MSAN 593: Assignment #1 SOLUTIONS

Paul Intrevado

July 19, 2018

Question 1

1.
```r
# Creating the bootcamp vectors
courseNum <- c(501, 502, 504, 593)
courseName <- c("Computation for Analytics", "Review of Linear Algebra",
    "Review of Probability and Statistics", "Exploratory Data Analysis")
courseProf <- c("Terence Parr", "David Uminsky", "Jeff Hamrick",
    "Paul Intrevado")
enrolled <- c(T, F, T, T)
anticipatedGrade <- c("A", NA, "B+", "A-")
anticipatedHours <- c(14, NA, 18, 16)
```

```r
# Creating vector of names of each bootcamp vector
bootcampNames <- c("courseNum", "courseName", "courseProf", "enrolled",
    "anticipatedGrade", "anticipatedHours")

# Creating vector of types of each bootcamp vector
Type <- c(typeof(courseNum), typeof(courseName), typeof(courseProf),
    typeof(enrolled), typeof(anticipatedGrade), typeof(anticipatedHours))

# Creating vector of classes of each bootcamp vector
Class <- c(class(courseNum), class(courseName), class(courseProf),
    class(enrolled), class(anticipatedGrade), class(anticipatedHours))

# Creating table of bootcamp vector types and classes
vector_info <- cbind(bootcampNames, Type, Class)
colnames(vector_info) <- c("**Variable Name**", "**Type**", "**Class**")
set.caption("Types and Classes of Bootcamp Vectors")
pandoc.table(vector_info)
```

Table 1: Types and Classes of Bootcamp Vectors

Variable Name	**Type**	**Class**
courseNum	double	numeric
courseName	character	character
courseProf	character	character
enrolled	logical	logical
anticipatedGrade	character	character
anticipatedHours	double	numeric

2.
```r
# Creating data frame of bootcamp vectors
bootcampDataFrame <- data.frame(courseNum, courseName, courseProf,
    enrolled, anticipatedGrade, anticipatedHours)

# Creating table of bootcampDataFrame types and classes
dataframe_info <- cbind(names(bootcampDataFrame), vapply(bootcampDataFrame,
    typeof, character(1)), vapply(bootcampDataFrame, class, character(1)))
colnames(dataframe_info) <- c("**Variable Name**", "**Type**",
    "**Class**")
row.names(dataframe_info) <- NULL
emphasize.strong.rows(c(2, 3))
set.caption("Types and Classes of `bootcampDataFrame` variables")
pandoc.table(dataframe_info)
```

Table 2: Types and Classes of `bootcampDataFrame` variables

Variable Name	**Type**	**Class**
courseNum	double	numeric
courseName	**integer**	**factor**
courseProf	**integer**	**factor**
enrolled	logical	logical
anticipatedGrade	integer	factor
anticipatedHours	double	numeric

After creating a dataframe of the boot-camp vectors from `Table 1`, we can see in `Table 2` that the `courseName` and `courseProf` vectors were each coerced from `character` vectors to `integer` factors. The rest of the boot-camp vectors maintained their original types and classes.

3.
```r
# Creating list of bootcamp vectors
bootcampDataList <- list(courseNum, courseName, courseProf, enrolled,
    anticipatedGrade, anticipatedHours)

# Assigning names to elements of bootcampDataList
names(bootcampDataList) <- bootcampNames

# Creating table of bootcampDataList types and classes
list_info <- cbind(names(bootcampDataList), vapply(bootcampDataList,
    typeof, character(1)), vapply(bootcampDataList, class, character(1)))
colnames(list_info) <- c("**Variable Name**", "**Type**", "**Class**")
row.names(list_info) <- NULL
set.caption("Types and Classes of `bootcampDataList` variables")
pandoc.table(list_info)
```

Table 3: Types and Classes of `bootcampDataList` variables

Variable Name	**Type**	**Class**
courseNum	double	numeric
courseName	character	character
courseProf	character	character
enrolled	logical	logical
anticipatedGrade	character	character
anticipatedHours	double	numeric

Creating a list of the boot-camp vectors in `Table 1` does not change the type or class of the vectors, as shown in `Table 3`.

4. Code and output for the following calculations:

   - The total number of hours you anticipate spending on coursework:
     - per week:

       ```
       sum(anticipatedHours, na.rm = TRUE)
       ```

       ```
       ## [1] 48
       ```

     - over all of bootcamp:

       ```
       5 * sum(anticipatedHours, na.rm = TRUE)
       ```

       ```
       ## [1] 240
       ```

   - A data frame with only the third row and first two columns of bootcampDataFrame:

     ```
     bootcampDataFrame[3, c(1, 2)]
     ```

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

   - The first value in the second element of bootcampDataList:

     ```
     bootcampDataList[[2]][1]
     ```

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

5.
   ```
   # Converting anticipatedGrade variable of bootcampDataFrame
   # to an ordinal factor
   bootcampDataFrame$anticipatedGrade <- factor(bootcampDataFrame$anticipatedGrade,
       levels = c("B-", "B", "B+", "A-", "A", "A+"), ordered = TRUE)
   ```

   - Maximum letter grade:

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

     ```
     ## [1] A
     ## Levels: B- < B < B+ < A- < A < A+
     ```

   - Course Name and Course Number of class with highest anticipated grade:

     ```
     # Course number of class with highest anticipated grade
     num <- bootcampDataFrame$courseNum[which.max(bootcampDataFrame$anticipatedGrade)]

     # Course name of class with highest anticipated grade
     name <- bootcampDataFrame$courseName[which.max(bootcampDataFrame$anticipatedGrade)]

     # Printing textual output of information for class with
     # highest anticipated grade
     paste("MSAN ", num, ": ", name, sep = "")
     ```

     ```
     ## [1] "MSAN 501: Computation for Analytics"
     ```

   The maximum letter grade I anticipate recieving in bootcamp is the letter grade A for MSAN 501: Computation for Analytics.

