

# Chapter 0

## Accessing R

Here we introduce **Cran R** (R Core Team, 2016) or, simply, **R**. We will utilize **R** to complete the statistical analyses we explore throughout the text. The reason we've chosen to use **R** mirrors the themes and examples we explore in the text – *we have enough faith in our students to do more*.

For decades, we have taught mathematics and statistics that can be done on a legal pad. Slowly, as technology becomes more prevalent, textbooks and classrooms have inched forward using the Texas Instruments line of calculators, Microsoft Excel, or online applets. All of these solutions allow us to answer more interesting questions, which empowered us to ask better questions. It is in this vein that we use **R** – to continue inching forward.

There are several programming languages we could have chosen, but there are many reasons we have chosen **R**. First, **R** is available for free download, which we think is reason enough. More importantly, however, **R** is designed specifically for statistical analysis and is perhaps the fastest way to disseminate newly developed statistical techniques that make the most cutting edge statistical analysis possible.

Using **R** allows us to explore real statistical analyses like political prediction and scientific research. This tool allows us to manipulate, analyze and visualize data in a way that mimics what a quantitative researcher would do. Exploring and practicing these techniques not only provide valuable knowledge, it leads us toward consuming statistical statements we encounter each day more responsibly.

While this isn't a computer science text, and we don't explain how to become a computer programmer, we will guide students to become *consumers* of technology. Students will explore how to use, edit, and find errors in **R** codes provided in the programming language's vast documentation. There are some important topics that this text doesn't cover. Instead, we focus on code that is essential to getting up and running quickly, and fitting the models discussed. We hope that this creates enough of a foundation that students can grow with **R**, by equipping students with the ability to pick up new skills and knowledge through the documentation.

### 0.1 Downloading R's Programming Files

To use **R** on our computers we need to download the program files from their website, <https://cran.r-project.org/>. Downloads of **R** are available for Windows, Mac, and Linux, but the program files are different for each operating system, so ensure you choose the correct download link on the

homepage.

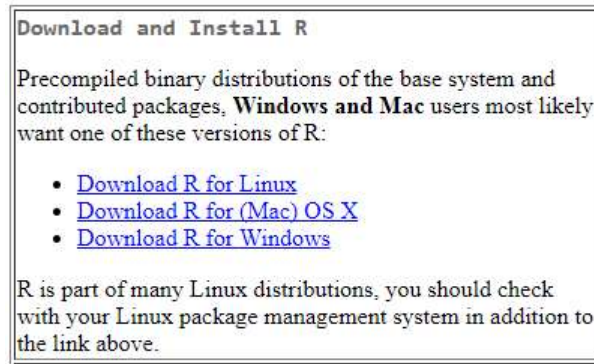


Figure 0.1.1: Download options by operating system on R's homepage.

If you're using Windows, you'll need to click "base" to go to the final download page, seen in Figure 0.1.3. If you're using Mac OSX, you'll just need to scroll down and click on the most recent version of R, seen in Figure 0.1.4. Note that these images were taken when R 3.4.3 was the most recent version available, follow the same instructions for whatever "#.#.#" version R has currently released; updates happen quite often!



Figure 0.1.2: Windows – click “base” to go to the download page.



Figure 0.1.3: Windows – click on the most recent “Download R #.#.# for Windows” to download.

Files:	
<a href="#">R-3.4.3.pkg</a>	<b>R 3.4.3</b> binary for OS X 10.11 (El Capitan) and higher, signed package. Contains R 3.4.3 framework, R.app GUI 1.70 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 5.2. The latter two components are optional and can be omitted when choosing "custom install", they are only needed if you want to use the tcltk R package or build package documentation from sources.
MD5-hash: d51d0869f3cbe0d782eede113897393a SHA- hash: d2694cd4b8d5539deab0e68a73bd79eb715fe62f (ca. 74MB)	

Figure 0.1.4: Mac – click on the most recent “R-#.#.#.pkg” to download.

## 0.2 Downloading RStudio

After following the steps above, you’ve installed R’s program files and all of its wonderful functionality. A tool that is very nice as a programmer of any language, and something that allows Windows, Mac and Linux operating systems a similar user experience, is an integrated development environment (IDE). RStudio (RStudio Team, 2016) is a nice IDE for R and we recommend using it as we explore solutions using R.

