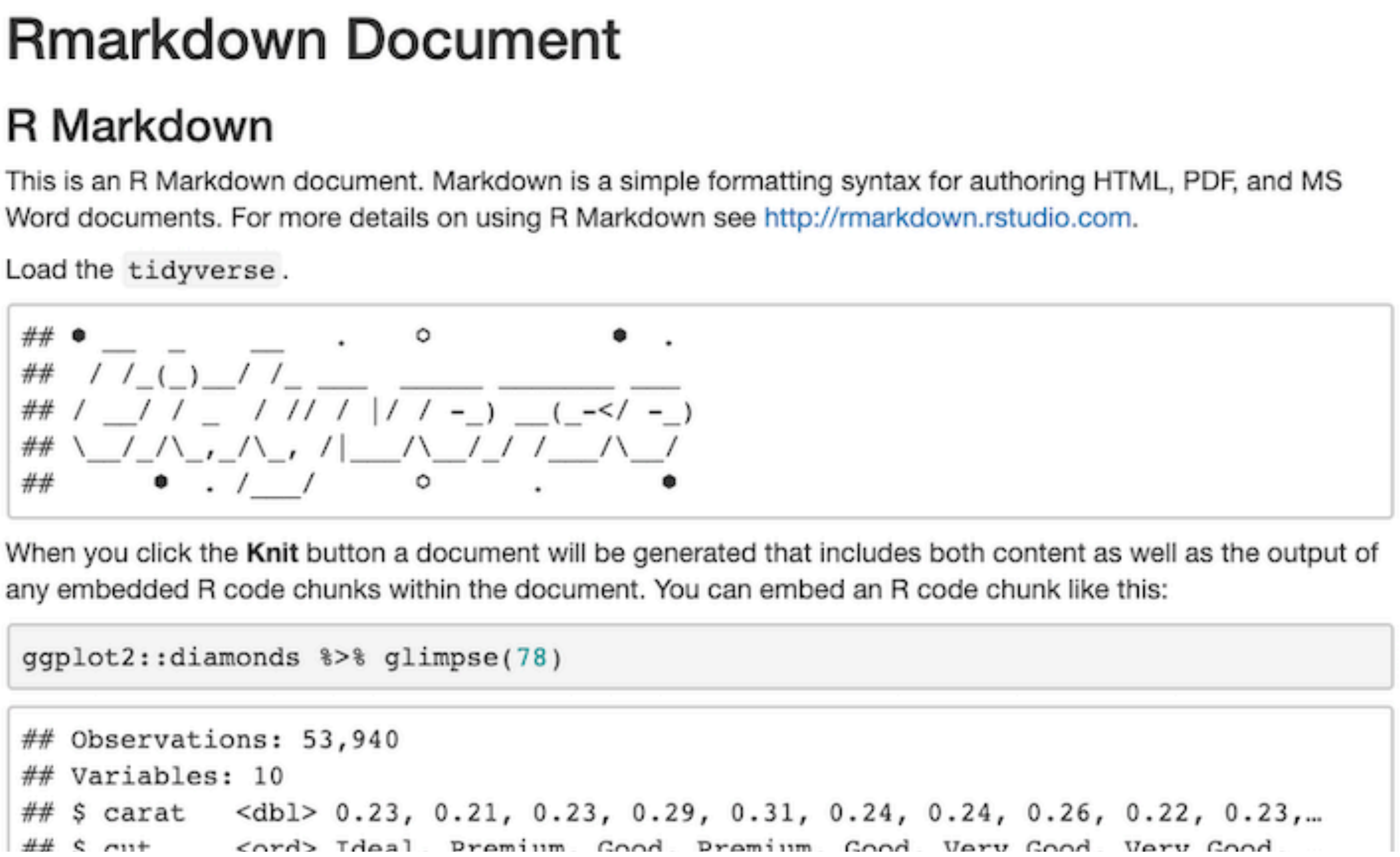
To learn more, see Markdown Syntax Documentation on John Gruber’s site).

We recommend learning markdown before any other programming language because it’s the lingua franca for asking questions. Stackoverflow, RStudio community, Reddit, Github, and many other sites use markdown to post questions and answers. We recommend experimenting with StackEdit, a browser-based markdown editor that gives you the ability to write in markdown and see the syntax rendered as HTML.

