Math 458 - Financial Mathematics for Actuaries II - Project # 0

Due: 11:59PM on Monday September 11, 2023.
Objective: Use R to accomplish basic calculations, learn how to manipulate various data structures, and basic visualization with `ggplot2`.

Introduction: In this project your group with get familiar with some of the most basic operations used in R. This will include installation of packages/libraries; construction and labeling of vectors, matrices, and data frames; generation of random numbers; averages, standard deviations; basic plots including the package `ggplot2`.

Project Instructions: Below are step-by-step instructions for objectives using R.

1. R Phase: For ALL work in R, always be sure to comment after your commands to tell the reader what you are doing.

   (a) Open RStudio and create new blank R script called

   "`ProjectZero_LastNamesOfGroupMembers.R`"

   In the first three lines of the script use the comment feature of R to indicate what this script is for, as shown below:

   ```
   # Darren Mason
   # Math458
   # Project 0 - Due on September 11, 2023 by midnight.
   ```

   NOTE: Commenting/documenting code is incredibly important in programming. It is essential for the reader of the code - you, me, or someone else - to be able to understand its function and modify. All code you write in R this semester must include comments!

   (b) This section of your script should be preceded by the comment `#Part (b) - Arithmetic`.

   i. Use R to compute $3-5\cdot 3+99$, $\ln(2)$, $e^{0.15}$, and $\frac{6\cdot 2.3+42}{3^{4.2-3.62}}$ by typing in the raw expression and evaluating that line of your script. The exponential function e^x in R is `exp(x)` and the natural logarithm $\ln(x)$ in R is `log(x)`.
   ii. Copy and paste the four expressions you typed in and assign them (using the `<-` operator) respectively to `w, x, y, z`. Note that these variables are now listed in the Environment Panel of RStudio with their current values.
   iii. Compute $x^x - y^z/w^2$ and assign the answer to v, where in R, x^y is written as `x^y`. Verify that you get v = 0.764612297700663.

(c) This section of your script should be preceded by the comment `#Part (c) - Creating, Manipulating, & Subsetting Vectors/Arrays`.

i. Use the `c()` command to create an 8 element vector containing the last 8 digits of your student ID. Assign this to the variable name `studentID`. Recall that to create the vector `(1,2,3,4)`, type in R the command `c(1,2,3,4)`.

ii. Create a vector `biday` consisting of the eight digits of your birthday in the format MMDDYEAR.

iii. Verify the length and class of these vectors using the `class()` and `length()` commands.

iv. Use the `rbind` command to create a 2×8 matrix called `rowarray` consisting of `studentID` as the first row and `biday` as the second row. Repeat this process to create a 8×2 matrix called `colarray` using the `cbind` command. Notice that these two arrays now show up in your RStudio environment tab under the heading "`data`" Double-click on the two arrays and observe that they are displayed in the coding/script pane of RStudio.

v. Use `cbind` to append to the right hand side of `colarray` any 8-digit vector. Your matrix should now have 8 rows and 3 columns. Name it `bigcolarray`. Then change the column names to `"ID","BDAY","FAV VECTOR"` using the command

```
colnames(bigcolarray)<-c("ID","BDAY","FAV VECTOR").
```

vi. Use the `dim` command to verify the dimensions of `rowarray`, `colarray`, and `bigcolarray`.

vii. Extract the first column of `bigcolarray` with the command `bigcolarray[,1]`, and the third row of `bigcolarray` with the command `bigcolarray[3,]`. Then extract upper right sub-matrix consisting of the second-fourth rows of the last two columns of `bigcolarray` with the command `bigcolarray[2:4,2:3]`.

viii. Use the `seq` command to generate a vector called `sequence1` with 6101 elements, with the first element being -3, the last element being 58, and with increments of 0.01. Specifically type

```
sequence1<-seq(from=-3,to=58,by=0.01)
```

Create a similar sequence called `sequence2` with first number 55, last number -6, with decrements of 0.01.

ix. Create vector `sequencesum` by adding `sequence1` to `sequence2`. Verify that the resulting vector has the sum of each pair of elements from the first two vectors. Next create vector `sequencediv` by dividing `sequence1` by `sequence2`. Verify that the resulting vector has the ratio of each pair of elements from the first two vectors.

x. Create new vectors `sequence1short` and `sequencedivshort` where you have selected every 5th element of the first 500 elements of the original sequences of length 6101. Your new vectors should have length 100.* Then use

*Hint: If you wanted to extract every other element of a vector `myvec`, you would use a vector of the

the `plot` command with the new 100-element length vectors and the options `xlab, ylab, main, col, cex=2, cex.lab=2, cex.axis=2, & cex.main=2` to create the plot below.

