Assignment #1

Tianqi Wang

July 16, 2018

Question 1

1.1

```
# Creating a vector to store all course numbers
courseNum <- c("MSAN501", "MSAN502", "MSAN504", "MSAN593")

# Creating a vector to store all course names
courseName <- c("Computation for Analytics", "Review of Linear Algebra",
    "Review of Probability and Statistics", "Explonatory Data Analysis")

# Creating a vector to store names of instructors for each
# course
courseProf <- c("Terence J Parr", "Xuemei Chen", "Jeff Hamrick",
    "Paul Intrevado")

# Creating a vector indicating which courses I formally
# enrolled in
enrolled <- c(T, T, F, T)

# Creatomg a vector to store my anticipated letter grade in
# each course
anticipatedGrade <- c("A", "B", NA, "C")

# Creating a vector to store my anticipated hours spent on
# each class per week
anticipatedHours <- c(20, 10, NA, 10)
```

Create a table summarizing the type and class for each vector.

Vector	Class	Type
courseNum	character	character
courseName	character	character
courseProf	character	character
enrolled	logical	logical
anticipatedGrade	character	character
anticipatedHours	numeric	double

1.2

```
bootcampDataFrame <- data.frame(courseNum, courseName, courseProf,
    enrolled, anticipatedGrade, anticipatedHours)
```

Vector	Class	Type
courseNum	factor	integer
courseName	factor	integer

Vector	Class	Type
courseProf	factor	integer
enrolled	logical	logical
anticipatedGrade	factor	integer
anticipatedHours	numeric	double

In the data frame, some variable retain their original classes/types while some do not.

1.3

```
bootcampDataList <- list(courseNum = courseNum, courseName = courseName,
    courseProf = courseProf, enrolled = enrolled, anticipatedGrade = anticipatedGrade,
    anticipatedHours = anticipatedHours)
```

Vector	Class	Type
courseNum	character	character
courseName	character	character
courseProf	character	character
enrolled	logical	logical
anticipatedGrade	character	character
anticipatedHours	numeric	double

In the list, all variables retain the original classes/types.

1.4

```
# The total number of hours you anticipate spending on
# coursework, both per week, and over all of boot camp.
knitr::kable(data.frame(courseName, anticipatedHours_perweek = anticipatedHours,
    anticipatedHours_total = anticipatedHours * 5))
```

courseName	anticipatedHours_perweek	anticipatedHours_total
Computation for Analytics	20	100
Review of Linear Algebra	10	50
Review of Probability and Statistics	NA	NA
Explonatory Data Analysis	10	50

```
# A data frame with only the third row and first two columns
# of bootcampDataFrame
knitr::kable(bootcampDataFrame[3, 1:2])
```

	courseNum	courseName
3	MSAN504	Review of Probability and Statistics

```
# The first value in the second element of bootcampDataList
bootcampDataList[[2]][1]
```

```
## [1] "Computation for Analytics"
```

1.5

```
# Convert the anticipatedGrade variable in bootcampDataFrame
# into an ordinal factor
bootcampDataFrame$anticipatedGrade <- factor(bootcampDataFrame$anticipatedGrade,
    levels = c("C", "B", "A"), ordered = T)

# The maximum letter grade you anticipate receiving in
# boot-camp
max(bootcampDataFrame$anticipatedGrade, na.rm = T)
```

```
## [1] A
## Levels: C < B < A
```

```
# The name and course number of that class
paste(bootcampDataFrame$courseNum[which.max(bootcampDataFrame$anticipatedGrade)],
    ":", bootcampDataFrame$courseName[which.max(bootcampDataFrame$anticipatedGrade)])
```

```
## [1] "MSAN501 : Computation for Analytics"
```

Question 2

2.1

Read in the file `titanic.csv` and store the data in the data frame `titanicData`.

```
titanicData <- read.csv("~/Desktop/titanic.csv")
```

2.2

Rows in this data frame

```
nrow(titanicData)
```

```
## [1] 891
```

2.3

Columns in this data frame

```
ncol(titanicData)
```

```
## [1] 12
```

2.4

Variable that has the most `NA` entries.

```
names(titanicData)[which.max(apply(is.na(titanicData), 2, FUN = sum))]
```

```
## [1] "Age"
```

2.5

Variables should be converted to a different type than the default type they were importedas.

```r
# Check types of variables when they were imported.
knitr::kable(sapply(titanicData, typeof), col.names = c("Type"),
    row.names = T)
```

	Type
PassengerId	integer
Survived	integer
Pclass	integer
Name	integer
Sex	integer
Age	double
SibSp	integer
Parch	integer
Ticket	integer
Fare	double
Cabin	integer
Embarked	integer

`Name` should be converted to character;

`Sex` should be converted to character;

`Cabin` should be converted to character;

`Embarked` should be converted to character;

```r
# Convert types of those variables and store the related
# information in the type data frame
titanicData$Name <- as.character(titanicData$Name)
titanicData$Sex <- as.character(titanicData$Sex)
titanicData$Cabin <- as.character(titanicData$Cabin)
titanicData$Embarked <- as.character(titanicData$Embarked)

type <- data.frame(Changed_Variable = c("Name", "Sex", "Cabin",
    "Embarked"), DefaultType = rep("Integer", 4), ChangedType = rep("Character",
    4))
knitr::kable(type)
```

Changed_Variable	DefaultType	ChangedType
Name	Integer	Character
Sex	Integer	Character
Cabin	Integer	Character
Embarked	Integer	Character

2.6

Coerce the `survived` variable into type `logical`

```r
titanicData$Survived <- as.logical(titanicData$Survived)
```

