Assignment 1
CSCI E 63 – Big Data Analytics

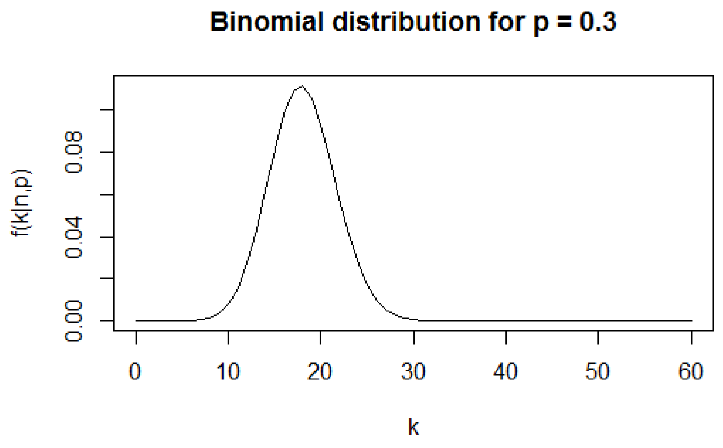
Problem 1:

Binomial distribution describes coin tosses with potentially doctored or altered coins. Value of p is the probability that head comes on top. If both the head and the tail have the same probability, p = 0.5. If the coin is doctored or altered, p could be larger or smaller. Plot on three separate graphs the binomial distribution for p = 0.3, p = 0.5 and p = 0.8 for the total number of trials n = 60 as a function of k, the number of successful (head up) trials. Subsequently, place all three curves on the same graph. For each value of p, determine 1st Quartile, median, mean, standard deviation and the 3rd Quartile. Present those values as a vertical box plot with the probability p on the horizontal axis.

Answer:

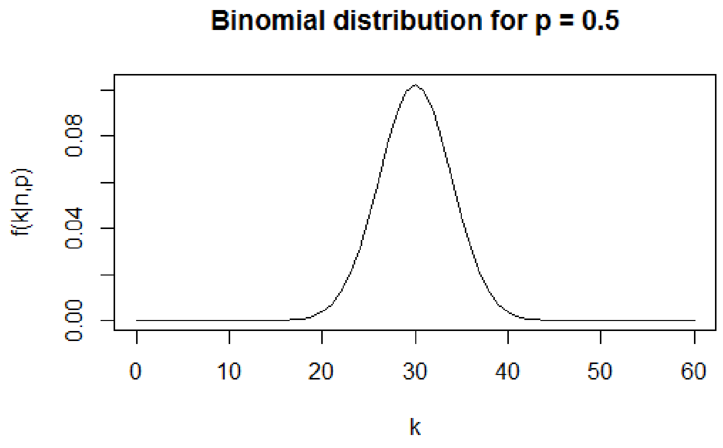
A] Plot on three separate graphs the binomial distribution for p = 0.3, p = 0.5 and p = 0.8

```
#total number of trials n = 60
n = 60
# Create a sample of 60 numbers which are incremented by 1 to represent successful trials.
k <- seq(0,60,by = 1)
# Create the binomial distribution.
binomDistr <- dbinom(k, n, prob = 0.3)
# Plot the graph for this sample.
plot(k, binomDistr, main = "Binomial distribution for p = 0.3", xlab = "k", ylab = "f(k|n,p)", type = "l")
```

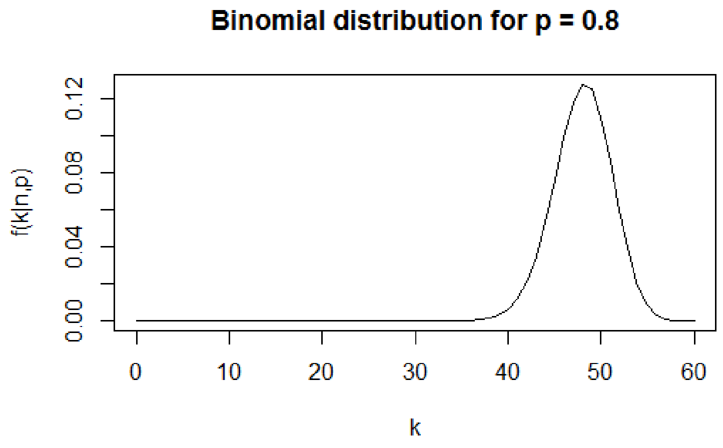


```
#total number of trials n = 60
n = 60
# Create a sample of 60 numbers which are incremented by 1 to represent successful trials.
k <- seq(0,60,by = 1)
# Create the binomial distribution.
binomDistr <- dbinom(k, n, prob = 0.5)
# Plot the graph for this sample.
```

plot(k, binomDistr, main = "Binomial distribution for p = 0.5", xlab = "k", ylab = "f(k|n,p)", type = "l")