Question 2

1. ```r
   # Reading in the titanic.csv file
   titanicData <- read.csv("titanic.csv", stringsAsFactors = FALSE)
   ```

2. There are 891 rows in the `titanicData` data frame.

3. There are 12 columns in the `titanicData` data frame.

4. ```r
   # Finding variable with most NAs
   names(which.max(apply(is.na(titanicData), 2, sum)))
   ```

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

   The `Age` variable in the `titanicData` data frame has the most `NA`s. It has 177 `NA`s.

5. ```r
   # Creating table of types and classes of the titanicData
   # variables
   titanic_info <- cbind(names(titanicData), vapply(titanicData,
       typeof, character(1)), vapply(titanicData, class, character(1)))
   colnames(titanic_info) <- c("**Variable Name**", "**Type**",
       "**Class**")
   row.names(titanic_info) <- NULL
   set.caption("Types and Classes of `titanicData` variables")
   pandoc.table(titanic_info)
   ```

   Table 4: Types and Classes of `titanicData` variables

Variable Name	**Type**	**Class**
PassengerId	integer	integer
Survived	integer	integer
Pclass	integer	integer
Name	character	character
Sex	character	character
Age	double	numeric
SibSp	integer	integer
Parch	integer	integer
Ticket	character	character
Fare	double	numeric
Cabin	character	character
Embarked	character	character

   In `Table 4` we can see that some of the variables have types that we would like to convert. In particular:

   - `Survived` has default type `integer`, but the only values it takes are `0` and `1`. We can convert this variable's type to `logical`.
   - `Pclass` has default type `integer`, but it only takes values `1`, `2`, or `3`. We can convert this variable to an ordinal factor with levels `1`, `2`, and `3` (since there is a natural ordering of the passenger class).
   - `Sex` has default type `character`, but it only takes values `male` or `female`. We can convert this to a nominal factor with levels `male` and `female` (since there is no natural ordering of passenger's sex).
   - `Embarked` has default type `character`, but it only takes values `C`, `Q`, `S`, and `""` (missing value). We can convert this to a nominal factor with levels `C`, `S`, and `Q` (since there is no natural ordering of the port of embarkation). Note: the act of converting this variable to a factor with the specified
   ```

levels will automatically convert the missing values, `""`, to `NA`.

```r
# Creating original copy of titanicData
titanicData_original <- titanicData

# Changing types of survived, Pclass, Sex, and Embarked
# variables
titanicData$Survived <- as.logical(titanicData$Survived)
titanicData$Pclass <- factor(titanicData$Pclass, levels = c(1,
    2, 3), ordered = TRUE)
titanicData$Sex <- factor(titanicData$Sex, levels = c("male",
    "female"))
titanicData$Embarked <- factor(titanicData$Embarked, levels = c("C",
    "Q", "S"))

# Creating table showing new and orginal types and classes of
# titanicData variables
titanic_new_info <- cbind(names(titanicData), vapply(titanicData_original,
    typeof, character(1)), vapply(titanicData, typeof, character(1)),
    vapply(titanicData_original, class, character(1)), vapply(titanicData,
        function(x) {
            paste(class(x), collapse = " ")
        }, character(1)))
colnames(titanic_new_info) <- c("**Variable Name**", "**Original Type**",
    "**New Type**", "**Original Class**", "**New Class**")
row.names(titanic_new_info) <- NULL
emphasize.strong.rows(c(2, 3, 6, 12))
set.caption("New and Original Types and Classes of `titanicData` variables")
pandoc.table(titanic_new_info, split.table = Inf)
```

Table 5: New and Original Types and Classes of `titanicData` variables

Variable Name	Original Type	New Type	Original Class	New Class
PassengerId	integer	integer	integer	integer
Survived	**integer**	**logical**	**integer**	**logical**
Pclass	**integer**	**integer**	**integer**	**ordered factor**
Name	character	character	character	character
Sex	character	integer	character	factor
Age	**double**	**double**	**numeric**	**numeric**
SibSp	integer	integer	integer	integer
Parch	integer	integer	integer	integer
Ticket	character	character	character	character
Fare	double	double	numeric	numeric
Cabin	character	character	character	character
Embarked	**character**	**integer**	**character**	**factor**

The new and original types and classes of variables are shown in `Table 5`. The highlighted rows are the variables whose types and classes changed.

