STAT 4410/8416 Homework 1

Gerjol, Nicholas

Due on Sep 20, 2021

1. Based on your reading assignments answer the following questions:

   a) What is data science?
   Answer: Data Science is the Science of extracting meaningful (a data product) information from data.

   b) Explain with an example what you mean by data product.
   Answer: A data product is the meaningful information we get from data. For example I am a high school math teacher. In one of my old school districts we had some software that took all of the data from the grade books in the school and created data products, such as graphs of grade distributions based on school or teacher. The organization of the data made it useful for us teachers to determine what teachers were doing well or how well our own instruction is. Those graphs, charts, and reports are all data products that take everything posted to our online system and puts out the data that is important for us to know.

   c) Carefully read Cleveland's paper shown in lecture 2 and discuss what he suggested about the field of statistics and data science. **Answer:** Cleveland had six suggestions for the fields of statistics and data science. First is to require data be analyzed. Work cannot be entirely theoretical in data science. He brings up how important discoveries in statistics were generated from statistical application to other fields and uses that to justify that working with real data, with real intent, has to be a part of learning data science. Cleveland then suggests that more work be put into the development of different buildings of models for data. He discuses how this specification of data is the first of the two critical tasks, followed by estimation and distribution, and that while extensive work has gone into estimation and distribution of data, the development of tools for building models of data is very needed in the field.
   Third he brings up how data science has to combine statistics and computer science. Data scientists have to be part of the development of the tools they will use to create data products in order for innovation in the field to happen. Learning data science in conjunction with R seems to be achieving this goal. Fourth he discuses that data science has to develop the education of data science. Data science has to be developed as its own concentration of statistics, with its own pedagogy. Fifth he discuses that the application of statistics used to develop new tools and methods in other fields have to be turned inwards to analyze the tools and methods used in data science. By analyzing the statistics of the way statistics is done statisticians can improve the way they do statistics.
   Finally, he brings up how in order for innovation to happen in data science, data scientists need to learn the mathematical theories that can provide a framework for the new models and innovation to happen. Data science in Cleveland's mind requires a combination of statistics, data science, and mathematics and all are needed for the field to evolve.

   d) Explain in a short paragraph how data science is different from computer science.
   Answer: Data science is different from computer science in that data science is about using computers to extract meaninful information from data. Computer science can include creating software of websites for various uses, but those creations don't have to have any data to be created. Data science requires that data is being analyzed. In my definition of data science it is even

possible to be data science without the use of computers at all. A data sciencest could hand create a data product, albiet much less effeciently then a computer would.

e) What is data literacy? Is it important to be data literate in this modern world? Explain why or why not.
Answer: Data literacy is the ability to read, write, and understand data. A modern world has so much data availalbe to everyone thatit is important for everyone to become data literate. While it is possible to get through life without understanding data, a modern society works best if everyone learns this skill. In the united states being data literate is critical to being well informed in politics, which is vital for being a contributing member in a democratic society. Expanding from the united states, interacting and making decisions that affect other people, either democratic actions or individual actions, being able to understand the vast amounts of data at all of our fingerprints, having the skill to do it, and the understanding of why being data literate is important are all vital parts of living in a modern society.

f) In his article, Donoho talked about the Common Task Framework. Explain what it is and why he mentioned it.
Answer: A common task framework is a situation where there is a publicly available common dataset, a group of people competing to infer some rule from that data, and a referee who determines who uses the inferered rules and scores them against each other. Donoho brings this up because its a great way in data science to allow for tested methods and rules to be created for a problem quickly and that it needs to be explicitly talked about in the training of data scientists.

g) According to Donoho, what are the activities of greater data science?
Answer:
He had six activities: Data exploration and preperation, Data represenation and transformation, computing with data, data modeling, data visualization and presenation, and science about data science.

2. What are the very first few steps one should take once data is loaded onto **R**? Demonstrate them by loading tips data from http://www.ggobi.org/book/data/tips.csv.

