A Little Book of R for Bioinformatics 2.0

Avirl Coghlan, with contributions by Nathan L. Brouwer

2021-08-15

Contents

Preface to version 2.0

Welcome to *A Little Book of R for Bioinformatics 2.0*!.

This book is based on the original *A Little Book of R for Bioinformatics* by Dr. Avril Coghlan (Hereafter "ALBRB 1.0"). Dr. Coghlan's book was one of the first and most thorough introductions to using *R* for bioinformatics and computational biology, and was generously published under the Creative Commons 3.0 Attribution License (CC BY 3.0). In addition to describing how to do bioinformatics in R, Coghlan provided numerous functions to facilitate important tasks, practice questions, and references to further reading.

ALBRB 1.0 was extremely useful to me when I was learning bioinformatics and computational biology. In this version of the book, which I'll refer to as **ALBRB 2.0**, I have adapted Dr. Coghlan's original book to suit my own teaching needs, updated it with current packages now available in R, added background materials from other open-access sources, and added in my original materials.

Below I've outlined the general types of changes I've made to the original book. I have tried to link back to the original content that these updates are derived from and note how changes were made. Any errors or inconsistencies should be ascribed to me, not Dr. Coghlan. If you have any feedback, please email me at brouwern@gmail.com

~Nathan Brouwer, June 2021

Changes implemented in ALBRB 2.0 by Nathan Brouwer

1. Converted the entire book to RMarkdown and published it via bookdown.
2. Added instructions for using RStudio and RStudio Cloud.
3. Updated instructions to reflect any changes in software, including changes to how the bioinformatics repository Bioconductor now works.
4. Split up chapters into smaller units.
5. Reorganized the order of some material.
6. Added biological background information by integrating information from the Open Access textbook LibreText General Biology
7. Added links to the books I am developing, *Get R Done!* and *Computational Biology for All.*
8. Moved functions written by Coghlan and datasets to my teaching package compbio4all.
9. Functions names changed from camelCase to snake_case
10. Functions re-written so as not to use Bioconductor to reduce/eliminate dependency of compbio4all on Bioconductor.
11. Changed some plotting to ggplot2 or ggpubr.
12. Added additional subheadings
13. Added vocab and function lists to the beginning of many chapters
14. At times replaced non-biological examples with biological ones.
15. Change from British to American English (Sorry! Couldn't help myself - the spellchecker made me do it!)
16. Provided additional links to external resources.
17. Added use of rentrez for querying NCBI databases

Chapter 1

Downloading R

By: Avril Coghlan

Adapted, edited and expanded: Nathan Brouwer (brouwern@gmail.com)

1.1 Preface

The following introduction to *R* is based on the first part of "How to install *R* and a Brief Introduction to R" by Avril Coghlan, which was released under the Creative Commons 3.0 Attribution License (CC BY 3.0). For additional information see the Appendices and "Getting *R* onto your computer".

1.2 Introduction to R

R (www.r-project.org) is a commonly used free statistics software. *R* allows you to carry out statistical analyses in an interactive mode, as well as allowing programming.

1.3 Installing R

To use R, you first need to install the *R* program on your computer.

1.3.1 Installing *R* on a Windows PC

These instructions will focus on installing *R* on a Windows PC. However, I will also briefly mention how to install *R* on a Macintosh or Linux computer (see below).

These steps have not been checked as of 8/13/2019 so there may be small variations in what the prompts are. Installing R, however, is basically that same as any other program. Clicking "Yes" etc on everything should work.

> **PROTIP:** Even if you have used *R* before its good to regularly update it to avoid conflicts with recently produced software.

Minor updates of *R* are made very regularly (approximately every 6 months), as *R* is actively being improved all the time. It is worthwhile installing new versions of *R* a couple times a year, to make sure that you have a recent version of *R* (to ensure compatibility with all the latest versions of the *R* packages that you have downloaded).

To install *R* on your **Windows** computer, follow these steps:

1. Go to https://cran.r-project.org/
2. Under "Download and Install R", click on the "Windows" link.
3. Under "Subdirectories", click on the **"base"** link.
4. On the next page, you should see a link saying something like "Download *R* 4.1.0 for Windows" (or *R* X.X.X, where X.X.X gives the version of the program). Click on this link.
5. You may be asked if you want to save or run a file "R-x.x.x-win32.exe". Choose "Save" and save the file. Then double-click on the icon for the file to run it.
6. You will be asked what language to install it in.
7. The *R* Setup Wizard will appear in a window. Click "Next" at the bottom of the *R* Setup wizard window.
8. The next page says "Information" at the top. Click "Next" again.
9. The next page says "Select Destination Location" at the top. By default, it will suggest to install *R* on the C drive in the "Program Files" directory on your computer.
10. Click "Next" at the bottom of the *R* Setup wizard window.
11. The next page says "Select components" at the top. Click "Next" again.
12. The next page says "Startup options" at the top. Click "Next" again.
13. The next page says "Select start menu folder" at the top. Click "Next" again.
14. The next page says "Select additional tasks" at the top. Click "Next" again.
15. *R* should now be installing. This will take about a minute. When *R* has finished, you will see "Completing the *R* for Windows Setup Wizard" appear. Click "Finish".
16. To start R, you can do one of the following steps:
17. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try the next step instead.
18. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start *R* by selecting "R" (or *R* X.X.X, where X.X.X gives the version of R) from the menu of programs.
19. The *R* console (a rectangle) should pop up:

