A stats book

Marc Kissel and Kaitie Schuping

2019-10-14

Contents

Preface

This is the very first part of the book.

Chapter 1

Introduction

An Introduction to Statistical Software

Unit Questions:

1. Why Use R? (R versus Excel)
2. What is the role of statistics in Anthropology?
3. How do you formulate statistical questions?
4. How do you download the program?

R is an interactive statistical program that assists in the organization and analysis of data. R is a language that exists within an environment. The word environment is associated with the R software in that R allows the user to run linear models, nonlinear regressions, time series analysis, parametric tests, etc. within an open network. The R environment is a "planned and coherent system, rather than an incremental accretion of a very specific and inflexible tools, as is frequently the case with other data analysis software." (cran.r-project.org).

R differs from **Excel** in that it technically is a more user-friendly program [see chpt XXX on spreadsheets] . While Excel may appear to be an easier software to use, if a mistake or problem occurs in Excel, it can be very hard to find the origin of the issue and correct it. R does have a learning curve and it can be intimidating, but it can be a really amazing data tool once the basics are understood. R can do everything that excel does and more. Since many people prefer to use Excel, many of the data files that you will use and upload into R will be stored in an unreadable format. In order to upload an Excel file into R, you need to save the Excel file as a csv. There are packages that allow you to *read in* an Excel file as well.

The word "code" refers to the text used in computer programming language which is called the source code. Metadata is the data about data, it contains analytical descriptions about the original data. (explain base R verses Tidy Verse)

There are multiple packages available for download within R. These include base R, compiler, datasets, grDevices, graphics, graid, methods, parallel, splines, stats, stats4, tcltk, tools, utils.

Most of readR's functions are concerned with turning flat files into data frames. The command read_csv( ) creates a pathway by which a file is able to be transferred and read within the R environment.

It is important to note that everything within R is an object. Everything object within R is either a call, an expression, a name or a null.

There is a lot of available information on how to use R but filtering through online search results can be a challenge. Almost all of your questions can be answered by a google search and there are many online forums where R experts discuss the problems that they have had with R.

If posed with a problem, try not to think big as it is likely that there is a singular small problem causing your issue.

As we learned R we found ourselves struggling a lot. Don't think that you are alone.

1.1 Ok, how do i get R?

1.1.1 install R and R Studio

- https://cloud.r-project.org, (this shld make most of the install automatic)

note: I have multiple verions of R on computer. sometimes useful for packages that don't work w/ certain versions

install R studio here: http://www.rstudio.com/download. - get the free version

1.2 R Studio

R studio is an integrated development environment (IDE) for R that makes it easier to work with R.

It also does a lot of other things, likes let you build websites, create documents, and makes slides all with embeded stats and visualizations

There are different panels. + the source - this is where we we write most of our code. R Studio allows for all sorts of tips and tricks here. That is one of the reasons we use this and not R itself + The console - this is where the R code is run. all lines in the console start with a '>'. You can send stuff from the source to the console in a few ways. this is what R looks like without R studo

lets look at what we can do in the panel - hit up to do histor - text then ctrl up - tab completion

- the history - shows what has been run
- the Enviornment
- plots
- and many others that become useful over time…

> Examine each of these and think about what they do?

1.2.1 lets look around…

1. it should open and see panels
2. if you go to profile -> Global options can see all kinds of stuff

- take a look at what is there

3. general layout of the program….what each pane is and how it works
4. where is R looking for stuff?

1.3 R projects

Probably the best thing you can do is learn to think in terms of **projects**. When we make a project we create a self-contained unit

This helps us with a key feature - reproducibility

If you have ever used Excel you may have made a chart. But can you remake that chart? not always easy. In R since everything is written down it is super easy to update/fix/edit things on the fly. This is a happy thing, especially if you want to ever publish something

1.4 Short overview of how R works

One way that i think about R that helps me to work it is to remember that everything in R is an "object." What this means is that in R we make objects (or we assign values to objects) and then do things to these objects. for example check out the code below

```
a <- 4
```

what this says is that we are going to create the object 'a' and then assign the value '4' to it. if you run the code in R Studio you will see that the 'environment' panel now lists 'a' and gives its value.