Answer: Read the data, then use the head, tail, summary, str, and plot functions.

```
url <- "http://www.ggobi.org/book/data/tips.csv"
dat <- read.table(url, header=T, sep=",")
head(dat)
```

```
##   obs totbill  tip sex smoker day  time size
## 1   1   16.99 1.01   F     No Sun Night    2
## 2   2   10.34 1.66   M     No Sun Night    3
## 3   3   21.01 3.50   M     No Sun Night    3
## 4   4   23.68 3.31   M     No Sun Night    2
## 5   5   24.59 3.61   F     No Sun Night    4
## 6   6   25.29 4.71   M     No Sun Night    4
```

```
tail(dat)
```

```
##     obs totbill  tip sex smoker day  time size
## 239 239   35.83 4.67   F     No Sat Night    3
## 240 240   29.03 5.92   M     No Sat Night    3
## 241 241   27.18 2.00   F    Yes Sat Night    2
## 242 242   22.67 2.00   M    Yes Sat Night    2
## 243 243   17.82 1.75   M     No Sat Night    2
## 244 244   18.78 3.00   F     No Thu Night    2
```

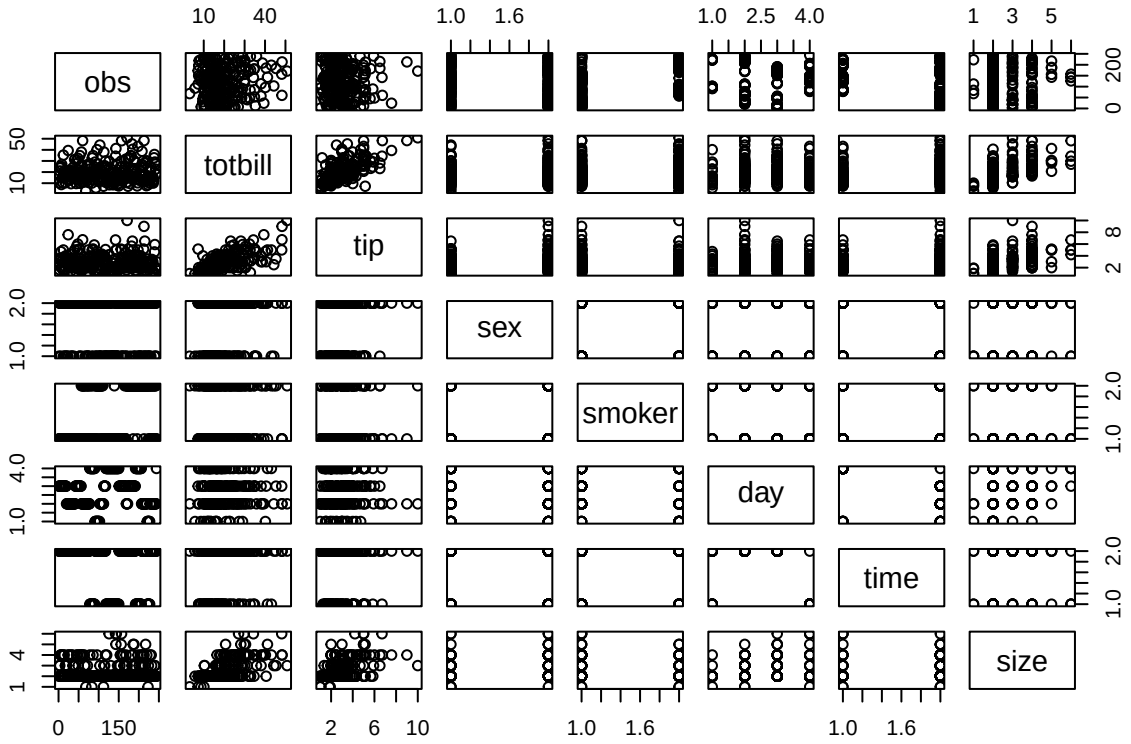
```
summary(dat)
```

```
##       obs            totbill           tip             sex
##  Min.   :  1.00   Min.   : 3.07   Min.   : 1.000   Length:244
##  1st Qu.: 61.75   1st Qu.:13.35   1st Qu.: 2.000   Class :character
##  Median :122.50   Median :17.80   Median : 2.900   Mode  :character
##  Mean   :122.50   Mean   :19.79   Mean   : 2.998
##  3rd Qu.:183.25   3rd Qu.:24.13   3rd Qu.: 3.562
##  Max.   :244.00   Max.   :50.81   Max.   :10.000
##     smoker             day                time                size
##  Length:244         Length:244         Length:244         Min.   :1.00
##  Class :character   Class :character   Class :character   1st Qu.:2.00
##  Mode  :character   Mode  :character   Mode  :character   Median :2.00
##                                                           Mean   :2.57
##                                                           3rd Qu.:3.00
##                                                           Max.   :6.00
```

```
str(dat)
```

```
## 'data.frame':    244 obs. of  8 variables:
##  $ obs    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ totbill: num  17 10.3 21 23.7 24.6 ...
##  $ tip    : num  1.01 1.66 3.5 3.31 3.61 4.71 2 3.12 1.96 3.23 ...
##  $ sex    : chr  "F" "M" "M" "M" ...
##  $ smoker : chr  "No" "No" "No" "No" ...
##  $ day    : chr  "Sun" "Sun" "Sun" "Sun" ...
##  $ time   : chr  "Night" "Night" "Night" "Night" ...
##  $ size   : int  2 3 3 2 4 4 2 4 2 2 ...
```

```
plot(dat)
```



