R Lab 0 - Introduction to R

Introduction to Causal Inference

Goals:
1. Introduce students to basic functionality in R.
2. Generate data.
3. Download and load an R package.

Next lab:
We will review the structural causal model (SCM), counterfactuals, and causal parameters defined with and without a working marginal structural model (MSM). Given the data generating mechanism, we will then obtain the values of two causal target parameters both analytically and by simulation.

Reminder:
This is not an R class. However, software is an important bridge between the statistical concepts and implementation.

1 Generating Data

We will first generate a test data set. This data set will contain 100 independent and identically distributed observations of the form (W, A, Y) generated as follows:

- $W \sim Unif(-1, 1)$
- $A|W \sim Bernoulli(1/2)$
- $Y|A, W$ is a mixture distribution
  - If $W \geq 0$: $Normal(A, 1)$
  - If $W < 0$: $Exponential(10|W|)$, where $10|W|$ is the rate of the exponential random variable.

We will generate the 100 observations of each variable as a **vector**.

1. Set the seed for the random number generator to 1: `set.seed(1)`.

Solution:

```
> # 1.
> set.seed(1)
```

Why learn about `set.seed()`*?* R mimics randomness based on an algorithm that takes an initial seed as input and outputs a seemingly random string of characters based on this seed. Setting this seed to a fixed value (generally) ensures that running the same chunk of code twice will return the exact same output, even if the code involves "random" numbers. This will also help your TA help you with your labs/homework assignments/projects.

2. Set the number of observations n to 100.
   `Hint: The assignment operator in R can be written as '=' or '<-'. To set the variable x to 0, you could write 'x<-0'.`
3. Generate a vector of n $Uniform(-1, 1)$ random variables and assign them to the vector W.
   `Hint: Try typing '?runif' in your R console.`
4. Generate a vector of n Bernoulli random variables and assign them to the vector A.
   `Hint: A Bernoulli random variable is a binomial random variable with 1 possible success. Use your favorite search engine to find how to generate a binomial random variable in R. You will likely end up on a page that begins like the one below. This is a standard R documentation page.`

Binomial {stats} R Documentation

The Binomial Distribution

Description

Density, distribution function, quantile function and random generation for the binomial distribution with parameters `size` and `prob`.

This is conventionally interpreted as the number of 'successes' in `size` trials.

Usage

```
dbinom(x, size, prob, log = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)
```

Solution:

```
> # 2.
> n <- 100
> # 3.
> W <- runif(n,min=-1,max=1)
> # 4.
> A <- rbinom(n,1,1/2)
```

We now have two vectors of length 100, one for W and one for A. To generate Y, we need to stratify individuals by the sign of W. We will do this two ways. First we will use a **for loop**. In R, for loops and **if statements** have the form

```
> # This is a comment. It will not be run.
> # Note 1:6 outputs a vector containing integers from 1 to 6.
> # We could also write this as c(1,2,3,4,5,6)
> #
> for(i in 1:6){
+   # Output ``Time to get started.'' if i is equal to 1.
+       # Note in the conditional we use '==' rather than a single equal sign.
+   # Output the current value of i in the console if i is greater than 3.
+   # Otherwise print out ``i not 1 or >3''
+   if(i==1){
+     print("Time to get started.")
+   } else if(i>3){
+     print(i)
+   } else {
+     print('i not 1 or >3')
+   }
+ }
```

```
[1] "Time to get started."
[1] "i not 1 or >3"
[1] "i not 1 or >3"
[1] 4
[1] 5
[1] 6
```

We will also need to select elements from a vector.

```
> # Define a vector consisting of integers from 6 to 10
> my.vec <- 6:10
> my.vec
```

```
[1]  6  7  8  9 10
```

```
> # What is the first element of my.vec?
> my.vec[1]
```

```
[1] 6
```

```
> # The third element?
> my.vec[3]
```

```
[1] 8
```

```
> # The first through third elements?
> my.vec[1:3]
```

```
[1] 6 7 8
```

5. Initialize an empty vector Y of length n.
   `Hint: An empty vector of length 2 is given by rep(NA,2).`
6. Write a for loop over i from 1 to n.
   - For each i, check if $W[i]$ is nonnegative
   - **If** the covariate W for individual i is greater than or equal to 0, then set $Y[i]$ equal to an instance of a $Normal(A[i], 1)$ random variable.
     `Hint: In R, greater than or equal to is written as >=.`
   - **Else** set $Y[i]$ equal to an instance of a $Exponential(10|W[i]|)$ random variable.
     `Hint: Type ?rexp and ?abs into your console.`

Solution:

```
> # 5.
> Y <- rep(NA,n)
> # 6.
```

```
> for(i in 1:n){
+   if(W[i]>=0){
+     Y[i] <- rnorm(1,A[i],1)
+   } else {
+     Y[i] <- rexp(1,10*abs(W[i]))
+   }
+ }
```

You should now have a vector Y that contains 100 numbers and no `NA`s. You can type Y in your console to check if this is correct.