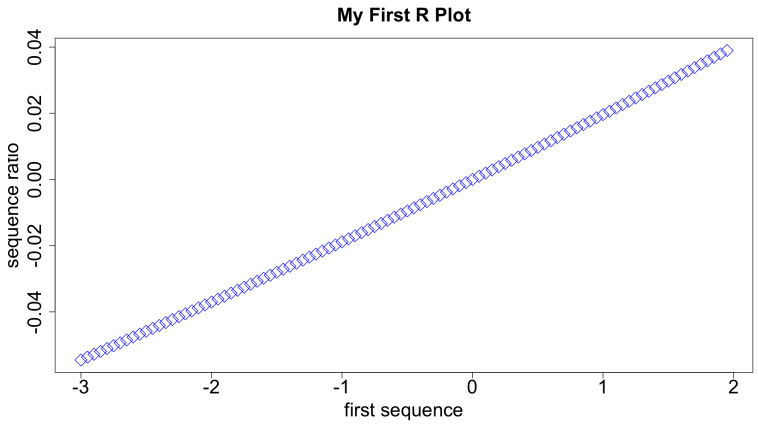

Figure 1: First R Plot in Project #0.

The basic syntax of the `plot` command is

```
plot(sequence1short, sequencedivshort,xlab="x-axis
    text",ylab="y-axis text",main="Title Text",col="plot color
choice",pch="point symbol choice",cex="number representing size of
        point symbols",cex.lab="size of labels relative to
 cex",cex.main="size of plot title relative to cex",cex.axis="size
                of axis markings relative to cex")
```

NOTE: Available `pch` choices are below:

xi. Change the point shapes in the plot to upward pointing hollow triangles and choose at least one other option in the `plot` not already used to modify the below plot. command You will likely need to use the help command `?plot` to figure out what each of these options accomplishes.

(d) This section of your script should be preceded by the comment `#Part (d) - Basic Coding`.

i. If statements: Here we will set up simple `if/then` control structure, along with generating a message as a result.

A. Create a variable called `mynumber` which contains any number you choose between 0 and 10.

B. Create a variable `rannum` of 1 random number between 0 and 10 using the command

relevant logical values TRUE and FALSE command to choose them via

```
myvec[c(TRUE,FALSE)]
```

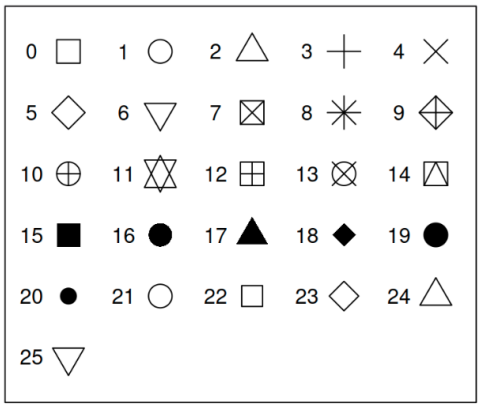

Figure 2: Plot point style choices for `pch` option in the `plot` command. Source: https://r-charts.com/base-r/pch-symbols/.

```
rannum<-runif(n=1,min=0,max=10)
```

C. Use the `if/else` and `cat` commands to return the statement "My number `mynumber` is less than the random number `rannum`." when `mynumber<rannum` and returns the statement "My number `mynumber` is greater than or equal to the random number `rannum`." when `mynumber`$\geq$`rannum`.

ii. For loops: Here we will program three "for loops", one basic, and one that interrogates a large vector of random numbers.

A. Write a `for` loop that computes the product of all odd numbers from 1 to 51. At the end of the loop, your code should produce the final message "The product of all odd numbers between 1 and 51 is X", where X is your answer.