3. In our **R** class, we learned about recursive functions that produce a sequence of numbers up to a given number, say n, as demonstrated with the following code:

```
foo <- function(x) {
  print(x)
  if(x > 1) {
    foo(x - 1)
  }
}

moo <- function(x) {
  if(x > 1) {
    moo(x - 1)
  }
  print(x)
}

foo(3)
```

```
## [1] 3
## [1] 2
## [1] 1
```

```
moo(3)
```

```
## [1] 1
## [1] 2
## [1] 3
```

Explain why `moo()` prints 1 through 3 while `foo()` prints from 3 to 1. **Answer:**

In function foo, the printing happens before the recursion does, so we start with x as 3. The function prints 3 then sends 2 to the function foo again which prints the number 2 before sending 1 to the function, which prints 1. X is now not greater than one, so the third iteration of foo ends, returning to the second iteration which ends, then the first iteration which ends. In the function moo, the printing happens after the recursion, so we have the first layer of the function with input 3, which starts a second layer of the function with input 2, which starts a third layer of the function with input 1, the third layer has X not greater than one, so it goes to print. The output of that print is 1, because we are in the third layer or iteration of the function which has input of 1, than that finishes and we return to the second iteration of the function, which prints 2 before finishing and returning us to the first layer, which still had input 3 and thus prints 3 third. The thing here is that X isn't some global variable that changes, we are sending numerical inputs, each layer or iteration keeps the same input in its x value.

4. The function `sqrt()` provides the square root of a non-negative number. Note what happens when you try `sqrt(-1)`. We want to create our own function that either finds the square root of a non-negative number or provides a custom message if we pass it a negative number.
   a) Create a new `R` function `getRootNotVectorized()` that will return the square root of any non-negative number and 'not possible' for a negative number. Further, `getRootNotVectorized()` should **only** successfully return 'not possible' if the negative value is the first element that you pass to the function. Otherwise, your function should return `NaN` for negative values. Demonstrate that your function produces the following outputs:

