Chapter 1

Preface

The purpose of this book is to help you learn R from the ground-up.

1.1 Where did this book come from?

Let me make something very, very clear…

I did not write this book.

This whole story started in the Summer of 2015. I was taking a late night swim on the Bodensee in Konstanz and saw a rusty object sticking out of the water. Upon digging it out, I realized it was an ancient usb-stick with the word YaRrr inscribed on the side. Intrigued, I brought it home and plugged it into my laptop. Inside the stick, I found a single pdf file written entirely in pirate-speak. After watching several pirate movies, I learned enough pirate-speak to begin translating the text to English. Sure enough, the book turned out to be an introduction to R called The Pirate's Guide to R.

This book clearly has both massive historical and pedagogical significance. Most importantly, it turns out that pirates were programming in R well before the earliest known advent of computers. Of slightly less significance is that the book has turned out to be a surprisingly up-to-date and approachable introductory text to R. For both of these reasons, I felt it was my duty to share the book with the world.

If you or spot any typos or errors, or have any recommendations for future versions of the book, please write me at YaRrr.Book@gmail.com or tweet me @YaRrrBook.

1.2 Who is this book for?

While this book was originally written for pirates, I think that anyone who wants to learn R can benefit from this book. If you haven't had an introductory course in statistics, some of the later statistical concepts may be difficult, but I'll try my best to add brief descriptions of new topics when necessary. Likewise, if R is your first programming language, you'll likely find the first few chapters quite challenging as you learn the basics of programming. However, if R is your first programming language, that's totally fine as what you learn here will help you in learning other languages as well (if you choose to). Finally, while the techniques in this book apply to most data analysis problems, because my background is in experimental psychology I will cater the course to solving analysis problems commonly faced in psychological research.

What this book is

This book is meant to introduce you to the basic analytical tools in R, from basic coding and analyses, to data wrangling, plotting, and statistical inference.

What this book is not

This book does not cover any one topic in extensive detail. If you are interested in conducting analyses or creating plots not covered in the book, I'm sure you'll find the answer with a quick Google search!

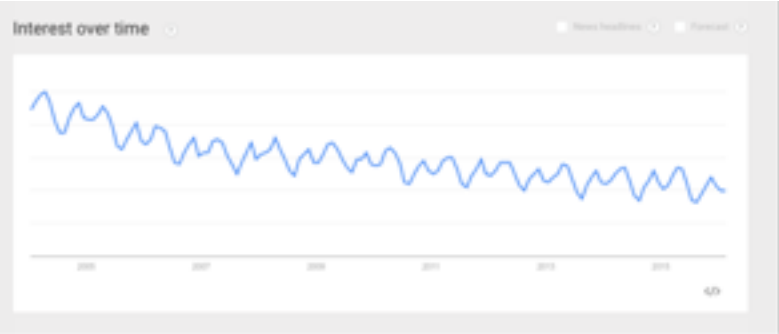
1.3 Why is R so great?

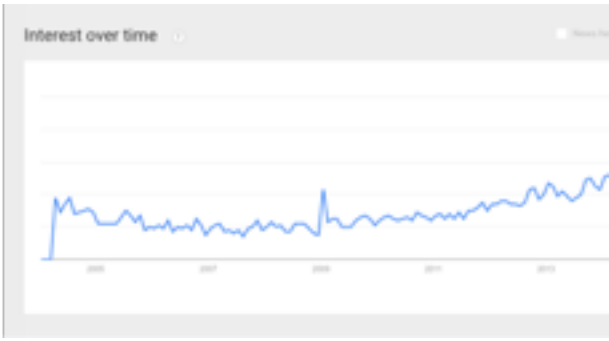
As you've already gotten this book, you probably already have some idea why R is so great. However, in order to help prevent you from giving up the first time you run into a programming wall, let me give you a few more reasons:

1. R is 100% free and as a result, has a huge support community. Unlike SPSS, Matlab, Excel and JMP, R is, and always will be completely free. This doesn't just help your wallet - it means that a huge community of R programmers will constantly develop an distribute new R functionality and packages at a speed that leaves all those other packages in the dust! Unlike Fight Club, the first rule of R is "Do

talk about R!" The size of the R programming community is staggering. If you ever have a question about how to implement something in R, a quick Poogle[1] search will lead you to your answer virtually every single time.

2. R is the present, and future of statistical programming. To illustrate this, look at the following three figures. These are Google trend searches for three terms: R Programming, Matlab, and SPSS. Try and guess which one is which.





3. R is incredibly versatile. You can use R to do everything from calculating simple summary statistics, to performing complex simulations to creating gorgeous plots like the chord diagram on the right. If you can imagine an analytical task, you can almost certainly implement it in R.

4. Using RStudio, a program to help you write R code, You can easily and seamlessly combine R code, analyses, plots, and written text into elegant documents all in one place using Sweave (R and Latex) or RMarkdown. In fact, I translated this entire book (the text, formatting, plots, code...yes, everything) in RStudio using Sweave. With RStudio and Sweave, instead of trying to manage two or three programs, say Excel, Word and (sigh) SPSS, where you find yourself spending half your time copying, pasting and formatting data, images and test, you can do everything in one place so nothing gets misread, mistyped, or forgotten.

```
circlize::chordDiagram(matrix(sample(10),
                              nrow = 2, ncol = 5))
```

5. Analyses conducted in R are transparent, easily shareable, and reproducible. If you ask an SPSS user how they conducted a specific analyses, they will either A) Not remember, B) Try (nervously) to construct an analysis procedure on the spot that makes sense - which may or may not correspond to what they actually did months or years ago, or C) Ask you what you are doing in their house. I used to primarily use SPSS, so I speak from experience on this. If you ask an R user (who uses good programming techniques!) how they conducted an analysis, they should always be able to show you the exact code they used. Of course, this doesn't mean that they used the appropriate analysis or interpreted it correctly, but with all the original code, any problems should be completely transparent!

6. And most importantly of all, R is the programming language of choice for pirates.

1.4 Why R is like a relationship...

Yes, R is very much like a relationship. Like relationships, there are two major truths to R programming:

1. There is nothing more *frustrating* than when your code does *not* work
2. There is nothing more *satisfying* than when your code *does* work!

Anything worth doing, from losing weight to getting a degree, takes time. Learning R is no different. Especially if this is your first experience programming, you are going to experience a *lot* of headaches when

[1] I am in the process of creating Poogle - Google for Pirates. Kickstarter page coming soon...

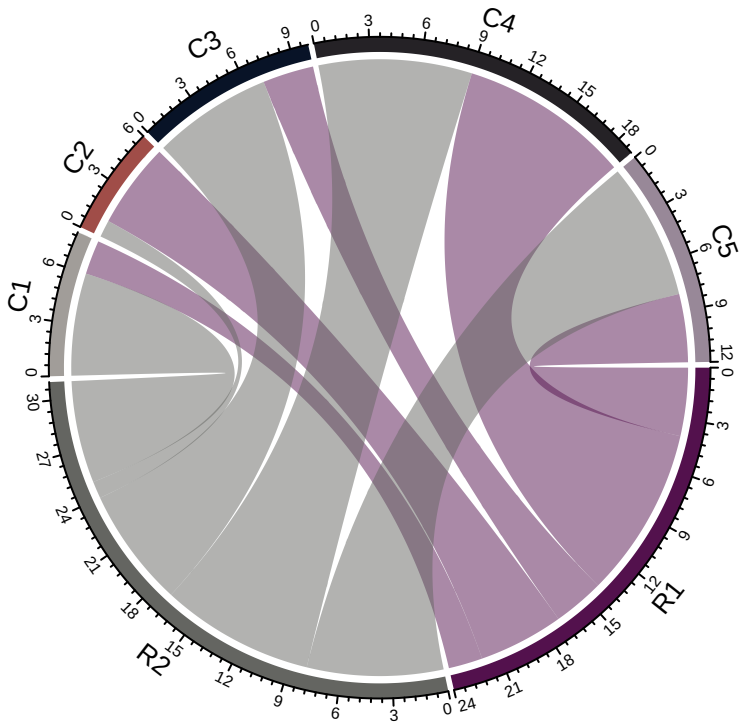


Figure 1.1: A super cool chord diagram from the circlize package

you get started. You will run into error after error and pound your fists against the table screaming: "WHY ISN'T MY CODE WORKING?!?!? There must be something wrong with this stupid software!!!" You will spend hours trying to find a bug in your code, only to find that - frustratingly enough, you had had an extra space or missed a comma somewhere. You'll then wonder why you ever decided to learn R when (::sigh::) SPSS was so "nice and easy."