1.4.1 Incomplete Code

If you mess up (e.g. leave off a parenthesis), R might show a `+` sign prompting you to finish the command:

```
> (11+17-8
+
```

Finish the command or hit `Esc` to get out of this.

look at these fucntions and think about what they will do before running them.. LIST HERE

2. some data analysis (http://datacarpentry.org/semester-biology/assignments/r-intro/) use the bird banding example maybe but turn it into an archy example

take a dataset and import it into R. then do something to it. use my example above as a start

1.5 lets practice

```
#library(ggnet)
```

1. start R and open the project folder
2. open a new R script and title it LASTNAME_FIRSTNAME_HW1
3. follow along with the info below. when it is time to do the excersie simply type in the Source and then run the command. This way you have a easy way to keep track of what you did

to keep track of where you are, use # to make a comment like this

```r
# question 1
5 + 3
```

```
## [1] 8
```

4. make sure to save your file often
5. when you reach the end, save the file again!

note: some of this comes from datacamp and other sources

1.5.0.1 Running Code

The first thing we want to do is to learn how to run something in R. take a look at the code below:

```r
6 + 7
```

```
## [1] 13
```

the part in grey is the code that is entered into R. If you copy that over to your R console and hit enter you should see the answer

a. how can you send a command from the Source pane to the Console Pane?

1.5.0.2 Basic math

Now lets learn how to do basic math. Using whatever sources you can think of, figure out how to add, subtract, divide, multiply, take an exponent (i.e. square a number), get the square root. In other words, lets learn how to take an expensive computer and turn it into a $10 calculator

> solve the following using R

a. 3-4
b. 7 divided by 10
c. 6 times 89
d. 8 rasied to the 7th power
e. the square root of 52

1.5.0.3 Assigning variables:

One of the key things you will do is assign a value to an object.

To do this we use this symbol <-.

```r
x <- 42
```

Read the above like: "assign the value of 42 to the object x." or "x gets the value 42."

a. How would you create a variable y and assign it the value 334
b. How you you add the x and y together to get the sum of the to numbers?
c. How would you create a new varibale (x) that stores the result of x +y
d. Now, change the value of x to be 500 and see what happens when you add x + y together? if you ask for the value of x now what do you get? why?
e. What are the rules for what the name of an object in R can be?

1.5.0.4 Types of data

Whe you assign a value to a variable, it is given a specific *class*. the class is VERY important and is probably the #1 reason for having troubles with R. it is always good to check the class of an object. to do this, we can use a specific *function* in R called class

class types: (note: maybe make a table?) 1. intergers - 'natural numbers': 5, 6, 987 2. numerics - decimal numbers: 4.5, 8.76666 3. characters - text (sometimes called strings): "hello", "goodbye" 4. logical - boolean: True or False 5. factors - categorical data. This is different from characters because factors are given numbers (or *levels*) that are assocated with that factor and then used for analysis.....we will come back to this later

```
x <- 42
class(x)
```

```
## [1] "numeric"
```

Note that the class of X here is numeric rather than an interger (which might not be want you expected). why? well, it has to do with some inside stuff on computer languages but in general R is going to store everything as numeric unless you tell it differently. In order to make it be an interger we need to use a trick, which is to add the suffix L to the number

```
xx <- 42L
is.integer(xx)
```

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

a. assign my_value to be "hello". then check its class
b. you can also ask R directly if something is a specific type.

```
my_value <- 1+ 3
is.numeric(my_value)
```

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

```
my_name <- "Marc Kissel"
is.numeric(my_name)
```

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

```
is.character(my_name) #and how!
```

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

a. below is some R code. guess what each type of object will be and then use R to find the answer

```
a <- 1.333
b <- TRUE
c <-  "my name is"
d <- Sys.Date() # tricky
```

remember: A class defines what kinds of operations can be implemented on an object & how a function will return a value.

It is important to keep track of the classes of your objects. Class mistakes are probably the most common kind of problem in R

1.5.0.5 What's your vector, Victor?

so far we have only stored one value into an object. but most of the time we are going to have to work with a lot of data. say you have a series of numbers and want to add 7 to each of them. it would be a pain to have to do that manually. we can use r to store a series of vaules (called a *vector*)

```
my_vector <- c(1,2,3,4,5)
my_new_vector <- c(6,7,8,9,10)
```

let's say you are doing some research and want to record the biological sex of the skeltons in your study. after analyzing them you decide that this is the correct designation

male male female male female female female

a. make a new object called **my__study** and make it a vector of the recorded biological sex
b. print the object to R

congrats! you now have a *vector*!

lets say you want to figure out how what the sex of the 5th skeleton is. you could print the object and count, but that takes time and an get difficult. R makes things easier for people like me who are lazy and what the computer to do it all.

to get an *element* from a list we use square brackets

```
my_study[2]
```

```
##    two
## "male"
```

c. how would you get the sex of the third skeleton?
d. can you figure out how, in one line of code, to get the sex of the 1st and 4th skeleton?