6.
```r
# Mean age of survivors
mean(titanicData$Age[titanicData$Survived], na.rm = TRUE)
```

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

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

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

```r
# Computing minimum and maximum ages for setting up axes of
# histograms
minAge <- min(titanicData$Age, na.rm = TRUE)
maxAge <- max(titanicData$Age, na.rm = TRUE)

# Plotting side-by-side histograms of ages
par(mfrow = c(1, 2))

hist(titanicData$Age[titanicData$Survived], right = FALSE, freq = TRUE,
    breaks = minAge:(maxAge + 1), main = "Ages of Titanic Survivors",
    xlab = "Age", xlim = c(minAge, maxAge + 1), ylab = "Number of Survivors",
    ylim = c(0, 20))

hist(titanicData$Age[!titanicData$Survived], right = FALSE, freq = TRUE,
    breaks = minAge:(maxAge + 1), main = "Ages of Titanic Casualties",
    xlab = "Age", xlim = c(minAge, maxAge + 1), ylab = "Number of Casualties",
    ylim = c(0, 20))
```

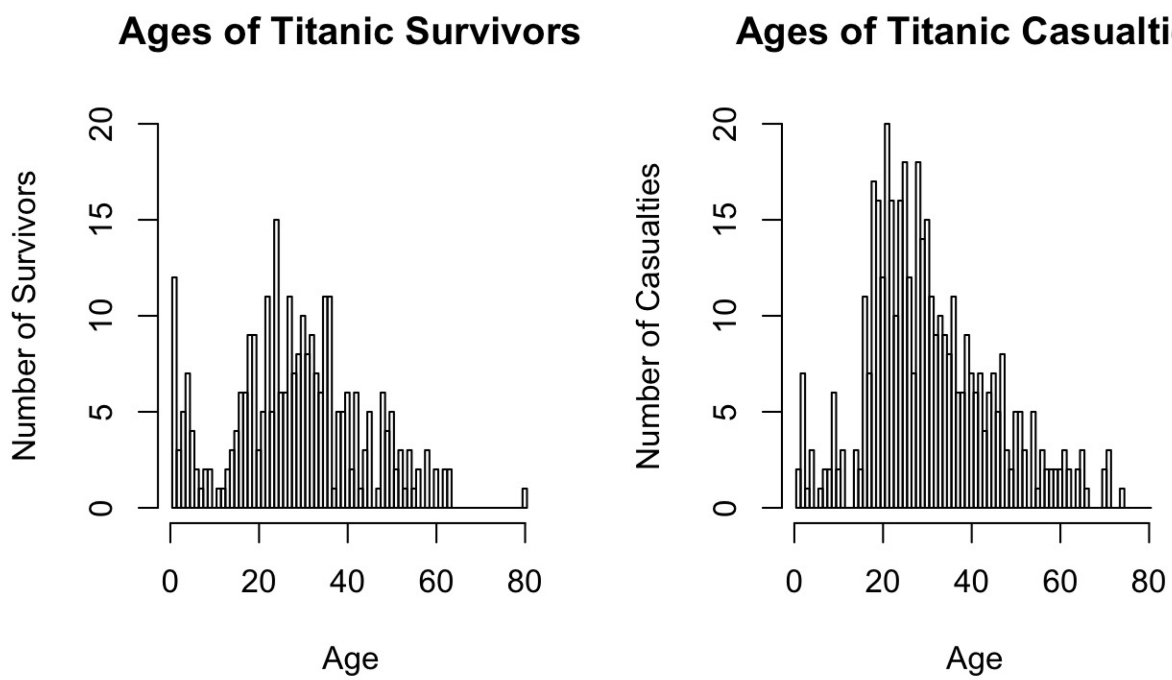

Figure 1: Ages of survivors and casualties from titanicData

7.
```r
titanicData$Cabin[1:10]
```

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

```r
# Replacing blanks with NA
titanicData[titanicData == ""] <- NA

titanicData$Cabin[1:10]
```

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

8.
```r
round(sum(is.na(titanicData$Age))/nrow(titanicData) * 100, digits = 2)
```

```
## [1] 19.87
```

Out of all observations for `Age`, 19.87% of them are `NA`s.

```r
# Calculating the mean and standard deviation of Age
meanAge <- mean(titanicData$Age, na.rm = TRUE)
sdAge <- sd(titanicData$Age, na.rm = TRUE)

# Replacing NAs with mean of Age
titanicData$Age[is.na(titanicData$Age)] <- meanAge

# Calculating new standard deviation
sdAgenew <- sd(titanicData$Age)

age_info <- cbind(sdAge, sdAgenew)
colnames(age_info) <- c("**Std. dev. before imputation**", "**Std. dev. after imputation**")
set.caption("Comparing Std. Dev. Before and After Imputation")
pandoc.table(age_info, justify = "center", digits = 5)
```

Table 6: Comparing Std. Dev. Before and After Imputation

Std. dev. before imputation	**Std. dev. after imputation**
14.526	13.002

This method of imputation is called “mean imputation”. The downside of using this particular method of imputation is that it affects the standard deviation of the `Age` data (as shown in `Table 6`) since data values that were once blank, and thus ignored in the calculation of variance, now enter the calculation as the mean. As a result, the variance decreases because of the increase in the number observations close to the mean.

