Assignment One - MSAN 593

Louise Y Lai

July 18, 2018

Question 1

Create the following vectors, populated with information about the four MSAN boot-camp classes, create a table summarizing the type and class for each vector

```
courseNum <- c("593", "501", "504", "502")
courseName <- c("Exploratory Data Analysis", "Computation for Analytics",
                "Review of Probability and Statistics", "Linear Algebra")
courseProf <- c("Paul Intrevado", "Terrence Parr", "Jeff Hamrick", "Xuemei Chen")
enrolled <- as.logical(c(1, 1, 1, 0))
anticipatedGrade <- c("A", "C", "B", NA)
anticipatedHours <- c(15, 10, 15, NA)

# make Matrix
names <- c("courseNum", "courseName", "courseProf", "enrolled", "anticipatedGrade",
           "anticipatedHours")
types <- c(typeof(courseNum), typeof(courseName), typeof(courseProf), typeof(enrolled),
           typeof(anticipatedGrade), typeof(anticipatedHours))
class <- c(class(courseNum), class(courseName), class(courseProf), class(enrolled),
           class(anticipatedGrade), class(anticipatedHours))

myMatrix <- matrix(c(names, types, class), nrow=6, ncol=3, byrow=FALSE)
colnames(myMatrix) <- c("items", "type", "class")
myMatrix
```

```
##      items              type        class
## [1,] "courseNum"        "character" "character"
## [2,] "courseName"       "character" "character"
## [3,] "courseProf"       "character" "character"
## [4,] "enrolled"         "logical"   "logical"
## [5,] "anticipatedGrade" "character" "character"
## [6,] "anticipatedHours" "double"    "numeric"
```

1.2) Create a data frame called bootcampDataFrame by combining all of the above vectors and create another table summarizing the type and class for the data frame. Do the data frame variables retain their original types/classes?

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

bootcampDataFrame
```

```
##   courseNum                         courseName     courseProf enrolled
```

```
## 1       593            Exploratory Data Analysis Paul Intrevado     TRUE
## 2       501            Computation for Analytics  Terrence Parr     TRUE
## 3       504 Review of Probability and Statistics   Jeff Hamrick     TRUE
## 4       502                       Linear Algebra    Xuemei Chen    FALSE
##   anticipatedGrade anticipatedHours
## 1                A               15
## 2                C               10
## 3                B               15
## 4             <NA>               NA
```

```
types_2 <-  c()
class_2 <-  c()

for(i in 1:length(names)){
  types_2[i] <- typeof(bootcampDataFrame[,i])
  class_2[i] <- class(bootcampDataFrame[,i])
}

myMatrix_2 <- matrix(c(names, types_2, class_2), nrow=6, ncol=3, byrow=FALSE)
colnames(myMatrix_2) <- c("items", "type", "class")

# no, character in list turns into type integer and type factor in list
```

1.3) Combine the vectors from 1.1 into a list called bootcampDataList, where each vector is an element of the list. Assign the names of each element to be the names of the original vectors. Do the elements of the list maintain their original types/classes?

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

names(bootcampDataList) <- names
types_3 <- c()
class_3 <- c()

for(i in 1:length(names)){
  types_3[i] <- typeof(bootcampDataList[[i]])
  class_3[i] <- class(bootcampDataList[[i]])
}

# yes, they do retain their types from the initial matrix, unlike when we did the dataframe
```

1.4) Write code that returns the following values in code chunks using echo = TRUE so that your code as well as your output is displayed after each calculation:

```
# number of hours anticpate spending on coursework, per week
sum(bootcampDataFrame$anticipatedHours, na.rm=TRUE) #40
```

```
## [1] 40
```

```
# number of hours anticpate spending on coursework, over all bootcamp
sum(bootcampDataFrame$anticipatedHours, na.rm=TRUE) * 5 #200
```

```
## [1] 200
```

```
# data frame with onlt the third row and first two columns of `bootcampDataFrame`
bootcampDataFrame[3,1:2]
```

```
##   courseNum                          courseName
## 3       504 Review of Probability and Statistics
```

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

```
## [1] "Exploratory Data Analysis"
```

```
bootcampDataFrame$anticipatedGrade <- factor(bootcampDataFrame$anticipatedGrade,
                                             levels=c("C", "B", "A"), order=TRUE)

bootcampDataFrame
```

```
##   courseNum                          courseName     courseProf enrolled
## 1       593            Exploratory Data Analysis Paul Intrevado     TRUE
## 2       501            Computation for Analytics  Terrence Parr     TRUE
## 3       504 Review of Probability and Statistics   Jeff Hamrick     TRUE
## 4       502                       Linear Algebra    Xuemei Chen    FALSE
##   anticipatedGrade anticipatedHours
## 1                A               15
## 2                C               10
## 3                B               15
## 4             <NA>               NA
```

1.5) If you haven't already, convert the anticipatedGrade variable in bootcampDataFrame into an ordinal factor

```
maxGrade <- max(bootcampDataFrame$anticipatedGrade, na.rm=TRUE)

highestCourseName <- toString(bootcampDataFrame[bootcampDataFrame$anticipatedGrade == maxGrade,2])
highestCourseNum <- bootcampDataFrame[bootcampDataFrame$anticipatedGrade==maxGrade,1]

printf <- function(...) invisible(print(sprintf(...)))
printf("MSAN %d : %s", highestCourseNum, highestCourseName)
```

```
## [1] "MSAN 4 : Exploratory Data Analysis, NA"
## [2] "MSAN NA : Exploratory Data Analysis, NA"
```

Question 2

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

```
# assumes titanic datafile is in the same diretory
#setwd("/home/louiselai88gmail/Desktop/programming/USF/r")
titanicData <- read.csv("/home/louiselai88gmail/Desktop/programming/USF/r/titanic.csv", na='\\N')
```

2.2) How many rows are in this data frame?

```
nrow(titanicData)
```

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

2.3) How many columns are in this data frame?

```
ncol(titanicData)
```

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

2.4) Which variable has the most NA entries?

```
maxNA  = 0

for (i in c(1:ncol(titanicData))) {
  if(sum(is.na(titanicData[i])) > maxNA){
    maxNA = sum(is.na(titanicData[i]))
    print(colnames(titanicData[i]))
    print(sum(is.na(titanicData[i])))
  }
}
```

```
## [1] "Age"
## [1] 177
```

```
print(sum(is.na(titanicData$Age))) # there are 117 NAs for age
```

```
## [1] 177
```

