Section 1 (STAT 171)

- TF: Jiaze Qiu
- Email: jiazeqiu@g.harvard.edu

- Section/OH: Wed 9-12 pm (ET)
- Notice: Time is subject to change

- Acknolegement: This handout is partially based on notes created by Ganghua Wang, Lisa Ruan and Christy Huo.

1. A Very Brief Stat 110 Review

Concepts

- Fundamental bridge: let I_A be the indicator function of event A, i.e., it's 1 if A happens and 0 otherwise. Then $\mathbf{E}[I_A] = \mathbf{P}(A)$.
- Inclusion-exclusion principle:

$$P\left(\cup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} P\left(A_i\right) - \sum_{i<j\leq n} P\left(A_i \cap A_j\right) + \sum_{i<j<k\leq n} P\left(A_i \cap A_j \cap A_k\right) - \cdots + (-1)^{n-1} P\left(\cap_{i=1}^{n} A_i\right)$$

- Law of total probability: let $\{B_n\}$ be a partition of the probability space, then

$$P(A) = \sum_{n} P\left(A \cap B_n\right) = \sum_{n} P\left(A \mid B_n\right) P\left(B_n\right)$$

- Bayes' theorem: if $\mathbf{P}(A) \neq 0$,

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)}$$

- Adam's law (Tower rule): $\mathbf{E}(X) = \mathbf{E}(\mathbf{E}[X|Y])$
- Law of total variance (Eve's law): $\mathrm{Var}(X) = \mathbf{E}[\mathrm{Var}(X \mid Y)] + \mathrm{Var}(\mathbf{E}[X \mid Y])$

Excercise

- Let Φ be the cumulative distribution function (CDF) of a standard normal distribution, i.e., $\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$. Let $X \sim N(\mu, \sigma^2)$. Find $\mathbf{E}(\Phi(X))$.
- Let X, Y, Z be i.i.d. $N(0,1)$. Let $W = \frac{X+YZ}{\sqrt{1+Z^2}}$. Find the distribution of W.

2. Introduction to R and R Studio

What's R

R is a free statistical programming language used in statistics, machine learning, and data science. R studio is a user-friendly environment for R. Students are recommended to use R to complete their practical assignments for STAT 171, while it is also okey to use other programing lauguages.

Installation

Please click on the link below to learn how to install R and R studio if you haven't.

https://youtu.be/d-u_7vdag-0

Getting started

After installing both R and R studio, you can start R studio. There should be three windows: Console (left window), Environment (top right window), Files (bottom right window).

- Console is where any work is processed and shows outputs. If there is any errors in your R command, the error message will appear in the console.
- Environment shows datasets and objects created. You can click on 'History' tab to view previous R commands.
- Files show any files in your working directory by default. Notice other tabs such as Plots, Packages, Help, etc.

R as a calculator

Copy and paste or retype each line of the following R commands in your console and hit [Enter] and predict what R will produce.

```
9+7
```

```
## [1] 16
```

```
sqrt(16)
```

```
## [1] 4
```

```
abs(-3)
```

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

```
3^2
```

```
## [1] 9
```

```
3**2
```

```
## [1] 9
```

```
3/2
```

```
## [1] 1.5
```

Creating objects

You can use either '<-' or '=' to create objects.

```
x<-1
y=3
xy<-x*y
```

Two commands above create two objects 'x' and 'y', where 'x' is defined as 1 and 'y' is defined as 3. 'xy' is defined as x multiplied by y. You can check 'Environment' Window (top right) to see all three objects.

Try each line of commands below.

```
print(x)
```

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

```
x
```

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

```
y<-10 #now y is 10
y
```

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

Note that both print(x) and x return the value of x. When you type 'y<-10', R console does not print the value of y.

Note that R is case-sensitive; hence each of the following commands will produce an error message.

```
print(X)
X
X+3
```

To remove objects

```
rm(x)
rm(y, z)
```

```
## Warning in rm(y, z): object 'z' not found
```

Different types of objects

Most values in `R` are of the following data types:

- Logical: `TRUE`, `FALSE`
- Integer: `110`, `111`
- Decimal: `2.718`
- Character: `"I love statistics"`

You can check that `y` is numeric with the `class()` command.

```
y <- 3.14159
class(y)
```

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

When you want to store more than one number/character to an object, you can create a vector. Suppose you have four students' names and their exam scores.

```
names=c("Adam", "Ben", "Chuck", "Dan")
exam=c(78, 51, 60, 90)
```

Whenever you want to create a vector, you need 'c()'. The letter 'c' stands for 'combine' or 'column'.