RStudio makes R easier to use as it combines a code editor, the console for running R’s functionality, plotting windows and workspace information in one window. We’ve included their simplification in Figure 0.2.5 so you might consider the perks of such a set up – you have all the information in view at all times. RStudio also has helpful tools such as syntax highlighting, code suggestions as well as integrated R help and documentation. Again, we note and love that it, like R, is also free.



Figure 0.2.5: RStudio’s simplified rendition of its IDE.

You can download RStudio at <https://www.rstudio.com/>. Click “Download RStudio” on the home-

page (Figure 0.2.6), scroll down to the download options on the resulting page (Figure 0.2.7) and choose the download option corresponding to your operating system.

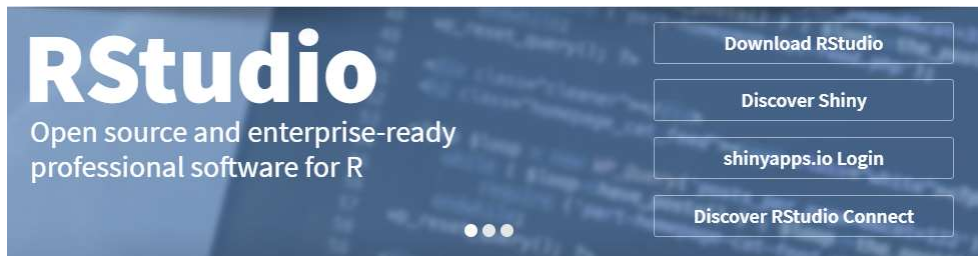



Figure 0.2.6: RStudio– click “Download RStudio” on the homepage.

### Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.1.414 - Windows Vista/7/8/10	85.8 MB	2018-01-17	7687a26828002b0d1e14e6a4556d64a4
RStudio 1.1.414 - Mac OS X 10.6+ (64-bit)	74.5 MB	2018-01-17	08a74c18ddade04524a7f06b1e82d61e
RStudio 1.1.414 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	89.2 MB	2018-01-17	44d16f29fcdbe82b2652adbef7cf53d
RStudio 1.1.414 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	97.4 MB	2018-01-17	759fd531bfbfc47a6f211953750de9e4
RStudio 1.1.414 - Ubuntu 16.04+/Debian 9+ (64-bit)	64.4 MB	2018-01-17	9376803d8203b47b94522d4cfea37254
RStudio 1.1.414 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.1 MB	2018-01-17	87cce60d3458a731a2ac31c846855f4c
RStudio 1.1.414 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	90.6 MB	2018-01-17	adc1b3e43d2af860c4bb3d33b524bd42

Figure 0.2.7: RStudio–scroll down to the download options on the resulting page and choose the download option corresponding to your operating system.

## 0.3 What It Should Look Like

When you’ve completed the installation of both R and RStudio your screen should look something like Figure 2.1.1. If you don’t see a pane for “Untitled1”, open a new R script. You can open a new R script by clicking  in the upper left corner then “R Script”, pressing “File” then “New File”, or pressing “CTRL+Shift+n,” on Windows or “Command + Shift + n” on Mac OSX.