1.3.2 How to install *R* on non-Windows computers (eg. Macintosh or Linux computers)

These steps have not been checked as of 8/13/2019 so there may be small variations in what the prompts are. Installing R, however, is basically that same as any other program. Clicking "Yes" etc on everything should work.

The instructions above are for installing *R* on a Windows PC. If you want to install *R* on a computer that has a non-Windows operating system (for example, a Macintosh or computer running Linux, you should download the appropriate *R* installer for that operating system at https://cran.r-project.org/ and follow the *R* installation instructions for the appropriate operating system at https://cran.r-project.org/doc/FAQ/R-FAQ.html#How-can-R-be-installed_003f .

1.4 Starting *R*

To start R, Check if there is an *R* icon on the desktop of the computer that you are using. If so, double-click on the *R* icon to start *R*. If you cannot find an *R* icon, try the next step instead.

You can also start *R* from the Start menu in Windows. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start *R* by selecting "R" (or *R* X.X.X, where X.X.X gives the version of R, e.g.. *R* 2.10.0) from the menu of programs.

Say "Hi" to *R* and take a quick look at how it looks. Now say "Goodbye", because we will never actually do any work in this version of *R*; instead, we'll use the **RStudio IDE (integrated development environment)**.

Chapter 2

Installing the RStudio IDE {$installRStudio}

By: Nathan Brouwer

The name "R" refers both to the programming language and the program that runs that language. When you download *it*R* there is also a basic **GUI** (graphical user interface) that you can access via the *R* icon.

Other GUIs are available, and the most popular currently is **RStudio.** RStudio a for-profit company that is a main driver of development of R. Much of what they produce has free basic versions or is entirely free. They produce software (RStudio), cloud-based applications (**RStudio Cloud**), and web server infrastructure for business applications of R.

A brief overview of installing RStudio can be found here "Getting RStudio on to your computer"

2.1 Getting to know RStudio

For a brief overview of RStudio see "Getting started with RStudio"

A good overview of what the different parts of RStudio can be seen in the image in this tweet: https://twitter.com/RLadiesNCL/status/1138812826917724160?s=20

2.2 RStudio versus RStudio Cloud

RStudio and RStudio cloud work almost identically, so anything you read about RStudio will apply to RStudio Cloud. RStudio is easy to download an use, but RStudio Cloud eliminates even the minor hiccups that occur. Free accounts with RStudio Cloud allow up to 15 hours per month, which is enough for you to get a taste for using R.

Chapter 3

Installing *R* packages

By: Avril Coghlan.

R is a programming language, and **packages** (aka **libraries**) are bundles of software built using *R*. Most sessions using *R* involve using additional *R* packages. This is especially true for bioinformatics and computational biology.

> **NOTE**: If you are working in an RStudio Cloud environment organized by someone else (e.g. a course instructor), they likely are taking care of many of the package management issues. The following information is still useful to be familiar with.

3.1 Downloading packages with the RStudio IDE

There is a point-and-click interface for installing *R* packages in RStudio. There is a brief introduction to downloading packages on this site: http://web.cs.ucla.edu/~gulzar/rstudio/

I've summarized it here:

1. "Click on the"Packages" tab in the bottom-right section and then click on "Install". The following dialog box will appear.
2. In the "Install Packages" dialog, write the package name you want to install under the Packages field and then click install. This will install the package you searched for or give you a list of matching package based on your package text.

3.2 Downloading packages with the function `install.packages()`

The easiest way to install a package if you know its name is to use the *R* function install .packages()‘. Note that it might be better to call this "download.packages" since after you install it, you also have to load it!

Frequently I will include install .packages (...) at the beginning of a chapter the first time we use a package to make sure the package is downloaded. Note, however, that if you already have downloaded the package, running install .packages (...) will download a new copy. While packages do get updated from time to time, but its best to re-run install .packages (...) only occassionaly.

