STAT167 Lab #1 - Spring 2022

Victoria Nguyen

2022/3/31

Contents

Acknowledgment: Part of this R Markdown template is adapted from David Dalpiaz (UIUC).

Discussion week 1 instructions

This week, you will first learn the basic R Markdown syntax.

- First, download the `rmd` file from Canvas.
- Open this `rmd` file in RStudio and click `Knit -> Knit to PDF` to render it to PDF format. You need to have `LaTex` installed on the computer to render it to PDF format. If not, you can also render it to HTML format.
- Read this `rmd` file and the rendered `pdf` file side-by-side, to see how this document was generated!
- Be sure to play with this document! Change it. Break it. Fix it. The best way to learn R Markdown (or really almost anything) is to try, fail, then find out what you did wrong.

Next, you will review some example code from this week's lectures.

- Read over the code and the output. If you have any questions about certain functions or parameters, it is the time to ask!
- There are some exercises through out this document. Replace **INSERT_YOUR_ANSWER** with your own answers. Knit the file to PDF, and check your results.

Please comment your R code thoroughly, and follow the R coding style guideline (https://google.github.io/styleguide/Rguide.xml). Partial credit may be deducted for insufficient commenting or poor coding styles.

Lab submission guideline

- After you completed all exercises, save your file to `FirstnameLastname-SID-lab1.rmd` and save the rendered pdf file to `FirstnameLastname-SID-lab1.pdf`. If you can not knit it to pdf, knit it to html first and then print it to pdf format.
- Submit **BOTH your source `rmd` file and the knitted `pdf` file** to **GradeScope**. Do NOT create a zip file. You can submit multiple times, you last submission will be graded.

RMarkdown Basics

RMarkdown at its core is a combination of `R` and Markdown used to generate reproducible reports for data analyses.

Markdown and `R` are mixed together in a `.rmd` file, which can then be rendered into a number of formats including `.html`, `.pdf`, and `.docx`. There is a strong preference for using `.html` in this course.

Have a look at this `.rmd` to see how this document was generated! It should be read alongside the rendered `.html` to best understand how everything works. You can also modifying the `.rmd` along the way, and see what effects your modifications have.

Formatting text is easy. **This is bold.** *This is italics.* `This text appears as monospaced.`

Test Header 1

Test Header 2

Test Header 3

- Unordered list element 1.

- Unordered list element 2.
- Unordered list element 3.

1. Ordered list element 1.
2. Ordered list element 2.
3. Ordered list element 3.

Packages

Packages are key to using `R`. The community generated packages are a large part of `R`'s success, and it is extremely rare to perform an analysis without using at least some packages. Once installed, packages must be loaded before they are used.

Install necessary packages

```r
# If you can knit the html file successfully, do not change this R code chunk

# Otherwise, uncomment the two lines and install the packages.
# You only need to run the install.packages() commands once.
# Then you can comment them out by adding '#' back at the beginning of each line

#install.packages("rmarkdown", repos="http://cran.rstudio.com/")
#install.packages("yaml", repos="http://cran.rstudio.com/")
```

Note that `rmarkdown` is actually a package in `R`! If `R` never prompts you to install `rmarkdown` and its associated packages when first creating an RMarkdown document, use the above command to install them manually.

Adding `R`

So far we have only used Markdown to create html. This is useful by itself, but the real power of RMarkdown comes when we add `R`. There are two ways we can do this. We can use `R` code chunks, or run `R` inline.

`R` Code Chunks

The following is an example of an `R` code chunk

```r
# generate random normals
set.seed(167) # feel free to change 167 to your lucky number
x <- rnorm(100) # use <- for assignment instead of =
```

There is a lot going on here. In the `.rmd` file, notice the syntax that creates and ends the chunk. Also note that `example_chunk` is the chunk name. Everything between the start and end syntax must be valid `R` code. Chunk names are not necessary, but can become useful as your documents grow in size.

Inline `R`

`R` can also be ran in the middle of exposition. For example, the mean of the data we generated is -0.094945.

Chunk Options

There are many chunk options. Here we first introduce two options which are frequently used: `eval` and `echo`.

```
?log
x
```

Using `eval = FALSE` the above chunk displays the code, but it is not run. The `?` code pulls up documentation of a function. The `x` code prints all values inside the vector x. With `eval = FALSE` (or simply `eval = F`), there is no output displayed.

```
## [1] "Hello World!"
```

Above, we see output, but no code! This is done using `echo = FALSE` (or simply, `echo=F`), which is often useful.

Adding Math with `LaTeX`

Another benefit of RMarkdown is the ability to add Latex for mathematics typesetting. Like `R` code, there are two ways we can include Latex; displaystyle and inline.

Note that use of `LaTeX` is somewhat dependent on the resulting file format. For example, it cannot be used at all with `.docx`. To use it with `.pdf` you must have LaTeX installed on your machine.