2.5) Which variables, if any, should be converted to a different type than the default type they were imported as? Include of list of those you wish to change, what type they were previously, and what type you changed them to.

```
for (i in c(1:ncol(titanicData))) {
  print(colnames(titanicData[i]))
  print(class(titanicData[i][1,]))
}
```

```
## [1] "PassengerId"
## [1] "integer"
## [1] "Survived"
## [1] "integer"
## [1] "Pclass"
## [1] "integer"
## [1] "Name"
## [1] "factor"
## [1] "Sex"
## [1] "factor"
## [1] "Age"
## [1] "numeric"
## [1] "SibSp"
## [1] "integer"
## [1] "Parch"
## [1] "integer"
## [1] "Ticket"
## [1] "factor"
## [1] "Fare"
## [1] "numeric"
## [1] "Cabin"
## [1] "factor"
## [1] "Embarked"
## [1] "factor"
```

```r
titanicTypes <- sapply(titanicData, typeof)

# survived & sex should be logical (binary), instead of integers, becuase there are only two options
# Pclass should be a factor with levels, as there is a natural hiearchy for the cabin classes
```

2.6) If you haven't already, coerce the survived variable into type logical.

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

avgAgeSurvivor <- mean(titanicData$Age[titanicData$Survived==TRUE], na.rm=TRUE)
# mean age of survivors = 28.34

avgAgeNonsurvivor <- mean(titanicData$Age[titanicData$Survived==FALSE], na.rm=TRUE)
# mean age of survivors = 30.63

sum(titanicData$Survived==TRUE) # 342
```

```
## [1] 342
```

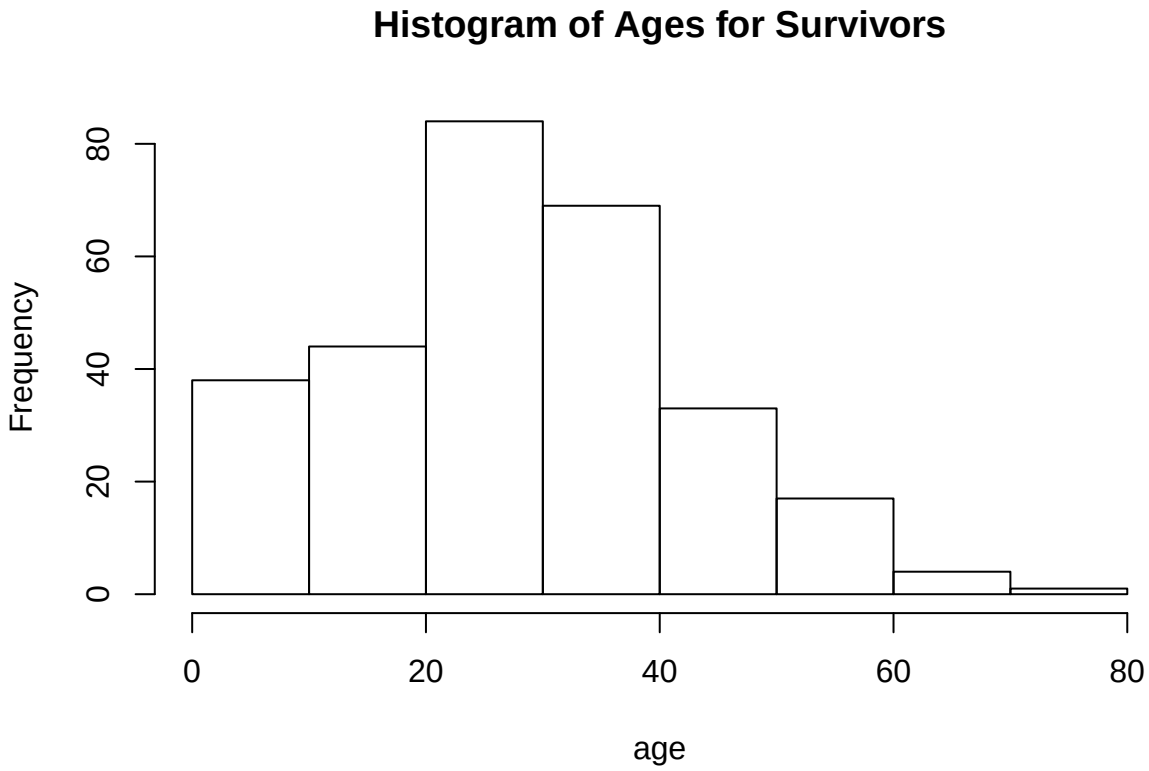
```r
sum(titanicData$Survived==FALSE) # 549
```

```
## [1] 549
```

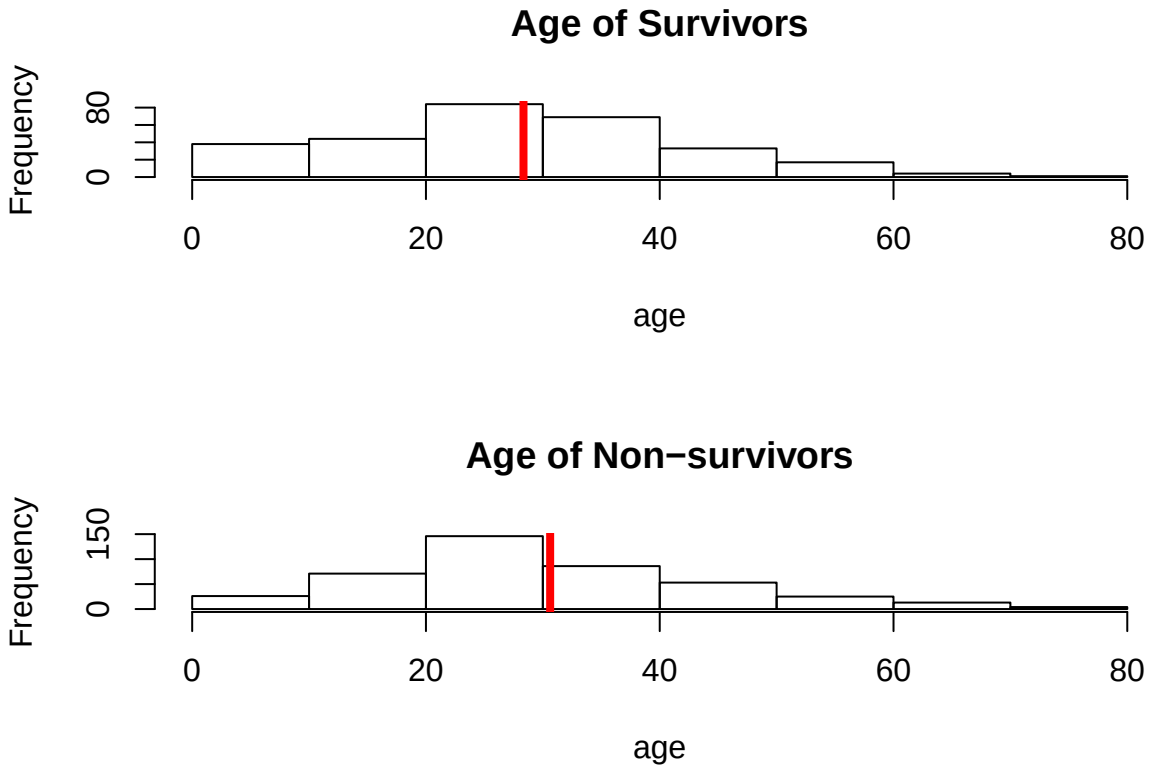
```r
sum(is.na(titanicData$Survived)) # 0
```