      ```
      getRootNotVectorized(4) = 2
      getRootNotVectorized(-4) = "not possible"
      getRootNotVectorized(c(-1, -4)) = "not possible"
      getRootNotVectorized(c(0, 1, -1, 4, -4)) = 0 1 NaN 2 NaN.
      ```

      Don't worry about the warning messages that accompany vector inputs with more than one element for now.
      Answer:

```
getRootNotVectorized<- function(x) {
  if(x<0) {
return("not possible")
  }
  y <- sqrt(x)
  print(y)
}
getRootNotVectorized(4)
```

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

```
getRootNotVectorized(-4)
```

```
## [1] "not possible"
```

```
getRootNotVectorized(c(-1,4))
```

```
## Warning in if (x < 0) {: the condition has length > 1 and only the first element
## will be used
## [1] "not possible"
```

```
getRootNotVectorized(c(0,1,-1,4,-4))
```

```
## Warning in if (x < 0) {: the condition has length > 1 and only the first element
## will be used
## Warning in sqrt(x): NaNs produced
## [1]   0   1 NaN   2 NaN
```

   b) Now create a second function `getRootVectorized()` that will return the square root of any

non-negative number and 'not possible' for a negative number **regardless** of the number's position in a numeric vector of arbitrary length. Demonstrate that your function produces the following outputs:

getRootVectorized(4) = 2
getRootVectorized(-4) = "not possible"
getRootVectorized(c(-1, -4)) = "not possible" "not possible"
getRootVectorized(c(0, 1, -1, 4, -4)) = "0" "1" "not possible" "2" "not possible".

Answer:

```
getRootVectorized<- function(x) {
  L <- length(x)
  for (i in 1:L) {
   if (i==1) {
 if (x[1] < 0) {
   ans <- "not possible"
   }
 if (x[1] >= 0) {
   ans <- sqrt(x[i])
   }
   }
   if (i > 1) {
 if (x[i] < 0) {
   ans <- c(ans,"not possible")
   }
 if (x[i] >= 0) {
   ans <- c(ans,sqrt(x[i]))
   }
   }
  }
  print(ans)
}
getRootVectorized(4)
```

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

```
getRootVectorized(-4)
```

```
## [1] "not possible"
```

```
getRootVectorized(c(-1,-4))
```

```
## [1] "not possible" "not possible"
```

```
getRootVectorized(c(0,1,-1,4,-4))
```

```
## [1] "0"            "1"            "not possible" "2"            "not possible"
```

c) Describe the differences in your code between `getRootNotVectorized()` and `getRootVectorized()` that allowed you to get the desired message output for any negative element of a vector in the latter function but not the former. Knowing whether or not functions that you use will handle vectors in the way that you expect will be very important as you continue working with `R`.
Answer: The difference between my code is that the non vectorized function just looks at the first element in the vector input, whereas the vectorized function has to go through individually and check each element and compile the answers into a list.

d) Why do you see a difference between the output of the two following lines of code?

```
is.numeric(getRootVectorized(c(0, 1, 4)))

is.numeric(getRootVectorized(c(0, 1, -4)))
```

Answer

In order to print "not possible" we have to change from a vector of numeric values to a list which is no longer numeric.

5. This problem will give you some practice with creating and manipulating vectors.

   a) Using `seq()`, create a vector consisting of an arithmetic sequence of integers from 5 to 50 with a common difference of 5 stored in a variable called `mySeq`. **Report** `mySeq`.

   b) Describe how the different arguments in each of the three following commands changes the output of `rep()`: `rep(mySeq, 5)`, `rep(mySeq, each = 5)`, and `rep(mySeq, mySeq)`.

   c) Concatenate the sequence `1:14` to the end of the vector described by `rep(mySeq,mySeq)` and store the resulting vector in the same `mySeq` variable. **Report** the length of `mySeq`.

   d) Create a square matrix populated row-wise from your `mySeq` vector and store it in a variable called `sqMtrx`. **Report** the vector of values described by the column sums of `sqMtrx`

