Introduction to R: Yale Math Camp 2023

August 2023

Contents

1 Downloading R and LaTeXto Personal Computer

Download the latest version of R to your personal laptop here.

To use LaTeXin R Markdown, please also download some sort of Tex-related system onto your computer. I use TexShop.

You can also use Overleaf for your problem sets (as I did here), but you cannot access data, so this is probably not advisable.

2 Difference between R and R Markdown

Once you download R to your laptop, you should open up your console.

You will likely be using R Markdown to typeset your problem sets in 500 and 503. Note that R Markdown is merely an add-on to R–it is not a package.

However, you should try to run all code for problem sets in a R Script, and *not* in R Markdown. Particularly as your code takes longer than a few seconds to run, R Markdown can really slow things down!

Then you can insert a code chunk into the R Markdown script, and you can knit the text.

3 Installing Packages

To install a package in R, there are two main ways:

1. Install directly
2. Use CRAN

I would stick to installing directly. For example, if you wanted to use `fixest`, a common regression package, you need to do the following two steps:

```
install.packages("fixest") # first, install packages


## Installing package into '/usr/local/lib/R/site-library'
## (as 'lib' is unspecified)

## Warning in install.packages("fixest"):  'lib = "/usr/local/lib/R/site-library"' is not
writable

## Error in install.packages("fixest"):  unable to install packages

library(fixest) # library the package


## Error in library(fixest):  there is no package called 'fixest'
```

Once you install a package into R, you will not need to do it again (unless you update R).

However, you will need to library a package every time you run a script in R.

4 Setting Working Directory

You need to set working directory if you plan to use any data, whether this is coming as a .csv file, an Excel document, or something more difficult to work with.

For the purposes of 500, you should create a folder on your computer for problem sets in which any needed data is stored.

```
# setwd(\computer_path_goes_here)
# read.csv(\document_in_computer_path)
```

Note that I comment out these lines with .

To assign the data frame so that it exists in my environment, I need to assign it with the ¡- operator. More on this next.

```
# df <- read.csv(\document_in_computer_path)
```

5 Assigning Variables and Vector Operations

To assign a value in R, the operator $<-$ is generally used. However, $=$ does the same thing.

```
X <- 1:10
print(X)

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

Note also that R is case-sensitive, so this won't work:

```
print(x)

## Error in print(x):  object 'x' not found
```

At first glance, R can be used as a fancy calculator. Some of the operations performed are shown here:

```
y <- 2
z <- 4
y + z

## [1] 6

y * z

## [1] 8

y / z

## [1] 0.5
```

6 Introduction to Functions

Many functions come built into R. For example, to calculate the mean value of a vector X:

```
X = 1:10 # create vector of values [1,10]

print(X)

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

mean(X)

## [1] 5.5
```

However, one can also build their own function in R. Say, for example, that I want to generalize a function so that I can find the mean, standard deviation, and range of a vector, and print these values in a table:

```
X = rnorm(1:10) # select random values within normal dist.
print(X) # print randomly drawn values

##  [1]  0.79134195 -0.31530114 -1.98570079 -2.77734146  1.45875027 -1.16953298
##  [7]  0.84063324  0.09900246  0.82073434  1.77141763

find_descriptives = function(vector) {
mean <- mean(vector)
sd <-  sd(vector)
range <- range(vector)
df <- data.frame(cbind(mean, sd, range))
print(df)
}

find_descriptives(X) # absolutely need to call the function every time

##          mean       sd     range
## 1 -0.04659965 1.504966 -2.777341
## 2 -0.04659965 1.504966  1.771418
```

The general syntax of a function is as follows:

```
name_your_function <- function(x) {
############################
}

name_your_function(x) # calls function

## NULL
```

Note that the value x can be numeric, a string, a dataframe, and a few other objects.

It can be difficult getting used to how to run functions so that the proper object returns. We can run through a few examples of this.

7 Useful Functions in Base R

The following code chunk shows some of the functions that I use the most frequently in base R:

1. cbind() - column binds
2. rbind() - row binds
3. data.frame() - makes into data frame
4. print() - shows object
5. colnames() - can be used to rename column names

Putting this all together:

```
x <- 4
y <- 3
z <- 7

cbind(x, y, z)


##      x y z
## [1,] 4 3 7


rbind(x, y, z)


##   [,1]
## x    4
## y    3
## z    7


df <- data.frame(cbind(x, y, z))
print(df) # print df to examine how it differs from only column-bind


##   x y z
## 1 4 3 7
```

8 Linear Algebra in R

We have so far considered vectors in R.

Several functions are useful in reference to matrices:

1. matrix() (make matrix)
2. t() (take transpose)
3. solve() (take inverse)

```
A <- matrix(c(1, 3, 2, 2, 8, 9), ncol = 3)
print(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    8
## [2,]    3    2    9
```

8.1 Multiplying by Scalar

As we learned, we can multiply the matrix by a scalar value, e.g. 7.

```
7 * A
```

```
##      [,1] [,2] [,3]
## [1,]    7   14   56
## [2,]   21   14   63
```

```
# we know that this is the same as
A*7
```

```
##      [,1] [,2] [,3]
## [1,]    7   14   56
## [2,]   21   14   63
```

8.2 Transpose of A

We can take the transpose of A using the t() function.

```
t(A)
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    2
## [3,]    8    9
```

8.3 Vector * Matrix

Why can't we multiply the following case?

```
a = c(5, 6)
A %*% a
```

```
## Error in A %*% a:  non-conformable arguments
```

```
a = c(5, 6)
a %*% A
```

```
##      [,1] [,2] [,3]
## [1,]   23   22   94
```

8.4 Matrix Multiplication

```
A <- matrix(c(1, 3, 2, 2, 8, 9), ncol = 2)
B <- matrix(c(5, 8, 4, 2), ncol = 2)
A %*% B
```

```
##      [,1] [,2]
## [1,]   21    8
## [2,]   79   28
## [3,]   82   26
```

8.5 Diagonal of Matrix

In certain cases, you may need to take the diagonal of a matrix:

```
B <- diag(1, 3) # creates identity matrix
B
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
diag(B) # takes the diagonal of the identidy matrix
```

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

8.6 Matrix Inversion

In certain cases, you may need to take the diagonal of a matrix:

```r
A <- matrix(c(1, 3, 2, 4), ncol = 2)
A
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

To do so, use the function solve().

8.7 Solving a System of Linear Equations

In class, we learned that we can solve a system of linear equations using A.

```r
A <- matrix(c(1, 2, 3, 4), ncol = 2)
b <- c(7, 10)
x <- solve(A) %*% b
x
```

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

We'll do an example of this in R given a system of equations on the board.

9 Linear Regression

I am considering years of education to be X and income (in $) to be Y.

I have created a fake data set in the following way:

```r
set.seed(1234)
num_samples <- 3 # sample function randomly samples
education <- sample(1:3, num_samples, replace = TRUE) # randomly samples between 0 to 12
income <- sample(5:50, num_samples, replace = TRUE) # randomly samples
df <- data.frame(education = education, income = income)
print(head(df))
```

```
##   education income
## 1         2     48
## 2         2     13
## 3         1      9
```

Your task:

Given X and Y, compute $\hat{\beta} = (X'X)^{-1}(X'Y)$.