Fun Fact! SPSS stands for "Shitty Piece of Shitty Shit". True story.

This is perfectly normal! Don't get discouraged and DON'T GO BACK TO SPSS! That would be quitting on exercise altogether because you had a tough workout.

Trust me, as you gain more programming experience, you'll experience fewer and fewer bugs (though they'll never go away completely). Once you get over the initial barriers, you'll find yourself conducting analyses much, much faster than you ever did before.

1.5 R resources

1.5.1 R Cheatsheets

Over the course of this book, you will be learning *lots* of new functions. Wouldn't it be nice if someone created a Cheatsheet / Dictionary of many common R functions? Yes it would, and thankfully several friendly R programmers have done just that. Below is a table of some of them that I recommend. I highly encourage you to print these out and start highlighting functions as you learn them!

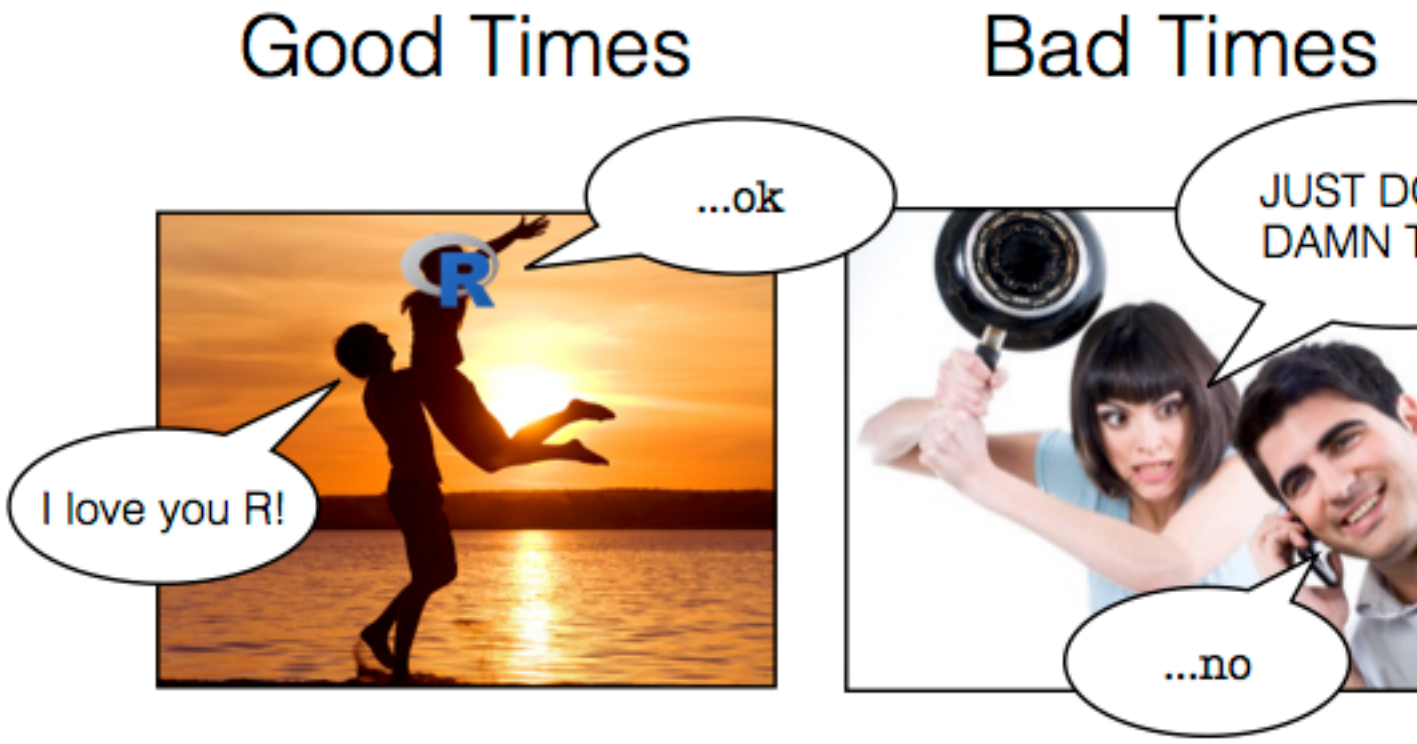


Figure 1.2: Yep, R will become both your best friend and your worst nightmare. The bad times will make the good times oh so much sweeter.