   Answer:

   a)

```
myseq <- seq(from = 5, to = 50, by = 5)
myseq
```

```
##  [1]  5 10 15 20 25 30 35 40 45 50
```

   b)

   The second argument just being the number states the number of times to repeat the first argument. Adding the word each to the number in the second element repeats each element in the first arguemnt that number of times before moving the the second element. (so the first example is the sequence 5 times in a row form start the finish, the second is each element listed 5 times before going to the next element.) The third case takes each element and repeats it the number of times that element is. So 5 5's, 10 10's etc. So the number repeats the sequence that number of times, adding each takes each element and repeats it that number, and having two sequences takes the corresponding elements and repeats the first sequences element and repeats it the second sequences element number of times.

   c)

```
myseq <- seq(from = 5, to = 50, by = 5)
myseq <- c(rep(myseq, myseq),1:14)
myseq
```

```
##   [1]  5  5  5  5  5 10 10 10 10 10 10 10 10 10 10 15 15 15 15 15 15 15 15 15 15
##  [26] 15 15 15 15 15 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
##  [51] 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
##  [76] 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
## [101] 30 30 30 30 30 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
## [126] 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 40 40 40 40 40 40 40 40 40 40
## [151] 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
## [176] 40 40 40 40 40 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45
## [201] 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45
## [226] 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
## [251] 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
## [276]  1  2  3  4  5  6  7  8  9 10 11 12 13 14
```

   d)

```
sqMtrx <- matrix(myseq, nrow=sqrt(length(myseq)))
sqMtrx
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    5   15   20   25   25   30   30   35   35    40    40    45    45
##  [2,]    5   15   20   25   25   30   30   35   35    40    40    45    45
##  [3,]    5   15   20   25   25   30   30   35   35    40    40    45    45
##  [4,]    5   15   20   25   25   30   35   35   35    40    40    45    45
##  [5,]    5   15   20   25   25   30   35   35   40    40    40    45    45
##  [6,]   10   15   20   25   25   30   35   35   40    40    40    45    45
##  [7,]   10   15   20   25   25   30   35   35   40    40    40    45    45
##  [8,]   10   15   20   25   30   30   35   35   40    40    40    45    45
##  [9,]   10   15   20   25   30   30   35   35   40    40    40    45    45
## [10,]   10   15   20   25   30   30   35   35   40    40    40    45    45
## [11,]   10   15   20   25   30   30   35   35   40    40    45    45    45
## [12,]   10   15   20   25   30   30   35   35   40    40    45    45    45
## [13,]   10   15   20   25   30   30   35   35   40    40    45    45    45
## [14,]   10   20   20   25   30   30   35   35   40    40    45    45    45
## [15,]   10   20   20   25   30   30   35   35   40    40    45    45    45
## [16,]   15   20   20   25   30   30   35   35   40    40    45    45    45
## [17,]   15   20   25   25   30   30   35   35   40    40    45    45    45
##       [,14] [,15] [,16] [,17]
##  [1,]    45    50    50    50
##  [2,]    45    50    50    50
##  [3,]    45    50    50    50
##  [4,]    45    50    50     1
##  [5,]    50    50    50     2
##  [6,]    50    50    50     3
##  [7,]    50    50    50     4
##  [8,]    50    50    50     5
##  [9,]    50    50    50     6
## [10,]    50    50    50     7
## [11,]    50    50    50     8
## [12,]    50    50    50     9
## [13,]    50    50    50    10
## [14,]    50    50    50    11
## [15,]    50    50    50    12
## [16,]    50    50    50    13
## [17,]    50    50    50    14
```

```
colSums(sqMtrx)
```

```
##  [1] 155 275 345 425 475 510 580 595 660 680 715 765 765 830 850 850 255
```

6. Write a program that will do the following. Include your codes and necessary outputs to demonstrate your work.

   a) Generate 350,000 random numbers from a gamma distribution with `shape = 2` and `scale = 3` and store these numbers in a vector called `myVector`. **Report** a histogram of the numbers you just generated.

   b) Convert `myVector` into a matrix with 5,000 rows and assign it to an object called `myMatrix`. **Report** the dimensions of `myMatrix`.

