*Last update: September 3, 2019*

# 201-DDD-05 - Statistical Methods
# COMPREHENSIVE PROJECT - PART 1

***Part 1 is due Tuesday October 22, before 9:00am***

## CONTENTS

# 1. PART 1: INSTALLING AND LEARNING R

The project will require that you install the statistical software R on your personal computer. Some of the computers in the Math Study Area have this software installed, but you should not rely on those computers as the study area is often very busy.

You will also need to familiarize yourself with the language. **BOTH** teammates should learn the basics of R. Part 3 of the project will rely heavily on being comfortable with this software.

**Installing and Learning R:**

- Install R on your personal computer/laptop. You can download the software from http://cran.utstat.utoronto.ca/. If you are installing R on a Mac, you will first need to install XQuartz.
- Some of the basic commands are listed in Appendix A. Download the script `tutorial.R` from LEA and try the commands one by one! It contains most of the commands necessary for the project.
- Perform the following exercises. Note that you are NOT allowed to calculate things on paper or using your calculator, everything has to be done using R commands.

**Exercise 1: Simulating Probabilities**

For the following questions, the function `sample` may be useful.

(a) Rolling a die!

(i) Simulate rolling a fair 12-sided die 500 times. Generate a pie chart of the results.

(ii) Simulate rolling an unfair 12-sided die 500 times, where rolling a 1 is three times more likely to occur than the rest of the outcomes. Generate a pie chart of the results.

(iii) Simulate rolling a pair of fair 12-sided die 10000 times. From the results, estimate the probability that the sum of the two dice is equal to 10. What is the exact theoretical probability?

(b) Playing the lottery!

(i) Complete the following code of a function that simulates playing the lotto 6/49 with the same ticket until you win the jackpot. You have to insert code where is says `HERE`.

```
play_lotto <- function(){
   n <- 0

   ticket <- # HERE: create a vector of your 6 favorite numbers in increasing order!

   draw <- c(0,0,0,0,0,0) # initializes the vector that will contain the numbers drawn

   while (!all(ticket==draw)){
      n <- n + 1
      draw <- # HERE: draw a random sample of 6 numbers from 1:49 in increasing order
   }

   return(n)
}
```

(ii) To test the code, first modify it so that it simulates the lotto 4/49 (sample of 4 numbers instead of 6). It should complete faster, and help you debug any problems you may have.

(iii) Run the code twice for the lotto 6/49. The value returned by the function gives the number of times you had to play before winning the jackpot. The code will probably take several minutes to terminate, so be patient (do something else while it runs).

(iv) Assuming you play once a week, how many years did it take before winning the jackpot?

(v) Assuming each ticket cost \$3, and that the jackpot is \$16,000,000, what is your net earning after playing this many times?

**Exercise 2: Creating functions in R**

For this exercise, you will have to write the code defining functions as specified. I expect you to use either a `FOR` or a `WHILE` loop in items (a), (b), and (c). Make sure you test your code to verify it is working properly and give different names to each function.

(a) Write a function in R that receives as input a number `n` (which is assumed to be a positive integer) and as an output returns the sum $1 + 2 + \cdots + n$.

(b) Write a function in R that receives as input a number `n` (which is assumed to be a positive integer) and as an output returns the sum $1^2 + 2^2 + \cdots + n^2$.

(c) Write a function in R that receives as input numbers `k` and `n` (which are assumed to be positive integers such that $k \leq n$) and as an output returns the sum $k^2 + (k+1)^2 + \cdots + n^2$.

(d) Rewrite the function in (b) to simply call the function in (c) instead of doing a loop.

**Exercise 3: Probability Distributions**

For this exercise, look in particular at section 8 of the script `tutorial.R`.

(a) For the Tim Hortons' *Roll Up The Rim* contest this year, a total of 33,357,600 cups were made for the province of Québec, with exactly 1/6 of them being winners of a coffee or food prize. Suppose that a random sample of 8,000 cups were sent to a Tim Hortons store in Kirkland. Calculate the (exact) probability that less than 1,300 of these cups were winning a coffee or food prize.

(b) The number of shots on goal by the Canadiens during a hockey game follows a Poisson distribution with an average of 29 shots per game. Considering a full season of 82 games, calculate the probability that the total number of shots on goal made by the Canadiens will exceed 2400 shots?