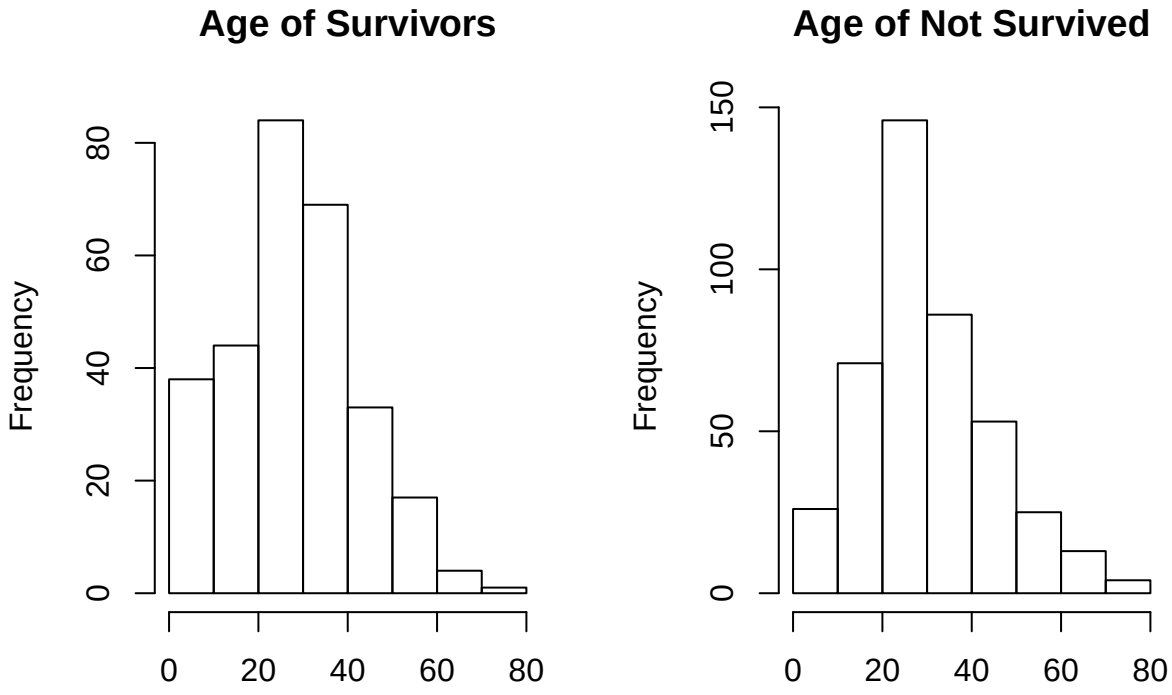
```r
# Mean age of survivors.
mean(titanicData$Age[titanicData$Survived == TRUE], na.rm = T)
```

```
## [1] 28.34369
```

```r
# The mean age of those who did not survive.
mean(titanicData$Age[titanicData$Survived != TRUE], na.rm = T)
```

```
## [1] 30.62618
```

```r
# Plot side-by-side histograms of the ages of survivors and
# non-survivors.
par(mfrow = c(1, 2))
hist(titanicData$Age[which(titanicData$Survived == TRUE)], main = "Age of Survivors",
    xlab = NULL)
hist(titanicData$Age[which(titanicData$Survived == FALSE)], main = "Age of Not Survived",
    xlab = NULL)
```



2.7

```r
# First 10 values in cabin.
head(titanicData$Cabin, 10)
```

```
##  [1] ""     "C85"  ""     "C123" ""     ""     "E46"  ""     ""     ""
```

```r
# Replace all blanks with NAs.
titanicData[titanicData == ""] = NA
```

2.8

```r
# The percentage of the observations for age are NAs.
scales::percent(sum(is.na(titanicData$Age))/length(titanicData$Age))
```

```
## [1] "19.9%"
```

```r
# Imuptation
titanicData$Age[is.na(titanicData$Age)] = mean(titanicData$Age,
    na.rm = T)
```

Since `mean` could be highly influenced by some extreme values, if we fill all `NA`s with the mean, which could magnify those influence from extreme values, cause the result to be biased.

Question 3

3.1

Generate 100 random variables $\sim \mathcal{U}\{-1, 1\}$ and compute the mean and variance (no need to set the seed for this exercise).

```
r100 <- runif(100, min = -1, max = 1)
mean(r100)
```

```
## [1] -0.05166717
```

```
var(r100)
```

```
## [1] 0.3397863
```

Repeat the previous step for sample sizes of 1,000, 10,000, 100,000 and 1,000,000, computing the mean and variance for each sample size.

```
r1000 <- runif(1000, min = -1, max = 1)
mean(r1000)
```

```
## [1] 0.03687712
```

```
var(r1000)
```

```
## [1] 0.3162699
```

```
r10000 <- runif(10000, min = -1, max = 1)
mean(r10000)
```

```
## [1] 0.004504532
```

```
var(r10000)
```

```
## [1] 0.3373948
```

```
r100000 <- runif(100000, min = -1, max = 1)
mean(r100000)
```

```
## [1] 0.002760251
```

```
var(r100000)
```

```
## [1] 0.3332065
```

```
r1000000 <- runif(1000000, min = -1, max = 1)
mean(r1000000)
```

```
## [1] -0.0005336553
```

```
var(r1000000)
```