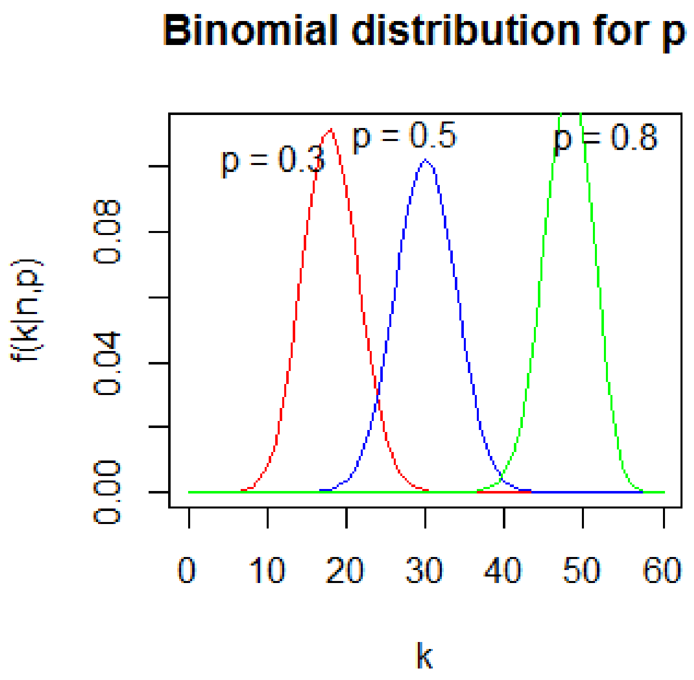
#total number of trials n = 60
n = 60
# Create a sample of 60 numbers which are incremented by 1 to represent successful trials.
k <- seq(0,60,by = 1)
# Create the binomial distribution.
binomDistr <- dbinom(k, n, prob = 0.8)
# Plot the graph for this sample.
plot(k, binomDistr, main = "Binomial distribution for p = 0.8", xlab = "k", ylab = "f(k|n,p)", type = "l")



B] Place all three curves on the same graph
#total number of trials n = 60
n = 60
# Create a sample of 60 numbers which are incremented by 1 to represent successful trials.
k <- seq(0,60,by = 1)
# Create the binomial distribution.

Assignment 1
CSCI E 63 – Big Data Analytics

```
binomDistr1 <- dbinom(k, n, prob = 0.3)
binomDistr2 <- dbinom(k, n, prob = 0.5)
binomDistr3 <- dbinom(k, n, prob = 0.8)
# Plot the graph for this sample.
plot(k, binomDistr1, main = "Binomial distribution for p", xlab = "k", ylab = "f(k|n,p)", type = "l",
col="red")
lines(k, binomDistr2, col = "blue")
lines(k, binomDistr3, col = "green")
# Label each plot
text(locator(), labels = c("p = 0.3", "p = 0.5", "p = 0.8"))
```



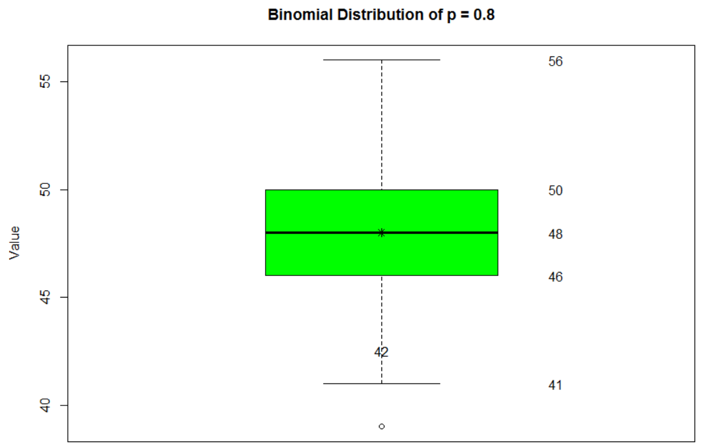
C] Determine 1st Quartile, median, mean, standard deviation and the 3rd Quartile. Present those values as a vertical box plot

	p = 0.3	p = 0.5	P = 0.8
Mean of distribution	0.01639344	0.01639344	0.01639344
k at Mean	18	30	48
Median of distribution	8.35738e-06	4.613852e-05	1.572006e-07
Standard Deviation of distribution	0.03239755	0.03062992	0.03527981
Standard Deviation in k	3.549648	3.872983	3.098387
1st Quartile	7.460887e-13	3.349811e-10	6.585109e-20
3rd Quartile	0.009613404	0.01227688	0.005842579