(c) On the same graph, plot the **cumulative** density function for two different continuous distributions (either uniform, normal, t, chi-squared or F). Make sure to carefully select the scale of the x-axis so that your graph shows both cumulatives going from 0 to 1.

**Report:**

Submit your results for Exercises 1, 2 and 3 in a clear and well-structured report. In this report, include your data, your code (R commands/script), your results and your graphs. Print your report on paper, double-sided if possible. This part is worth 5% of your final grade. There will be a late penalty of 10% (0.5 point out of 5) for each day late.

## 2. GRADING SCHEME

Below is a tentative grading scheme for this part of the project.

| **PART 1** | |
|---|---|
| **Exercise 1** (*correctness, presentation*) | 2 |
| **Exercise 2** (*correctness, presentation*) | 1.5 |
| **Exercise 3** (*correctness, presentation*) | 1.5 |
| **TOTAL (Part 1)** | **5** |

# 3. APPENDIX: BASICS COMMANDS AND TIPS FOR R

The following is available as a script (`tutorial.R`) on LEA; download it and run the commands one by one to see what they do.

```
########################################
# R TUTORIAL WITH MANY USEFUL COMMANDS #
########################################

# Note that any text following the character "#" on a line is a comment.


# SECTION 1: GETTING HELP
# -----------------------

# If you know the name of a command (say mean), here's how to get more information.

help(mean)
?mean           # same as help(mean)
??mean          # searches for all functions related to the keyword 'mean'
example(mean)   # shows examples on how to use a command

# If you are looking to do something in R, say drawing a histogram, but you don't know
# the specific command name, in my opinion the easiest method is to use Google!


# SECTION 2: DATA STRUCTURES
# --------------------------

# You can store numbers, character strings, vectors and matrices into variables.
# The following are called assignment operations; they assign a value or object
# to a name.

x <- 5
y <- "Hello World!"
z <- x + sqrt(16) + cos(pi)

# You may then print the value of a variable simply by typing the name (e.g. z)
# or using the function print (e.g. print(z)).
# To construct a vector of values, they are several ways. Try the following commands.

vec1 <- 1:10
vec2 <- c(1,4,2,-10,-5)
vec3 <- c("Alice", "Bob", "Charles")
vec1
vec3[2]
vec2[3:5]
log(vec1)

# You can append elements to the end of a vector:
vec1 <- append(vec1, 40)
vec1
```

```
# And you can take the "union" of two vectors (viewed as sets)
vec4 <- union(vec1, vec2);
vec4;

# You can also create a list of vectors
vec_list <- list(vec1, vec2, vec3)
vec_list

# To access individual elements in this list, you use [[ ]]
vec_list[[1]]
vec_list[[2]]
vec_list[[3]]

vec_list[[3]][2]

# A very useful data structure in R is the data frame. Think of a data frame
# as a spreadsheet, loosely speaking. Suppose I have a sample of 7 students, and I have
# measured the eye color and R score for each. Below is an example on how to creat
# a data frame.

eyecolor <- c("brown","brown","blue","brown","brown","brown","blue")
Rscore <- c(27.1,28.7,24.0,31.2,29.5,30.9,34.7)
MyData <- data.frame(eyecolor,Rscore)
MyData

# Note that the $ operator will access individual columns of the data frame.
MyData$eyecolor
MyData$Rscore
mean(MyData$Rscore)


# SECTION 3: SAMPLING AND RANDOM NUMBERS
# --------------------------------------

# You may generate random samples from a given vector of outcomes.

outcomes <- 10:100
sample1 <- sample(outcomes,6)   # sampling without replacement! (by default)
sample1
marbles <- c("blue", "blue", "red")
sample2 <- sample(marbles, 10, replace=TRUE)   # sampling with replacement!
sample2

# All of the above assume by default that all outcomes are equally likely.
# We can also take a sample assuming not all outcomes are equally likely.
# Let's see the case of a (non-fai) die and assume that
# the outcome 1 has probability 0.5 and all other outcomes have probability 0.1
# Let's roll this die 20 times

outcomes <- 1:6
sample3 <- sample(outcomes, 20, replace=TRUE, prob = c(0.5, 0.1, 0.1, 0.1, 0.1, 0.1));