Question 3

1.
```r
# Only use scientific notation on numbers with more than 10
# digits
options(scipen = 10)

# Vector of sample sizes
sampleSize <- c(100, 1000, 10000, 1e+05, 1e+06)
sampleMean <- NULL
sampleVariance <- NULL

# Looping through sample sizes and appending means and
# variances
for (n in sampleSize) {
    unifrvs <- runif(n, min = -1, max = 1)
    sampleMean <- c(sampleMean, mean(unifrvs))
    sampleVariance <- c(sampleVariance, var(unifrvs))
}

# Creating data frame of the results
unifDataFrame <- data.frame(sampleSize, sampleMean, sampleVariance)

# Adding theoretical means and variances and their difference
# from computed means and variances
unifDataFrame$theoreticalMean <- rep((-1 + 1)/2, nrow(unifDataFrame))
unifDataFrame$theoreticalVariance <- rep((1 - (-1))^2/12, nrow(unifDataFrame))
unifDataFrame$deltaMean <- abs(unifDataFrame$sampleMean - unifDataFrame$theoreticalMean)
unifDataFrame$deltaVariance <- abs(unifDataFrame$sampleVariance -
    unifDataFrame$theoreticalVariance)

# Reordering the columns of unifDataFrame
unifDataFrame <- unifDataFrame[c("sampleSize", "theoreticalMean",
    "sampleMean", "deltaMean", "theoreticalVariance", "sampleVariance",
    "deltaVariance")]
```

```r
# Plotting sampleSize vs. deltaMean
par(mfrow = c(1, 2), cex.main = 0.75, cex.lab = 0.75, cex.axis = 0.75)
plot(sampleSize, unifDataFrame$deltaMean, main = "sampleSize vs. deltaMean",
    xlab = "sampleSize", ylab = "deltaMean", log = "x")

# Plotting sampleSize vs. deltaVariance
plot(sampleSize, unifDataFrame$deltaVariance, main = "sampleSize vs. deltaVariance",
    xlab = "sampleSize", ylab = "deltaVariance", log = "x")
```

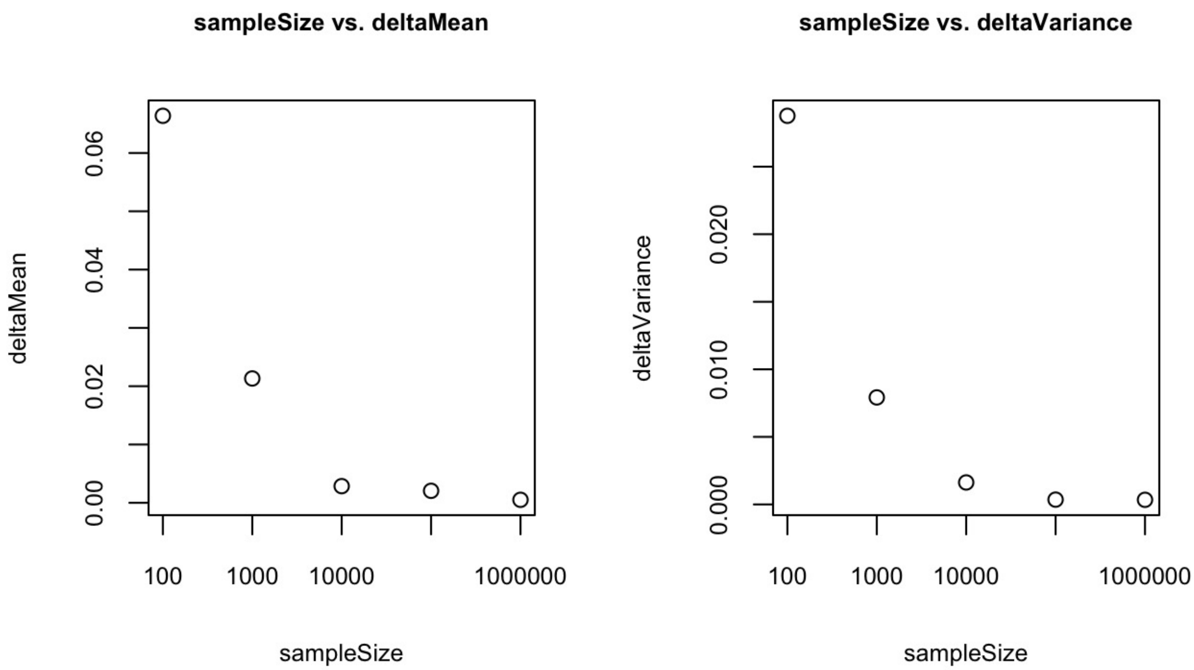

Figure 2: Difference in sample and theoretical measurements

In `Figure 2`, we took samples from $Unif(-1, 1)$ and plotted the difference in the sample and theoretical mean as well as the difference in the sample and theoretical variance. We can see that these differences converge to 0 as we increase the sample size.