Ok, but it is kinda confusing what these stand for. you know that they are skeletons 1-7, but maybe someone else doesn't. We can assign names to objects using a special function called *names*

```
names(my_study) <- c("one", "two", "three", "four", "five", "six", "seven")
```

d. print my__study now and see how it differs.

Note: now that they have names we can also get the values that way

```
my_study[2]
```

```
##    two
## "male"
```

```
my_study["two"]
```

```
##    two
## "male"
```

```
#these are the same

my_study[2] == my_study["two"] # the '==' asks R to tell you if the value on the left and the value on
```

```
##  two
## TRUE
```

to be fair, most of the time there are easier ways to name things, but having a basic understanding of how R works helps a lot

1.5.0.6 Matrices

A vector is simply a list of numbers A matrix is a **rectangular** array of numbers

```
cx1980 <- c(7, 13, 8, 13, 5, 35, 9)
cx1988 <- c(9, 11, 15, 8, 9, 38, 0)
chimp <- cbind(cx1980, cx1988) # cbind binds the vectors together a columns
class(chimp)
```

```
## [1] "matrix"
```

```
chimp
```

```
##      cx1980 cx1988
## [1,]      7      9
## [2,]     13     11
## [3,]      8     15
## [4,]     13      8
## [5,]      5      9
## [6,]     35     38
## [7,]      9      0
```

One thing you want to learn is how to read a matrix and identify elements. Lets say you want to get a vector of just the ages from 1980. we can use the square brackets again but need to know a trick.

```
chimp[1,] #note the comma
```

```
## cx1980 cx1988
##      7      9
```

```
chimp[,1]
```

```
## [1]  7 13  8 13  5 35  9
```

```
#putting the number before the comma gets us the row. putting it after the comma gets column.
#one way to remeber that Rows come first is the menominc Railway Cars
chimp[3,2] # third row, second column
```

```
## cx1988
##     15
```

a. how would you make a matrix by row rather than column (use google if need be)
b. There are others ways to make a matirx look at the code below and figure out how it works

```
freq <- c(32,11,10,3,  38,50,25,15,  10,10,7,7,  3,30,5,8)
hair <- c("Black", "Brown", "Red", "Blond")
eyes <- c("Brown", "Blue", "Hazel", "Green")
freqmat <- matrix(freq, nr=4, nc=4, byrow=TRUE)
dimnames(freqmat)[[1]] <- hair
dimnames(freqmat)[[2]] <- eyes
freqmat
```

```
##       Brown Blue Hazel Green
## Black    32   11    10     3
## Brown    38   50    25    15
## Red      10   10     7     7
## Blond     3   30     5     8
```

Now, create your own matrix with madeup data…

1.5.0.7 Dataframes

A dataframe stores data! it can hold different kinds of varibales/classes so it is different from a matrix. You cna think of it as a list of varibales that are all the same length. Data frames are probably the most common way we will work with R

```
bone <- c("humerus", "radius", "ulna", "femur", "tibia", "fibula")
size_inches <- c(14.4, 10.4, 11.1, 19.9, 16.9, 15.9)
injury <- sample(c("yes","no"),6,replace=TRUE)
sample_letter <- LETTERS[1:6]
my_sample <- data.frame(bone, size_inches, injury, sample_letter)
my_sample
```

```
##      bone size_inches injury sample_letter
## 1 humerus        14.4    yes             A
## 2  radius        10.4     no             B
## 3    ulna        11.1     no             C
## 4   femur        19.9     no             D
## 5   tibia        16.9    yes             E
## 6  fibula        15.9    yes             F
```

If you use the function *View* you can see a spreadsheet of the data frame you just made

```
View(my_sample)
```

you can view a specifc column/vector using the $

```r
my_sample$bone
```

```
## [1] humerus radius  ulna    femur   tibia   fibula
## Levels: femur fibula humerus radius tibia ulna
```

a.run the code below. it should show an error. Why? rewrite the code so it works!

NOTE: make this appear but not run?

```r
#num <- c(1,2,3,4,5)
#food <- c("bread", "butter", "milk", "cheese","coffee", "tea")
#quantity <-  c(1,1,3,5,7,1)
#shopping <- data.frame(num, food, quantity)
```

b. what is the class type of the different vectors in the my_sample dataframe?

1.5.0.8 Comparing values

Often times we are going to want to compare things.

a. run the code below

```r
a <- 5
b <- 9
c <- 7
d <- sqrt(49)
```

Now, figure out how to have R evaluate the following: - is a bigger than b? - is c equal to d (careful with this one..) - is c less than or equal to b?

b. make a new vector called *temp* with the values of 1,5,7,9,11,14,6,8. then write a single line of code that evalautes if 3 is greater than each of the values in the vector

c. how would you ask R if the 5th value in *temp* is larger than 5?