sample3
```

```
# SECTION 4: MORE ON VECTORS AND INDICES
# --------------------------------------

# Run the following code and discover what each command does by printing each variable.
# You can also read the help pages for each new command, e.g. help(rep).

U <- rep("toto", 5)
V <- rep("foo", 8)
C <- c(U, V)
C <- sample(C)
indices <- which(C == "foo")
C[indices]


# SECTION 5: RESTRICTING A VECTOR
# -------------------------------

eyecolor <- c("brown","brown","blue","brown","brown","brown","blue")
Rscore <- c(27.1,28.7,24.0,31.2,29.5,30.9,34.7)
MyData <- data.frame(eyecolor,Rscore)
MyData$age <- c(18,17,17,16,19,18,18)

# Suppose we would like to find the mean and standard deviation of a variable
# (say the age), but categorized by eye color. In other words, we want to calculate
# the average of people with brown eyes, and the average age of people with
# blue eyes separately.

# A first way is to restrict the variable as follows.
iblue <- which( MyData$eyecolor == "blue" )
ibrown <- which( MyData$eyecolor == "brown" )
mean( MyData$age[ iblue ] )
sd( MyData$age[ iblue ] )
mean( MyData$age[ ibrown ] )
sd( MyData$age[ ibrown ] )

# Another convenient way is to use the aggregate function.
aggregate( MyData$age, list(Name=MyData$eyecolor), mean, na.rm=TRUE)
aggregate( MyData$age, list(Name=MyData$eyecolor), sd, na.rm=TRUE)

# Note that if the data contains some NA values, they need to be ignored in
# most calculations, and therefore the need for the argument na.rm=TRUE
# in many functions. Another possibility is to directly remove those NA values:
vec <- c(0,1,NA,3,4,5,6,NA,NA,9,10)
vec <- vec[ !is.na(vec) ]         # keep only the non-NA values
vec


# SECTION 6: PLOTS
# ----------------

# Here are a few commands to generate some useful plots using
```

```
# the previously created data frame MyData.
hist(MyData$Rscore)                    # HISTOGRAM
pie(table(MyData$eyecolor))            # PIE CHART
barplot( table(MyData$age) )           # BAR CHART
plot(MyData$eyecolor, MyData$Rscore)   # BOX-PLOTS
plot(MyData$age, MyData$Rscore)        # SCATTER PLOT

x <- 1:20
y <- sample(1:20, replace=T)

# You can make your graphs better by adding your own title and labels for the axes.
plot(x,y, main="Scatter plot of random numbers vs integers",
     xlab="Integers", ylab="Random")

# And also change the range of x-values and y-values, which is particularly useful
# when zooming in a graph.
plot(x,y, xlim=c(5,15), ylim=c(0,12))

# Finally you can also produce plots in log-scale by specifying the option
# log='x' or log='y' or log='xy' in a plotting function.
x <- 2**(0:5)
y <- 10**(0:5)
plot(x,y, log='xy', main="Plot in log-log scale")

# You can also draw two functions on the same plot, here's an example
x <- seq(0, 3*pi, 0.1)                 # values from 0 to 3pi by increments of 0.1
plot(x, sin(x), type='l', col='blue')  # first function
lines(x, cos(x), type='l', col='red')  # add second function on the same plot


# SECTION 7: Intro to Programming
# -------------------------------

# Programming is fun. Let's see some interesting things you can do when you program...

#Let's simulate rolling a fair die:
rolled_value <- sample(1:6, 1)

rolled_value

# Now we learn about the IF statement
if (rolled_value == 1){
   print("The number 1 was rolled")
} else {
   print("A number other than 1 was rolled")
}

# Another example (the "!=" logical operator represents "NOT EQUAL")
if (rolled_value != 1){
   print("The number 1 was not rolled")
} else {
   print("The number 1 was rolled")
}
```

```
# And another one... (the "&" logical operator represents "AND")
if ( (rolled_value >= 2) & (rolled_value <= 4) ) {
   print("A number between 2 and 4 was rolled")
} else {
   print("Either 1 or 5 or 6 was rolled")
}

# Another useful gadget in programming is the FOR loop
# Let's use it to print all numbers from 1 to 15
for (i in 1:15){
   print(i)
}

# There is another type of loop that is useful: it is the WHILE loop
# Let's use it to print all numbers from 1 to 15 again
i <- 1
while (i <= 15){
   print(i)
   i <- i + 1
}