We'll download a package used for plotting called ggplot2, which stands for "Grammar of Graphics." ggplot2 was developed by Dr. Hadley Wickham, who is now the Chief Scientists for RStudio.

To download ggplot2, run the following command:

```
install.packages("ggplot2") # note the " "
```

Often when you download a package you'll see a fair bit of angry-looking red text, and sometime other things will pop up. Usually there's nothing of interest here, but sometimes you need to read things carefully over it for hints about why something didn't work.

3.3 Using packages after they are downloaded

To actually make the functions in package accessible you need to use the library() command. Note that this is *not* in quotes.

```
library(ggplot2) # note: NO " "
```

Chapter 4

Installing Bioconductor

By: Avril Coghlan.

Adapted, edited and expanded: Nathan Brouwer under the Creative Commons 3.0 Attribution License (CC BY 3.0), including details on install Bioconductor and common prompts and error messages that appear during installation.

4.1 Bioconductor

R **packages** (aka "libraries") can live in many places. Most are accessed via **CRAN**, the **Comprehensive R Archive Network**. The bioinformatics and computational biology community also has its own package hosting system called Bioconductor. *R* has played an important part in the development and application of bioinformatics techniques in the 21th century. Bioconductor 1.0 was released in 2002 with 15 packages. As of winter 2021, there are almost 2000 packages in the current release!

> **NOTE**: If you are working in an RStudio Cloud environment organized by someone else (eg a course instructor), they likely are taking care of most of package management issues, inlcuding setting up Bioconductor. The following information is still useful to be familiar with.

To interface with Bioconductor you need the BiocManager package. The Bioconductor people have put BiocManager on CRAN to allow you to set up interactions with Bioconductor. See the BiocManager documentation for more information (https://cran.r-project.org/web/packages/BiocManager/vignettes/BiocManager.html).

Note that if you have an old version of R you will need to update it to interact with Bioconductor.

4.2 Installing BiocManager

BiocManager can be installed using the install .packages() packages command.

```
install.packages("BiocManager") # Remember the " "; don't worry about the red
    text
```

Once downloaded, BioManager needs to be explicitly loaded into your active R session using library ()

```
library(BiocManager) # no quotes; again, ignore the red text
```

Individual Bioconductor packages can then be downloaded using the install () command. An essential packages is Biostrings. To do this ,

```
BiocManager::install("Biostrings")
```

4.3 The ins and outs of package installation

IMPORANT Bioconductor has many **dependencies** - other packages which is relies on. When you install Bioconductor packages you may need to update these packages. If something seems to not be working during this process, restart R and begin the Bioconductor installation process until things seem to work.

Below I discuss the series of prompts I had to deal with while re-installing Biostrings while editing this chapter.

4.3.1 Updating other packages when downloading a package

When I re-installed Biostrings while writing this I was given a HUGE blog of red test that contained something like what's shown below (this only aout 1/3 of the actual output!):

```
'getOption("repos")' replaces Bioconductor standard repositories,
see '?repositories' for details

replacement repositories:
    CRAN: https://cran.rstudio.com/

Bioconductor version 3.11 (BiocManager 1.30.16), R 4.0.5 (2021-03-31)
Old packages: 'ade4', 'ape', 'aster', 'bayestestR',
  'bio3d', 'bitops', 'blogdown',
  'bookdown', 'brio', 'broom', 'broom.mixed',
  'broomExtra', 'bslib', 'cachem', 'callr',
  'car', 'circlize', 'class', 'cli', 'cluster',
  'colorspace', 'corrplot', 'cpp11', 'curl',
  'devtools', 'DHARMa', 'doBy', 'dplyr', 'DT',
  'e1071', 'ellipsis', 'emmeans', 'emojifont',
  'extRemes', 'fansi', 'flextable', 'forecast',
  'formatR', 'gap', 'gargle', 'gert', 'GGally'
```

Hidden at the bottom was a prompt: "Update all/some/none? [a/s/n]:"

Its a little vague, but what it wants me to do is type in a, s or n and press enter to tell it what to do. I almost always chose a, though this may take a while to update everything.

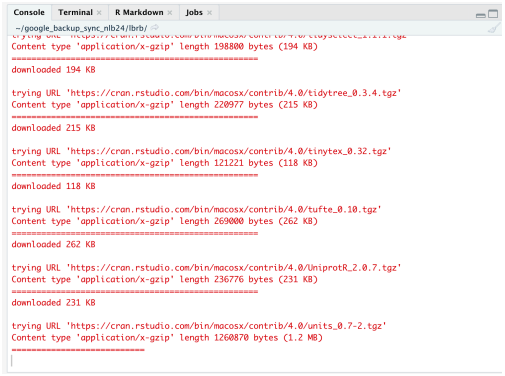
4.3.2 Packages "from source"

You are likely to get lots of random-looking feedback from R when doing Bioconductor-related installations. Look carefully for any prompts as the very last line. While updating Biostrings I was told: "*There are binary versions available but the source versions are later:*" and given a table of packages. I was then asked "*Do you want to install from sources the packages which need compilation? (Yes/no/cancel)*"

I almost always chose "no".