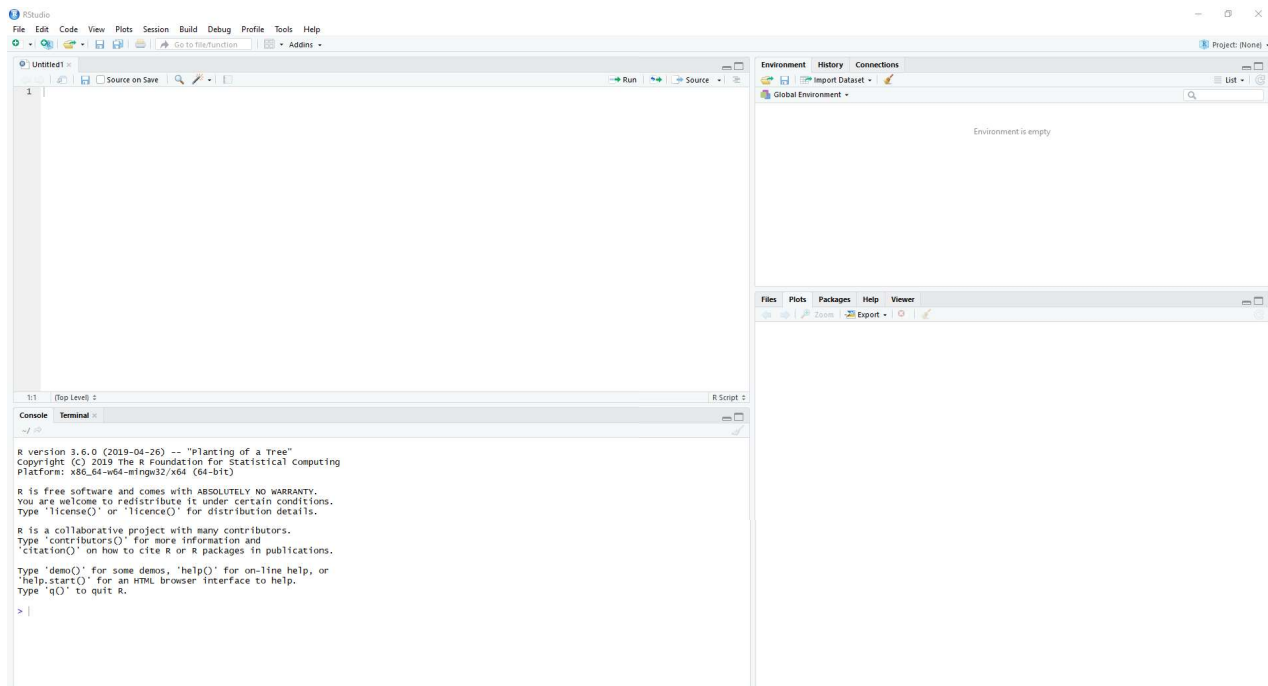



Figure 0.3.8: RStudio– the first screen.

The upper left portion of the screen is the code editor; this is where code is written and edited. We will want to save this, with a meaningful name, so that we might revisit what we’ve done. For example, when working on the homework problems for this chapter you may want to save the code as “HW0” and if you’re submitting it to your professor you may want to save the code as “lastname-HW0” so that they might easily identify the file too. This may seem like a small detail but it’s immensely helpful – there will be many homework assignments and making it easier to pull from previous work might be really helpful, particularly if it’s easy to find. This extends beyond schooling – quick recall of past work is helpful for quantitative researchers and consultants as well.

The lower left portion of the screen is the terminal, which is where code is run. The ‘>’ in the terminal indicates that R is waiting for your commands and a ‘+’ means that R is waiting for you to *finish* a command you have already started. This is a nice feature when you intend to continue a line of thought, but you’ll often see it when you’ve forgotten a closing parenthesis in a calculation. A line starting with a number enclosed with brackets, e.g. [1], indicates that what follows is an answer to a question you’ve asked.

It’s important to know which state R is in so that you can communicate with it effectively. You can type code into the terminal directly, copy and paste code from the code editor above, or run highlighted code in the code editor by clicking  above the code editor or pressing “CTRL+Enter” on Windows, or “Command + Enter” on Mac OSX.

The upper right portion of the screen shows the environment. The environment will show the objects we’ve defined in the current R session, something like assigning  $x$  the value 6. We won’t refer to this very often but sometimes it’s helpful to see what we have and haven’t defined yet, especially when we want to ask for it by name.

The lower right portion of the screen is a viewer. When we ask R to produce publication-quality graphs, they will pop up in this window along with several options. We can view, copy, and save

the image in a variety of sizes and formats. This is helpful as graphs are very important in any statistical analysis.

Often times users want to minimize the right part of the screen to use more of their screen real estate for text editing. This however, may cause an error when R is asked to create a graph. The error reads, “Error in plot.new() : figure margins too large” and indicates that the viewing pane needs to be increased in size so that R has space to draw the image.

## 0.4 Basic Operations in R

The question and answer style of the R programming language is surprisingly similar to that of a Texas Instruments calculator. A point of contention with R among some skeptical students, however, is that moving from a comfortable technology to a new technology is sometimes uncomfortable; we like what we know and what we’re familiar with.

Perhaps the most important benefit from switching to R is that we can write, save and edit everything we do. Once we’ve found a way to solve a problem, we can save our code and revisit it when a similar problem arises. Something that helps us remember how the code we saved works and what it does is writing it down; this is called commenting your code. In R and most other programming languages, commenting your code is paramount to any solution as you revisit and share them with your peers, professors, or collaborators. Commenting code in R is done by use of the hashtag or pound sign, #.

### Example 0.1. Commenting in R.

```
# This is a comment -- nothing after a # on the same line will be run.  
# If I go to the next line and I’m still commenting, I need to put another #  
# at the front of that line.
```

Algebraic calculations are completed in R similarly to those done on any calculator. The operations  $+$ ,  $-$ ,  $\div$ , and  $\times$  are completed in R using  $+$ ,  $-$ ,  $/$ , and  $*$ . R follows the order of operations automatically, like a graphing calculator might, but it can’t read our minds so we must be careful to use parentheses when necessary. Recall PEMDAS, “Please Excuse My Dear Aunt Sally,” or however you remembered the order of operations.