Chapter 2

Technical Details

here we can put a full code for students to check out

2.1 notes while working on book

currently trying different formats used msmbstyle (https://github.com/grimbough/msmbstyle) by installing package via devtools::install_github("grimbough/msmbstyle")

Chapter 3

How Computers (and R) Think

What is going on behind the scenes

This chapter has a lot of background info on using computers. A lot of this is basically random tips and tricks and info that we've picked up over the years, Mostly as non-computer science folks coming across terms and ideas that are basic in the comp-sci world but not in the anthropology world. In other words, this is a repository of things that you might one day care about. As always a work in progress so let us know if something is missing/wrong/needs to be updated etc.

3.1 file/folder management

One of the first things you want to start doing to work more efficiently in R (and in general) is to think about how best to manage files and folders. Once you get the hang of R it is easy to get a data set and just dump it into a temp folder to check it out. But this can get messy. This section gives tips on how to name and use file management so that your code is easy to use by co-authors (which could include yourself 2 years from now)

3.1.1 file names

In programming it is often a bad idea to have spaces in function names. It also is sometimes difficult to 'read in' files with spaces in their names.[1] Because of this many times we don't want spaces in file names. There are a few ways to combine words, however, to make it easy to read

Name	example
Snake case	my_new_data
Camel case	myNewData
Kebab case	my-new-data
Pascal case	MyNewData

> Study question: What are the differences between these four types? Which do you like the best? Choose one and stick with that!

It isn't always easy, but keeping consistent file naming helps in the long run. It also helps to know a bit

[1]This is because a space often means something special in computer languages.

about how computers organize files. If you want a series of files to be able to be sorted in order a good trick is to label them like this:

- 01_first
- 02_second
- 03_third
- 04_fourth

This allows you to sort the files by name and have them show up in order

> Thinking question: What are some examples of *bad* file names? in other words, what would be an example of a file name that wouldn't help the you understand what is in it

As you know @ref(R_basics) {FIX}

As you know **??**

3.1.2 dates

Dealing with dates is not easy

```r
knitr::include_graphics("images/iso_8601.png")
```

The ISO 8601 standard is YYYY-MM-DD.

Computers deal with dates in different ways depending upon the operating system. However, in general they rely on **system time**, which is the amount of time (in seconds or nanoseconds) that have elapsed since a certain day. If you find yourself working with date data it is useful to learn more about this. In chpt XXX we talk a bit about this in terms of how Excel can cause problems with date time fields

3.2 file types

A computer file is, in esscene, a string of 1s and 0s. Computers can tell what a file is (is it an image, a text, a video) by the **extention**, which is the name after the period in a full file name. For example, for the file *nyc_temp_data.txt*, the ".txt" tells the computer the file is a text file. MAYBE HAVE IMAGE HERE

3.3 types of files you might see in the wild and what they are

- JSON - JavaScript Object Notation
  - This is common way to store info on webpages
  - it looks like this, with a key and a value: {“name”: “Mary”, “Major”: “Anthropology”}
  - To read JSON files into R you can use the **jsonlite** package and the *fromJSON* function.[2]
- EXE - executable file
  - a file that can be run by clicking on it
- CSV - Comma-separated values
  - This is one of the most common ways to send and receive data. A CSV file is a simple text file that uses commas to deliminate, or separate, values. Such files store numbers and text only. It is useful since anyone can open them and doesn’t require proprietary software.
- sql - Structured Query Language
  - Used for buidling databases
  - where you might see it: exported data from Filemaker?
- html
- zip
- tar - Tape ARchive
  -
- tar.gz - Tape ARchive compressed with gzip, a Unix function
  - compressed file that o ften has multiple files stored in a single file
- iso -
  - contains image that was from an optical disc like a DVD. a copy of everything on that disk
  - You may want to *mount* the ISO file, which means it gets treated as if you are opening the real disc. you can also burn the ISO file onto an actual DVD
  - Where you might see it: Large files, installing operating systems
- md - Markdown
  - A file written in any one of several types of *Markdown language*, which allows someone to write a file in a text document but then convert to another format like HTML or PDF.
  - Where you might see it: GitHub,
- rmd - R Markdown
  - Markdown file that easily runs R code.
  - Fun fact: this book was written in R Markdown
  - Where you might see it: The chpts of this book

[2]for more info: https://stackoverflow.com/questions/16947643/getting-imported-json-data-into-a-data-frame