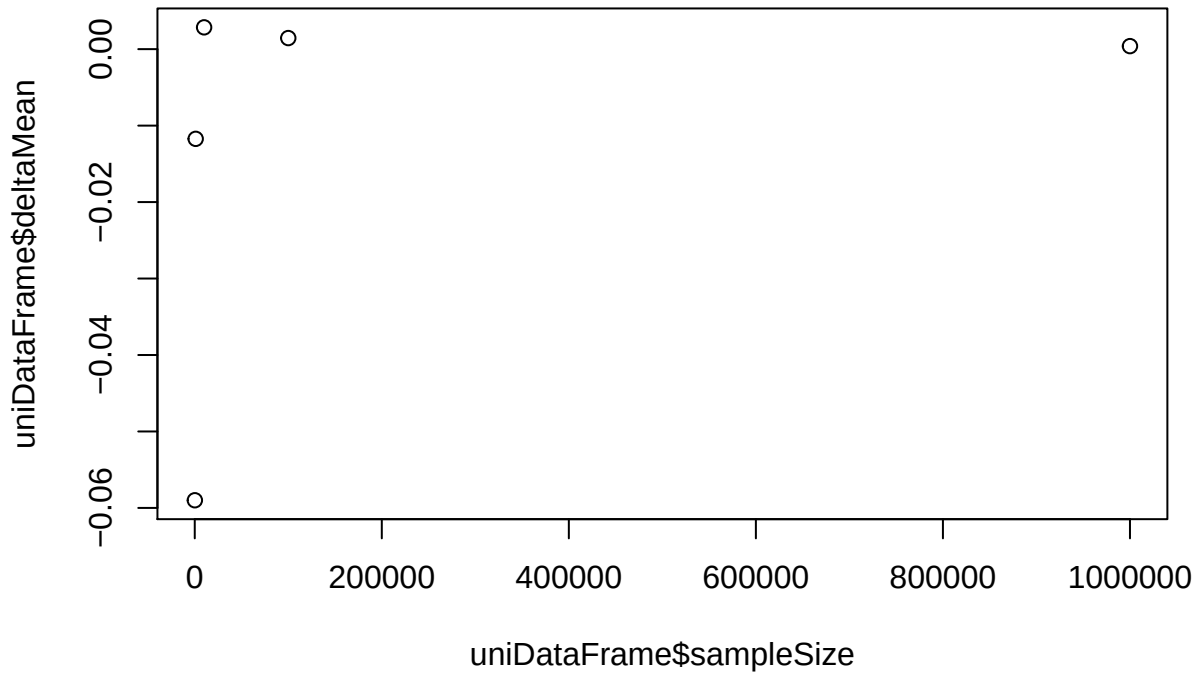
```
## [1] 0.3328572
```

Create a data frame called `unifDataFrame` with seven variables: `sampleSize`, `theoreticalMean`, `sampleMean`, `deltaMean`, `theoreticalVariance`, `sampleVariance`, `deltaVariance`.

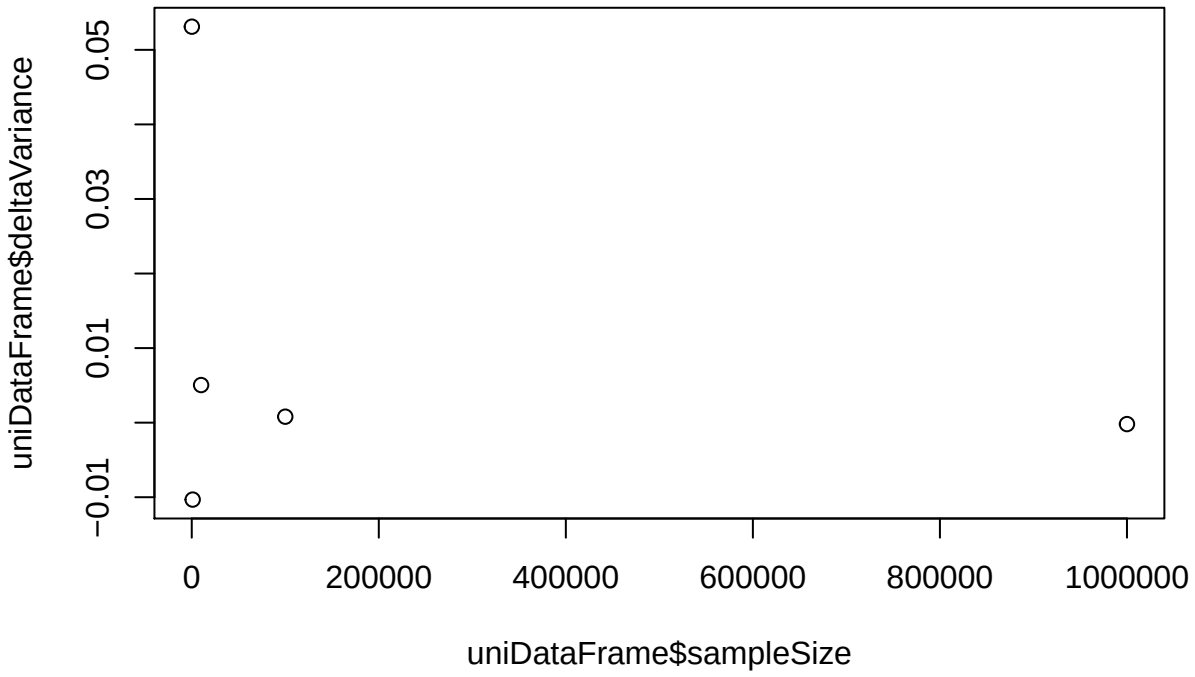
```
# Create the blank data frame uniDataFrame
uniDataFrame <- data.frame(sampleSize=integer(5),theoreticalMean=numeric(5),
                sampleVariance=numeric(5),deltaVariance=numeric(5))
```

```r
# Using a loop to generate data in uniDataFrame
for (i in (1:5)){
    uniDataFrame$sampleSize[i]=10^(i+1);
    uniDataFrame$theoreticalMean[i]=10^(i+1)*0; #
    uniDataFrame$sampleMean[i]=mean(runif(10^(i+1),-1,1));
    uniDataFrame$deltaMean[i]=uniDataFrame$sampleMean[i]-uniDataFrame$theoreticalMean[i];
    uniDataFrame$theoreticalVariance[i]=(1+1)^2/12;
    uniDataFrame$sampleVariance[i]=var(runif(10^(i+1),-1,1));
    uniDataFrame$deltaVariance[i]=uniDataFrame$sampleVariance[i]-
        uniDataFrame$theoreticalVariance[i]
}
```

```r
# Plot the deltaMean.
plot(uniDataFrame$deltaMean ~ uniDataFrame$sampleSize)
```