```
## [1] 0
```

```r
# plotting histogram
hist(titanicData$Age[titanicData$Survived == TRUE],
     main="Histogram of Ages for Survivors", xlab="age")
```



```r
attach(titanicData)
par(mfrow=c(2,1)) # 2 rows 1 col
hist(titanicData$Age[titanicData$Survived == TRUE], main="Age of Survivors", xlab="age")
abline(v=avgAgeSurvivor, lwd=4, col="red")

hist(titanicData$Age[titanicData$Survived == FALSE], main ="Age of Non-survivors", xlab="age")
abline(v=avgAgeNonsurvivor, lwd=4, col="red")
```



2.7) Include the first 10 value of the cabin variable in this deliverable, observing that many are blank. Write and run a script that replaces all blanks in the entire data frame titanicData with NAs.

```r
titanicData$Cabin[1:10]
```

```
##  [1]       C85       C123           E46
## 148 Levels:  A10 A14 A16 A19 A20 A23 A24 A26 A31 A32 A34 A36 A5 A6 ... T
```

```r
# could just load the data again, subbing spare "" with NA
# data <- read.csv("data.csv", na.strings="")
# but we won't do that becuase we don't want to reload file
replace(titanicData$Cabin, titanicData$Cabin == "", NA) # adds 687 NAs to Cabin
replace(titanicData$Embarked, titanicData$Embarked == "", NA)
```

2.8) What percent of the observations for age are NAs? Replace all NAs with the mean age?

```r
percentAgeNA <- 100 * (sum(is.na(titanicData$Age)) / nrow(titanicData))
percentAgeNA # 19.865%
```

```
## [1] 19.86532
```

```r
# replace NAs with average age
titanicData$Age[is.na(titanicData$Age)] <- mean(titanicData$Age, na.rm = TRUE)
sum(is.na(titanicData$Age)) # 0, shows that no NAs left
```

```
## [1] 0
```

Question 3

3.1) Generate random variables for different sample sizes

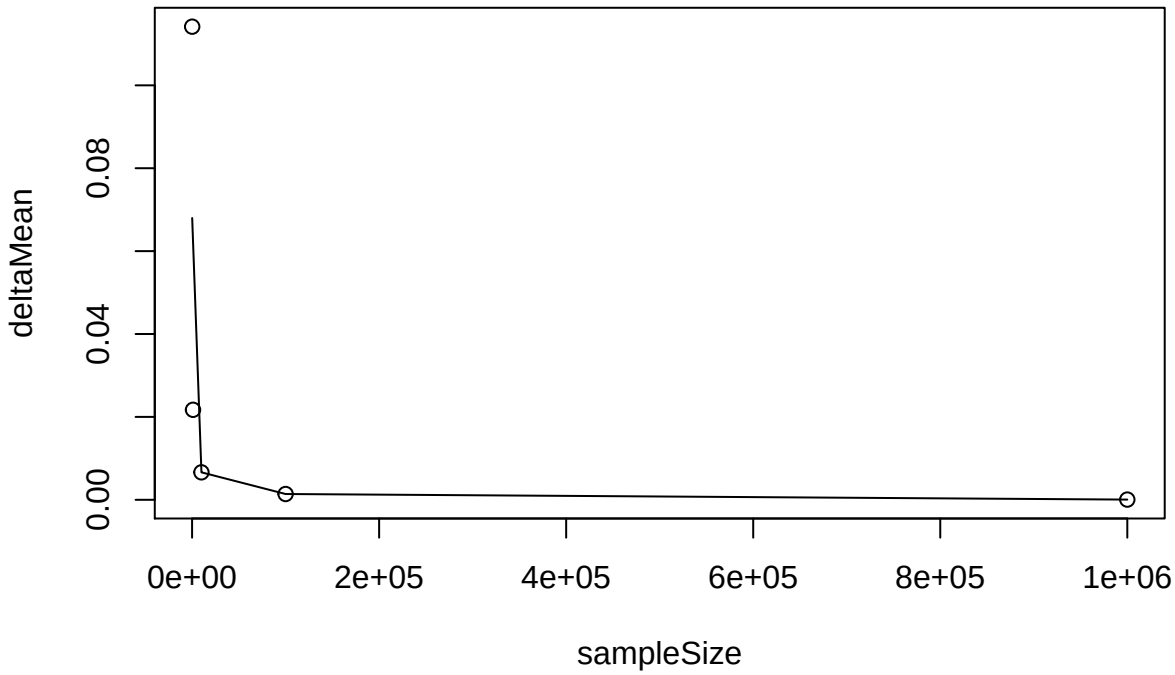
```
distA <- runif(100, -1, 1)
distA <- runif(100, -1, 1)
distB <- runif(1000, -1, 1)
distC <- runif(10000, -1, 1)
distD <- runif(100000, -1, 1)
distE <- runif(1000000, -1, 1)

distribution <- list(distA, distB, distC, distD, distE)

sampleSize <- c(100, 1000, 10000, 100000, 1000000)