Solution:

```
> Y

  [1]  0.16273096  0.61493770 -0.91092165  1.15802877  0.12044487  0.57671878 -1.28074943
  [8]  1.62544730  0.49930340  0.02688821  0.12635704  0.26433166 -0.46164473  0.07703847
 [15] -1.68018272  1.97997524  0.68000713 -0.27911330  0.15789080  1.13805271  0.88120797
 [22]  0.02721970  1.34303883  0.18071253  0.44542193  0.34469623  0.05426937  0.43758972
 [29] -1.23290120  0.28592840  2.24613975  0.79791644 13.21099705  0.33634171  1.68937270
 [36]  0.04416090 -0.23170706  0.22200655  0.47172010  0.40677522  0.08931932  1.74127631
 [43]  0.06851153 -0.32375075 -0.08650305 -1.01592895  0.15496980  0.49773669 -1.28630053
 [50] -0.64060553  0.77618096 -0.31142059  0.76993773  0.35209265  0.07300720  0.27146118
 [57]  0.35240794 -1.07519230  1.00002880  0.40892833 -0.01252935  0.26858097  0.26457209
 [64]  0.39256080  3.65865803  0.11202581  1.78202877  0.71324052  0.10433709 -0.25502703
 [71]  0.71891832  0.65140532  0.53484371  0.04945420  0.25859127  0.70580355  0.38504973
 [78]  0.80023985  0.66599916  0.96527397  0.43548207 -0.20618900  0.41231828  0.01631967
 [85]  0.92958262  0.17259206  1.98389557  0.02300642  0.13230533  0.31089615  0.07788082
 [92]  0.04507822 -0.09045612  1.15649049 -0.73731169  0.79865879  0.13730620  0.60293277
 [99]  2.02476139  0.29630575
```

Let's try generating Y again without a for loop. The key is that we can select multiple elements of a vector at a time.

```
> x <- 1:6
> x

[1] 1 2 3 4 5 6

> # Let y contain 20, 19, ..., 15
> y <- seq(20,15,by=-1)
> y

[1] 20 19 18 17 16 15

> # Select elements of x where x is larger than three
> x[x>3]
```

```
[1] 4 5 6


> # Now select elements of y where x is larger than three
> y[x>3]


[1] 17 16 15


> # Equivalently:
> # Record the indices for which x>3
> inds <- which(x>3)
> # Now print out the corresponding output
> x[inds]


[1] 4 5 6


> y[inds]


[1] 17 16 15
```

7. Remove the current Y vector from memory with `rm(Y)`.
8. Initialize a new vector Y of length n.
   `Hint: See step 5.`
9. Let `Wpos` denote the indices for which $W \geq 0$.
10. Calculate the number of individuals m for which $W \geq 0$.
    `Hint: Try the length command.`
11. For the indices of Y corresponding to people with $W \geq 0$, set Y equal to a $Normal(A, 1)$ random variable.
    `Hint: You should only generate length(Wpos) normals, with mean A[Wpos].`
12. For the indices of Y corresponding to people with $W < 0$, set Y equal to an $Exponential(10|W|)$ random variable.
    `Hint: Y[-Wpos] will select individuals with W<0.`

Solution:

```
> # 7.
> rm(Y)
> # 8.
> Y <- rep(NA,n)
> # 9.
> Wpos <- which(W>=0)
> # 10.
> m <- length(Wpos)
> # 11.
> Y[Wpos] <- rnorm(m, A[Wpos], 1)
> # 12.
> Y[-Wpos] <- rexp(n-m,10*abs(W[-Wpos]))
```

The above code is known as **vectorized** code because it exploits the vector structure of the variables. Vectorized code will generally run faster in `R`, and is often faster to write than non-vectorized code. That said, this is not an `R` class so it is up to you to decide how to write your code.

You should now have vectors W, A, and Y, each of length 100. Let's take a look at our data.

13. Check that W, A, and Y have length 100.
    `Hint: Use the length command.`
14. Explore your data by looking at the first or last few entries of each vector.
    `Hint: Use the head or tail command.`
15. Make a histogram of the outcome Y.
    `Hint: hist(x) will make a histogram of the data in a vector x.`
16. Make a histogram of Y among people with $W \geq 0$.
    `Hint: Use Wpos to subset` Y.
17. Make a histogram of Y among people with $W < 0$.