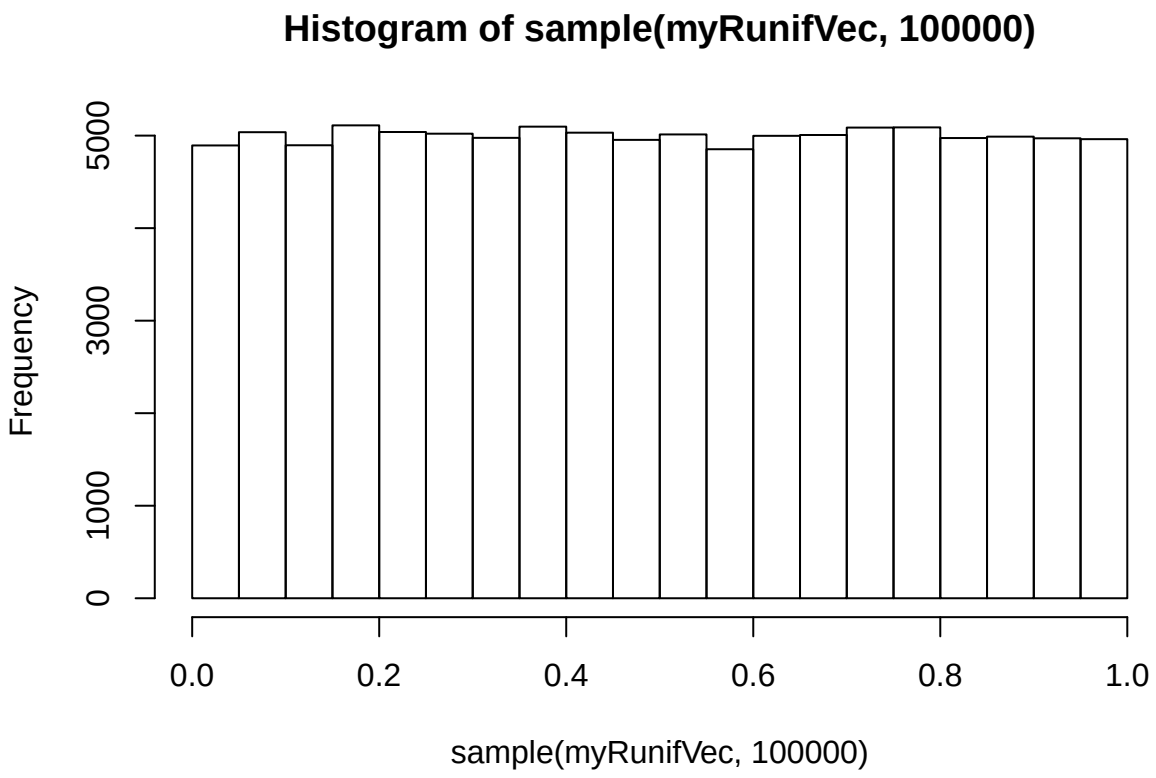

```r
# Plot the deltaVariance.
plot(uniDataFrame$deltaVariance ~ uniDataFrame$sampleSize)
```



3.2

```
# Generate the vector of random variables and store them in
# myRunifVec
myRunifVec = runif(10000000, 0, 1)

# Create the histogram of the subsample
hist(sample(myRunifVec, 100000))
```

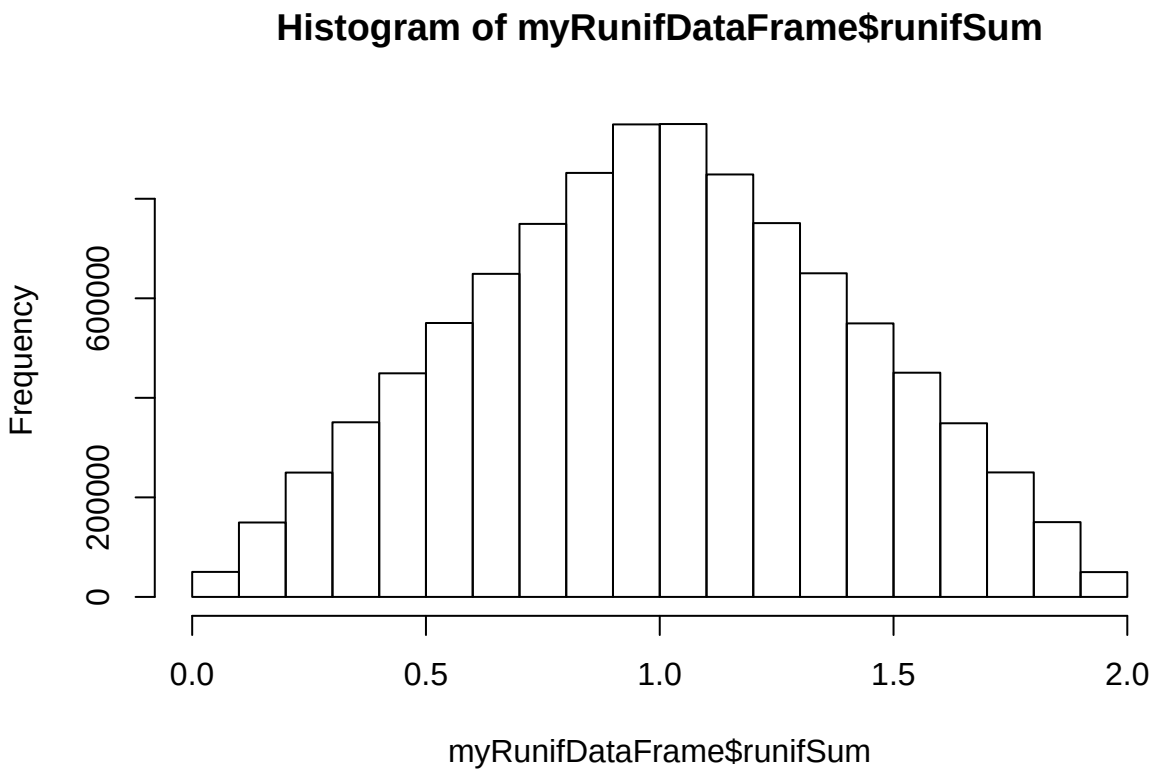


3.3

```r
# Create the data frame store two random variables
myRunifDataFrame <- data.frame(col1 = runif(10000000, 0, 1),
    col2 = runif(10000000, 0, 1))

# Create the third variable to sum up col1 and col2
myRunifDataFrame$runifSum = myRunifDataFrame$col1 + myRunifDataFrame$col2

# Create the histogram of runifSum to check the distribution
hist(myRunifDataFrame$runifSum)
```