# compute mean and variance
theoreticalMean <-c(0, 0, 0, 0, 0)
sampleMean <- sapply(distribution, function(x) mean(x))
deltaMean <- abs(theoreticalMean - sampleMean)

# create unifDataFrame
theoreticalVariance <- c(1/3, 1/3, 1/3, 1/3, 1/3)
sampleVariance <- sapply(distribution, function(x) var(x))
deltaVariance <- abs(theoreticalVariance - sampleVariance)

unifDataFrame <- data.frame(sampleSize, theoreticalMean, sampleMean, deltaMean, theoreticalVariance, sa
```

Create a plot with sampleSize on the x-axis and deltaMean on the y-axis

```
# plot sampleSize v deltaMean
attach(unifDataFrame)
```

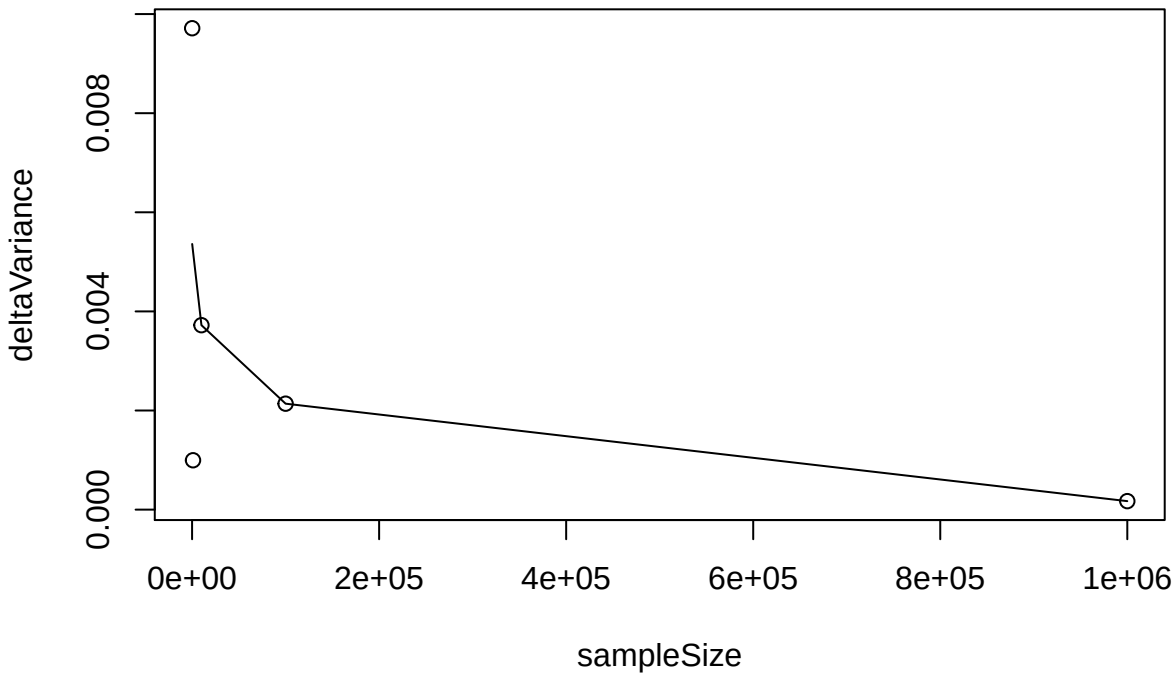
```
## The following objects are masked _by_ .GlobalEnv:
##
##     deltaMean, deltaVariance, sampleMean, sampleSize,
##     sampleVariance, theoreticalMean, theoreticalVariance
```

```
plot(sampleSize, deltaMean)
lines(lowess(deltaMean ~ sampleSize), col="black")
```



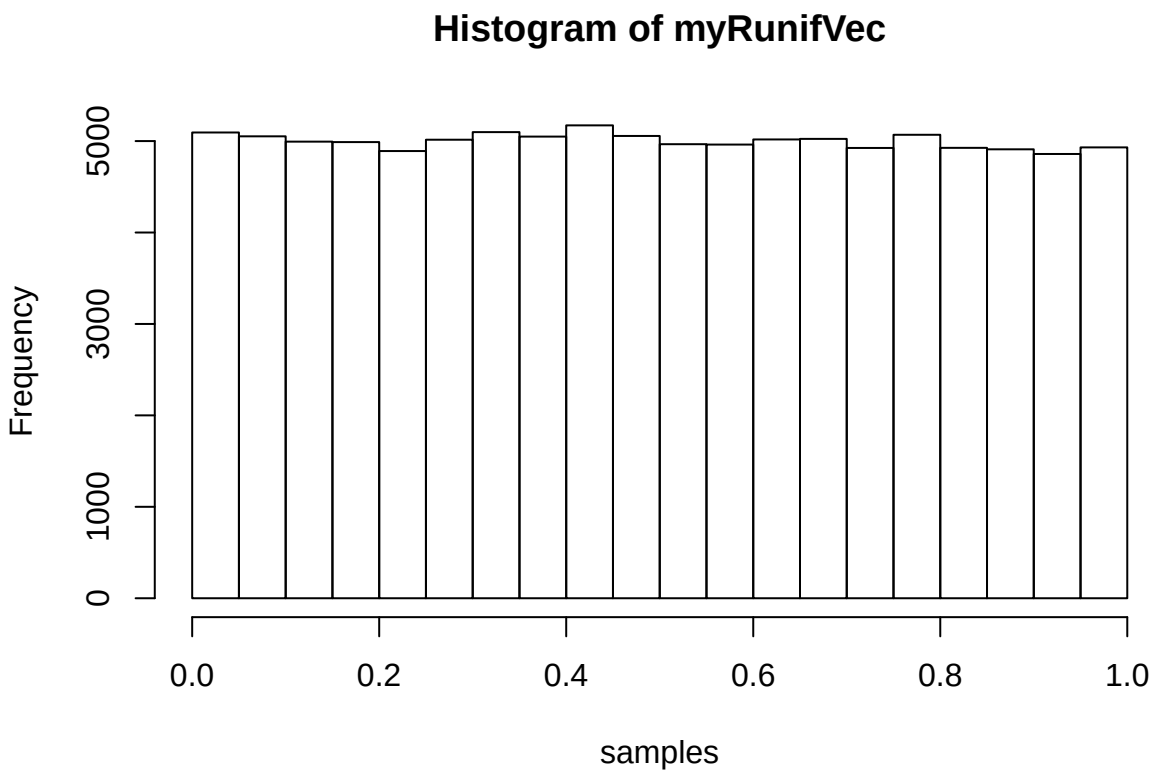
Create a plot with sampleSize on the x-axis and deltaVariance on the y-axis.