4.3.3 More on angry red text

After the prompt about packages from source, R proceeded to download a lot of updates to packages, which took a few minutes. Lots of red text scrolled by, but this is normal.

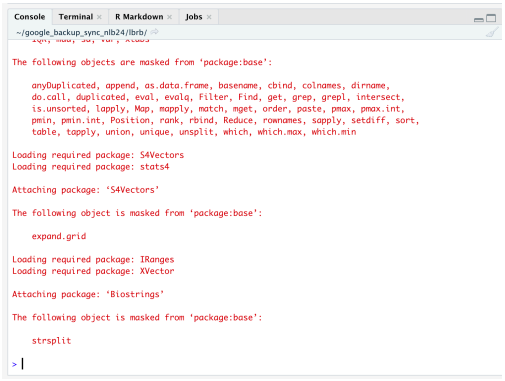


4.4 Actually loading a package

Again, to actually load the Biostrings package into your active R sessions requires the libary() command:

```
library(Biostrings)
```

As you might expect, there's more red text scrolling up my screen!



I can tell that is actually worked because at the end of all the red stuff is the R prompt of ">" and my cursor.

Chapter 5

A Brief introduction to R

By: Avril Coghlan.

This chapter provides a brief introduction to R. At the end of are links to additional resources for getting started with R.

5.1 Vocabulary

- scalar
- vector
- list
- class
- numeric
- character
- assignment
- elements of an object
- indices
- attributes of an object
- argument of a function

5.2 R functions

- <-
- [ ]
- $
- table()
- function
- c()
- log10()
- help(), ?
- help.search()
- RSiteSearch()
- mean()
- return()
- q()

5.3 Interacting with R

You will type *R* commands into the RStudio **console** in order to carry out analyses in *R*. In the RStudio console you will see the R prompt starting with the symbol ">". ">" will always be there at the beginning of each new command - don't try to delete it! Moreover, you never need to type it.

We type the **commands** needed for a particular task after this prompt. The command is carried out by *R* after you hit the Return key.

Once you have started R, you can start typing commands into the RStudio console, and the results will be calculated immediately, for example:

```
2*3
```

```
## [1] 6
```

Note that prior to the output of "6" it shows "[1]".

Now subtraction:

```
10-3
```

```
## [1] 7
```

Again, prior to the output of "7" it shows "[1]".

R can act like a basic calculator that you type commands in to. You can also use it like a more advanced scientific calculator and create **variables** that store information. All variables created by R are called **objects**. In R, we assign values to variables using an arrow-looking function <- the **assignment operator**. For example, we can **assign** the value 2*3 to the variable x using the command:

```
x <- 2*3
```

To view the contents of any R object, just type its name, press enter, and the contents of that R object will be displayed:

```
x
```

```
## [1] 6
```

5.4 Variables in R

There are several different types of objects in R with fancy math names, including **scalars**, **vectors**, **matrices** (singular: **matrix),** arrays**,** dataframes**,** tables**, and** lists**. The** scalar** variable x above is one example of an R object. While a scalar variable such as x has just one element, a **vector** consists of several elements. The elements in a vector are all of the same **type** (e.g.. numbers or alphabetic characters), while **lists** may include elements such as characters as well as numeric quantities. Vectors and dataframes are the most common variables you'll use. You'll also encounter matrices often, and lists are ubiquitous in R but beginning users often don't encounter them because they remain behind the scenes.

5.4.1 Vectors

To create a vector, we can use the c() (combine) function. For example, to create a vector called myvector that has elements with values 8, 6, 9, 10, and 5, we type:

```
myvector <- c(8, 6, 9, 10, 5) # note: commas between each number!
```

To see the contents of the variable myvector, we can just type its name and press enter:

```
myvector
```

```
## [1]  8  6  9 10  5
```

5.4.2 Vector indexing

The [1] is the **index** of the first **element** in the vector. We can **extract** any element of the vector by typing the vector name with the index of that element given in **square brackets** [...] .