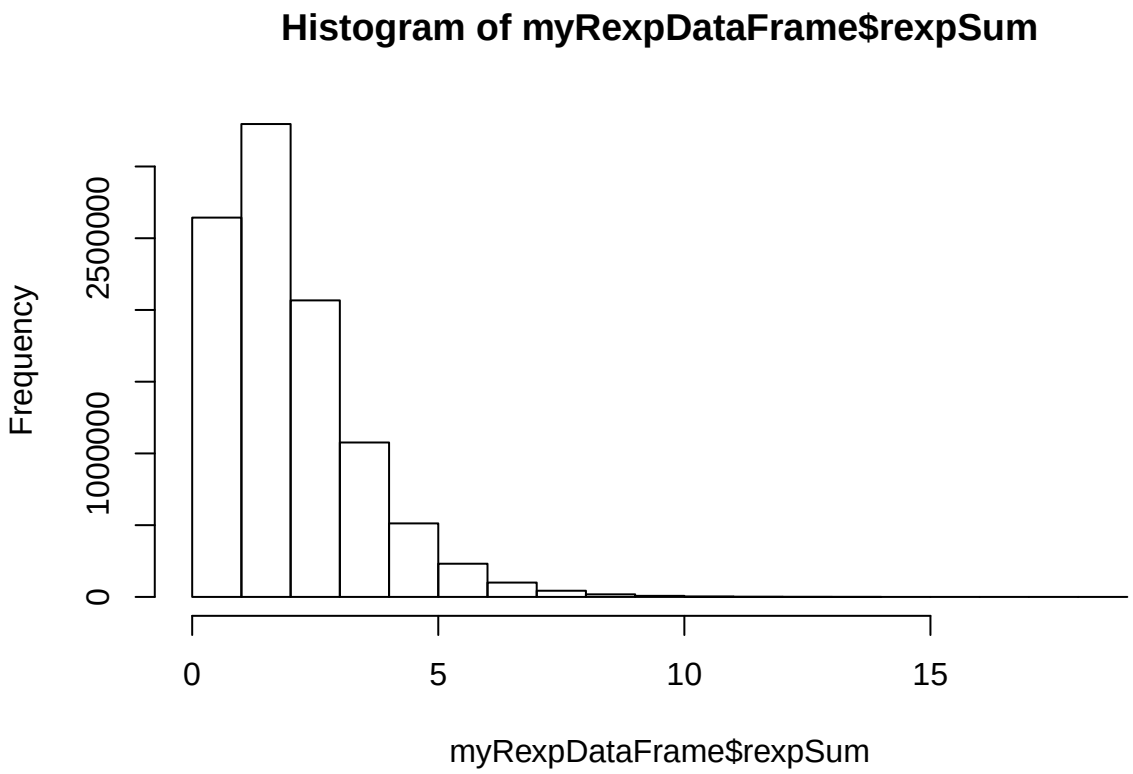


3.4

```
# Create the data frame store two random variables
myRexpDataFrame <- data.frame(col1 = rexp(10000000, 1), col2 = rexp(10000000,
    1))

# Create the third variable to sum up col1 and col2
myRexpDataFrame$rexpSum = myRexpDataFrame$col1 + myRexpDataFrame$col2

# Create the histogram of rexpum to check the distribution
hist(myRexpDataFrame$rexpSum)
```



Question 4

```
set.seed(100)
x_1 <- runif(100000, -100, 100)
y_1 <- rexp(100000, rate = 0.5)
```

4.1

```
# Compute the coefficients for the simple linear regression.
b1_1 <- as.numeric((x_1 - mean(x_1)) %*% (y_1 - mean(y_1))/((x_1 -
    mean(x_1)) %*% (x_1 - mean(x_1))))
b0_1 <- (sum(y_1 - b1_1 * x_1))/length(x_1)
```

4.2

Since we have

$$\hat{Y}_i = b_0 + b_1 X_i$$

$$SSE = \sum_{i=1}^{n} \left(Y_i - \hat{Y}\right)^2$$

$$SSR = \sum_{i=1}^{n} \left(\hat{Y}_i - \bar{Y}\right)^2$$

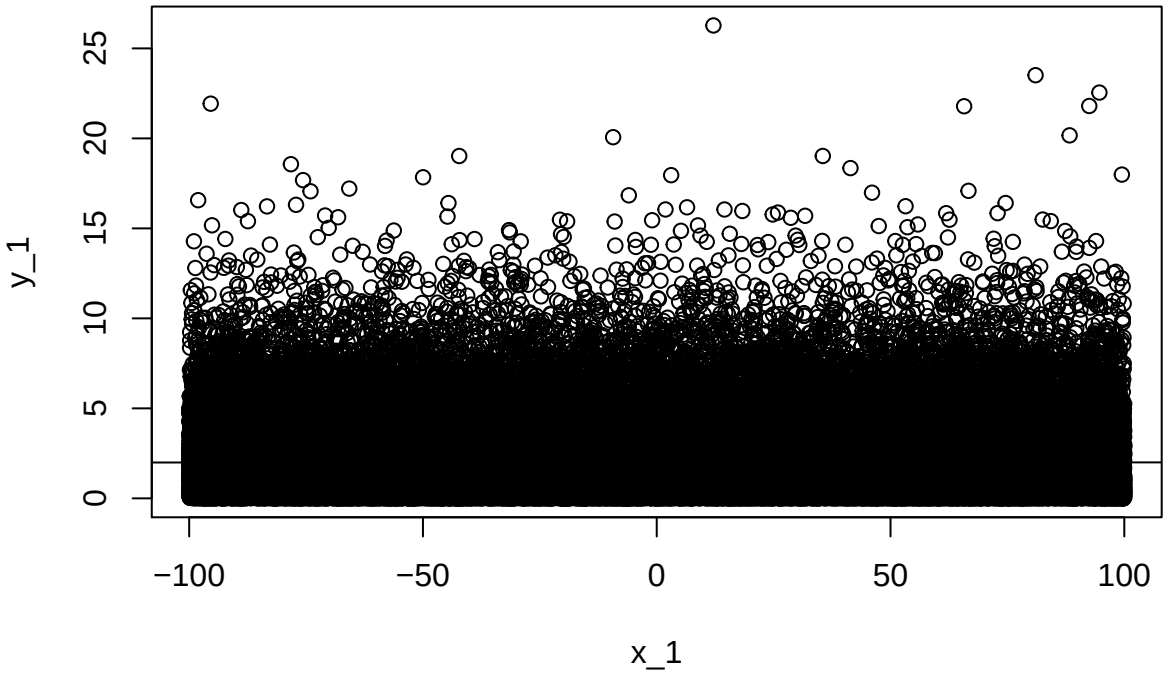
$$SST = SSR + SSE$$

$$R^2 = \frac{SSR}{SST}$$

```r
# Compute SSE, SSR, SST and R2
SSE_1 <- sum((y_1 - b0_1 - b1_1 * x_1) %*% (y_1 - b0_1 - b1_1 *
    x_1))
SSR_1 <- sum((b0_1 + b1_1 * x_1 - mean(y_1)) %*% (b0_1 + b1_1 *
    x_1 - mean(y_1)))
SST_1 <- sum((y_1 - mean(y_1)) %*% (y_1 - mean(y_1)))
R2_1 <- SSR_1/SST_1
```