2.
```r
par(mfrow = c(2, 2), cex.main = 1, cex.axis = 0.5, cex.lab = 0.75)

# Plotting sample from Unif(0,1)
myRunifVec <- runif(10000000, min = 0, max = 1)
hist(sample(myRunifVec, size = 100000), main = "Sampling 100,000 values from U(0,1)",
    xlab = "Values", ylab = "Density", xlim = c(0, 1), ylim = c(0,
        8000))

# Plotting sample from Unif(4,7)
myRunifVec <- runif(10000000, min = 4, max = 7)
hist(sample(myRunifVec, size = 100000), main = "Sampling 100,000 values from U(4,7)",
    xlab = "Values", ylab = "Density", xlim = c(4, 7), ylim = c(0,
        8000))

# Plotting sample from Unif(14,24)
myRunifVec <- runif(10000000, min = 14, max = 24)
hist(sample(myRunifVec, size = 100000), main = "Sampling 100,000 values from U(14,24)",
    xlab = "Values", ylab = "Density", xlim = c(14, 24), ylim = c(0,
        8000))

# Plotting sample from Unif(-1,1)
myRunifVec <- runif(10000000, min = -1, max = 1)
hist(sample(myRunifVec, size = 100000), main = "Sampling 100,000 values from U(-1,1)",
    xlab = "Values", ylab = "Density", xlim = c(-1, 1), ylim = c(0,
        8000))
```

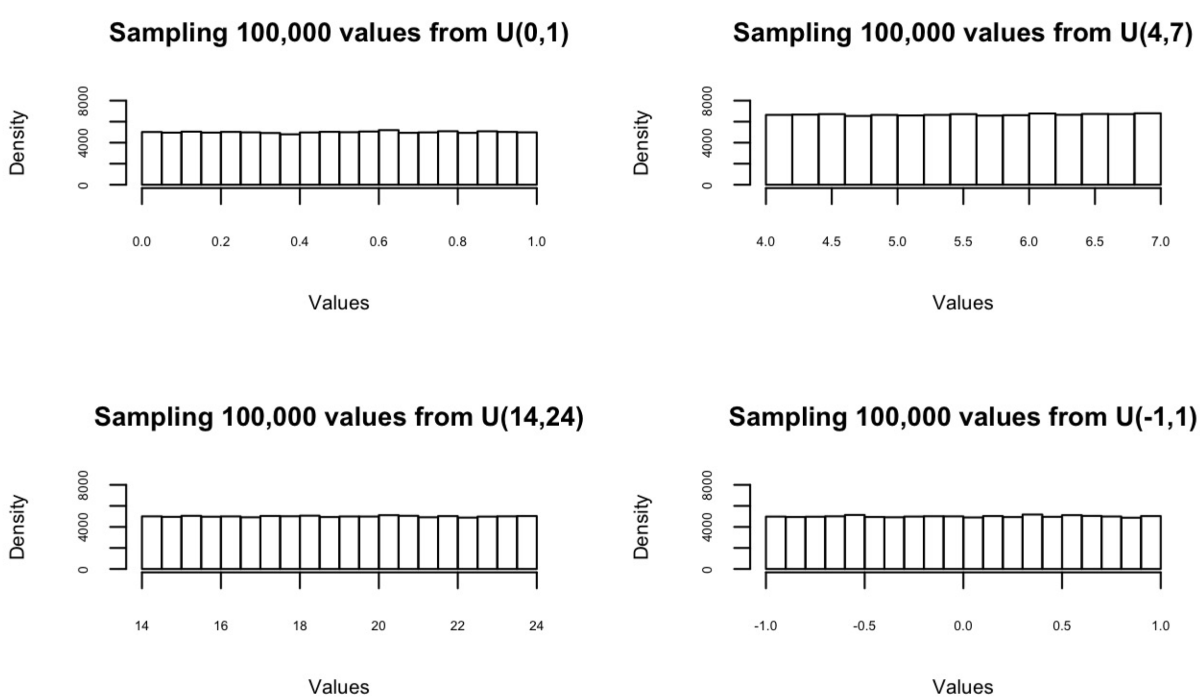

Figure 3: Sampling over different uniform distributions

For each histogram in `Figure 3`, we see that the distributions of our samples are approximately uniform over the same ranges of the uniform distributions from which they were sampled.

3.
```
# Sampling from Unif(0,1)
col1 <- runif(10000000, min = 0, max = 1)
col2 <- runif(10000000, min = 0, max = 1)
myRunifDataFrame <- data.frame(col1, col2)

# Taking the sum of the samples
myRunifDataFrame$runifSum <- myRunifDataFrame$col1 + myRunifDataFrame$col2

hist(myRunifDataFrame$runifSum, main = "Sum of Two U(0,1)", xlab = "Values",
    ylab = "Density", xlim = c(0, 2), ylim = c(0, 1400000))
```