```
# plot sampleSize v deltaVariance
plot(sampleSize, deltaVariance)
lines(lowess(deltaVariance ~ sampleSize), col="black")
```



3.2) Create a vector of 10,000,000 random variables $\sim \mathcal{U}0, 1$ and store them in the vector called myRunifVec.

```
myRunifVec <- runif(10000000, 0, 1)
samples <- sample(myRunifVec, 100000)
hist(samples, main="Histogram of myRunifVec")
```



3.3) and 3.4) omitted

Question 4

4.1) Manually compute the coefficients for the simple linear regression

```
# calculate b1 (slope)
get_b1 <- function(x_list, y_list){
  x_hat <- mean(x_list)
  y_hat <- mean(y_list)
  numerator = 0.0
  denominator = 0.0
  for (i in 1:length(x_list)){
    numerator = numerator + ( (x_list[i] - x_hat) * (y_list[i] - y_hat) ) # 8064.517
    denominator = denominator + ( (x_list[i]-x_hat)**2 ) # 333564449
  }
  b1 = numerator/denominator # 2.417679e-05
  return(b1)
}

# calculate b0 (intercept)
get_b0 <- function(x_list, y_list, b1_given){
  b0 = (sum(y_list) - b1_given*sum(x_list))/length(x_list) # 1.995036
  return(b0)
}
```

4.2) Manually compute SSE, SSR, SSTO and compute the simple coefficient of determination.

```
# generate predicted Y values from x values, using regression formula (i.e. coefficients found above)
get_predicted_y_values <- function(x_list, b0, b1){ # might break due to variable overlap b0 b1
  predictedY = c()
  for (i in 1:length(x_list)){
    predictedY[i] = b0 + b1*x_list[i]
  }
  return(predictedY)
}

# SSE
get_SSE <- function(x_list, y_list, predictedY){
  SSE = 0.0
  for (i in 1:length(x_list)){
    SSE = SSE + (y_list[i]- predictedY[i])**2 # y_1 = y actual
  }
  SSE # 396,806.1
}

# SSTo
get_SSTo <- function(x_list, y_list){
  SSTo = 0.0
  y_hat = mean(y_list)
  for (i in 1:length(x_list)){
    SSTo = SSTo + (y_list[i]-y_hat)**2
  }
  return(SSTo) # 396,806.3
}