4.3

```r
# Plot y on x, and draw the fitted regression line
plot(y_1 ~ x_1)
abline(b0_1, b1_1)
```

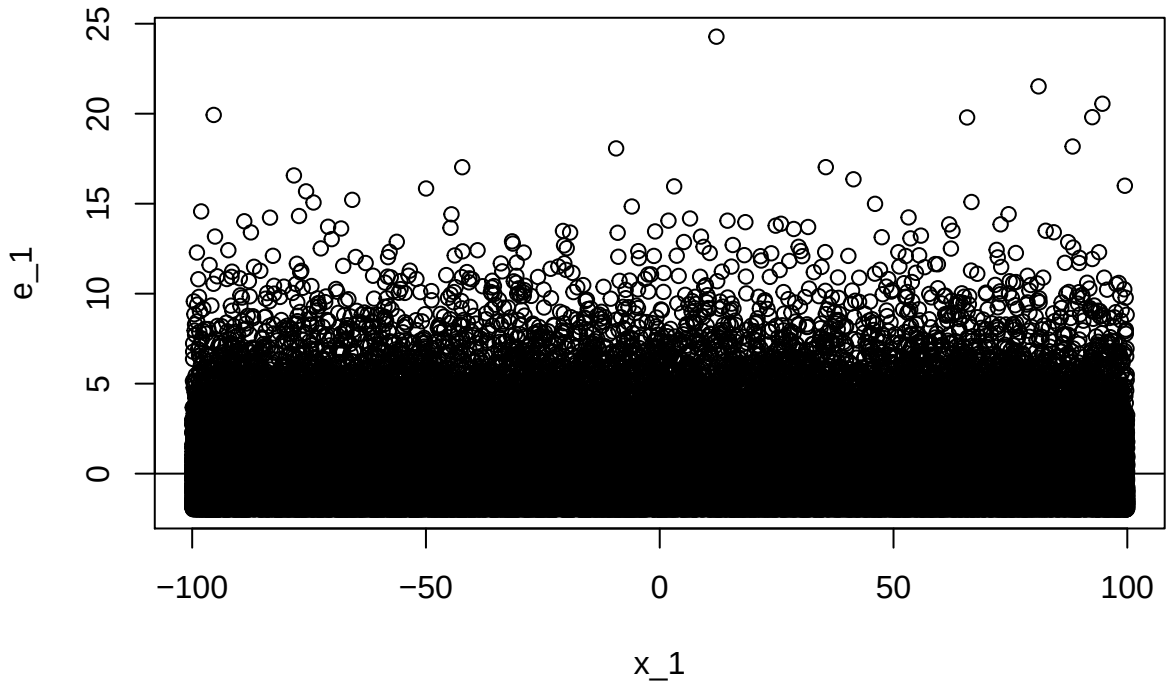


##4.4

```r
# Compute the residuals
e_1 <- y_1 - (b0_1 - b1_1 * x_1)
```

4.5

```r
# Generate a residual plot
plot(x_1, e_1)
abline(h = 0)
```



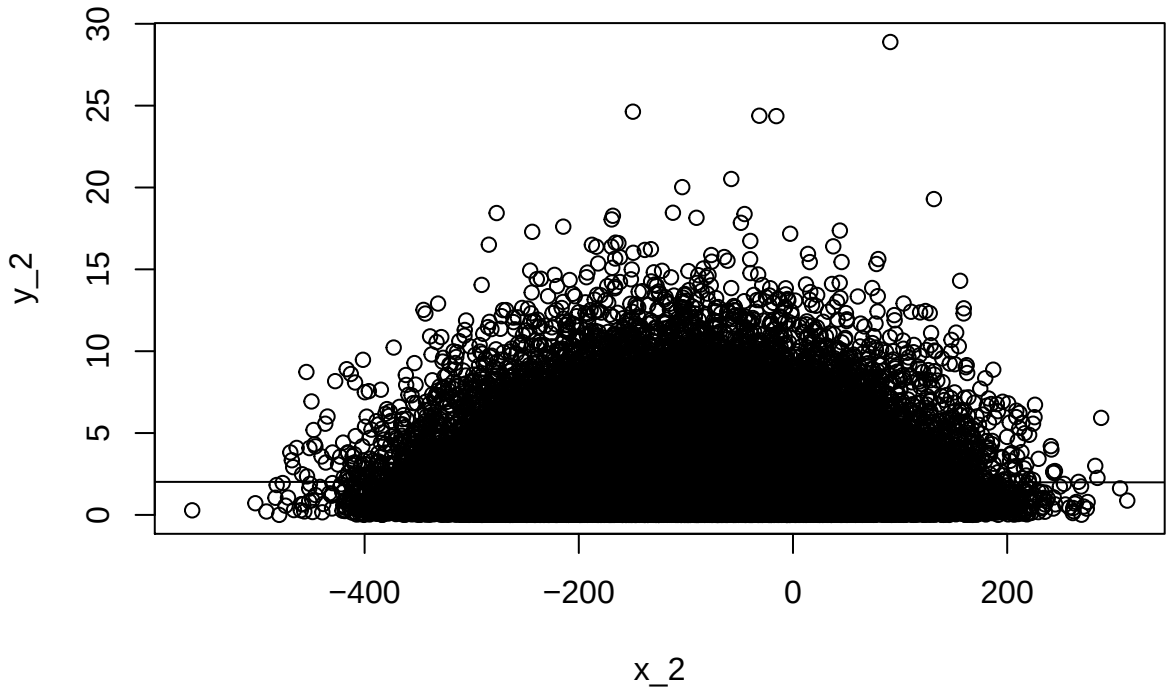
Repeat using other distribution

```
set.seed(999)
x_2 <- rnorm(100000, -100, 100)
y_2 <- rexp(100000, rate = 0.5)
```

```
# Compute the coefficients for the simple linear regression.
b1_2 <- as.numeric((x_2 - mean(x_2)) %*% (y_2 - mean(y_2))/((x_2 -
    mean(x_2)) %*% (x_2 - mean(x_2))))
b0_2 <- (sum(y_2 - b1_2 * x_2))/length(x_2)
```