For example, to get the value of the 4th element in the vector myvector, we type:

```
myvector[4]
```

```
## [1] 10
```

5.4.3 Character vectors

Vectors can contain letters, such as those designating nucleic acids

```
my.seq <- c("A","T","C","G")
```

They can also contain multi-letter **strings**:

```
my.oligos <- c("ATCGC","TTTCGC","CCCGCG","GGGCGC")
```

5.4.4 Lists

NOTE: *below is a discussion of lists in R. This is excellent information, but not necessary if this is your very very first time using R.*

In contrast to a vector, a **list** can contain elements of different types, for example, both numbers and letters. A list can even include other variables such as a vector. The list () function is used to create a list. For example, we could create a list mylist by typing:

```
mylist <- list(name="Charles Darwin",
               wife="Emma Darwin",
               myvector)
```

We can then print out the contents of the list mylist by typing its name:

```
mylist
```

```
## $name
## [1] "Charles Darwin"
##
## $wife
## [1] "Emma Darwin"
##
## [[3]]
## [1]  8  6  9 10  5
```

The **elements** in a list are numbered, and can be referred to using **indices**. We can extract an element of a list by typing the list name with the index of the element given in double **square brackets** (in contrast to a vector, where we only use single square brackets).

We can extract the second element from mylist by typing:

```
mylist[[2]]  # note the double square brackets [[...]]
```

```
## [1] "Emma Darwin"
```

As a baby step towards our next task, we can wrap index values as in the c() command like this:

```
mylist[[c(2)]]  # note the double square brackets [[...]]
```

```
## [1] "Emma Darwin"
```

The number 2 and c(2) mean the same thing.

Now, we can extract the second AND third elements from mylist. First, we put the indices 2 and 3 into a vector c(2,3), then wrap that vector in double square brackets: [c(2,3)]. All together it looks like this.

```
mylist[c(2,3)] # note the double brackets
```

```
## $wife
## [1] "Emma Darwin"
##
## [[2]]
## [1]  8  6  9 10  5
```

Elements of lists may also be named, resulting in a **named lists**. The elements may then be referred to by giving the list name, followed by "$", followed by the element name. For example, mylist$name is the same as mylist[[1]] and mylist$wife is the same as mylist[[2]]:

```
mylist$wife
```

```
## [1] "Emma Darwin"
```

We can find out the names of the named elements in a list by using the attributes() function, for example:

```
attributes(mylist)
```

```
## $names
## [1] "name" "wife" ""
```

When you use the attributes() function to find the named elements of a list variable, the named elements are always listed under a heading "$names". Therefore, we see that the named elements of the list variable mylist are called "name" and "wife", and we can retrieve their values by typing mylist$name and mylist$wife, respectively.

5.4.5 Tables

Another type of object that you will encounter in R is a **table**. The table() function allows you to total up or tabulate the number of times a value occurs within a vector. Tables are typically used on vectors containing **character data**, such as letters, words, or names, but can work on numeric data data.

5.4.5.1 Tables - The basics

If we made a vector variable "nucleotides" containing the of a DNA molecule, we can use the table() function to produce a **table variable** that contains the number of bases with each possible nucleotides:

```
bases <- c("A", "T", "A", "A", "T", "C", "G", "C", "G")
```

Now make the table

```
table(bases)
```

```
## bases
## A C G T
## 3 2 2 2
```

We can store the table variable produced by the function table(), and call the stored table "bases.table", by typing:

```
bases.table <- table(bases)
```

Tables also work on vectors containing numbers. First, a vector of numbers.

```
numeric.vecter <- c(1,1,1,1,3,4,4,4,4)
```

Second, a table, showing how many times each number occurs.

```
table(numeric.vecter)
```

```
## numeric.vecter
## 1 3 4
## 4 1 4
```

5.4.5.2 Tables - further details

To access elements in a table variable, you need to use double square brackets, just like accessing elements in a list. For example, to access the fourth element in the table bases.table (the number of Ts in the sequence), we type:

```
bases.table[[4]]  # double brackets!
```

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

Alternatively, you can use the name of the fourth element in the table ("John") to find the value of that table element:

```
bases.table[["T"]]
```

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

5.5 Arguments

Functions in R usually require **arguments**, which are input variables (i.e.. objects) that are **passed** to them, which they then carry out some operation on. For example, the log10() function is passed a number, and it then calculates the log to the base 10 of that number:

```
log10(100)
```

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

There's a more generic function, log(), where we pass it not only a number to take the log of, but also the specific **base** of the logarithm. To take the log base 10 with the log() function we do this

```
log(100, base = 10)
```

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

We can also take logs with other bases, such as 2:

```
log(100, base = 2)
```

```
## [1] 6.643856
```

5.6 Help files with `help()` and `?`

In *R*, you can get help about a particular function by using the help() function. For example, if you want help about the log10() function, you can type:

```
help("log10")
```

When you use the help() function, a box or web page will show up in one of the panes of RStudio with information about the function that you asked for help with. You can also use the ? next to the function like this

```
?log10
```

Help files are a mixed bag in R, and it can take some getting used to them. An excellent overview of this is Kieran Healy's "How to read an R help page."