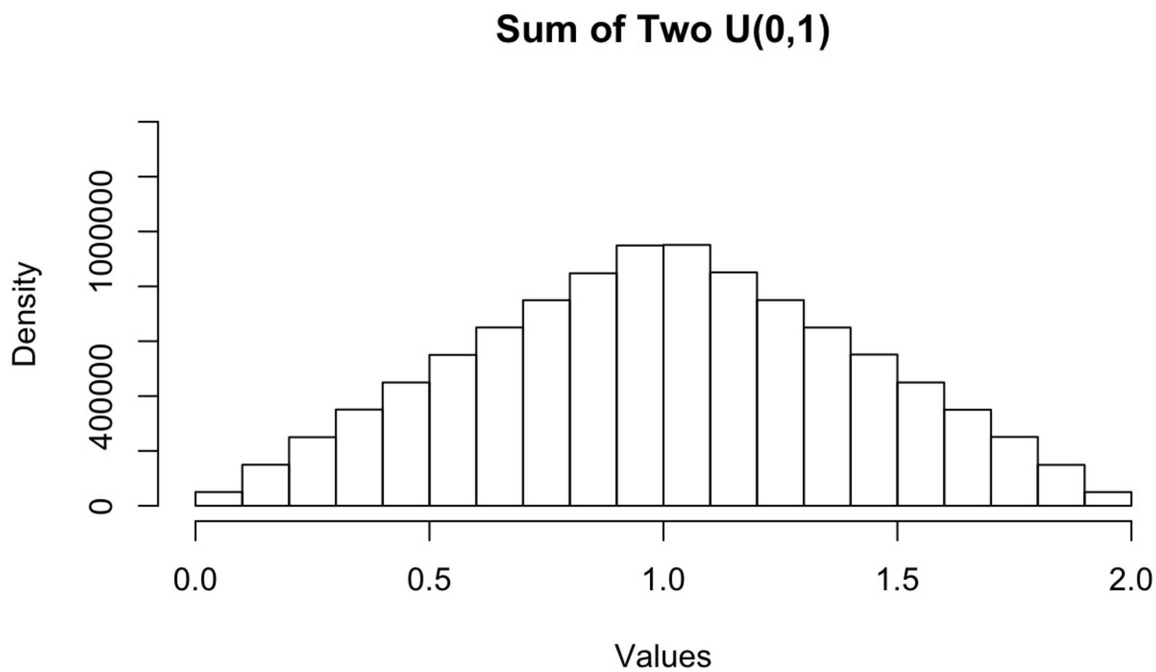

Figure 4: Sum of two random samples from U(0,1)

In `Figure 4`, we took the sum of two random samples from $Unif(0,1)$. The result no longer has a uniform distribution. Instead, we see that more of our observations are near 1 and less are near 0 and 2.

4.
```r
# Sampling from Exp(1)
col1 <- rexp(10000000, rate = 1)
col2 <- rexp(10000000, rate = 1)
myRunifDataFrame <- data.frame(col1, col2)

# Taking the sum of the samples
myRunifDataFrame$runifSum <- myRunifDataFrame$col1 + myRunifDataFrame$col2

# Plotting side-by-side histograms of samples
par(mfrow = c(1, 2), cex.main = 0.75, cex.lab = 0.75, cex.axis = 0.75)
hist(myRunifDataFrame$runifSum, main = "Sum of two random samples from Exp(1)",
    xlab = "Values", ylab = "Density", xlim = c(0, 25), ylim = c(0,
        3500000))

hist(rgamma(10000000, shape = 2, rate = 1), main = "Gamma Distribution: Gamma(2, 1)",
    xlab = "Values", ylab = "Density", xlim = c(0, 25), ylim = c(0,
        3500000))
```

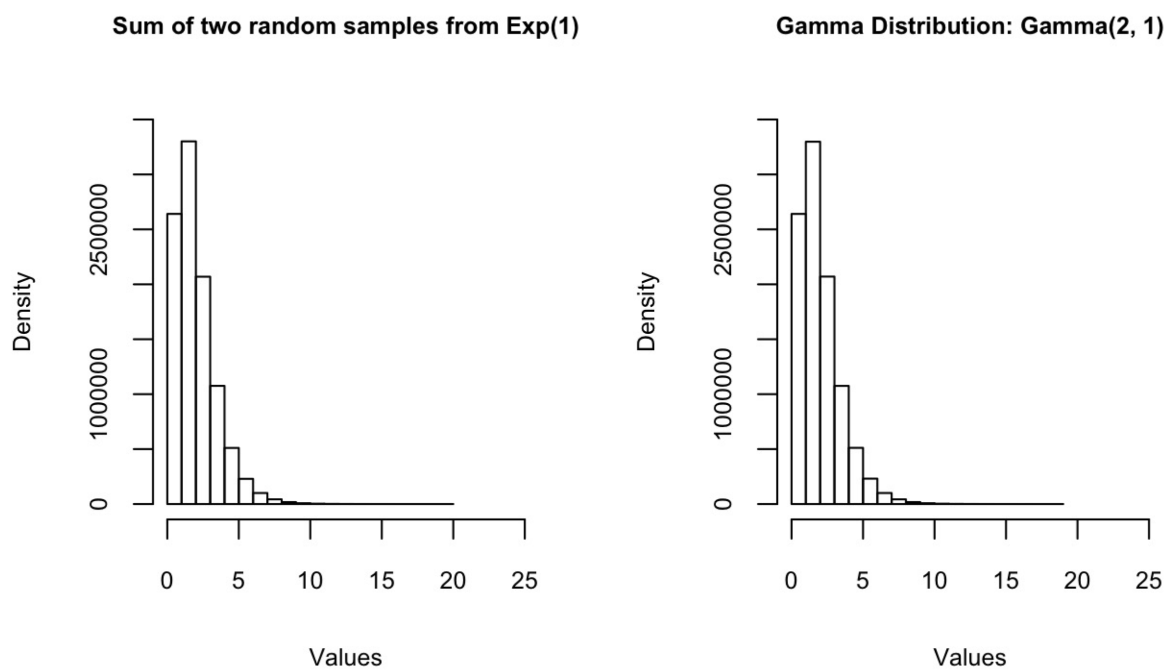

Figure 5: Sum of Exponential distribution samples compared to the Gamma distribution

In `Figure 5`, we took the sum of two samples from $Exp(1)$ and compared it to $Gamma(2, 1)$. We can see from the histograms that the sum of the two exponentially distributed samples seems to have a distribution very close to that of the gamma distribution.