Solution:

```
> # 13.
> length(W)

[1] 100

> length(A)

[1] 100

> length(Y)

[1] 100

> # 14.
> head(W)

[1] -0.4689827 -0.2557522  0.1457067  0.8164156 -0.5966361  0.7967794

> tail(W)

[1]  0.55782935  0.59461765 -0.08945109 -0.17983184  0.62174049  0.20986658

> head(A)

[1] 1 0 0 1 1 0
```

```
> tail(A)

[1] 0 1 0 1 0 1

> head(Y)

[1]  0.07423168  0.74467930  0.96079238  2.79048505  1.06116326 -1.06416516

> tail(Y)

[1] -0.70592859  0.83882149  0.32220339  0.45383957  0.50132183 -0.01353967

> # 15.
> hist(Y)

> # 16.
> hist(Y[Wpos])

> # 17.
> hist(Y[-Wpos])
```

Finally, let's merge the data into a single data frame that we could use downstream for an analysis. First, an introduction to data frames. A data frame is a two-dimensional object, where each column of the data frame represents a variable of interest. Typically a row in a data frame will represent an observational unit.

```
> df <- data.frame(col1=1:5,col2=2:6,col3=rnorm(5))
> # Whole data frame
> df

  col1 col2       col3
1    1    2  1.7238941
2    2    3 -0.2061446
3    3    4 -1.3141951
4    4    5  0.0634741
5    5    6 -0.2319775

> # Just the third column
> df$col1

[1] 1 2 3 4 5

> # First and second columns
> subset(df,select=c(col1,col2))

  col1 col2
1    1    2
```

```
2    2    3
3    3    4
4    4    5
5    5    6
```

```
> # First two rows
> df[1:2,]
```

```
  col1 col2       col3
1    1    2  1.7238941
2    2    3 -0.2061446
```

18. Define `data` as a data frame containing W, A, and Y, with column names W, A, and Y, respectively.
19. Select the A column from the data frame and print it to the console.
20. Look at the first few rows in `data`.
    `Hint: The head function works for data frames.`

Solution:

```
> # 18.
> data <- data.frame(W=W,A=A,Y=Y)
> # 19.
> print(data$A)
```

```
  [1] 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1
 [43] 0 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 0 0 1 1
 [85] 1 0 1 1 1 1 1 0 0 1 0 1 0 1 0 1
```

```
> # 20.
> head(data)
```

```
           W A           Y
1 -0.4689827 1  0.07423168
2 -0.2557522 0  0.74467930
3  0.1457067 0  0.96079238
4  0.8164156 1  2.79048505
5 -0.5966361 1  1.06116326
6  0.7967794 0 -1.06416516
```

2 Installing a Package

One of the reasons that `R` is such a popular statistical programming language is the wide range of freely available packages online. We will use several packages from the Comprehensive `R` Archive Network (CRAN) repository this semester.

Suppose we want to make a draw from a multivariate normal distribution. This feature is not built into the `R` base package.

1. Check to see if you already have access to the function `mvrnorm` by typing the name of the function into your console.
   `Hint: If you have not already loaded the MASS package, you should get an error!`

We will now install the `MASS` package from CRAN. This package includes the function `mvrnorm`.

1. Install the `MASS` from CRAN.
   `Hint: ?install.packages.`

Solution:

```
> # 1.
> install.packages('MASS')
```

You only need to install a package once. Once the package is installed, you just need to load it at the beginning of each session.

2. Load the package from your library.
   `Hint: ?library.`

Solution:

```
> # 2.
> library(MASS)
```

We will generate five draws from a bivariate normal distribution with mean $\mu = (0, 0)$ and covariance matrix

$$\Sigma = \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix}.$$

We can create the matrix in `R` as follows:

```
> # Bind column vectors containing (1,1/2) and (1/2,1) together
> Sigma <- cbind(c(1,1/2),c(1/2,1))
> Sigma

     [,1] [,2]
[1,]  1.0  0.5
[2,]  0.5  1.0

> # Alternatively, we could enter Sigma as
> Sigma <- matrix(c(1,1/2,1/2,1),nrow=2)
> Sigma

     [,1] [,2]
[1,]  1.0  0.5
[2,]  0.5  1.0
```

3. Use the `mvrnorm` function and the covariance matrix above to generate a draw from the $N(\mu, \Sigma)$ distribution.
   `Hint: Installed packages should come with documentation files. Try ?mvrnorm.`

Solution:

```
> # 3.
> mu = c(0,0)
> Sigma = matrix(c(1,1/2,1/2,1),nrow=2)
> mvrnorm(1,mu,Sigma)

[1] -0.2673438  1.3673005
```