5.7 Searching for functions with `help.search()` and `RSiteSearch()`

If you are not sure of the name of a function, but think you know part of its name, you can search for the function name using the help.search() and RSiteSearch() functions. The help.search() function searches to see if you already have a function installed (from one of the R packages that you have installed) that may be related to some topic you're interested in. RSiteSearch() searches *all* R functions (including those in packages that you haven't yet installed) for functions related to the topic you are interested in.

For example, if you want to know if there is a function to calculate the standard deviation (SD) of a set of numbers, you can search for the names of all installed functions containing the word "deviation" in their description by typing:

```
help.search("deviation")
```

Among the functions that were found, is the function sd() in the stats package (an R package that comes with the base R installation), which is used for calculating the standard deviation.

Now, instead of searching just the packages we've have on our computer let's search all R packages on CRAN. Let's look for things related to DNA. Note that RSiteSearch() doesn't provide output within RStudio, but rather opens up your web browser for you to display the results.

```
RSiteSearch("DNA")
```

The results of the RSiteSearch() function will be hits to descriptions of R functions, as well as to R mailing list discussions of those functions.

5.8 More on functions

We can perform computations with R using objects such as scalars and vectors. For example, to calculate the average of the values in the vector myvector (i.e.. the average of 8, 6, 9, 10 and 5), we can use the mean() function:

```
mean(myvector) # note: no " "
```

```
## [1] 7.6
```

We have been using built-in R functions such as mean(), length(), print(), plot(), etc.

5.8.1 Writing your own functions

NOTE: *Writing your own functions is an advanced skills. New users can skip this section.

We can also create our own functions in R to do calculations that you want to carry out very often on different input data sets. For example, we can create a function to calculate the value of 20 plus square of some input number:

```
myfunction <- function(x) { return(20 + (x*x)) }
```

This function will calculate the square of a number (x), and then add 20 to that value. The return() statement returns the calculated value. Once you have typed in this function, the function is then available for use. For example, we can use the function for different input numbers (e.g.. 10, 25):

```
myfunction(10)
```

```
## [1] 120
```

5.9 Quiting R

To quit R either close the program, or type:

```
q()
```

5.10 Links and Further Reading

Some links are included here for further reading.

For a more in-depth introduction to R, a good online tutorial is available on the "Kickstarting R" website, cran.r-project.org/doc/contrib/Lemon-kickstart.

There is another nice (slightly more in-depth) tutorial to R available on the "Introduction to R" website, cran.r-project.org/doc/manuals/R-intro.html.

Chapter 3 of Danielle Navarro's book is an excellent intro to the basics of R.

Chapter 6

Logarithms in R

By Nathan Brouwer

Logging splits up multiplication into addition. So, log(m*n) is the same as log(m) + log(n)

You can check this

```
m<-10
n<-11
log(m*n)
```

```
## [1] 4.70048
```

```
log(m)+log(n)
```

```
## [1] 4.70048
```

```
log(m*n) == log(m)+log(n)
```

```
## [1] TRUE
```

Exponentiation undos logs

```
exp(log(m*n))
```

```
## [1] 110
```

```
m*n
```

```
## [1] 110
```

The key equation in BLAST's E values is

u = ln(K*m*n)/lambda

This can be changed to

[ln(K) + ln(mn)]/lambda

We can check this

```
K <- 1
m <- 10
n <- 11
lambda <- 110

log(K*m*n)/lambda

## [1] 0.04273164

(log(K) + log(m*n))/lambda

## [1] 0.04273164

log(K*m*n)/lambda == (log(K) + log(m*n))/lambda

## [1] TRUE
```

Part I

Appendices

Appendix 01: Getting access to R

6.1 Getting Started With R and RStudio

- R is a piece of software that does calculations and makes graphs.
- RStudio is a GUI (graphical user interface) that acts as a front-end to R
- Your can use R directly, but most people use a GUI of some kind
- RStudio has become the most popular GUI

The following instructions will lead you click by click through downloading R and RStudio and starting an initial session. If you have trouble with downloading either program go to YouTube and search for something like "Downloading R" or "Installing RStudio" and you should be able to find something helpful, such as "How to Download R for Windows".

6.1.1 RStudio Cloud

TODO: Add RStudio cloud

6.1.2 Getting R onto your own computer

To get R on to your computer first go to the CRAN website at https://cran.r-project.org/ (CRAN stands for "comprehensive R Archive Network"). At the top of the screen are three bullet points; select the appropriate one (or click the link below)

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

Each page is formatted slightly differently. For a current Mac, click on the top link, which as of 8/16/2018 was "R-3.5.1.pkg" or click this link. If you have an older Mac you might have to scroll down to find your operating system under "Binaries for legacy OS X systems."