Question 4

```r
# Creating the variables

# Model 1
set.seed(100)
x_1 <- runif(100000, -100, 100)
y_1 <- rexp(100000, rate = 0.5)
model_1 <- list(x = x_1, y = y_1)

# Model 2
set.seed(999)
x_2 <- rnorm(100000, -100, 100)
y_2 <- rexp(100000, rate = 0.5)
model_2 <- list(x = x_2, y = y_2)

# Model 3
set.seed(543)
x_3 <- rnorm(100000, -100, 100)
y_3 <- rnorm(100000, -100, 100)
model_3 <- list(x = x_3, y = y_3)

# Creating list to store models
Model_list <- list(model_1, model_2, model_3)
```

- 1, 2, and 4

```r
# Only use scientific notation on numbers with more than 10
# digits
options(scipen = 10)

# Creating empty data frame to store values
model_data_frame <- data.frame(Model_1 = rep(NA, 6), Model_2 = rep(NA,
    6), Model_3 = rep(NA, 6))

# Creating empty lists for predictions and residuals
predictions <- list()
residuals <- list()

# Computing coefficients for simple linear regression, SSE,
# SSR, SSTO, and R^2 for each model
for (model in c(1, 2, 3)) {
    # Creating model name for placement of data into data frame
    model_name <- paste0("Model_", model)

    # Grabbing x and y vectors for model
    X <- Model_list[[model]]$x
    Y <- Model_list[[model]]$y

    # Computing coefficents for simple linear regression
    b_1 <- sum((X - mean(X)) * (Y - mean(Y)))/sum((X - mean(X))^2)
    b_0 <- (1/length(X)) * (sum(Y) - b_1 * sum(X))

    # Computing simple linear regression prediction
    Y_hat <- b_0 + b_1 * X
    predictions[[model]] <- Y_hat
    residuals[[model]] <- Y - Y_hat

    # Computing SSE
    SSE <- sum((Y - Y_hat)^2)

    # Computing SSR
    SSR <- sum((Y_hat - mean(Y))^2)

    # Computing SSTO
    SSTO <- sum((Y - mean(Y))^2)

    # Computing R^2
    R2 <- SSR/SSTO

    # Storing all info into data frame
    model_data_frame[, c(model_name)] <- c(b_0, b_1, SSE, SSR,
        SSTO, R2)
}
```

3.

```r
par(mfrow = c(1, 3), cex.main = 0.75, cex.lab = 0.75, cex.axis = 0.75)

for (model in c(1, 2, 3)) {
    # Grabbing x and y vectors for model
    X <- Model_list[[model]]$x
    Y <- Model_list[[model]]$y

    # Grabbing predictions for model
    Y_hat <- predictions[[model]]

    # Plotting y on x with fitted regression line for each model
    plot(X, Y, main = paste("Fitted Regression line for Model",
        model), xlab = "X", ylab = "Y")
    lines(X, Y_hat, col = "red")
}
```