B. Write a `for` loop that counts the number of negative numbers in a list of 5000 random numbers between -1 and 1, generated by the `runif` command. Use an `if/else` command in the loop to count "+1" when number is negative and does not count if the number is nonnegative. Your code for this part should end with the text "`The number of negative numbers is` Z", where Z is the number of negative numbers.

C. Repeat the previous step, except compute the number of negative numbers in 2000 randomly generated vectors, each of which contains 5000 numbers. As part of this process your code should create a vector with 2000 elements, each of which is the # of negative numbers in one of the randomly generated 5000-element vectors. Use the `mean` command to find the average number of negative numbers in the 2000 vectors. Your code should end with the sentence "`The average number of negative numbers in the 2000 random vectors is` Z", where Z is the average. You will need a `for` loop outside of the for loop from the previous step. This is called a *nested for loop*. What number is closest to your average? Is it what you expected?

D. R allows for a much faster way to count the number of elements in a vector that satisfy a certain condition. If `myvector` is your 5000 element vector of random numbers, the command `sum(myvector<0)` will count the number of elements less than 0. Use this fact to modify your code from the previous step so that you only use ONE `for` loop. But this time, generate 10000 random vectors, each with 10000 elements. Assign the name `numnegs` to the 10000 element vector that contains the number of negative numbers found in each of the 10000 randomly generated vectors of 10000 numbers between -1 and 1.

(e) This section of your script should be preceded by the comment `#Part (e) - Data Frames and ggplot2`. R comes with many data sets built in to the program. We will use one that contains information about diamonds to illustrate some of the features of the powerful graphics package `ggplot2`. You can find information about this dataset at

https://ggplot2.tidyverse.org/reference/diamonds.html.

i. Building Data Frames

A. Assign to `n` the maximum length of `bigcolarray, rowarray, & numnegs`.

B. Add blank rows with entries "`NA`" to `bigcolarray` so that it now has `n` rows, calling the output `newbigcolarray`. Since `bigcolarray` already has 8 rows, the command to do this is

```
newbigcolarray<-rbind(bigcolarray,matrix(data=NA,ncol=3,
                        nrow = n-8))
```

C. Repeat the above step so that `rowarray` is replaced with `newrowarray` which has enough rows of "`NA`" so that it now has the same number of rows as `bigcolarray`.

D. Convert `numnegs` into a column vector (10000 rows and 1 column instead of 1 row and 10000 columns) using the **transpose command** `t()`. Call the new vector `tnumnegs`.

E. Create a data frame `df` which binds together as columns `newbigcolarray, newrowarray`, and `tnumnegs`. You will need to use the `as.data.frame` command. Check the dimension of `df`.

F. Change the column names of `df` to, respectively, `ID, BDAY, FAV, Digit 1, Digit 2,..., Digit 8, Negative Count`. Then display the `head` of the data frame.

ii. Plotting Data Frames: R comes with a very powerful plotting package called `ggplot2` that includes a variety of built in data frames such as on car engines, flower geometry, or diamond prices. We will use the diamonds data frame for this exercise.

A. First install the `ggplot2` package with the command `install.packages("ggplot2")`. Then type in the command

`library(ggplot2)`. Note that you can also install the package using the "Packages" tab in the lower right window of RStudio. I will show you both methods in class.

B. Assign the diamonds data frame into the variable `df1`. Enter `head(df1)` to check out the data frame. You can also click on `df1` in the Environment tab.

C. Create your first plot of diamond data using the command

```
plot1<-ggplot(df1,aes(x=carat,y=price,color=clarity))
                    + geom_point()
```