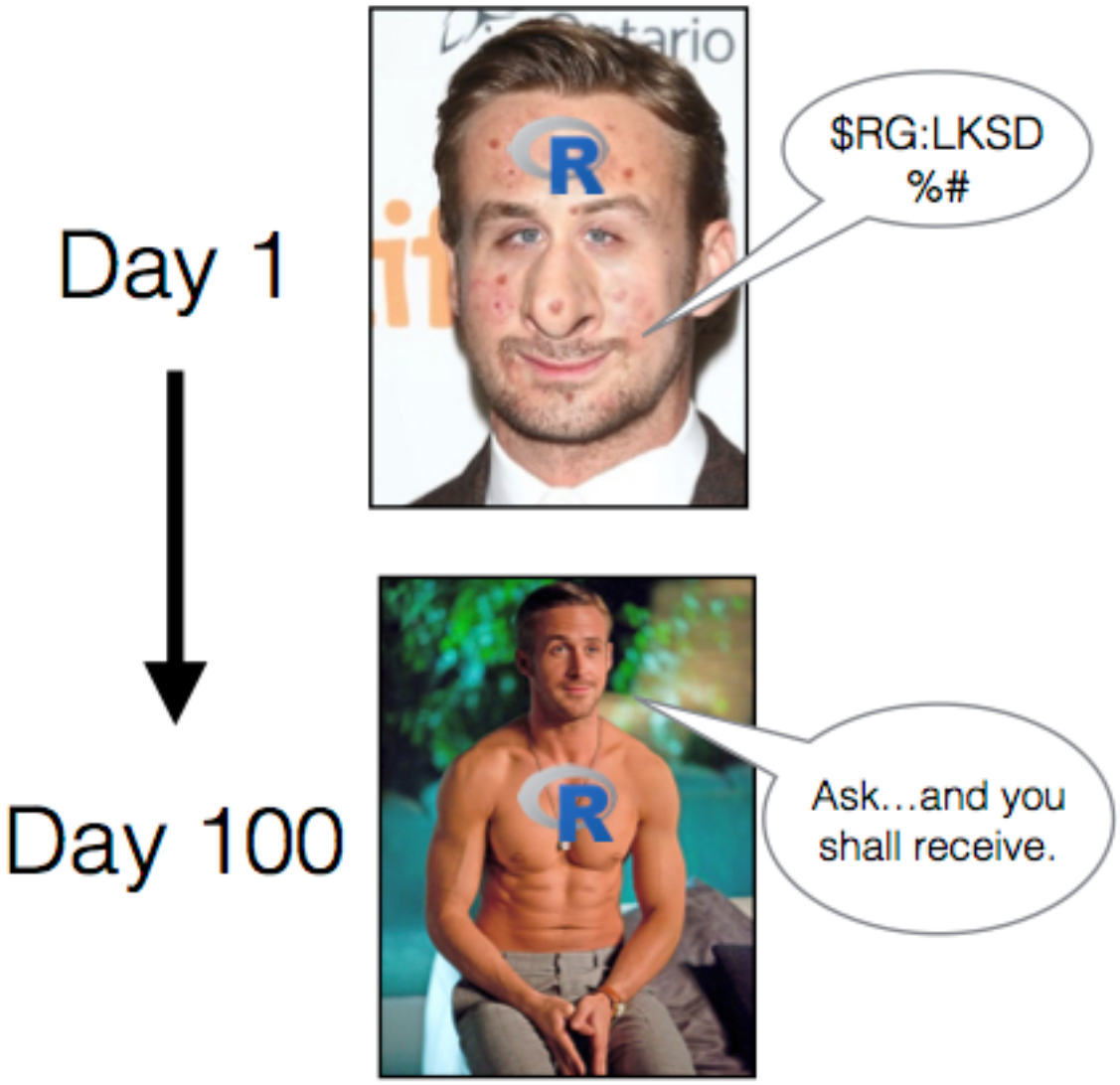
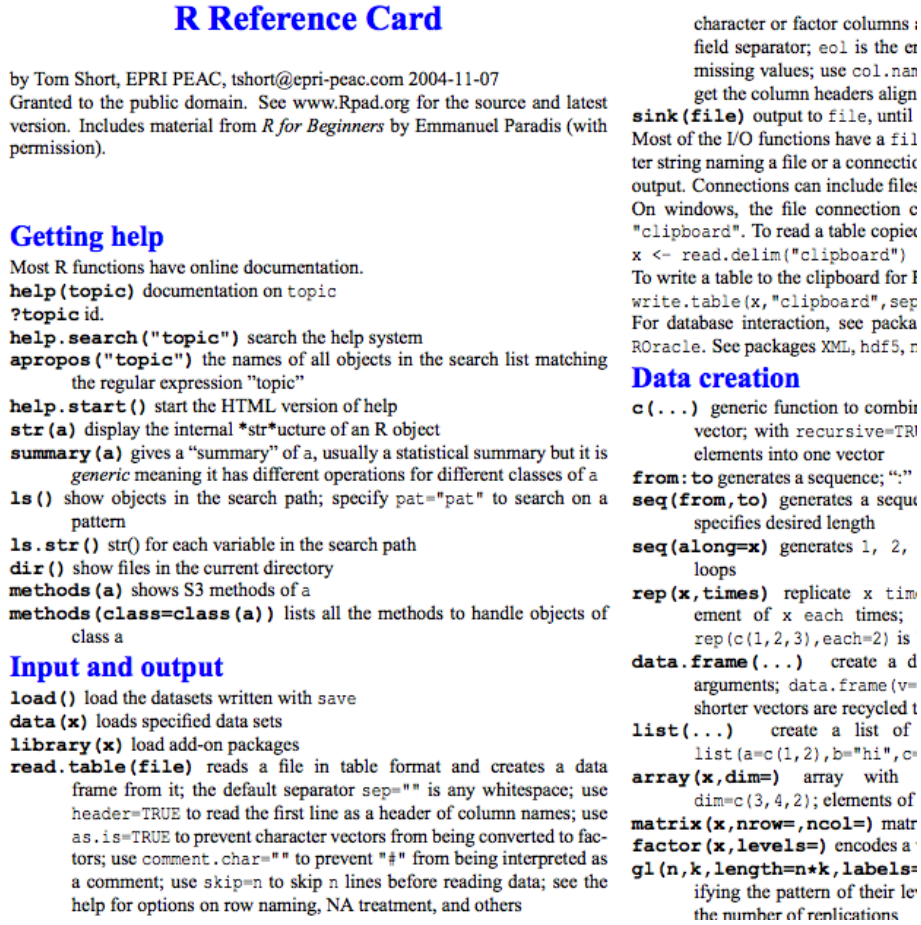


Figure 1.3: When you first meet R, it will look so fugly that you'll wonder if this is all some kind of sick joke. But trust me, once you learn how to talk to it, and clean it up a bit, all your friends will be crazy jealous.



R Reference Card

by Tom Short, EPRI PEAC, tshort@epri-peac.com 2004-11-07
Granted to the public domain. See www.Rpad.org for the source and latest version. Includes material from *R for Beginners* by Emmanuel Paradis (with permission).

Getting help

Most R functions have online documentation.
help(topic) documentation on topic
?topic id.
help.search("topic") search the help system
apropos("topic") the names of all objects in the search list matching the regular expression "topic"
help.start() start the HTML version of help
str(a) display the internal *str*ucture of an R object
summary(a) gives a "summary" of a, usually a statistical summary but it is *generic* meaning it has different operations for different classes of a
ls() show objects in the search path; specify pat="pat" to search on a pattern
ls.str() str() for each variable in the search path
dir() show files in the current directory
methods(a) shows S3 methods of a
methods(class=class(a)) lists all the methods to handle objects of class a

Input and output

load() load the datasets written with save
data(x) loads specified data sets
library(x) load add-on packages
read.table(file) reads a file in table format and creates a data frame from it; the default separator sep="" is any whitespace; use header=TRUE to read the first line as a header of column names; use as.is=TRUE to prevent character vectors from being converted to factors; use comment.char="" to prevent "#" from being interpreted as a comment; use skip=n to skip n lines before reading data; see the help for options on row naming, NA treatment, and others
character or factor columns
field separator; eol is the
missing values; use col.nam
get the column headers align
sink(file) output to file, until
Most of the I/O functions have a fil
ter string naming a file or a connecti
output. Connections can include file
On windows, the file connection c
"clipboard". To read a table copie
x <- read.delim("clipboard")
To write a table to the clipboard for
write.table(x,"clipboard",sep
For database interaction, see packa
ROracle. See packages XML, hdf5,

Data creation

c(...) generic function to combi
vector; with recursive=TR
elements into one vector
from:to generates a sequence; ":"
seq(from,to) generates a sequ
specifies desired length
seq(along=x) generates 1, 2,
loops
rep(x,times) replicate x tim
ement of x each times;
rep(c(1,2,3),each=2) is
data.frame(...) create a d
arguments; data.frame(v=
shorter vectors are recycled t
list(...) create a list of
list(a=c(1,2),b="hi",c=
array(x,dim=) array with
dim=c(3,4,2); elements of
matrix(x,nrow=,ncol=) matr
factor(x,levels=) encodes a
gl(n,k,length=n*k,labels=
ifying the pattern of their le
the number of replications

Figure 1.4: The R reference card written by Tom Short is absolutely indispensable!

CheatSheet	Link
R Basics by Tom Short	https://cran.r-project.org/doc/contrib/Short-refcard.pdf
R Basics by Mhairi McNeill	http://github.com/rstudio/cheatsheets/raw/master/source/pdfs/base-r.pdf
Advanced R by Arianne Colton and Sean Chen	hhttps://www.rstudio.com/wp-content/uploads/2016/02/advancedR.pdf
Plotting	https://www.rstudio.com/wp-content/uploads/2016/10/how-big-is-your-graph.pdf

1.5.2 Getting R help and inspiration online

Here are some great resources for R help and inspiration:

Site	Description
www.google.com	If you haven't heard of it, Google is this amazing site that gives you access to all R knowledge that has ever existed. Just ask it an R question and 99.9% of the time it will give you the answer!
www.r-bloggers.com	R bloggers is my go-to place to discover the latest and greatest with R.
blog.revolutionanalytics.com	Revolution analytics always has great R related material.
www.kaggle.com	Kaggle is a really cool website that posts data analysis challenges that anyone can try to solve. It also contains a wide range of real-world datasets and tutorials.

1.5.3 Other R books

There are many, many excellent (non-pirate) books on R, some of which are available online for free. Here are some that I highly recommend:

Book	Description
R for Data Science by Garrett Grolemund and Hadley Wickham	The best book to learn the latest tools for elegantly doing data science.
The R Book by Michael Crawley	As close to an R bible as you can get.