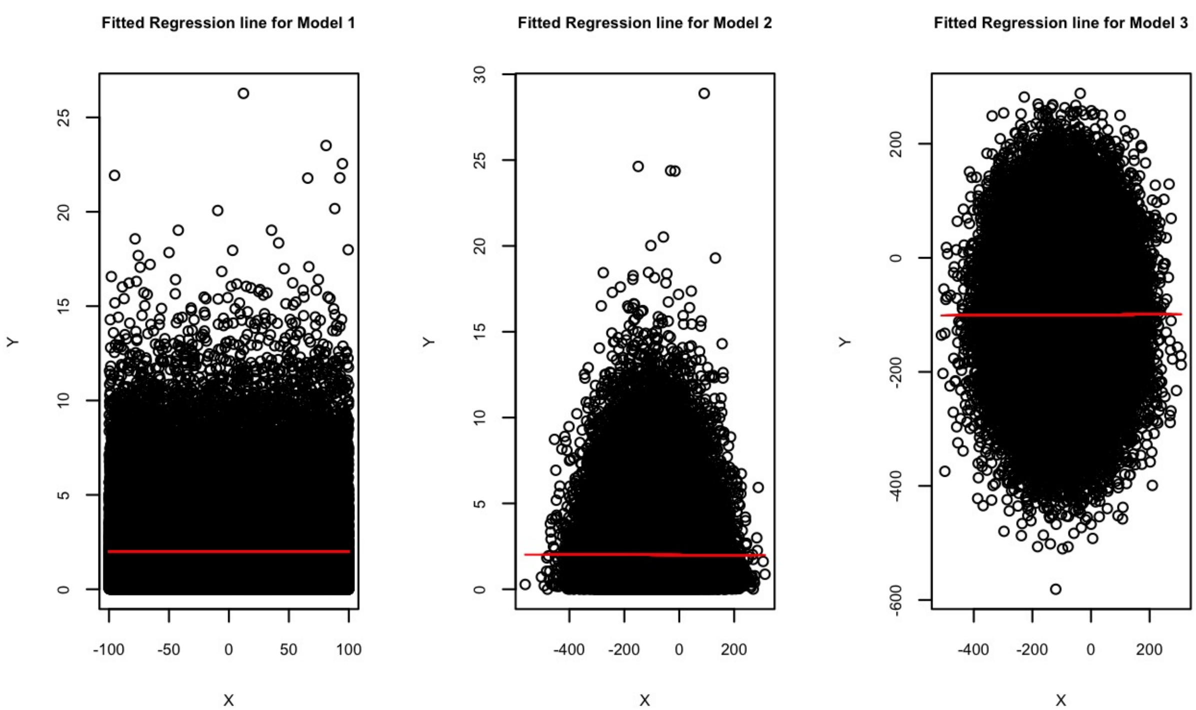

Figure 6: Fitted regression lines for each of the models

5.
```r
par(mfrow = c(1, 3), cex.main = 0.75, cex.lab = 0.75, cex.axis = 0.75)

for (model in c(1, 2, 3)) {
    # Grabbing x vector for model
    X <- Model_list[[model]]$x

    # Grabbing residuals for model
    e <- residuals[[model]]

    # Plotting residual plots of residuals on x with line through
    # residuals = 0
    plot(X, e, main = paste("Residual Plot for Model", model),
        xlab = "X", ylab = "Residuals")
    abline(h = 0, col = "red")
}
```

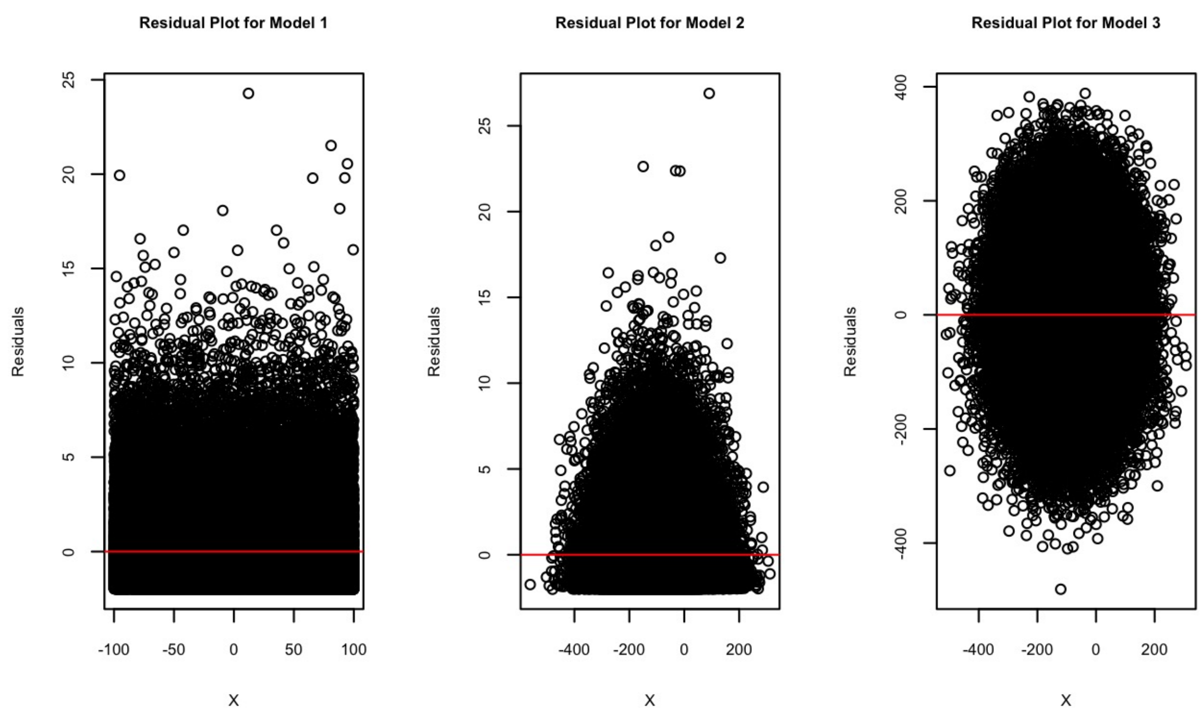

Figure 7: Residual plots for each of the models

```r
# Creating table of linear regression results for each model
row.names(model_data_frame) <- c("$b_0$", "$b_1$", "$SSE$", "$SSR$",
    "$SSTO$", "$R^2$")
colnames(model_data_frame) <- c("**Model 1**", "**Model 2**",
    "**Model 3**")
set.caption("Model simple linear regression results")
pandoc.table(model_data_frame, emphasize.rownames = FALSE, justify = "center")
```

Table 7: Model simple linear regression results

	Model 1	**Model 2**	**Model 3**
b_0	1.995	1.999	-99.87
b_1	0.00002418	-0.00003205	0.002599
SSE	396806	398628	999572342
SSR	0.195	1.032	6728
$SSTO$	396806	398629	999579070
R^2	0.0000004914	0.00000259	0.000006731

We can see in `Table 7` that none of the models performed well under simple linear regression. There is no evidence of a linear trend between the x and y variables as indicated by the low coefficient of determination for each model.