Built-in Functions in R

`R` comes with many built-in functions, which are often applied to data vectors. Common examples are `length()`, `mean()`, and `print()`. Suppose you want to find the average (mean) of those four students scores. You can use built-in function 'mean()'.

```
mean(exam)
```

```
## [1] 69.75
```

Packages

`R` comes with a set of built-in functions (such as `mean()` and `print()`) but many more functions can be used via packages. To install packages

```
install.packages("PackageName")
```

To add packages to your library

```
library(PackageName)
```

For instance, the `stats` package has a function `cor()` which finds the correlation between two vectors. To use `cor()` run the following.

```
# install.packages("stats") # only run once on your machine - only run if you haven't already
library(stats) # run once per session
cor(c(1,2,3), c(1,2,4))
```

```
## [1] 0.9819805
```

Help

In R, using 'help(FunctionName)' or '?', you can open help document of the function.

```
help(hist)
?hist
```

Matrices

To declare a matrix in `R` use the `matrix()` function. The first argument is the data for the matrix, which is typically read in as a vector. You can then specify how many rows or vectors the matrix should have and whether the data should be fillwed in "by row" or "by column" (the default). For instance,

```
X <- matrix(c(1,2,3,4,5,6), nrow = 2, byrow = TRUE)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Common matrix functions are shown below (output omitted).

```
X[2,] # extracts the second row of the matrix and all columns
X[,3] # extracts the third column of the matrix and all rows
dim(X) # finds the dimensions of X (rows, )
tX <- t(X) # finds the transpose of X
XtX <- X %*% tX # performs matrix multiplication of X by tX
inverse_XtX <- solve(XtX) # takes the inverse of XtX
```

Dataframes

Creating Dataframes

Dataframes are like matrices but prettier and often easier to work with. There are many ways to create a dataframe. One way is to make a dataframe from a matrix. Another way is to fill in the data manually by column. The two commands below make the same dataframe

```
# method 1
Y = data.frame(X)
colnames(Y) = c('A','B','C') # adds names for Y's columns

# method 2
Y = data.frame(A = c(1,4), B = c(2,5), C = c(3,6)) # names automatically included
```

```
print(Y)
```

```
##   A B C
## 1 1 2 3
## 2 4 5 6
```

It is most common, however, to read in a dataframe from your computer. Such as

```
Y = read.csv('/Path/To/File/filename.csv')
```

Accessing and Modifying Components

We can access a dataframe's columns by name with the $, by name with double brackets, or by index with double brackets. Each of the following returns the first column of `Y` as a vector (output omitted).

```
Y[['A']]
Y$A
Y[[1]]
```

We can also access specific elements within the dataframe with `matrix.name[row,column]`.

```
Y[2,3]
```

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

We can access all data where certain columns or rows satisfy a given criterion. For instance, the below accesses all rows where column `C` equals 6.

```
Y[Y$C == 6,]
```

```
##   A B C
## 2 4 5 6
```

We can change certain elements as below.

```
Y[1,'B'] = 0 # alternatively Y[1,2] = 0
Y
```

```
##   A B C
## 1 1 0 3
## 2 4 5 6
```

We can also add to a dataframe as below

```
Y$D = c(110,111)
Y
```

```
##   A B C   D
## 1 1 0 3 110
## 2 4 5 6 111
```

For Loops

For loops allow us to loop through a collection of items and do something to each element or to simply repeat some process a given number of times. We will extensively use for loops throughout Stat111. To loop through elements in a collection, we use the following general syntax.

```
for(i in vector){
  do.something
}
```

As an example,

```
numbers <- 1:3
for(number in numbers){
  print(number/2)
}
```

```
## [1] 0.5
## [1] 1
## [1] 1.5
```

To repeat some process `n` times, use the following syntax.

```
for(j in 1:n){
  do.something
}
```

As an example, to sum from 1 to 10 we could do

```
total <- 0
for(i in 1:10){
  total <- total + i
}
print(total)
```

```
## [1] 55
```

Often, we want to add the result of each iteration to some results vector. This is done with the following code.

```
n <- 100
results <- rep(NA, n) # makes a vector of length n with all NAs
for(i in 1:n){
  results[i] <- whatever
}
```

For instance, the following creates a vector of 10 sample means from random draws of size 100

```
sample_means <- rep(NA, 10)
for(i in 1:10){
  data <- rnorm(100)
  sample_means[i] <- mean(data)
}
```

Comments

By using '#', you can write comments.

```
# This is a comment line
mean(c(12, 4, 25, 0))
```

```
## [1] 10.25
```

Anything you write after '#' is a comment line and R will not run that line. A comment line is used to describe what you are doing.