# SSR
```

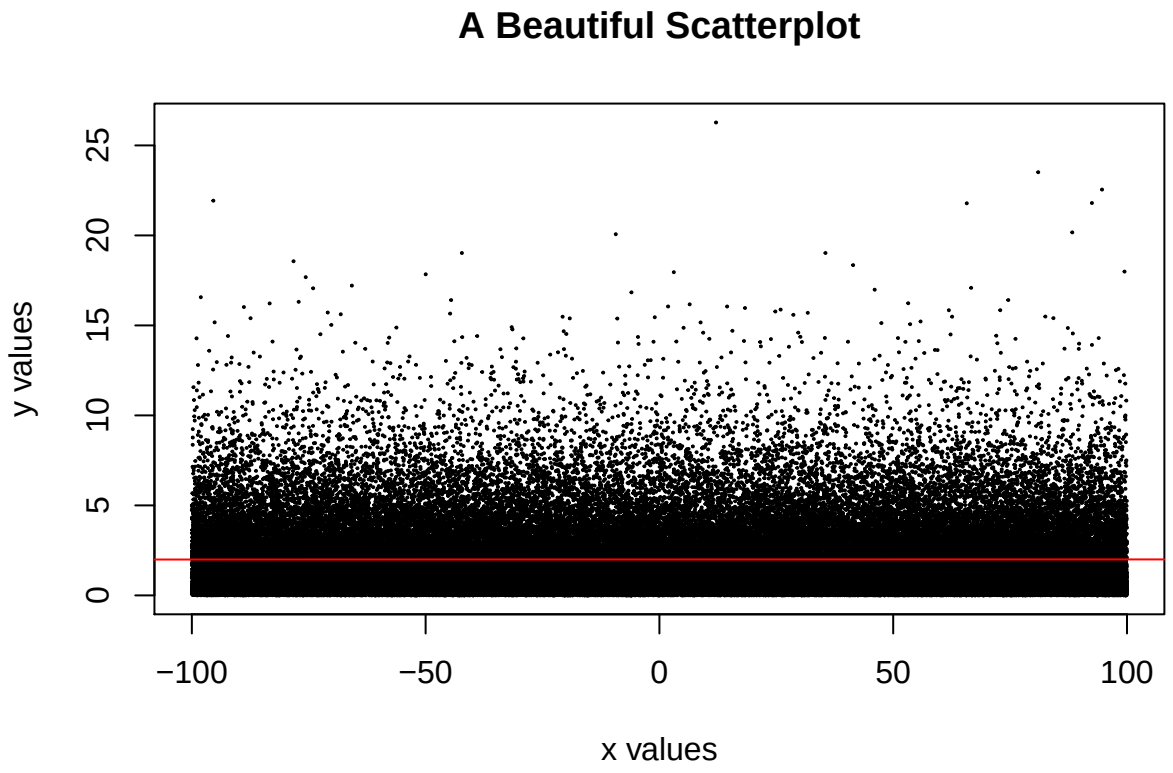
```r
get_SSR <- function(SSTo, SSE){
  return(SSTo-SSE) # 0.19497
}

# rSquared
get_rSquared <- function(SSR, SSTo){
  rSquared = SSR/SSTo # 4.913584e-07
  return(rSquared)
}
```

4.4) Manually compute the residuals for the fitted values.

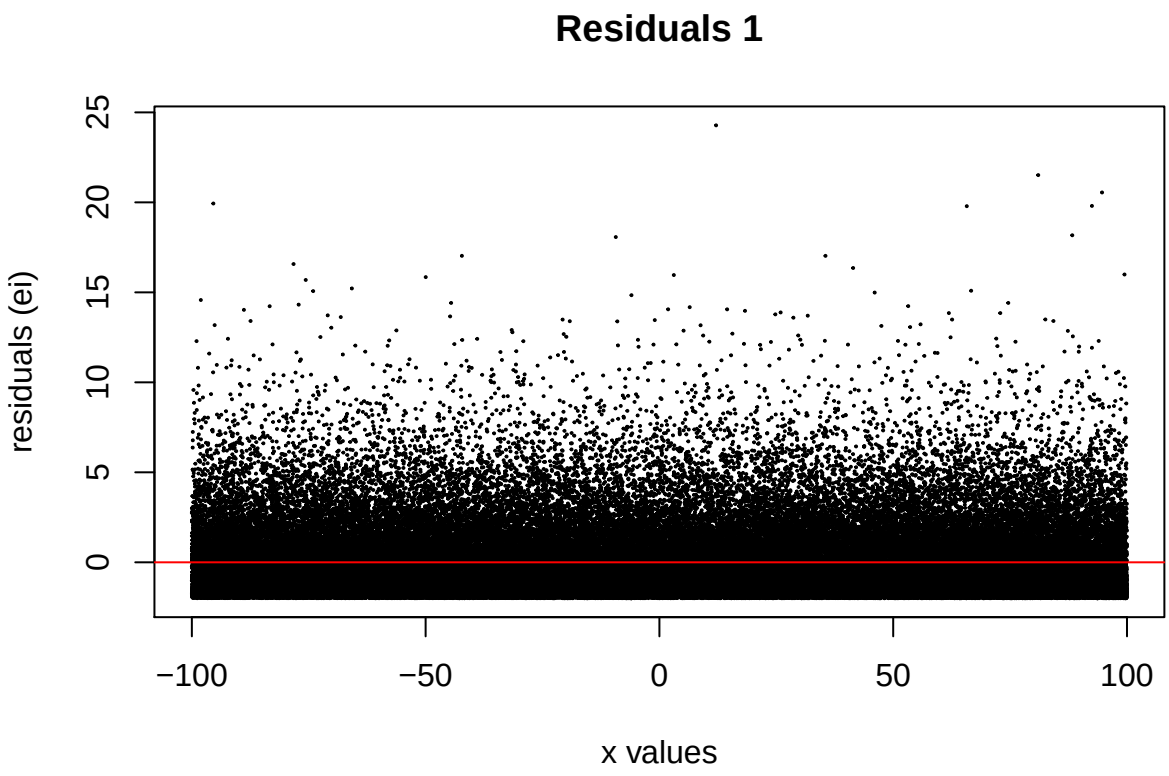
```r
# residuals
get_residuals <- function(x_list, y_list, predictedY){
  residuals = c()
  for(i in 1:length(x_list)){
    residuals[i] = y_list[i] - predictedY[i]
  }
  return(residuals)
}
```

Model 1

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

b1_1 <- get_b1(x_1, y_1) # -3.205058e-05
b0_1 <- get_b0(x_1, y_1, b1_1) # 1.999032
predictedY_1 <- get_predicted_y_values(x_1, b0_1, b1_1)
SSE_1 <- get_SSE(x_1, y_1, predictedY_1) # 398,628.3
SSTo_1 <- get_SSTo(x_1, y_1) # 398,629.4
SSR_1 <- get_SSR(SSTo_1, SSE_1) # 1.0323
rSquared_1 <- get_rSquared(SSR_1, SSTo_1) # 2.589701e-06
residuals_1 <- get_residuals(x_1, y_1, predictedY_1) # list

# scatterplot
plot(x_1, y_1, main="A Beautiful Scatterplot", xlab="x values", ylab="y values", pch=20, cex=0.2)
abline(b0_1, b1_1, col="red")
```

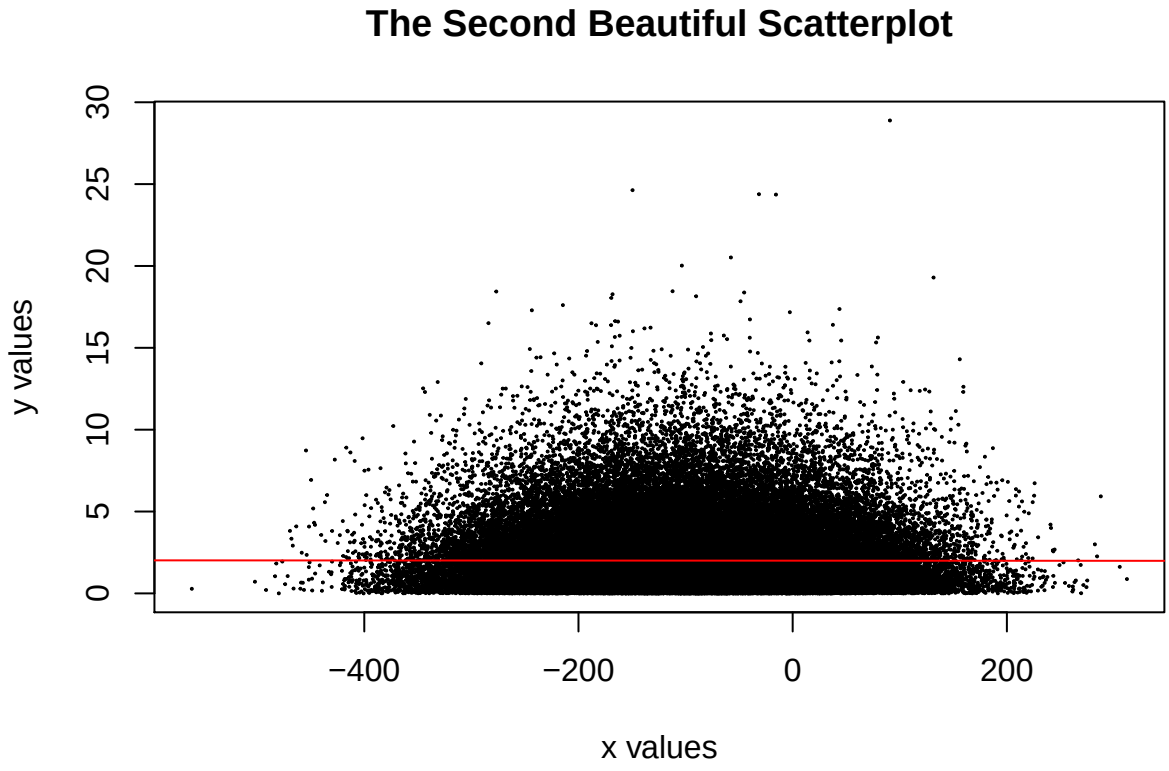



```
# residuals
plot(x_1, residuals_1, main="Residuals 1", xlab="x values", ylab="residuals (ei)", pch=20, cex=0.2)
abline(0,0, col="red")
```