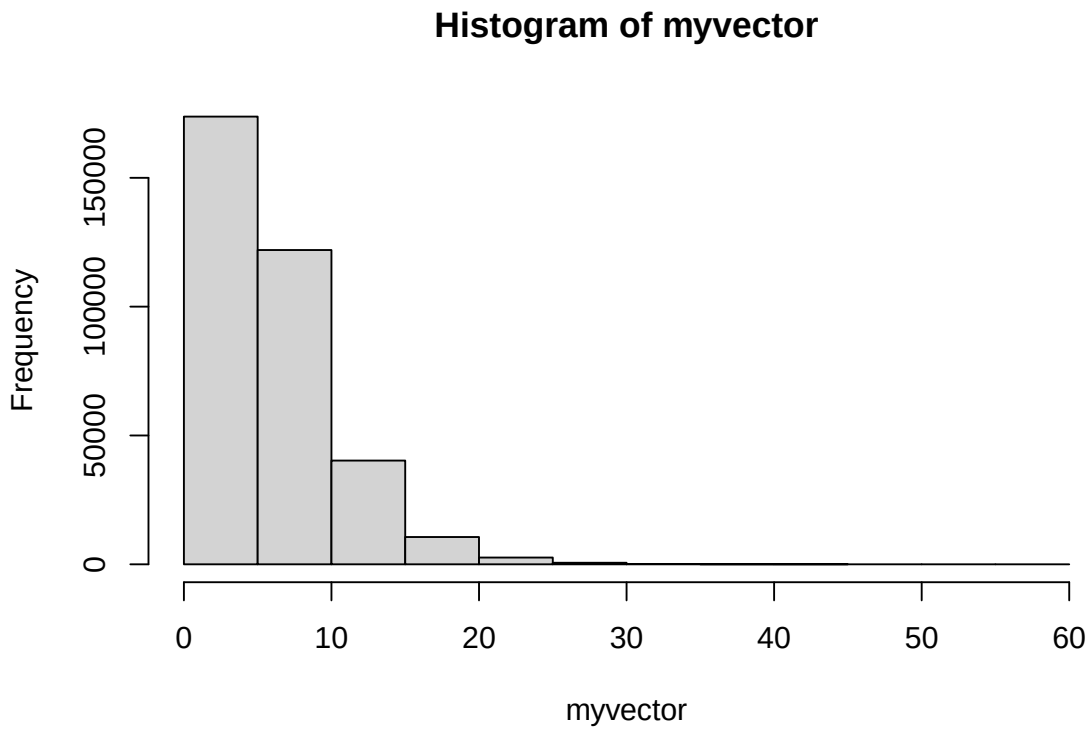
c) Compute the row means of `myMatrix` and **report** a histogram of those row means.

d) Explain why the two histograms you created in (6a) and (6c) have different shapes.

Answer:

a)

```
myvector <- rgamma(350000,shape=2, scale=3)
hist(myvector)
```



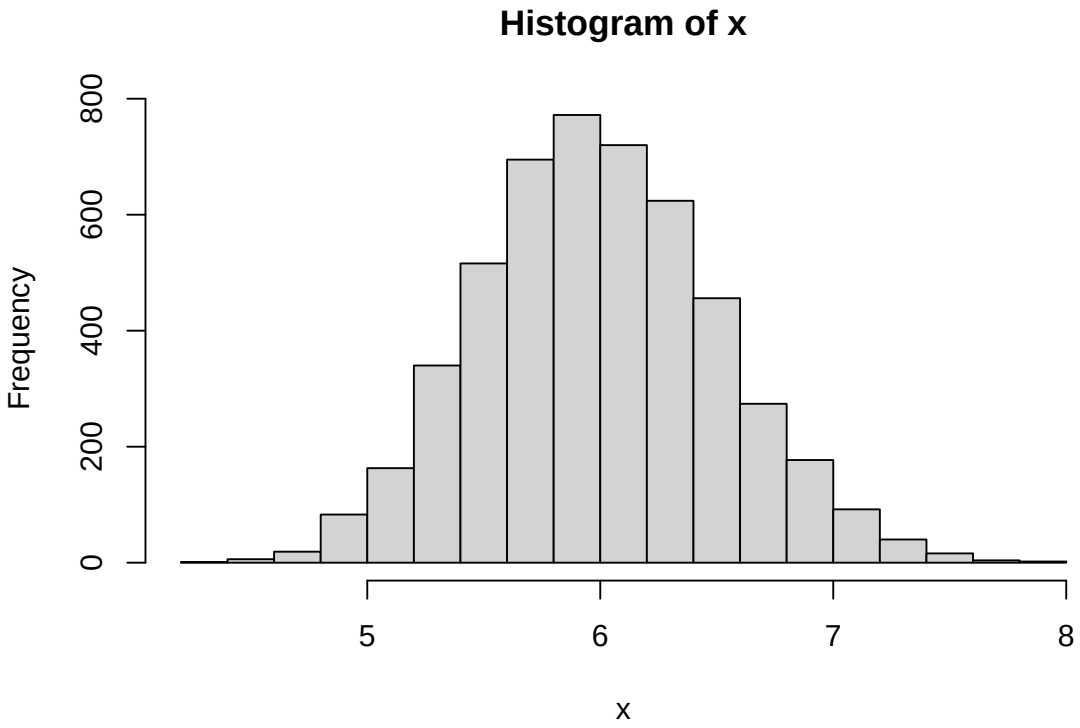
b)

```
mymatrix <- matrix(myvector, nrow=5000)
dim(mymatrix)
```

```
## [1] 5000   70
```

c)

```
x <- rowMeans(mymatrix)
hist(x)
```



d) By splitting the 350,000 elements into 5000 rows, we are taking 5000 samples of 350 random numbers generated by the gamma distribution. This number is large enough that the mean of these numbers should be close to the mean of the gamma distribution, which is always the shape multiplied by the scale, or in this case, three times two is 6. Central limit thereon can also be applied to explain why the second graph becomes the normal distribution, as as number of samples increases, averages converge to the normal distribution.

7. Perform the following reproducible procedure:

   a) Set a seed for the `R` random number generator using `set.seed()` and seed value 2019.

   b) Create a vector called `x` of 1,000 values from a normal distribution with mean 100 and standard deviation 20. **Report** the `summary()` of `x`.

   c) Create a second vector called `y` of 1,000 values from a normal distribution with mean 0 and standard deviation 4. **Report** the `summary()` of `y`.

   d) Create a data frame called `df` from your `x` and `y` vectors.

   e) Generate a scatterplot of `df`.

   f) **Report** the `tail()` of `df` as a nice table using `kable()`.

   Answer:

a)

```
set.seed(2019)
```

b)

```
x <- rnorm(1000,100,20)
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   35.28   85.45   97.36   98.44  111.97  170.83
```