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


Rmarkdown document
Markdown
Words for humans to read (prose, expectations, predictions...)
# Header 1
plain text
> quote
`code`
R Code
Code that translates our thoughts into something the computer can read
Data %>%
filter(var == 100) %>%
ggplot(aes(x = var)) + ...
Results
Imported data, summary tables, graphs, model results, etc.
More Markdown
Output to interpret, explain, makes sense of, generally ponder over, etc.
# Header 1
plain text
> quote
`code`

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:

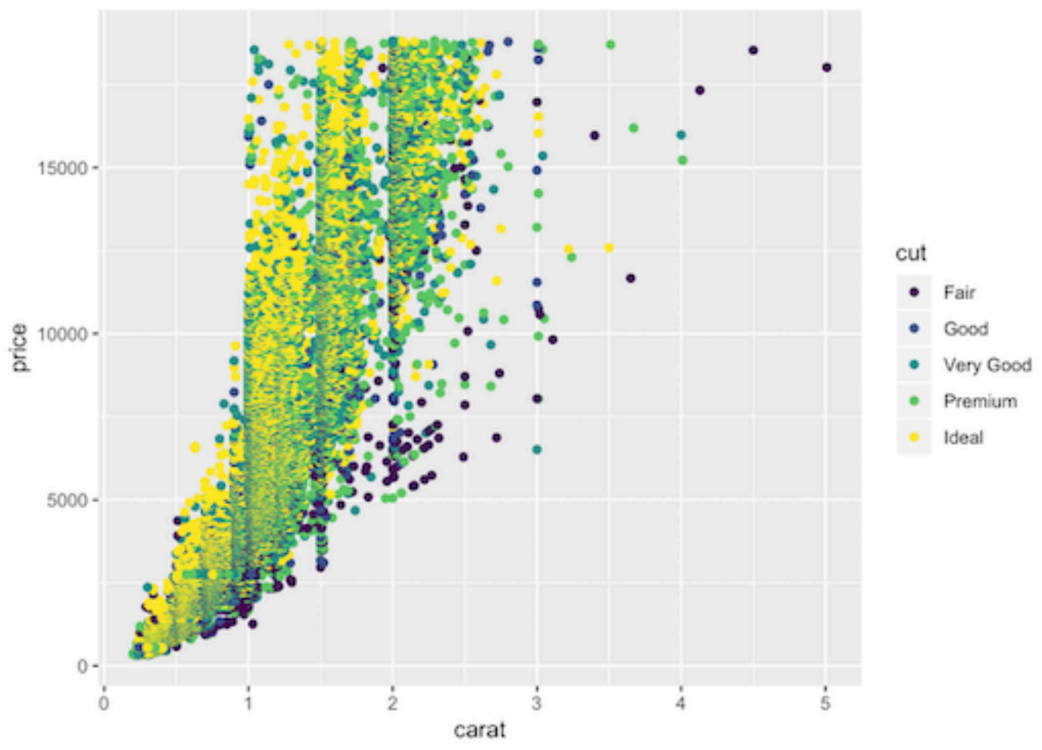
```
ggplot2::diamonds %>% glimpse(78)
```

```
## 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,
## $ 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()
```



Python & Jupyter vs. R & RStudio

Python is a neat language and a great tool to combine with R.

It's also helpful to know a little Python even if you're primarily working in R, because the benefits of being multilingual extend beyond just spoken languages, too.

We recommend R/RStudio because we wrote this book for people who have a data file and specific questions (or general curiosity). Thus, the entry point for our audience into data science is *with data they need to analyze*, and this is what R was made 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.

You can focus on your work

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

We recommend R/RStudio because of the time saved by switching between software applications. For example, when I was in graduate school, I'd have *a minimum of five applications open* to do data analysis. I would be using Word to write, Stata for statistics, Excel to create tables, the browser for internet research, and Adobe Acrobat for reading PDFs. That means I needed to learn five different GUIs, each with their design characteristics.

Each software application cost me valuable neurons whenever 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).

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, dialog 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):
  1) Why is it so hard to do my work? The challenge of attention residue when switching between work tasks
  2) Information, Attention, and Decision Making
  3) Causes, effects, and practicalities of everyday multitasking
- See Baumer et al. for an in-depth summary of why you should abandon a copy + paste workflow