Book	Description
Advanced R by Hadley Wickham	A truly advanced book for expert R users, especially those with a programming background. Hadley Wickham is *the* R guru.
Discovering Statistics with R by Field, Miles and Field	A classic text focusing on the theory and practice of statistical analysis with R
Applied Predictive Modeling by Kuhn and Johnson	A great text specializing in statistical learning aka predictive modeling aka machine learning with R.

1.6 Who am I?

My name is Nathaniel – not Nathan...not Nate...and *definitely* not Nat. I am a psychologist with a background in statistics and judgment and decision making. You can find my R (and non-R) related musings at http://ndphillips.github.io

1.6.1 Acknowledgements

I am deeply indebted to many people for either directly or indirectly helping me make this book happen. I would especially like to thank Captain Thomas Moore and Captain Wei Linn for my early training in both statistics and R, Captain Hansjoerg Neth for teaching me LaTeX and ultimately inspiring me to write (I mean translate) this book, and Captain Dirk Wulff for teaching me almost everything I know about R. If I hadn't been lucky enough to meet just one of these people, this book would not exist.

1.7 Please Contribute!

I am grateful for comments, questions, bug reports, and requests to future editions of the book! If there's anything you'd like to add or share, please contact me via email at yarrr.book@gmail.com, or if you are familiar with GitHub, post an issue at https://github.com/ndphillips/ThePiratesGuideToR/issues. Contributers will be added to the R pirate

Figure 1.5: Like a pirate, I work best with a mug of beer within arms' reach.

Chapter 2

Getting Started

2.1 Installing Base-R and RStudio

To use R, we'll need to download two software packages: **Base-R**, and **RStudio**. Base-R is the basic software which contains the R programming language. RStudio is software that makes R programming easier. Of course, they are totally free and open source.

2.1.1 Check for version updates

R and RStudio have been around for several years – however, they are *constantly* being updated with new features and bug-fixes. At the time that I am writing this sentence (09:48, Thursday, 23 February, 2017), the latest version of Base-R is 3.3.2 "Sincere Pumpkin Patch" (the versions all have funny names) which was released on 31 October, 2016, and the latest version of RStudio is 1.0.136 released on 21 December, 2016. If you have a (much) older version of R or RStudio currently installed on your computer, then you should update both R and RStudio to the newest version(s) by installing them again from scratch. If you don't, then some of the code and packages in this book might not work.

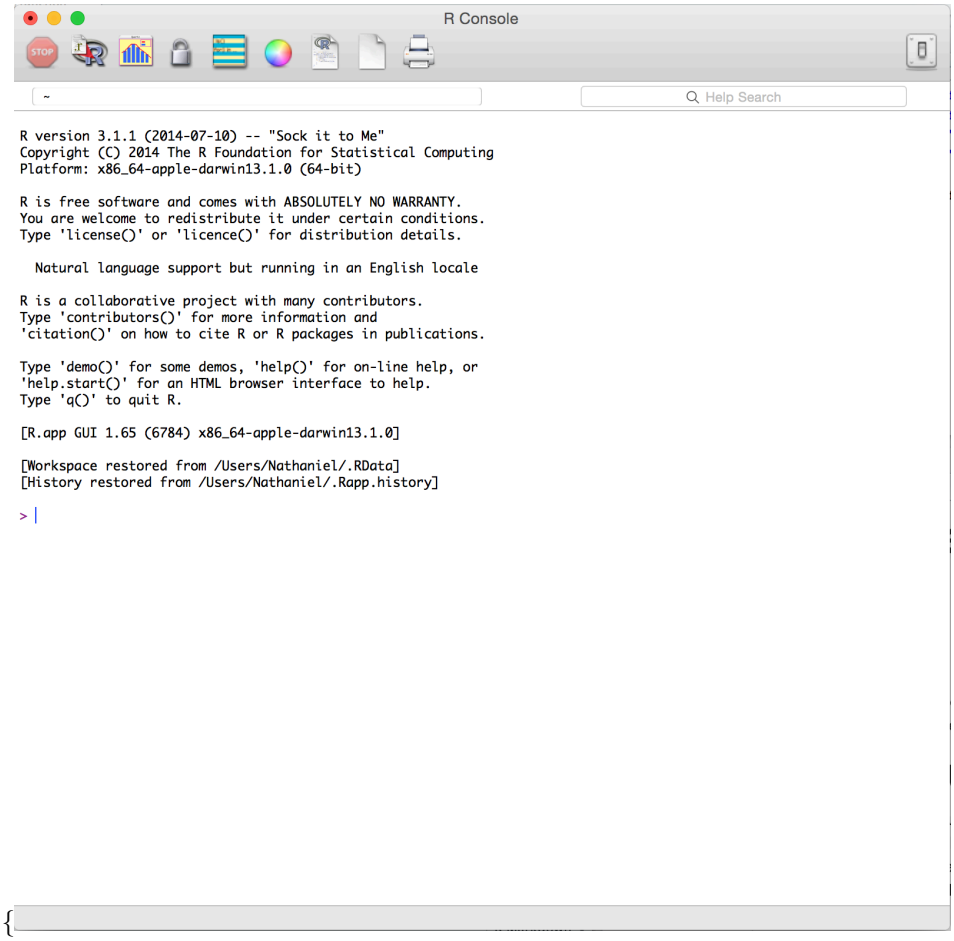
To install Base-R, click on one of the following links and follow the instructions.

Operating System	Link
Windows	http://cran.r-project.org/bin/windows/base/

Operating System	Link
Mac	http://cran.r-project.org/bin/macosx/

Once you've installed base-R on your computer, try opening it. When you do you should see a screen like the one in Figure 2.1.1 (this is the Mac version). As you can see, base R is very much bare-bones software. It's kind of the equivalent of a simple text editor that comes with your computer.

\begin{figure}



{

}

\caption{{Here is how the base R application looks. While you can use the base R application alone, most people I know use RStudio – software that helps you to write and use R code more efficiently!} \end{figure}

While you can do pretty much everything you want within base-R, you'll find that most people these days do their R programming in an application called RStudio. RStudio is a graphical user interface (GUI)-like interface for R that makes programming in R a bit easier. In fact, once you've installed RStudio, you'll likely never need to open the base R application again. To download and install RStudio (around 40mb), go to one of the links above and follow the instructions.

Operating System	Link
All	http://www.rstudio.com/products/rstudio/download/

Let's go ahead and boot up RStudio and see how she looks!

2.2 The four RStudio Windows

When you open RStudio, you'll see the following four windows (also called panes) shown in in Figure 2.1. However, your windows might be in a different order that those in Figure 2.1. If you'd like, you can change the order of the windows under RStudio preferences. You can also change their shape by either clicking the minimize or maximize buttons on the top right of each panel, or by clicking and dragging the middle of the borders of the windows.

Now, let's see what each window does in detail.

2.2.1 Source - Your notepad for code

The source pane is where you create and edit "R Scripts" - your collections of code. Don't worry, R scripts are just text files with the ".R" extension. When you open RStudio, it will automatically start a new Untitled script. Before you start typing in an untitled R script, you should always save the file under a new file name (like, "2015PirateSurvey.R"). That way, if something on your computer crashes while you're working, R will have your code waiting for you when you re-open RStudio.

You'll notice that when you're typing code in a script in the Source panel, R won't actually evaluate the code as you type. To have R actually evaluate your code, you need to first 'send' the code to the Console (we'll talk about this in the next section).

There are many ways to send your code from the Source to the console. The slowest way is to copy and paste. A faster way is to highlight the code you wish to evaluate and clicking on the "Run" button on the top right of the Source. Alternatively, you can use the hot-key "Command + Return" on Mac, or "Control + Enter" on PC to send all highlighted code to the console.