### Example 0.2. Algebraic Calculations in R.

```
#Adding in R is easy  
4+5 #Ask to add using +  
  
#Subtracting in R is easy, too  
10-15 #Ask to subtract using -  
  
#Multiplying in R is easy, too  
3*5 #Ask to multiply using *  
  
#Dividing in R is easy, too  
27/3 #Ask to divide using /
```

```
#Square Roots in R are easy, too
sqrt(16) #Ask for a sqrt with sqrt()

#We can do several calculations at once but we want to ensure that we're
#careful with order of operations.
(3+5)*3/3^2 #Just squares the 3 according to PEMDAS
((3+5)*3/3)^2 #Squares the result of the expression in parenthesis
#These give very different answers showing this is important!
```

When you run this code in R, you'll see the code and the output in the console window. From now on we'll just provide the code in this format to avoid printing it twice but we stress the importance of coding in the coding editor so that you might easily edit, comment, save and revisit the code that you use.

```
> #Adding in R is easy
> 4+5 #Ask to add using +
[1] 9
> #Subtracting in R is easy, too
> 10-15 #Ask to subtract using -
[1] -5
> #Multiplying in R is easy, too
> 3*5 #Ask to multiply using *
[1] 15
> #Dividing in R is easy, too
> 27/3 #Ask to divide using /
[1] 9
> #Square Roots in R are easy, too
> sqrt(16) #Ask for a sqrt with sqrt()
[1] 4
> #We can do several calculations at once but we want to ensure that we're
> #careful with order of operations.
> (3+5)*3/3^2 #Just squares the 3 according to PEMDAS
[1] 2.666667
> ((3+5)*3/3)^2 #Squares the result of the expression in parenthesis
[1] 64
> #These give very different answers showing this is important!
```

**Example 0.3. Spotting an Error in R.** Let's consider the algebraic calculation in Example 0.2, but this time there's a mistake.

```
> ((3+5)*3/3^2 #Squares the result of the expression in parenthesis
+
```

We've run the code but R is still waiting for us to finish the command, as the `+` in the console indicates. When this happens, we need to diagnose where we didn't finish, after all R is waiting. This usually occurs when we have more opening than closing parentheses; we see this is the case as the second closing parenthesis is missing. To retry, hit "ESC" on your keyboard, to let R know you made a mistake, and run the corrected code.



**Example 0.4. Spotting an Error in R.** Let's consider the algebraic calculation in Example 0.2, but this time there's a different mistake.

```
> ((3+5)3/3)^2 #Squares the result of the expression in parenthesis
Error: unexpected numeric constant in "((3+5)3"
```

We've run the code but R doesn't return a number, and instead we get an error message in red. When this happens, we need to assess our code in the context of the error R has thrown. Admittedly, the printed errors are sometimes unhelpful, but a majority of the time they at least points us in the right direction. Here, there's an issue with the way we've written the numbers  $((3 + 5)3$ .

This is common notation on paper, but a mistake in R. We might know that writing  $(3 + 5)3$  indicates that we want to multiply those two values but R does not, using `*` is necessary. R is going to do a lot of work for us as we explore using a variety of statistical techniques, the least we can do is ask nicely and in a way R understands.

When we come across an error, it is much more likely that it's how we asked it than R itself. To quote one of our favorite Mathematics professors "don't ever think you're so unique that you're the only one that has ever seen a problem. Google it." If the error doesn't pop right out at you, search the error R printed using Google or your favorite search engine, it's likely many others have seen that error code before and that someone has solved it.

### 0.4.1 Summary of R commands

Table 0.4.1 summarizes the operators used in this section.

Operator	Functionality	Example From Notes
#	comment	# This is not run as code
+	add	4+5
-	subtract	10-15
*	multiply	3*5
/	divide	27/3
^	power	4^2
sqrt( <i>x</i> )	square root of <i>x</i>	sqrt(16)

Table 0.4.1: A list of basic operators in R.

## 0.5 Downloading and Using Packages

R is designed specifically for statistical analysis, and is perhaps the fastest way to disseminate newly developed statistical techniques which makes the most cutting edge statistical analysis possible. This allows our learned skill to grow as the open-source software extends its functionality.