With `.html` the `LaTeX` is not actually rendered during knitting, but actually rendered in your browser using MathJax.

Displaystyle `LaTeX`

Displaystyle is used for larger equations which appear centered on their own line.

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Inline LaTex

We could mix LaTeX commands in the middle of exposition, for example: $t = 2$. We could actually mix `R` with Latex as well! For example: $\bar{x} = -0.094945$.

Lecture Review

Generate sample data

Recall that in the lecture, we generated 100 random normals, assigned the values into a vector `x`, then printed out some values in vector `x`.

```r
# generate random normals
set.seed(167) # feel free to change 167 to your lucky number
x <- rnorm(100) # use <- for assignment instead of =
# print out the data or part of it
x
```

```
##   [1] -0.99307913 -0.17111961 -0.10200147 -0.14990885 -0.21135557 -0.60078127
##   [7]  0.11739337 -0.21578390 -0.75237199  1.55294726  0.77671331  0.32967895
##  [13] -1.89405513 -1.99349413 -0.22007207  0.19850126  0.56371430 -0.88079442
##  [19] -0.64174738 -0.84737192  1.83233164 -0.53413321  0.30973709  0.01403343
##  [25] -1.09327949 -0.07513613  1.52135816  0.53832708 -0.26987651  0.26677743
##  [31] -0.70915890 -1.27608985  0.52580565  0.71607236 -0.60290030  0.14002073
##  [37] -1.99917225  0.42774676 -0.08996853 -0.23563275 -0.41061735  0.15146705
##  [43] -2.26942347 -0.93500192 -1.41073822 -2.32958937 -0.15649706 -0.94767522
##  [49]  0.73027385  0.41461206  0.65270923  0.64319947 -0.08089847  0.27330967
##  [55] -0.08968567 -0.37876327 -1.15709419  0.58358139 -0.07484820  0.85993573
##  [61]  0.70481069  0.66931003  0.38240296  0.01418130  1.05815293 -0.97596451
##  [67] -0.01815480 -1.60150608 -0.29764517 -0.97331417 -1.53441965  2.09839585
##  [73]  0.95002228  0.09173245  1.54627570 -1.42063598  0.73546857  0.12136776
##  [79]  1.03977273 -1.38440186  0.05776784  0.17496418 -0.19319262  0.06309704
##  [85]  0.52483624  0.12168457  1.60986947  0.96609625 -0.89391965 -0.89837740
##  [91] -1.56742899  1.73821583 -1.21415287  1.44463281 -1.61656091  0.40198802
##  [97]  0.15775583  0.57040588  0.07979506  0.43204664
```

```r
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
10:1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```r
seq(from=1, to=10, by=2)
```

```
## [1] 1 3 5 7 9
```

```r
seq(10, 1)
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```r
x[1:10]
```

```
##  [1] -0.9930791 -0.1711196 -0.1020015 -0.1499089 -0.2113556 -0.6007813
##  [7]  0.1173934 -0.2157839 -0.7523720  1.5529473
```

```r
y <- c(2,4,7)
y
```

```
## [1] 2 4 7
```

```r
x[y]
```

```
## [1] -0.1711196 -0.1499089  0.1173934
```

```r
length(x)
```

```
## [1] 100
```

```r
length(x[y])
```

```
## [1] 3
```

Exercise #1

Look at the output of the following code, what happens when you select with negative index?

```r
x[4]
```

```
## [1] -0.1499089
```

```r
x[-4]
```

```
##   [1] -0.99307913 -0.17111961 -0.10200147 -0.21135557 -0.60078127  0.11739337
##   [7] -0.21578390 -0.75237199  1.55294726  0.77671331  0.32967895 -1.89405513
##  [13] -1.99349413 -0.22007207  0.19850126  0.56371430 -0.88079442 -0.64174738
##  [19] -0.84737192  1.83233164 -0.53413321  0.30973709  0.01403343 -1.09327949
##  [25] -0.07513613  1.52135816  0.53832708 -0.26987651  0.26677743 -0.70915890
##  [31] -1.27608985  0.52580565  0.71607236 -0.60290030  0.14002073 -1.99917225
##  [37]  0.42774676 -0.08996853 -0.23563275 -0.41061735  0.15146705 -2.26942347
##  [43] -0.93500192 -1.41073822 -2.32958937 -0.15649706 -0.94767522  0.73027385
##  [49]  0.41461206  0.65270923  0.64319947 -0.08089847  0.27330967 -0.08968567
##  [55] -0.37876327 -1.15709419  0.58358139 -0.07484820  0.85993573  0.70481069
##  [61]  0.66931003  0.38240296  0.01418130  1.05815293 -0.97596451 -0.01815480
##  [67] -1.60150608 -0.29764517 -0.97331417 -1.53441965  2.09839585  0.95002228
##  [73]  0.09173245  1.54627570 -1.42063598  0.73546857  0.12136776  1.03977273
##  [79] -1.38440186  0.05776784  0.17496418 -0.19319262  0.06309704  0.52483624
##  [85]  0.12168457  1.60986947  0.96609625 -0.89391965 -0.89837740 -1.56742899
##  [91]  1.73821583 -1.21415287  1.44463281 -1.61656091  0.40198802  0.15775583
##  [97]  0.57040588  0.07979506  0.43204664
```