2.2.2 Console: R's Heart

The console is the heart of R. Here is where R actually evaluates code. At the beginning of the console you'll see the character `>`. This is a prompt that tells you that R is ready for new code. You can type code directly into the console after the `>` prompt and get an immediate response. For example, if you type `1+1` into the console and press enter, you'll see that R immediately gives an output of 2.

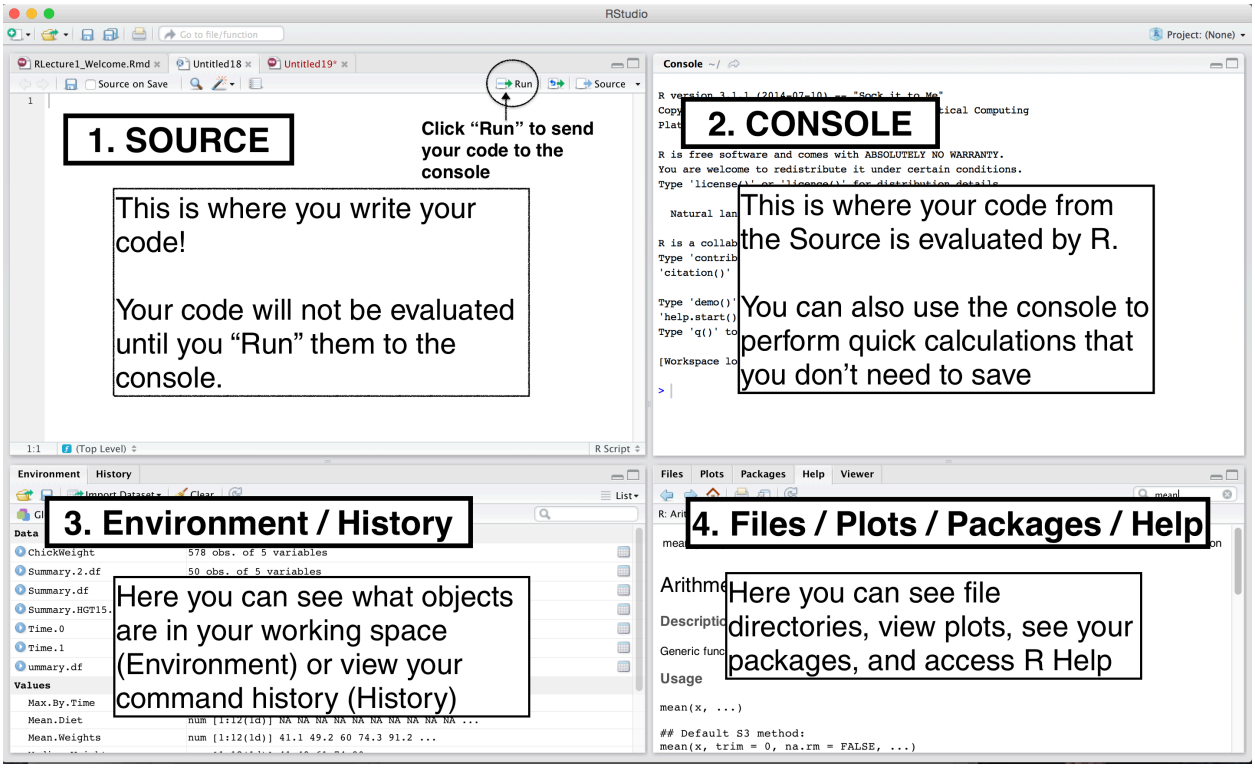


Figure 2.1: The four panes of RStudio.

```
1+1
## [1] 2
```

Try calculating 1+1 by typing the code directly into the console - then press Enter. You should see the result [1] 2. Don't worry about the [1] for now, we'll get to that later. For now, we're happy if we just see the 2. Then, type the same code into the Source, and then send the code to the Console by highlighting the code and clicking the "Run" button on the top right hand corner of the Source window. Alternatively, you can use the hot-key "Command + Return" on Mac or "Control + Enter" on Windows.

Tip: Try to write most of your code in a document in the Source. Only type directly into the Console to de-bug or do quick analyses.

So as you can see, you can execute code either by running it from the Source or by typing it directly into the Console. However, 99% most of the time, you should be using the Source rather than the Console. The reason for this is straightforward: If you type code into the console, it won't be saved (though you can look back on your command History). And if you make a mistake in typing code into the console, you'd have to re-type everything all over again. Instead, it's better to write all your code in the Source. When you are ready to execute some code, you can then send "Run" it to the console.

2.2.3 Environment / History

The Environment tab of this panel shows you the names of all the data objects (like vectors, matrices, and dataframes) that you've defined in your current R session. You can also see information like the number of observations and rows in data objects. The tab also has a few clickable actions like "Import Dataset" which will open a graphical user interface (GUI) for important data into R. However, I almost never look at this menu.

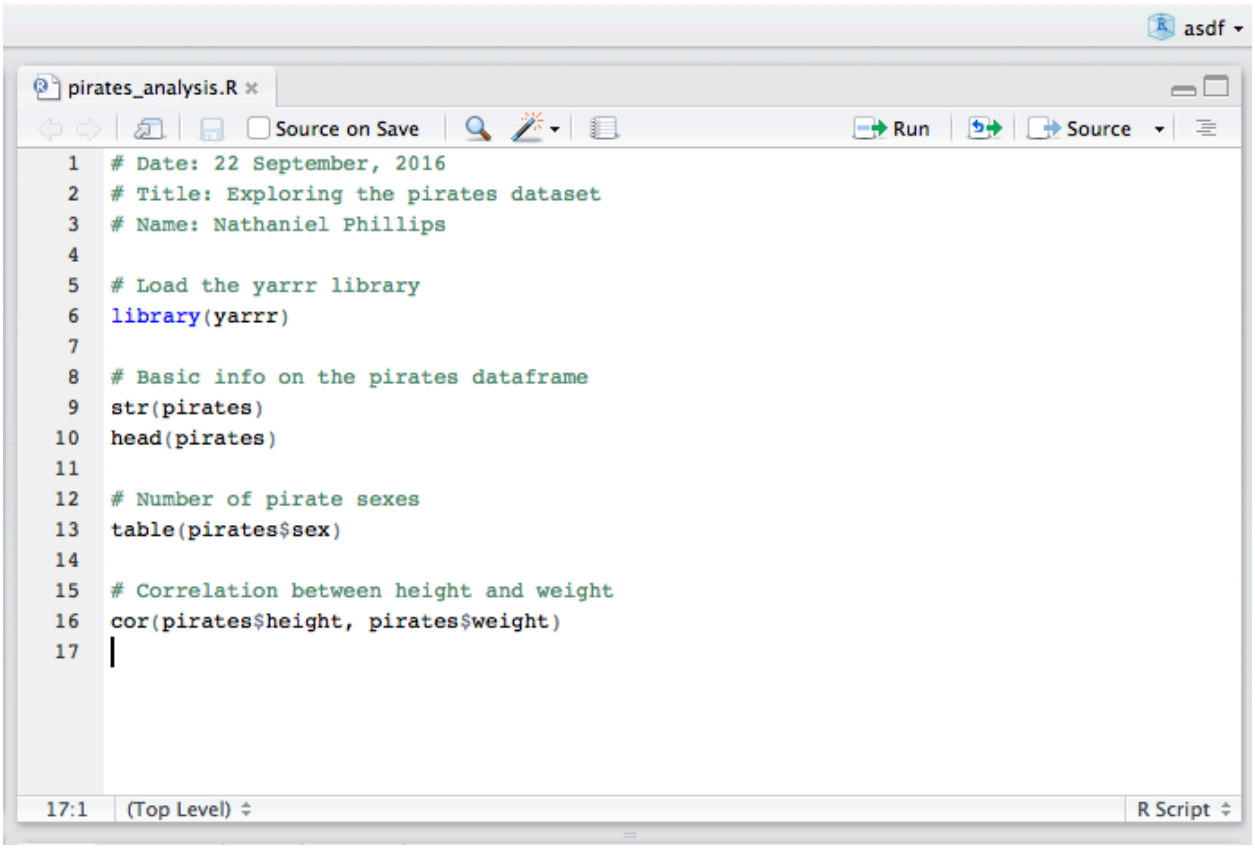


Figure 2.2: The Source contains all of your individual R scripts. The code won't be evaluated until you send it to the Console.