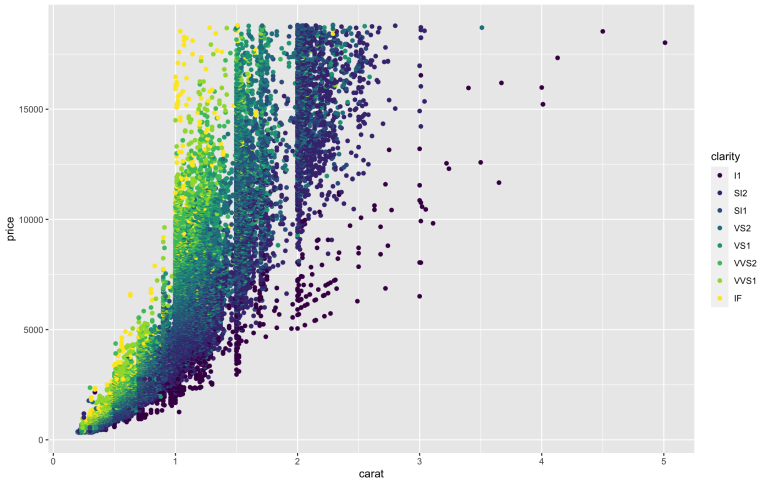

Figure 3: First ggplot2 Plot in Project #0.

For the `ggplot` command, `df1` is your data frame name, `aes` stands for "aesthetics", and then you assign to `x, y, color` your choice for the x axis as the variable `carat`, the y axis as the variable `price`, and then assign a color based on `clarity`. The "`+ geom_point()`" command adds to the plot the choice of plotting a point to represent your data. There are more options for `aes` and `geom_point()`. Your graph should look like the image above.

D. Most of the "action" in the above plot is for diamonds that are less than 2.5 carats in weight. So create a new data frame called `dfcarat` which excludes all diamonds with weight 2.5 carats or above. To do this, the command is

```
dfcarat<-df1[df1$carat<2.5,]
```

The $ is used in R to identify a subset of the data frame `df1` that you want to use as a filter. In this case, it is the carat variable that we want.

E. Now create a second plot using `ggplot` and the the `dfcarat` data frame to reproduce the previous plot, but with only diamonds of weight

less than 2.5 carats. Submit your graph and tell me what you found. Do you notice any differences from previous graphs?

F. Suppose you are only interested in diamonds less than 2.5 carats in weight, but with cut of grade "Fair" or "Ideal". You can create a corresponding data frame `dfcut` by requiring `dfcarat$cut` to be "in" the vector `c("Fair","Ideal")` via

```
dfcut<-dfcarat[dfcarat$cut %in% c("Fair","Ideal"), ]
```

G. Now create a third plot using `ggplot` and the the `dfcut` data frame to reproduce the previous plot, but with only diamonds of weight less than 2.5 carats. then add to the plot the option "`shape=cut`" to the aesthetics `aes` and the options `alpha=0.5, size=1.5` in `geom_point()`, where here `alpha` represents opacity on a scale from 0 to 1 and `size` represents the relative size of the points in the graph.

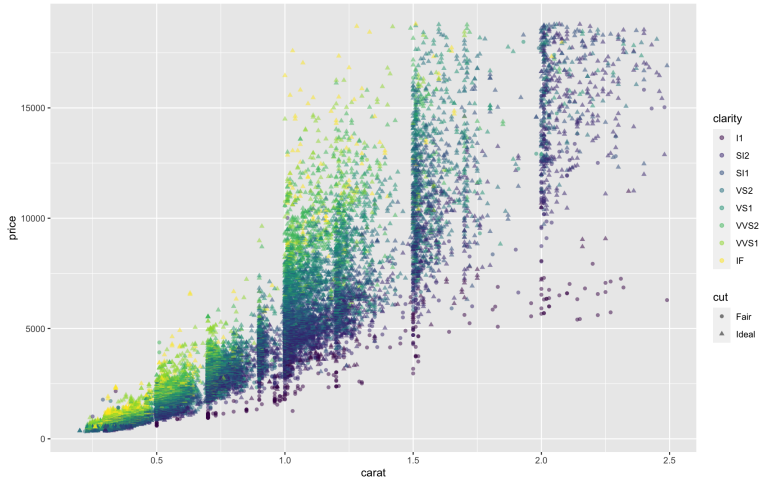

Figure 4: Third ggplot2 Plot in Project #0.

Submit your graph - it should look like the above graph. What can you infer about how diamonds are priced from this plot?

H. Add the option `geom_smooth()` to your second plot and include in your solution. What do you see? The curves in your plot is the result of a statistical technique called *Generalized Adaptive Model* that fits a generalized nonlinear model to a set of data. In this case you should find that the formula used is '`y ~ s(x, bs = "cs")`', where `cs` stands for 'cubic splines", a type of smooth curve fitting technique that uses piecewise smooth third-order polynomials.