ANSWERS: The positive index only shows one value whereas the negative index shows multiple (99)

Calculate summary statistics

We can calculate some summary statistics from the data. Here we set the option `collapse=T` for our code chunk, so that R Markdown will try to collapse all the source and output blocks from one code chunk into a single block.

```
mean(x)
## [1] -0.09494496
median(x)
## [1] -0.002060686

var(x) # sample variance
## [1] 0.8931129
sd(x) # sample standard deviation
## [1] 0.9450465

n <- length(x)
sum((x-mean(x))^2)/n # population variance
## [1] 0.8841817
mean(x^2)-(mean(x))^2 # population variance
## [1] 0.8841817
var(x)*(n-1)/n # population variance
## [1] 0.8841817
sd(x)^2*(n-1)/n # population variance
## [1] 0.8841817

summary(x)
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -2.329589 -0.776122 -0.002061 -0.094945  0.528936  2.098396
```

Exercise #2

Complete the following code to print out the maximum value and the minimum value in `x`.

ANSWERS

```
min(x)
```

```
## [1] -2.329589
```

```
max(x)
```

```
## [1] 2.098396
```

Exercise #3

The `sd()` function calculate the sample standard deviation.

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

How to calculate the population standard deviation?

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

ANSWERS

```
sd(x) # sample standard deviation s
## [1] 0.9450465

# calculate the population standard deviation
sd(x)*(sqrt((length(x)-1)/length(x)))
## [1] 0.9403094
```

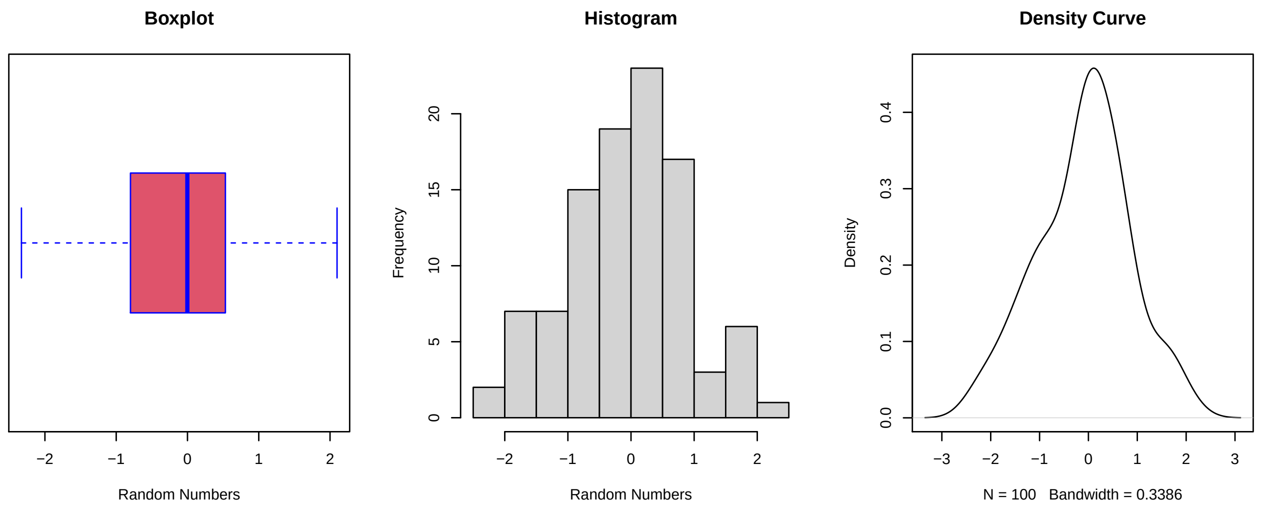
Plot single data set

Note that you can modify chunk options `fig.height` and `fig.width` to change the size of output plots from a particular chunk.

```
par(mfrow=c(1,3)) # 1-by-3 layout
# subfigure #1
boxplot(x, horizontal=T, col=2, border="blue", xlab="Random Numbers", main="Boxplot")

# subfigure #2
hist(x, xlab="Random Numbers", main="Histogram")

# subfigure #3
plot(density(x), main="Density Curve")
```


Boxplot
Histogram
Density Curve
Frequency
Density
Random Numbers
Random Numbers
N = 100   Bandwidth = 0.3386