For PC select "base" or click this link.

When its downloaded, run the installer and accept the defaults.

6.1.3 Getting RStudio onto your computer

RStudio is an R interface developed by a company of the same name. RStudio has a number of commercial products, but much of their portfolio is freeware. You can download RStudio from their website www.rstudio.com/ . The download page (www.rstudio.com/products/rstudio/download/) is a bit busy because it shows all of their commercial products; the free version is on the far left side of the table of products. Click on the big green DOWNLOAD button under the column on the left that says "RStudio Desktop Open Source License" (or click on this link).

This will scroll you down to a list of downloads titled "Installers for Supported Platforms." Windows users can select the top option RStudio 1.1.456 - Windows Vista/7/8/10 and Mac the second option RStudio

1.1.456 - Mac OS X 10.6+ (64-bit). (Versions names are current of 8/16/2018).

Run the installer after it downloads and accept the default. RStudio will automatically link up with the most current version of R you have on your computer. Find the RStudio icon on your desktop or search for "RStudio" from your task bar and you'll be read to go.

6.1.4 Keep R and RStudio current

Both R and RStudio undergo regular updates and you will occasionally have to re-download and install one or both of them. In practice I probably do this about every 6 months.

Getting started with R itself (or not)

Vocabulary

- console
- script editor / source viewer
- interactive programming
- scripts / script files
- .R files
- text files / plain text files
- command execution / execute a command from script editor
- comments / code comments
- commenting out / commenting out code
- stackoverflow.com
- the rstats hashtag

R commands

- c(…)
- mean(…)
- sd(…)
- ?
- read.csv(…)

This is a walk-through of a very basic R session. It assumes you have successfully installed R and RStudio onto your computer, and nothing else.

Most people who use R do not actually use the program itself - they use a GUI (graphical user interface) "front end" that make R a bit easier to use. However, you will probably run into the icon for the underlying R program on your desktop or elsewhere on your computer. It usually looks like this:

ADD IMAGE HERE

The long string of numbers have to do with the version and whether is 32 or 64 bit (not important for what we do).

If you are curious you can open it up and take a look - it actually looks a lot like RStudio, where we will do all our work (or rather, RStudio looks like R). Sometimes when people are getting started with R they will accidentally open R instead of RStudio; if things don't seem to look or be working the way you think they should, you might be in R, not RStudio

6.1.4.1 R's console as a scientific calculator

You can interact with R's console similar to a scientific calculator. For example, you can use parentheses to set up mathematical statements like

```
5*(1+1)
```

```
## [1] 10
```

Note however that you have to be explicit about multiplication. If you try the following it won't work.

```
5(1+1)
```

R also has built-in functions that work similar to what you might have used in Excel. For example, in Excel you can calculate the average of a set of numbers by typing "=average(1,2,3)" into a cell. R can do the same thing except

- The command is "mean"
- You don't start with "="
- You have to package up the numbers like what is shown below using "c(…)"

```
mean(c(1,2,3))
```

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

Where "c(…)" packages up the numbers the way the mean() function wants to see them.

If you just do the following R will give you an answer, but its the wrong one

```
mean(1,2,3)
```

This is a common issue with R – and many programs, really – it won't always tell you when somethind didn't go as planned. This is because it doesn't know something didn't go as planned; you have to learn the rules R plays by.

6.1.4.2 Practice: math in the console

See if you can reproduce the following results

Division

```
10/3
```

```
## [1] 3.333333
```

The standard deviation

```
sd(c(5,10,15)) # note the use of "c(...)"
```

```
## [1] 5
```

6.1.4.3 The script editor

While you can interact with R directly within the console, the standard way to work in R is to write what are known as **scripts.** These are computer code instructions written to R in a **script file.** These are save with the extension **.R** but area really just a form of **plain text file.**

To work with scripts, what you do is type commands in the script editor, then tell R to **excute** the command. This can be done several ways.

First, you tell RStudio the line of code you want to run by either * Placing the cursor at the end a line of code, OR * Clicking and dragging over the code you want to run in order highlight it.

Second, you tell RStudio to run the code by * Clicking the "Run" icon in the upper right hand side of the script editor (a grey box with a green error emerging from it) * pressing the control key ("ctrl)" and then then enter key on the keyboard

The code you've chosen to run will be sent by RStudio from the script editor over to the console. The console will show you both the code and then the output.