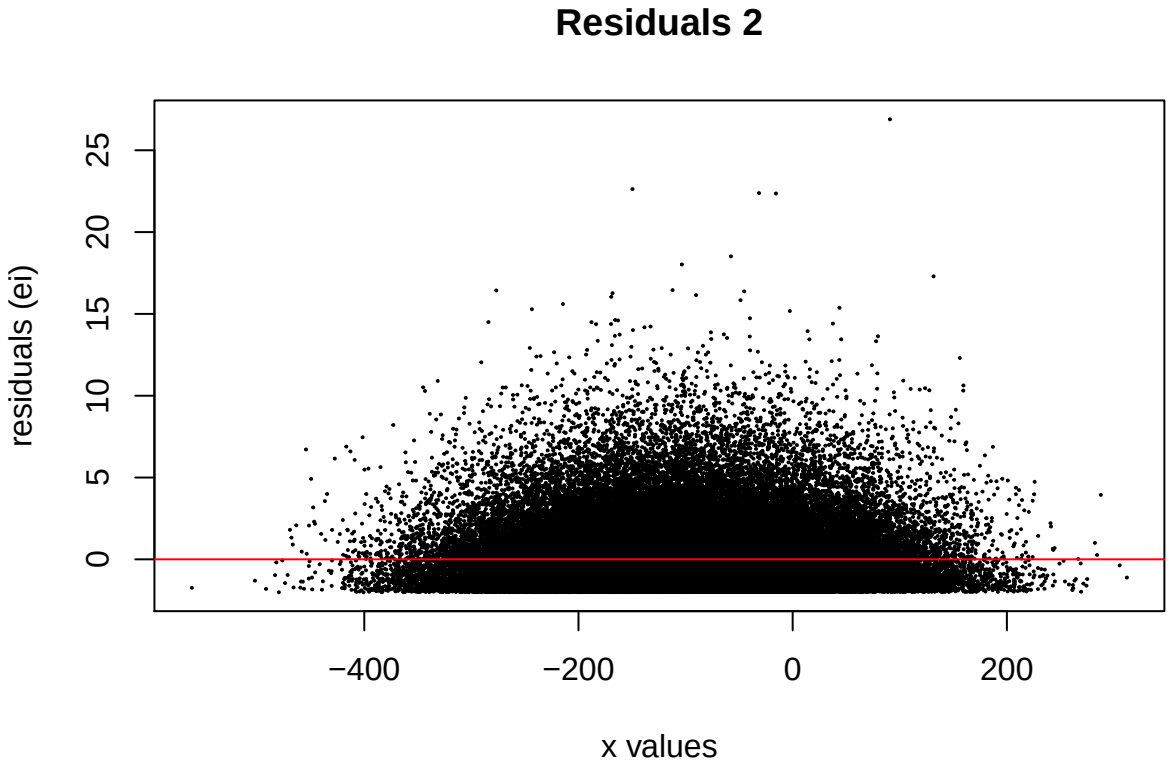

Residuals 1
residuals (ei)
0
5
10
15
20
25
−100
−50
0
50
100
x values

Model 2

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

b1_2 <- get_b1(x_2, y_2)
b0_2 <- get_b0(x_2, y_2, b1_2)
predictedY_2 <- get_predicted_y_values(x_2, b0_2, b1_2)
SSE_2 <- get_SSE(x_2, y_2, predictedY_2)
SSTo_2 <- get_SSTo(x_2, y_2)
SSR_2 <- get_SSR(SSTo_2, SSE_2)
rSquared_2 <- get_rSquared(SSR_2, SSTo_2)
residuals_2 <- get_residuals(x_2, y_2, predictedY_2) # list

# scatterplot 2
plot(x_2, y_2, main="The Second Beautiful Scatterplot", xlab="x values", ylab="y values", pch=20, cex=0
abline(b0_2, b1_2, col="red")
```

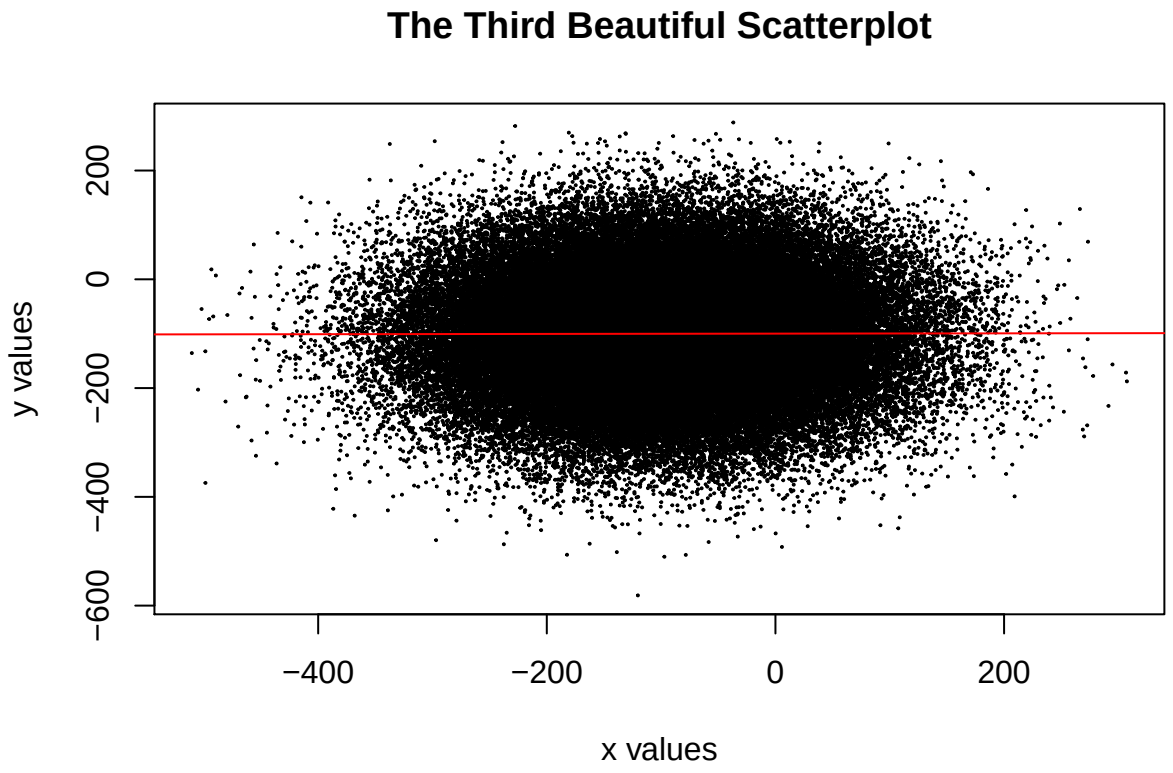