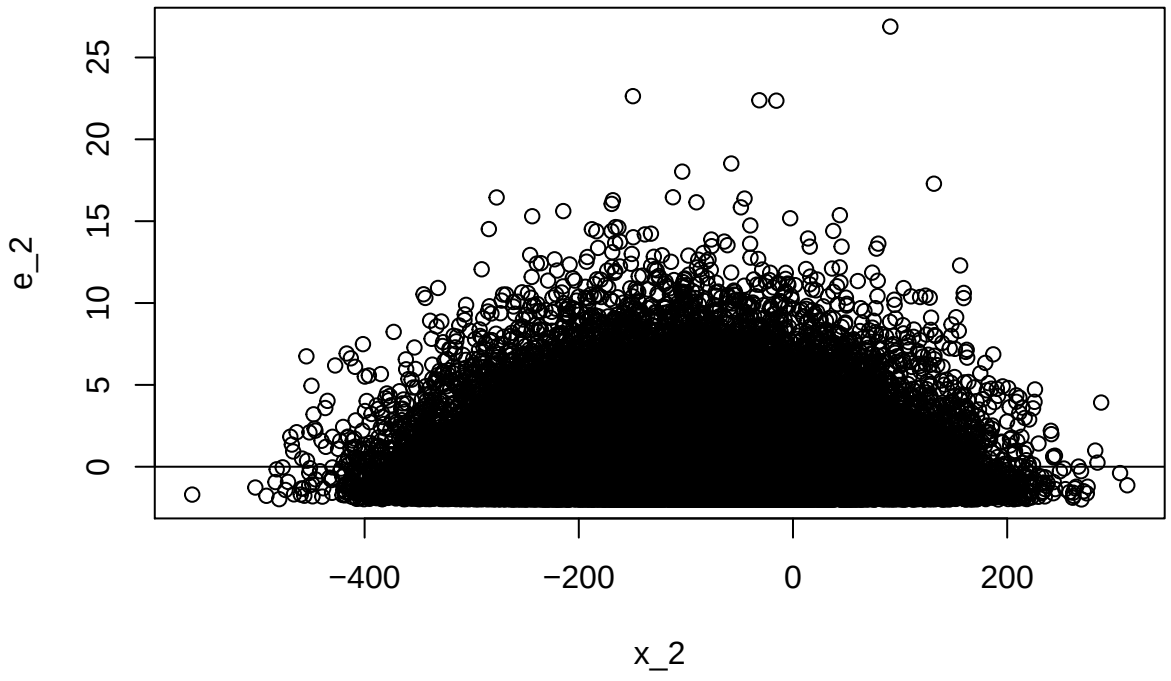
```
# Compute SSE, SSR, SST and R2
SSE_2 <- sum((y_2 - b0_2 - b1_2 * x_2) %*% (y_2 - b0_2 - b1_2 *
    x_2))
SSR_2 <- sum((b0_2 + b1_2 * x_2 - mean(y_2)) %*% (b0_2 + b1_2 *
    x_2 - mean(y_2)))
SST_2 <- sum((y_2 - mean(y_2)) %*% (y_2 - mean(y_2)))
R2_2 <- SSR_2/SST_2
```

```
# Plot y on x, and draw the fitted regression line
plot(y_2 ~ x_2)
abline(b0_2, b1_2)
```




```
# Compute the residuals
e_2 <- y_2 - (b0_2 - b1_2 * x_2)
```

```
# Generate a residual plot
plot(x_2, e_2)
abline(h = 0)
```



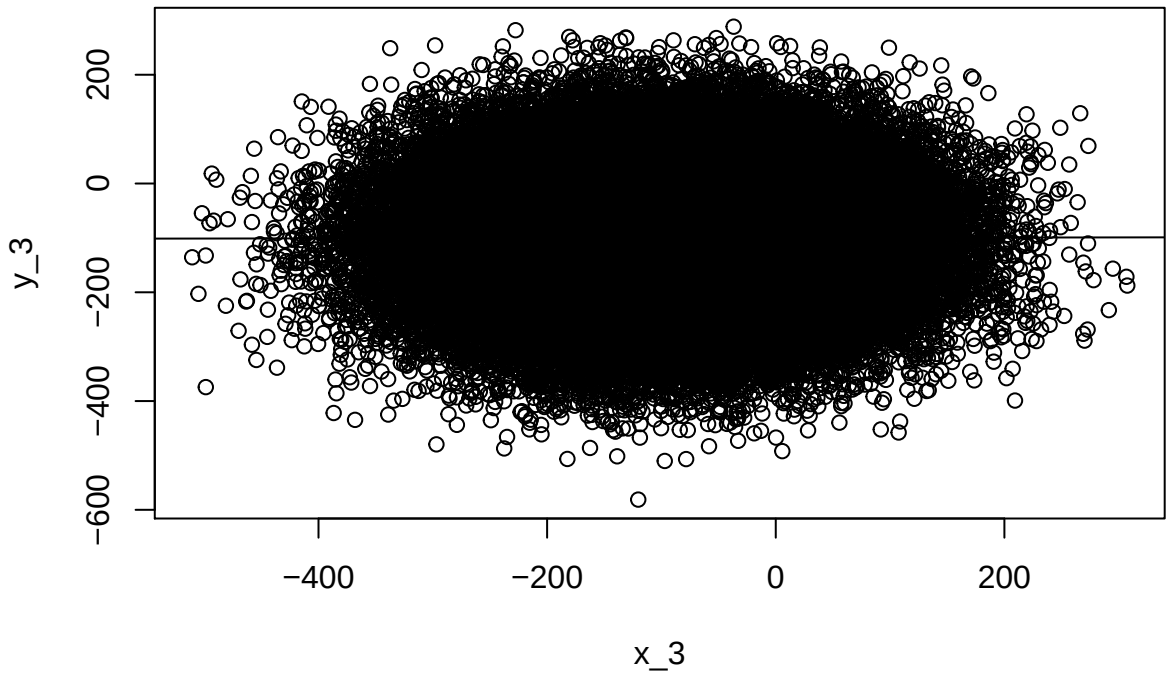
The third distribution

```
set.seed(543)
x_3 <- rnorm(100000, -100, 100)
y_3 <- rnorm(100000, -100, 100)
```

```r
# Compute the coefficients for the simple linear regression.
b1_3 <- as.numeric((x_3 - mean(x_3)) %*% (y_3 - mean(y_3))/((x_3 -
    mean(x_3)) %*% (x_3 - mean(x_3))))
b0_3 <- (sum(y_3 - b1_3 * x_3))/length(x_3)
```