c)

```
y <- rnorm(1000, 0, 4)
summary(y)
```

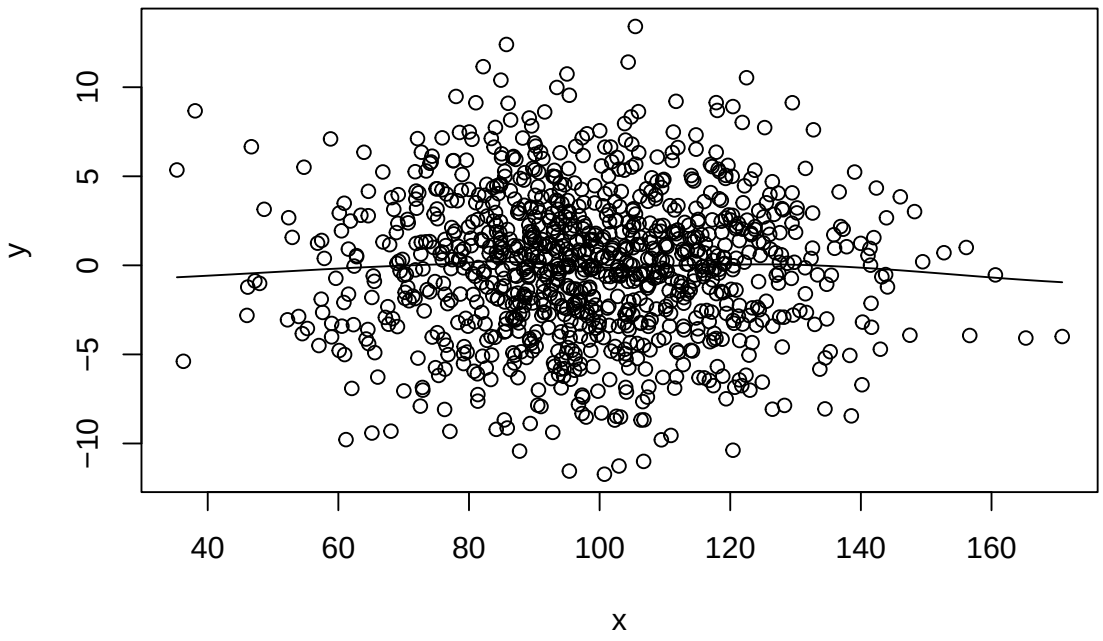
```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -11.72496  -2.76170   0.07867  -0.02481   2.55656  13.40945
```

d)

```
df <- data.frame(x,y)
```

e)

```
scatter.smooth(df)
```



f) Kable isnt a function that I have, is it a typo for table or did I need to use the library/package to get it? I tried downloading a few packages but wasn't able to get it to work.

```
table(tail(df))
```

```
##                   y
```

```
## x                 -5.82310585600454 -3.07351522862379 -2.85040516975856
##   81.1889336368027                 0                 0                 0
##   84.7300838601966                 0                 0                 0
##   87.0862997920432                 0                 1                 0
##   94.0085057350294                 1                 0                 0
##   119.208785474849                 0                 0                 0
##   121.248162923235                 0                 0                 1
##                   y
## x                 -2.06609188415735 1.13759511501715 1.82588275666864
##   81.1889336368027                 1                 0                 0
##   84.7300838601966                 0                 0                 1
##   87.0862997920432                 0                 0                 0
##   94.0085057350294                 0                 0                 0
##   119.208785474849                 0                 1                 0
##   121.248162923235                 0                 0                 0
```

8. Based on our lecture notes, answer the following questions. Show your answer presenting the relevant `R` code.

   a) We have a vector of values `x = c(2,4,5, "3.5")`. What would be the mode of the vector `x`?

   b) How do you load a package into `R`? Show that loading `ggplot2` package.

   c) Missing values are shown as `NA` in `R`. How can you check if there is any missing values in a vector `y = c(3,5,8, NA, 6)`?

   Answer:

a) Character

```
x <- c(2,4,5, "3.5")
mode(x)
```

```
## [1] "character"
```

b) If you leave the input blank you get a list of every package you can download.

```
install.packages("ggplot2", repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/lojre/OneDrive/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)
```

```
## package 'ggplot2' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\lojre\AppData\Local\Temp\RtmpC0RDDT\downloaded_packages
```

c)

```
y = c(3,5,8, NA, 6)
is.na(y)
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE
```