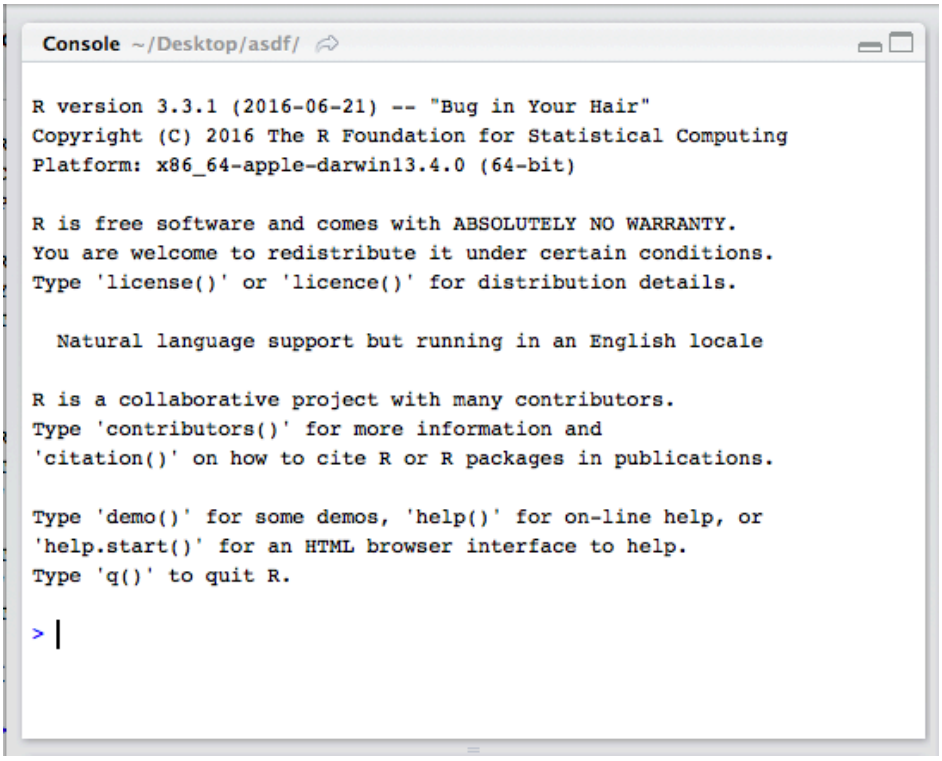


Figure 2.3: The console the calculation heart of R. All of your code will (eventually) go through here.

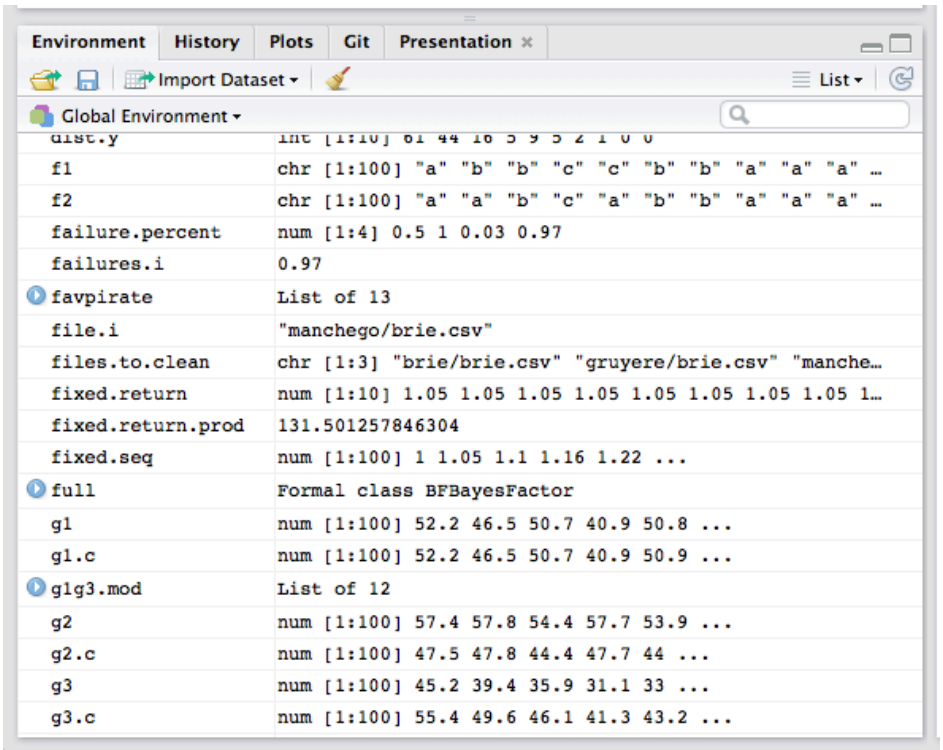


Figure 2.4: The environment panel shows you all the objects you have defined in your current workspace. You'll learn more about workspaces in Chapter 7.

The History tab of this panel simply shows you a history of all the code you've previously evaluated in the Console. To be honest, I never look at this. In fact, I didn't realize it was even there until I started writing this tutorial.

As you get more comfortable with R, you might find the Environment / History panel useful. But for now you can just ignore it. If you want to declutter your screen, you can even just minimize the window by clicking the minimize button on the top right of the panel.

2.2.4 Files / Plots / Packages / Help

The Files / Plots / Packages / Help panel shows you lots of helpful information. Let's go through each tab in detail:

1. Files - The files panel gives you access to the file directory on your hard drive. One nice feature of the "Files" panel is that you can use it to set your working directory - once you navigate to a folder you want to read and save files to, click "More" and then "Set As Working Directory." We'll talk about working directories in more detail soon.

2. Plots - The Plots panel (no big surprise), shows all your plots. There are buttons for opening the plot in a separate window and exporting the plot as a pdf or jpeg (though you can also do this with code using the `pdf()` or `jpeg()` functions.)

Let's see how plots are displayed in the Plots panel. Run the code on the right to display a histogram of the weights of chickens stored in the `ChickWeight` dataset. When you do, you should see a plot similar to the one in Figure 2.5 show up in the Plots panel.

```
hist(x = ChickWeight$weight,
     main = "Chicken Weights",
     xlab = "Weight",
     col = "skyblue",
     border = "white")
```

3. Packages - Shows a list of all the R packages installed on your harddrive and indicates whether or not they are currently loaded. Packages that are loaded in the current session are checked while those that are installed but not yet loaded are unchecked. We'll discuss packages in more detail in the next section.

4. Help - Help menu for R functions. You can either type the name of a function in the search window, or use the code `?function.name` to search for a function with the name `function.name`

```
?hist   # How does the histogram function work?
?t.test # What about a t-test?
```

2.3 Packages

When you download and install R for the first time, you are installing the Base R software. Base R will contain most of the functions you'll use on a daily basis like `mean()` and `hist()`. However, only functions written by the original authors of the R language will appear here. If you want to access data and code written by other people, you'll need to install it as a *package*. An R package is simply a bunch of data, from functions, to help menus, to vignettes (examples), stored in one neat package.

A package is like a light bulb. In order to use it, you first need to order it to your house (i.e.; your computer) by *installing* it. Once you've installed a package, you never need to install it again. However, every time you want to actually use the package, you need to turn it on by *loading* it. Here's how to do it.

Figure 2.5: The plot panel contains all of your plots, like this histogram of the distribution of chicken weights.