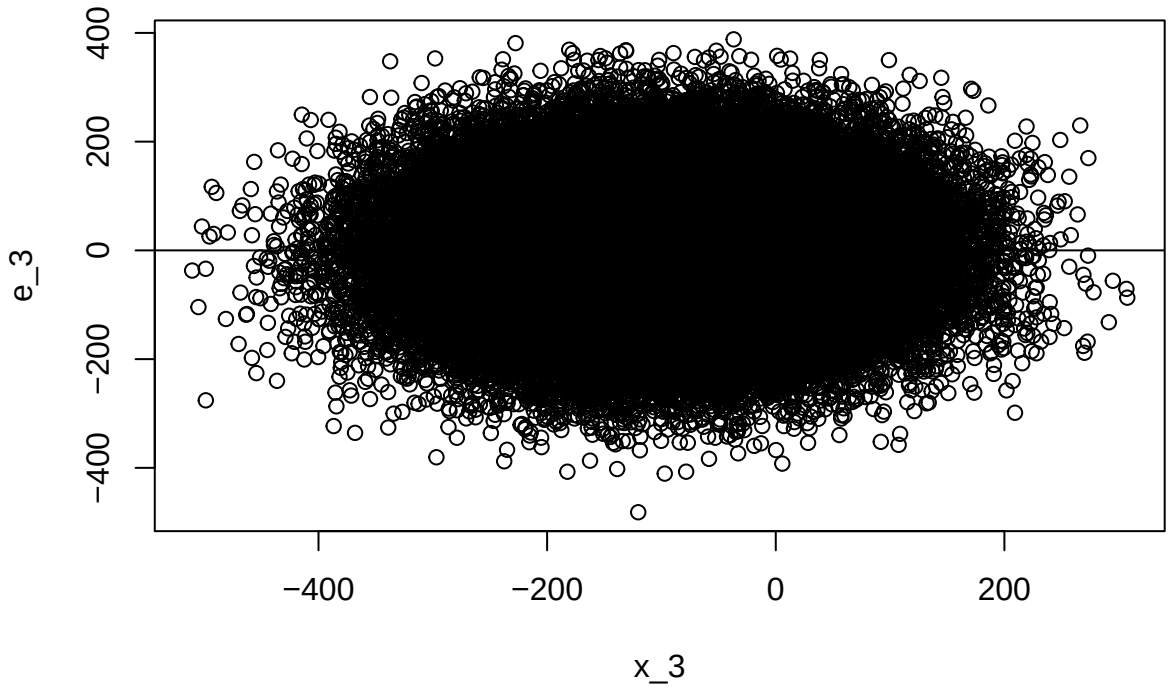
```r
# Compute SSE, SSR, SST and R3
SSE_3 <- sum((y_3 - b0_3 - b1_3 * x_3) %*% (y_3 - b0_3 - b1_3 *
    x_3))
SSR_3 <- sum((b0_3 + b1_3 * x_3 - mean(y_3)) %*% (b0_3 + b1_3 *
    x_3 - mean(y_3)))
SST_3 <- sum((y_3 - mean(y_3)) %*% (y_3 - mean(y_3)))
R2_3 <- SSR_3/SST_3
```

```r
# Plot y on x, and draw the fitted regression line
plot(y_3 ~ x_3)
abline(b0_3, b1_3)
```




```r
# Compute the residuals
e_3 <- y_3 - (b0_3 - b1_3 * x_3)
```

```r
# Generate a residual plot
plot(x_3, e_3)
abline(h = 0)
```



Create a table comparing the models

```
b_0 <- c(b0_1, b0_2, b0_3)
b_1 <- c(b1_1, b1_2, b1_3)
SSE <- c(SSE_1, SSE_2, SSE_3)
SSR <- c(SSR_1, SSR_2, SSR_3)
SSTO <- c(SST_1, SST_2, SST_3)
R2 <- c(R2_1, R2_2, R2_3)
finaltable <- data.frame(b_0, b_1, SSE, SSR, SSTO, R2, row.names = c("Model 1",
    "Model 2", "Model 3"))
names(finaltable) <- c("$b_0$", "$b_1$", "SSE", "SSR", "SSTO",
    "$R^2$")
knitr::kable(t(finaltable), )
```

	Model 1	Model 2	Model 3
b_0	1.9950361	1.9990315	-99.8692149
b_1	0.0000242	-0.0000321	0.0025987
SSE	396806.1241233	398628.3441648	999572341.9479262
SSR	0.1949741	1.0323308	6728.3929640
SSTO	396806.3190974	398629.3764956	999579070.3408700
R^2	0.0000005	0.0000026	0.0000067

Since the coefficient b_1 and R^2 are really small, those x has really weak explanation power toward y, in all three models. But in the third model, since the scale of y is higher, lead to higher scale of SSE,SSR and SST, but R^2 actually does not improve much.