We will often use libraries containing such techniques made available to us through R and we will spend much time thanking the authors of such libraries for all the time and paper we'll save. Below we describe the ways we can interact with these packages in R from installation to citation.

1. **Install the packages:** `install.packages("package name")`
  - This only needs to be run on a computer once
2. **Load the package for use:** `library("package name")`
  - this needs to be run in every new R session we want to use the package
3. **Ask for help with a package or function:** `help("name")`
4. **Ask for help with a package or function:** `?name`
5. **Ask if a vignette is available for a package or function:** `vignette("name")`
6. **Ask if a demo is available for a package or function:** `demo("name")`
7. **Cite a package:** `citation("package name")`

**Example 0.5.** For example we can run a demo from the “plotly” package (Sievert et al., 2017) in R that allows us to create interactive graphs. To see this we will also need to load the “tourr” package (Wickham et al., 2011) that contains a dataset from Lubischew (1962), containing the physical measurements on three species of flea beetles – *concinna*, *heptapotamica*, and *heikertingeri*. The dataset includes the species of the flea beetle, and six physical measurements from each of the 74 flea beetles observed.

- `tars1` – width of the first joint of the first tarsus in microns
- `tars2` – width of the second joint of the first tarsus in microns
- `head` – the maximal width of the head between the external edges of the eyes in 0.01 mm
- `ade1` – the maximal width of the aedeagus in the fore-part in microns
- `ade2` – the front angle of the aedeagus ( 1 unit = 7.5 degrees)
- `ade3` – the aedeagus width from the side in microns
- `species` – which species is being examined

**Research Question:** Are these measurements a good basis for discriminating between species? Can we clearly separate all three species in one figure? If we can, what does that say about developing objective criteria for a system of organisms – can we decide the species based on such physical measurements?

```
#Download and installs flea data
> install.packages("tourr")
#Loads flea data
> library("tourr")
#Download and installs plotting features
> install.packages("plotly")
```

```
#Loads plotting features
> library("plotly")
#Demonstrates the plotting ability of R further motivating us to learn it!
> demo("animation-tour-basic",package = "plotly")
```

The demonstration yields an animated plot that shows us the data with lines that help us discriminate between the species of flea based on the six physical measurements. Watching this demonstration it should be visually clear that the three species are separated in the plot. This indicates that we can, with these measurements, define a criteria that helps us decide what species of flea beetle we might be looking at.

### 0.5.1 Summary of R commands

Table 0.5.2 summarizes the operators used in this section.

Operator	Functionality
<code>install.packages(package.name)</code>	downloads the specified package
<code>library(package.name)</code>	loads the specified package
<code>help(...)</code>	provides the documentation for a function or package
<code>vignette(...)</code>	loads a text-based walk-through the functionality of a function or package, if available
<code>demo(...)</code>	loads a user-friendly interface for running code of a function or package, if available
<code>?...</code>	shorthand for <code>help()</code>
<code>citation(package.name)</code>	provides a citation for the package

Table 0.5.2: A list of commands for working with packages in R.

## 0.6 Summary

Using technology allows us to experience answering complicated questions without an assumed mastery of algebra or calculus. This learning experience allows us to ask and answer better and more meaningful questions while learning a coveted skill.

These types of monumental classroom changes face resistance in terms of buy-in from both faculty and students. Making monumental changes to a course causes both faculty and students to experience shifts in the classroom and possible growing pains. As teachers and learners, we should accept and manage the liability of pushing for change as we consider our approaches to teaching and learning whether or not the status quo is compliant with university policy or our comfort level. Support, development and coaching are necessary for many to make this jump and such programs are built by people that pioneer those changes.

We can already see the importance of technology in a world where data-driven solutions are becoming more prevalent. Using a statistical programming language like R allow us to import data

in seconds, instead of doing it manually as we saw in Example 2.3. We also saw a demonstration of graphing in R, in Example 0.5, which created an animation of over thirty graphs in seconds that are of publication quality.

We will continue to introduce meaningful strategies and techniques for asking and answering important questions about the world using technology. Technology like R is paramount to the success of a course centered on numeracy as our scientific literacy depends on this, particularly as mathematics and statistics become the grammar of science. As we explore real-world data through political polls and research papers we need the tools of the trade to do so, moving past the legal pad and contrived classroom examples.

Numeracy has become a moving target; the pace of scientific advancement leads us to asking what is science today? Let's equip ourselves with a tool that grows as the answer to this question changes.