Figure 2.6: An R package is light a lightbulb. First you need to order it with install.packages(). Then, every time you want to use it, you need to turn it on with library()

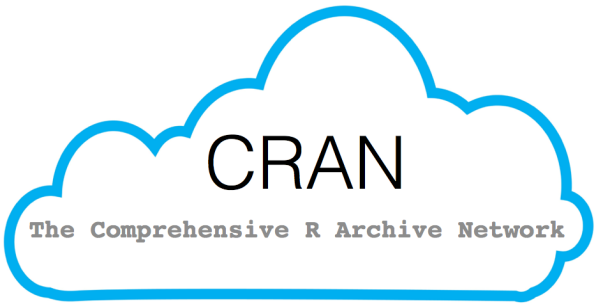


Figure 2.7: CRAN (Comprehensive R Archive Network) is the main source of R packages

```
> install.packages("circlize")
trying URL 'https://cran.rstudio.com/bin/macosx/mavericks/contrib/3.3/circlize_0.3.8.tgz'
Content type 'application/x-gzip' length 3856952 bytes (3.7 MB)
==================================================
downloaded 3.7 MB


The downloaded binary packages are in
        /var/folders/gy/bsyxftvn37q93cm6vt1v56fr0000gp/T//RtmpOzgZ2R/downloaded_packages
>
```

Figure 2.8: When you install a new package, you'll see some random text like this you the download progress. You don't need to memorize this.

2.3.1 Installing a new package

Installing a package simply means downloading the package code onto your personal computer. There are two main ways to install new packages. The first, and most common, method is to download them from the Comprehensive R Archive Network (CRAN). CRAN is the central repository for R packages. To install a new R package from CRAN, you can simply run the code `install.packages("name")`, where "name" is the name of the package. For example, to download the `yarrr` package, which contains several data sets and functions we will use in this book, you should run the following:

```
# Install the yarrr package from CRAN
#   You only need to install a package once!
install.packages("yarrr")
```

When you run `install.packages("name")` R will download the package from CRAN. If everything works, you should see some information about where the package is being downloaded from, in addition to a progress bar.

Like ordering a light bulb, once you've installed a package on your computer you never need to install it again (unless you want to try to install a new version of the package). However, every time you want to use it, you need to turn it on by loading it.

2.3.2 Loading a package

Once you've installed a package, it's on your computer. However, just because it's on your computer doesn't mean R is ready to use it. If you want to use something, like a function or dataset, from a package you *always* need to *load* the package in your R session first. Just like a light bulb, you need to turn it on to use it!

To load a package, you use the `library()` function. For example, now that we've installed the `yarrr`

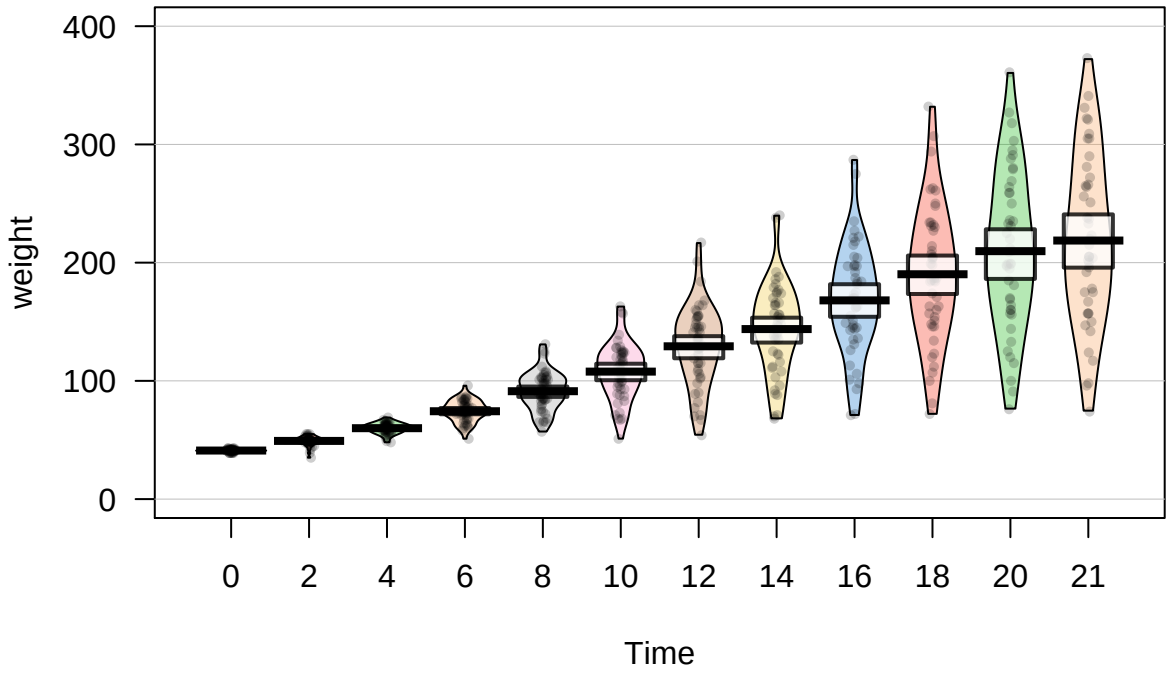
package, we can load it with `library("yarrr")`:

```
# Load the yarrr package so I can use it!
#   You have to load a package in every new R session!
library("yarrr")
```

Now that you've loaded the `yarrr` package, you can use any of its functions! One of the coolest functions in this package is called `pirateplot()`. Rather than telling you what a pirateplot is, let's just make one. Run the following code chunk to make your own pirateplot. Don't worry about the specifics of the code below, you'll learn more about how all this works later. For now, just run the code and marvel at your pirateplot.

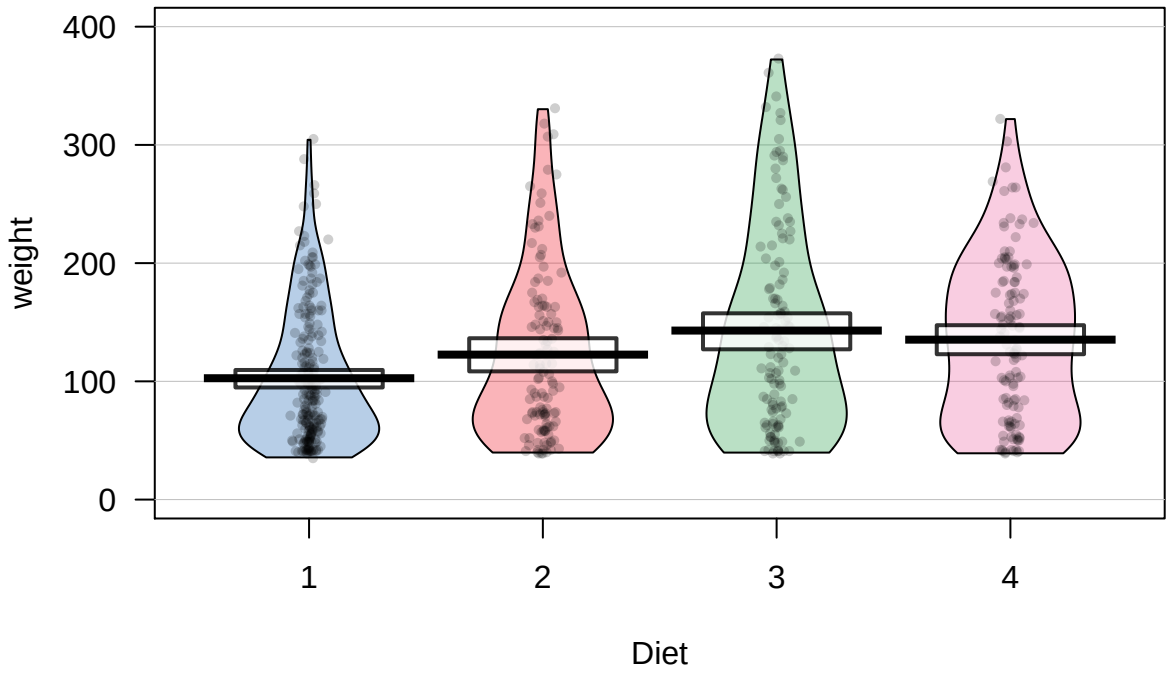
```
# Make a pirateplot using the pirateplot() function
#  from the yarrr package!

pirateplot(formula = weight ~ Time,
           data = ChickWeight,
           pal = "xmen")
```



There is one way in R to temporarily load a package without using the `library()` function. To do this, you can simply use the notation `package::function` notation. This notation simply tells R to load the package just for this one chunk of code. For example, I could use the `pirateplot` function from `yarrr` package as follows:

```
# Use the pirateplot() function without loading the yarrr package first
yarrr::pirateplot(formula = weight ~ Diet,
                  data = ChickWeight)
```



Again, you can think about the `package::function` method as a way to temporarily loading a package for a single line of code. One benefit of using the `package::function` notation is that it's immediately clear to anyone reading the code which package contains the function. However, a drawback is that if you are using a function from a package often, it forces you to constantly retype the package name. You can use whichever method makes sense for you.