I. Add a title and axis labels to the graph by adding the feature

```
+ labs(title="Diamonds - Clarity, Cut, & Price",
  x="Weight", y="Price ($)", color="Clarity")
```

Display the graph – it should look like the below graph:

Figure 5: Fifth ggplot2 Plot in Project #0.

J. The last graph you create is to add a `theme` element to the previous plot so that you can adjust the color, font, face, and size of each labeling element individually. You will need the elements `plot.title`, `legend.title`, `legend.text`, `axis.title`, & `axis.text`, each of which uses the function `element_text()` to set values. For example, if the previous plot is called `plot5`, then I could add a legend title that has color "turquoise3", face "bold", family/font "Arial", and size "12" with the command

```
plot6<-plot5 +
theme(legend.title=element_text(color="turquoise3",
    face = "bold", family = "Arial", size=18))
```

Using the above example as a template, change the plot title, axes labels, and legend text as indicated in the below table.

Feature	Color	Family	Face	Size
plot.title		Arial	bold	22
legend.text	cornflowerblue	Arial	italic	15
axis.title	tomato3	Arial	bold	18
axis.text	orchid3	Arial	bold.italic	15

Table 1: Table of `theme()` features for final R plot using `ggplo2`.

Your final graph should look like

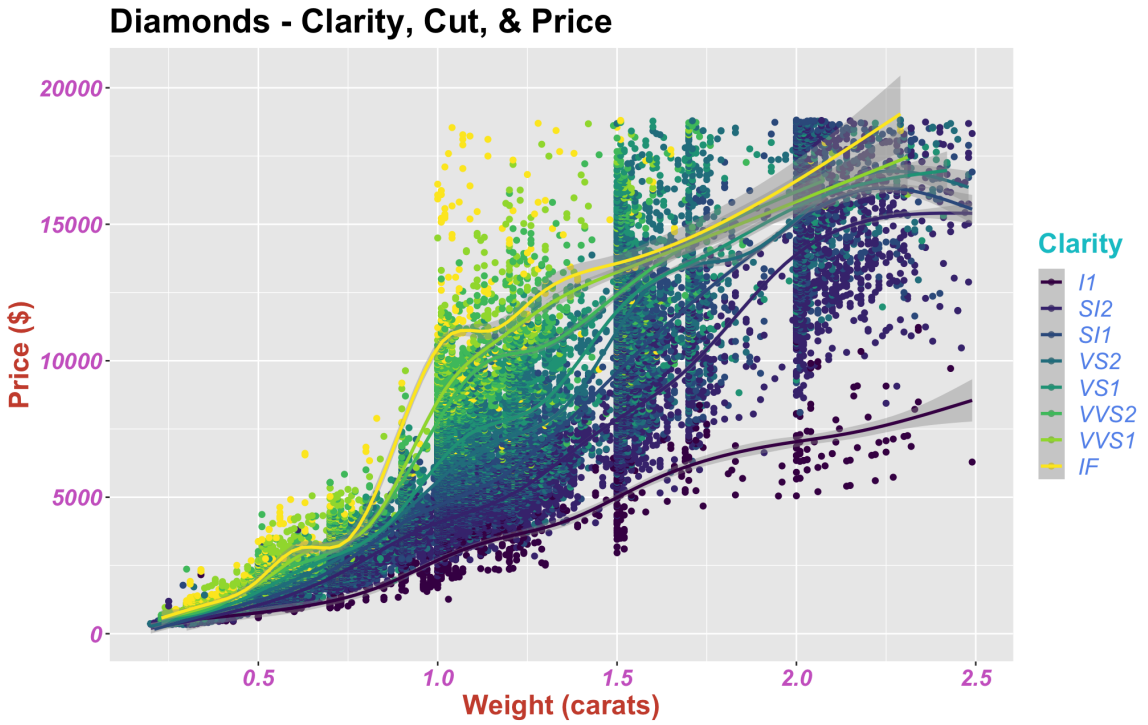

Figure 6: Sixth ggplot2 Plot in Project #0.