# Finally, a piece of code that is used often can be put in a "function"
# Much like a mathematical function, a function in this context
# can accept one or more inputs (called arguments) and then returns an output
# Let's define a function to simulate the following experiment (for a given number s):
#   a pair of dice is rolled until the sum of its values is equal to s
# The function will take as argument the number s and return the number of trials that were necessary

sumn <- function(s){
   # This IF statement is used to make sure the value of s given by user is valid
   # (the "|" logical operator represents "OR")
   if ( (s < 2) | (s > 12) ){
      print("ERROR: Invalid value of s")
   } else {
      n <- 0  # initializes the variable that will count the number of trials

      rolled_sum <- 0;  # this is the variable that will contain the sum of the numbers rolled

      while (rolled_sum != s){
         outcome <- sample(1:6, 2, replace = TRUE)
         rolled_sum <- outcome[1] + outcome[2] # sum of the two rolled numbers is put into variable ro

         n <- n + 1 # add 1 to counter of number of trials
      }

      return(n) # return the number of trials
   }
}

# Let's call this function:
sumn(5) # (this will call the function with the value of argument s equal to 5)
sumn(12) # (this will call the function with the value of argument s equal to 12)
```

```
# SECTION 8: Distributions
# ------------------------

# Binomial distribution
dbinom(5,10,0.4)   # p(5) for a binomial with n=10, p=0.4
pbinom(5,10,0.4)   # F(5) for a binomial with n=10, p=0.4
x = 0:10
plot(x, dbinom(x,10,0.4), type="h") # line plot of the binomial distribution

# Normal distribution
pnorm(2.123)    # cumulative standard normal probability up to z=2.123
qnorm(0.60)     # 60th percentile of the standard normal
x <- seq(-3,3,0.01)
plot(x, dnorm(x), type="l") # plot of the standard normal density function


# Here is a table of useful functions for the distributions we will see in class.
# To learn how to use these functions, remember to type help(<function_name>).

#-------------------------------------------------------------------------
# Distribution   | Probability/density | Cumulative | Percentiles | Sampling |
#-------------------------------------------------------------------------
# binomial       | dbinom              | pbinom     | qbinom      | rbinom   |
# hypergeometric | dhyper              | phyper     | qhyper      | rhyper   |
# Poisson        | dpois               | ppois      | qpois       | rpois    |
# uniform        | dunif               | punif      | qunif       | runif    |
# t              | dt                  | pt         | qt          | rt       |
# chi-squared    | dchisq              | pchisq     | qchisq      | rchisq   |
# F              | df                  | pf         | qf          | rf       |
#-------------------------------------------------------------------------


# SECTION 9: DESCRIPTIVE STATISTICS
# ---------------------------------

# Here are some useful commands to obtain some basic descriptive statistics.
# Note that you want these functions to ignore missing values (NA, meaning Not Available).

values <- c(0, -5, 4, 2, 1, 1, 10, 8, NA, 7, 2, -2, NA)
summary(values, digits=6)   # provide mean and five-number summary
sd(values, na.rm=TRUE)      # sample standard deviation
length(values)              # length of vector
sum( is.na(values) )        # number of NA's

eyecolor <- c("brown","brown","blue","brown","brown","brown","blue")
Rscore <- c(27.1,28.7,24.0,31.2,29.5,30.9,34.7)
MyData <- data.frame(eyecolor,Rscore)
MyData$age <- c(18,17,17,16,19,18,18)

# We can calculate the correlation coefficient between two variables as follows:
cor(MyData$age, MyData$Rscore, use="complete.obs")
```

```
# SECTION 10: MORE ON PLOTS
# -------------------------

# For a scatter plot, you can also find the coefficient of the regression by using
# the function lm, and then draw the line on top of the scatter plot.
# Here's an example:
x <- 1:20
y <- sample(1:20, replace=T)
plot(x,y)
lm(y ~ x)      # order is important here, y ~ x means y is the dependent variable
abline(lm(y ~ x), col=2)
```