3.4 Accessing internet resources

3.4.1 what is an api

An API is a “application programming interface.” For most of what we will use, we can think of it as a a code that lets us access information. we can send a request for the info we want

hints for api: load ‘usethis’ run edit_r_environ() add api there see: https://usethis.r-lib.org/articles/articles/usethis-setup.html

#accessing/using the terminal

#computer languages you might come across Python

#terms GUI IDE

#useful programs Wox - for PC. lets you open files easy and search a bit

#video and images

3.5 refs for this section

(Refs: https://www.tidyverse.org/articles/2017/12/workflow-vs-script/

Chapter 4

Working with Spreadsheets

Spreadsheets are interactive computer applications for organization, analysis, and storage of data in tabular form. They consist of a table of cells arranged into rows and columns. A cell is simply a box that holds data. Most spreadsheets use a standard convention where columns are are represented by letters, “A”, “B”, “C” and rows are normally represented by numbers, 1, 2, 3,.

(image from: https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwjAk6Wd3pnlAhUKKqwKHfnmBAkQjRx6BAgBEAQ&url=https%3A%2F%2Fstablemanagement.com%2Farticles%2Ftips-stable-owners-creating-spreadsheet-54801&psig=AOvVaw1IGNVA8ljGXcio0Vrdvhy8&ust=1571073279950004)

In most math and stats courses, we are taught to refer to the row first and then the column. This comes from linear algebra (the standard mnemonic is to remember the word “RailwayCars”, where the R comes before the C).

The fact is that you are actually looking at a spreadsheet now! The screen you are reading this on is a sort of spreadsheet, with the ‘cells’ filled with different values. The computer then translated this to color and shade. This is how digital photos etc work. Here is a fun video that talks about how pics on a computer are spreadsheets…

https://www.youtube.com/watch?v=UBX2QQHlQ_I

In general, spreadsheets are fine for storing data, but not so great for analyzing/visualizing data. The reason for this is that the format we use to record data are often made for ease of use of us rather than for ease of use of stats. One of us once got a spreadsheet where different info was connoted by the spaces within each cell. Trying to export that info into a workable format was not easy (see section 6.XXX for how to do this)

The big issue is how can we setup data to be read by both humans **and** computers?

Most people who have used spreadsheet have used Excel. One of the major drawbacks to Excel is its price (for researchers/students in college they can usually get it for free. But as with other examples once you no longer have access it is hard to get it). HOwever, there are many great things about Excel. I recommend this video which teaches a lot of nice tips and tricks:

https://www.youtube.com/watch?v=0nbkaYsR94c

Excel makes it easy for us to read the data but hard for the computer to read the data. And Excel can also cause some major issues. Follow along the example below (if you have Excel) for an example of what we mean

[insert example here from http://ecologybits.com/index.php/2016/07/06/beware-this-scary-thing-excel-can-do-to-your-data/

4.1 How to best use spreadsheets:

> Assignment: read Karl W. Broman & Kara H. Woo (2018) Data Organization in Spreadsheets, The American Statistician, 72:1, 2-10, DOI: 10.1080/00031305.2017.1375989
>
> What examples do they give of bad spreadsheets? What makes a spreadsheet good?
>
> Assignment: Build a spreadsheet using the tips of Browman & Woo

4.2 Types of data used in spreadsheets:

Dates: this is probably one of the largest issues. For example, Excel has 2 different date formats based upon the version of Excel you use. One counts 1900 at the start date and the other counts 1904 (https://support.microsoft.com/en-us/help/214330/differences-between-the-1900-and-the-1904-date-system-in-excel)

As we talked about in the "How computers think" section, 3.1.2 dates work by giving a computer a start time and then calculating how long it has been since that start

Check: [excel-options-advanced]

4.3 problems

Panko (2008) reported that in 13 audits of real-world spreadsheets, an average of 88% contained errors

examples:https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-5-80 https://www.bloomberg.com/news/articles/2013-04-18/faq-reinhart-rogoff-and-the-excel-error-that-changed-history

4.4 Spreadsheet programs

Excel GoogleSheets Numbers LibreOce OnlineCalc Airtable Calligra

Chapter 5

Working with data

How do we manipulate the data to tell a story

1. How to get data from your collaborators into a format for R…
2. how to organize data 3.how to explore data

As we talked about in the previous chapter, most data live in spreadsheets.

5.1 Tidy data

[maybe talk about this in the spreadsheet chpt?]

Data transformation is the key to R. we are going to take data in from a worksheet/csv/whatever and transform it.

There are a number of ways to do this. One method that gives us many options is using a package called *dplyr.* This is part of a larger group of packages known as the *Tidyverse.* we are also going to use a packaged called nycflights13