```
# residuals 2
plot(x_2, residuals_2, main="Residuals 2", xlab="x values", ylab="residuals (ei)", pch=20, cex=0.2)
abline(0,0, col="red")
```

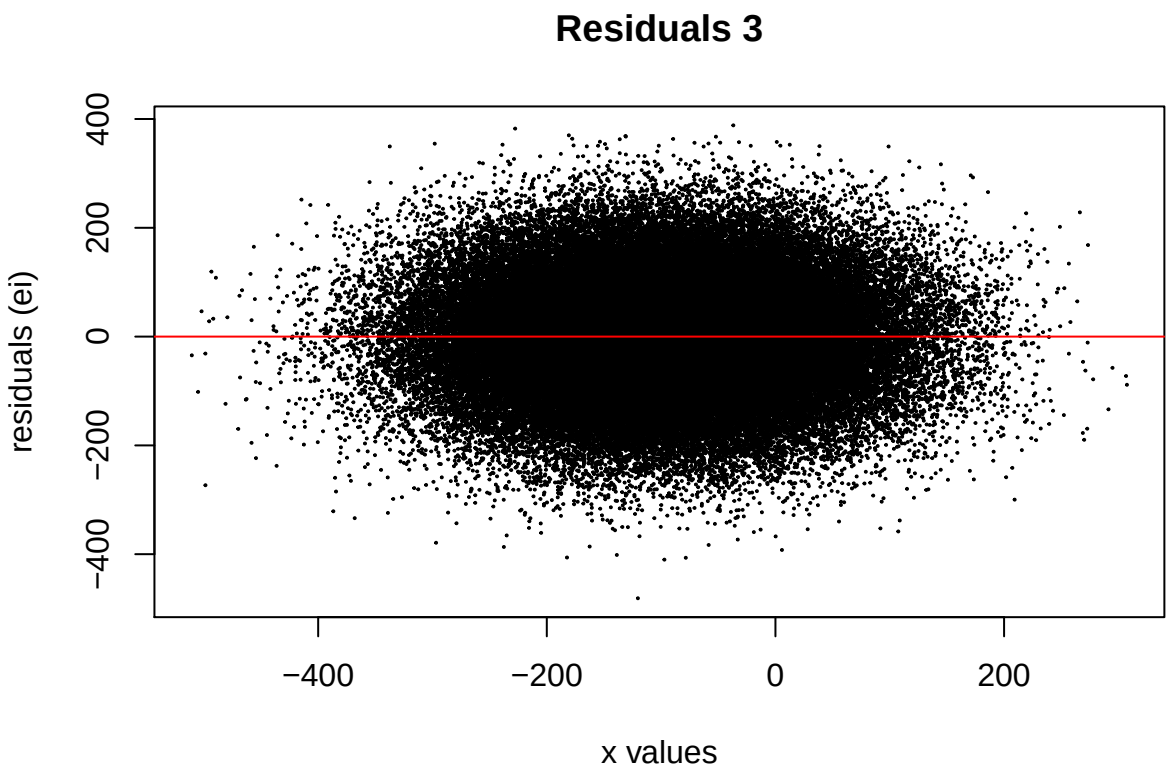

Residuals 2
residuals (ei)
0
5
10
15
20
25
−400
−200
0
200
x values

Model 3

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

b1_3 <- get_b1(x_3, y_3)
b0_3 <- get_b0(x_3, y_3, b1_3)
predictedY_3 <- get_predicted_y_values(x_3, b0_3, b1_3)
SSE_3 <- get_SSE(x_3, y_3, predictedY_3)
SSTo_3 <- get_SSTo(x_3, y_3)
SSR_3 <- get_SSR(SSTo_3, SSE_3)
rSquared_3 <- get_rSquared(SSR_3, SSTo_3)
residuals_3 <- get_residuals(x_3, y_3, predictedY_3) # list

# scatterplot 3
plot(x_3, y_3, main="The Third Beautiful Scatterplot", xlab="x values", ylab="y values", pch=20, cex=0.1
abline(b0_3, b1_3, col="red")
```




```
# residuals 3
plot(x_3, residuals_3, main="Residuals 3", xlab="x values", ylab="residuals (ei)", pch=20, cex=0.2)
abline(0,0, col="red")
```


Residuals 3
residuals (ei)
400
200
0
−200
−400
−400
−200
0
200
x values

Create final table

```
summaryTable <- data.frame(
  model <- c("Model 1", "Model 2", "Model 3"),
  b0 <- c(b0_1, b0_2, b0_3),
  b1 <- 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(SSTo_1, SSTo_2, SSTo_3),
  rSquared <- c(rSquared_1, rSquared_2, rSquared_3),
  stringsAsFactors = FALSE
)

colnames(summaryTable) <- c("model", "b0", "b1", "SSE", "SSR", "SSTo", "RSquared")

summaryTable
```

```
##     model         b0            b1         SSE          SSR        SSTo
## 1 Model 1   1.995036  2.417679e-05    396806.1    0.1949741    396806.3
## 2 Model 2   1.999032 -3.205058e-05    398628.3    1.0323308    398629.4
## 3 Model 3 -99.869215  2.598695e-03 999572341.9 6728.3929440 999579070.3
##       RSquared
## 1 4.913584e-07
## 2 2.589701e-06
## 3 6.731226e-06
```