2.4 Reading and writing Code

2.4.1 Code Chunks

In this book, R code is (almost) always presented in a separate gray box like this one:

```
# A code chunk

# Define a vector a as the integers from 1 to 5
a <- 1:5

# Print a
a
## [1] 1 2 3 4 5

# What is the mean of a?
mean(a)
## [1] 3
```

This is called a *code chunk*. You should always be able to copy and paste code chunks directly into R. If you copy a chunk and it does not work for you, it is most likely because the code refers to a package, function, or object that I defined in a previous chunk. If so, read back and look for a previous chunk that contains the missing definition.

2.4.2 Comments with

Lines that begin with # are comments. If you evaluate any code that starts with #, R will just ignore that line. In this book, comments will be either be literal comments that I write directly to explain code, or they will be *output* generated automatically from R. For example, in the code chunk below, you see lines starting with ##. These are the output from the previous line(s) of code. When you run the code yourself, you should see the same output in your *console*.

```
# This is a comment I wrote

1 + 2
## [1] 3

# The line above (## [1] 3) is the output from the previous code that has been 'commented out'
```

2.4.3 Element numbers in output [1]

The output you see will often start with one or more number(s) in brackets such as [1]. This is just a visual way of telling you where the numbers occur in the output. For example, in the code below, I will print a long vector containing the multiples of 2 from 0 to 100:

```
seq(from = 0, to = 100, by = 2)
##  [1]   0   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32
## [18]  34  36  38  40  42  44  46  48  50  52  54  56  58  60  62  64  66
## [35]  68  70  72  74  76  78  80  82  84  86  88  90  92  94  96  98 100
```

As you can see, the first line of the output starts with ## [1], and the next two lines start with [18] and [35]. This is just telling you that 0 is the [1]st element, 34 is the [18]th element, and 68 is the [35]th element. Sometimes this information will be helpful, but most of the time you can just ignore it.

2.5 Debugging

When you are programming, you will always, and I do mean always, make errors (also called bugs) in your code. You might misspell a function, include an extra comma, or some days…R just won't want to work with you (again, see section Why R is like a Relationship).

Debugging will always be a challenge. However, over time you'll learn which bugs are the most common and get faster and faster at finding and correcting them.

Here are the most common bugs you'll run into as you start your R journey.

2.5.1 R is not ready (>)

Another very common problem occurs when R does not seem to be responding to your code. That is, you might run some code like `mean(x)` expecting some output, but instead, nothing happens. This can be very frustrating because, rather than getting an error, just nothing happens at all. The most common reason for this is because R isn't *ready* for new code, instead, it is *waiting* for you to finish code you started earlier, but never properly finished.

Think about it this way, R can be in one of two states: it is either **Ready** (>) for new code, or it is **Waiting** (+) for you to finish old code. To see which state R is in, all you have to do is look at the symbol on the console. The `>` symbol means that R is Ready for new code – this is usually what you want to see. The `+` symbol means that R is Waiting for you to (properly) finish code you started before. If you see the `+`

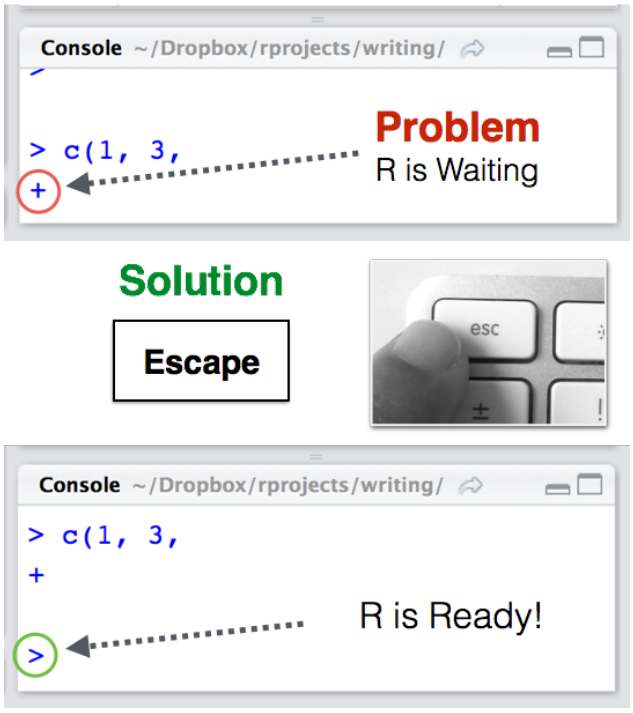


Figure 2.9: To turn R from a Waiting (+) state to a Ready (>) state, just hit Escape.

symbol, then no matter how much new code you write, R won't actually evaluate it until you finish the code you started before.

Thankfully there is an easy solution to this problem (See Figure 2.9): Just hit the escape key on your keyboard. This will cancel R's waiting state and make it Ready!

2.5.2 Misspelled object or function

If you spell an object or function incorrectly, you'll receive an error like `Error: could not find function` or `Error: object 'x' not found`.

In the code below, I'll try to take the mean of a vector `data`, but I will misspell the function `mean()`

```
data <- c(1, 4, 3, 2, 1)

# Misspelled function: should be mean(x), not meeen(x)
meeen(data)
```

Error: could not find function "meeen"

Now, I'll misspell the object `data` as `dta`:

```
# Misspelled object: should be data, not dta
mean(dta)
```

Error: object 'dta' not found

R is case-sensitive, so if you don't use the correct capitalization you'll receive an error. In the code below, I'll use `Mean()` instead of the correct version `mean()`

```
# Capitalization is wrong: should be mean(), not Mean()
Mean(data)
```

Error: could not find function "Mean"

Here is the correct version where both the object `data` and function `mean()` are correctly spelled:

```
# Correct: both the object and function are correctly spelled
mean(data)
## [1] 2.2
```

2.5.3 Punctuation problems

Another common error is having bad coding "punctuation". By that, I mean having an extra space, missing a comma, or using a comma (,) instead of a period (.). In the code below, I'll try to create a vector using periods instead of commas:

```
# Wrong: Using periods (.) instead of commas (,)
mean(c(1. 4. 2))
```

Error: unexpected numeric constant in "mean(c(1. 4."

Because I used periods instead of commas, I get the above error. Here is the correct version

```
# Correct
mean(c(1, 4, 2))
## [1] 2.3
```

If you include an extra space in the middle of the name of an object or function, you'll receive an error. In the code below, I'll accidentally write `Chick Weight` instead of `ChickWeight`:

```
# Wrong: Extra space in the ChickWeight object name
head(Chick Weight)
```

Error: unexpected symbol in "head(Chick Weight"

Because I had an extra space in the object name, I get the above error. Here is the correction:

```
# Correct:
head(ChickWeight)
```