You can run several lines of code if you want; the console will run a line, print the output, and then run the next line. First I'll use the command mean(), and then the command sd() for the standard deviation:

```
mean(c(1,2,3))
```

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

```
sd(c(1,2,3))
```

```
## [1] 1
```

6.1.4.4 Comments

One of the reasons we use script files is that we can combine R code with **comments** that tell us what the R code is doing. Comments are preceded by the hashtag symbol #. Frequently we'll write code like this:

```
#The mean of 3 numbers
mean(c(1,2,3))
```

If you highlight all of this code (including the comment) and then click on "run", you'll see that RStudio sends all of the code over console.

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

Comments can also be placed at the *end* of a line of code

```
mean(c(1,2,3)) #Note the use of c(...)
```

Sometimes we write code and then don't want R to run it. We can prevent R from executing the code even if its sent to the console by putting a "#" *infront* of the code.

If I run this code, I will get just the mean but not the sd.

```
mean(c(1,2,3))
#sd(c(1,2,3))
```

Doing this is called **commenting out** a line of code.

6.2 Help!

There are many resource for figuring out R and RStudio, including

- R's built in "help" function
- Q&A websites like **stackoverflow.com**
- twitter, using the hashtag #rstats
- blogs
- online books and course materials

6.2.1 Getting "help" from R

If you are using a function in R you can get info about how it works like this

```
?mean
```

In RStudio the help screen should appear, probably above your console. If you start reading this help file, though, you don't have to go far until you start seeing lots of R lingo, like "S3 method","na.rm", "vectors". Unfortunately, the R help files are usually not written for beginners, and reading help files is a skill you have to acquire.

For example, when we load data into R in subsequent lessons we will use a function called "read.csv"

Access the help file by typing "?read.csv" into the console and pressing enter. Surprisingly, the function that R give you the help file isn't what you asked for, but is read.table(). This is a related function to read.csv, but when you're a beginner thing like this can really throw you off.

Kieran Healy as produced a great cheatsheet for reading R's help pages as part of his forthcoming book. It should be available online at http://socviz.co/appendix.html#a-little-more-about-r

6.2.2 Getting help from the internet

The best way to get help for any topic is to just do an internet search like this: "R read.csv". Usually the first thing on the results list will be the R help file, but the second or third will be a blog post or something else where a usually helpful person has discussed how that function works.

Sometimes for very basic R commands like this might not always be productive but its always work a try. For but things related to stats, plotting, and programming there is frequently lots of information. Also try searching YouTube.

6.2.3 Getting help from online forums

Often when you do an internet search for an R topic you'll see results from the website www.stackoverflow.com, or maybe www.crossvalidated.com if its a statistics topic. These are excellent resources and many questions that you may have already have answers on them. Stackoverflow has an internal search function and also suggests potentially relevant posts.

Before posting to one of these sites yourself, however, do some research; there is a particular type and format of question that is most likely to get a useful response. Sadly, people new to the site often get "flamed" by impatient pros.

6.2.4 Getting help from twitter

Twitter is a surprisingly good place to get information or to find other people knew to R. Its often most useful to ask people for learning resources or general reference, but you can also post direct questions and see if anyone responds, though usually its more advanced users who engage in twitter-based code discussion.

A standard tweet might be "Hey #rstats twitter, am knew to #rstats and really stuck on some of the basics. Any suggestions for good resources for someone starting from scratch?"

6.3 Other features of RStudio

6.3.1 Ajusting pane the layout

You can adjust the location of each of RStudio 4 window panes, as well as their size.

To set the pane layout go to 1. "Tools" on the top menu 1. "Global options" 1. "Pane Layout"

Use the drop-down menus to set things up. I recommend 1. Lower left: "Console"" 1. Top right: "Source" 1. Top left: "Plot, Packages, Help Viewer" 1. This will leave the "Environment…" panel in the lower right.

6.3.2 Adjusting size of windows

You can clicked on the edge of a pane and adjust its size. For most R work we want the console to be big. For beginners, the "Environment, history, files" panel can be made really small.

6.4 Practice (OPTIONAL)

Practice the following operations. Type the directly into the console and execute them. Also write them in a script in the script editor and run them.

Square roots

```
sqrt(42)
```

```
## [1] 6.480741
```

The date Some functions in R can be executed within nothing in the parentheses.

```
date()
```

```
## [1] "Sun Aug 15 15:36:40 2021"
```

Exponents The ^ is used for exponents

```
42^2
```

```
## [1] 1764
```

A series of numbers A colon between two numbers creates a series of numbers.

```
1:42
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
```

logs The default for the log() function is the natural log.

```
log(42)
```

```
## [1] 3.73767
```

log10() gives the base-10 log.

```
log10(42)
```

```
## [1] 1.623249
```

exp() raises e to a power

```
exp(3.73767)
```

```
## [1] 42.00002
```

Multiple commands can be nested

```
sqrt(42)^2
log(sqrt(42)^2)
exp(log(sqrt(42)^2))
```