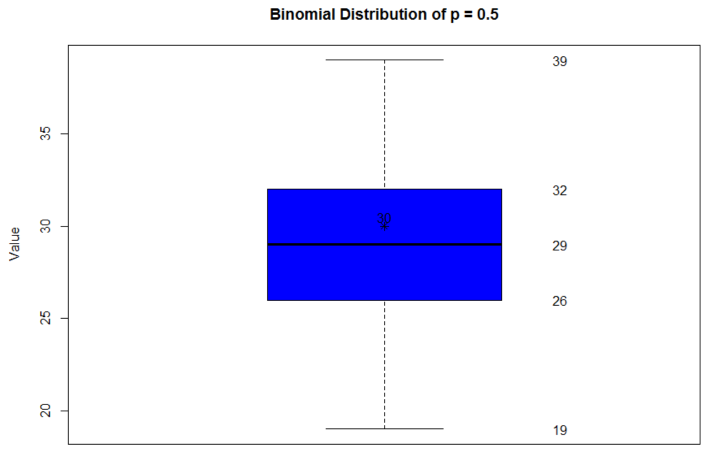
Assignment 1
CSCI E 63 – Big Data Analytics

```
# Total number of trials n = 60
n = 60
# Create a sample of 60 numbers which are incremented by 1 to represent successful trials.
k <- seq(0,60,by = 1)
# Probabilities to be considered
p1 = 0.3
p2 = 0.5
p3 = 0.8
# Create the binomial distribution.
binomDistr1 <- dbinom(k, n, prob = p1)
binomDistr2 <- dbinom(k, n, prob = p2)
binomDistr3 <- dbinom(k, n, prob = p3)
# Calculate mean, median, standard deviation, 1st quartile and 3rd quartile.
# p = 0.3
f_mean1 = mean(binomDistr1)
k_mean1 = n*p1
median1 = median(binomDistr1)
f_stdDev1 = sd(binomDistr1)
k_stdDev1 = sqrt(n*p1*(1-p1))
quantile1 = quantile(binomDistr1)
firstQuartile1 = quantile1[2]
thirdQuartile1 = quantile1[4]
# p = 0.5
f_mean2 = mean(binomDistr2)
k_mean2 = n*p2
median2 = median(binomDistr2)
f_stdDev2 = sd(binomDistr2)
k_stdDev2 = sqrt(n*p2*(1-p2))
quantile2 = quantile(binomDistr2)
firstQuartile2 = quantile2[2]
thirdQuartile2 = quantile2[4]
# p = 0.8
f_mean3 = mean(binomDistr3)
k_mean3 = n*p3
median3 = median(binomDistr3)
f_stdDev3 = sd(binomDistr3)
k_stdDev3 = sqrt(n*p3*(1-p3))
quantile3 = quantile(binomDistr3)
firstQuartile3 = quantile3[2]
thirdQuartile3 = quantile3[4]
```

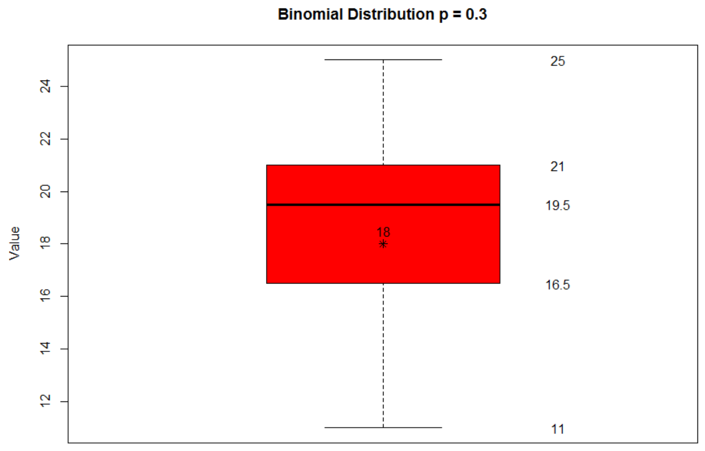
Assignment 1
CSCI E 63 – Big Data Analytics



p = 0.8, mean=48
sd=3.79



p = 0.5, mean=30
sd=3.87



p = 0.3, mean=18
sd=3.55

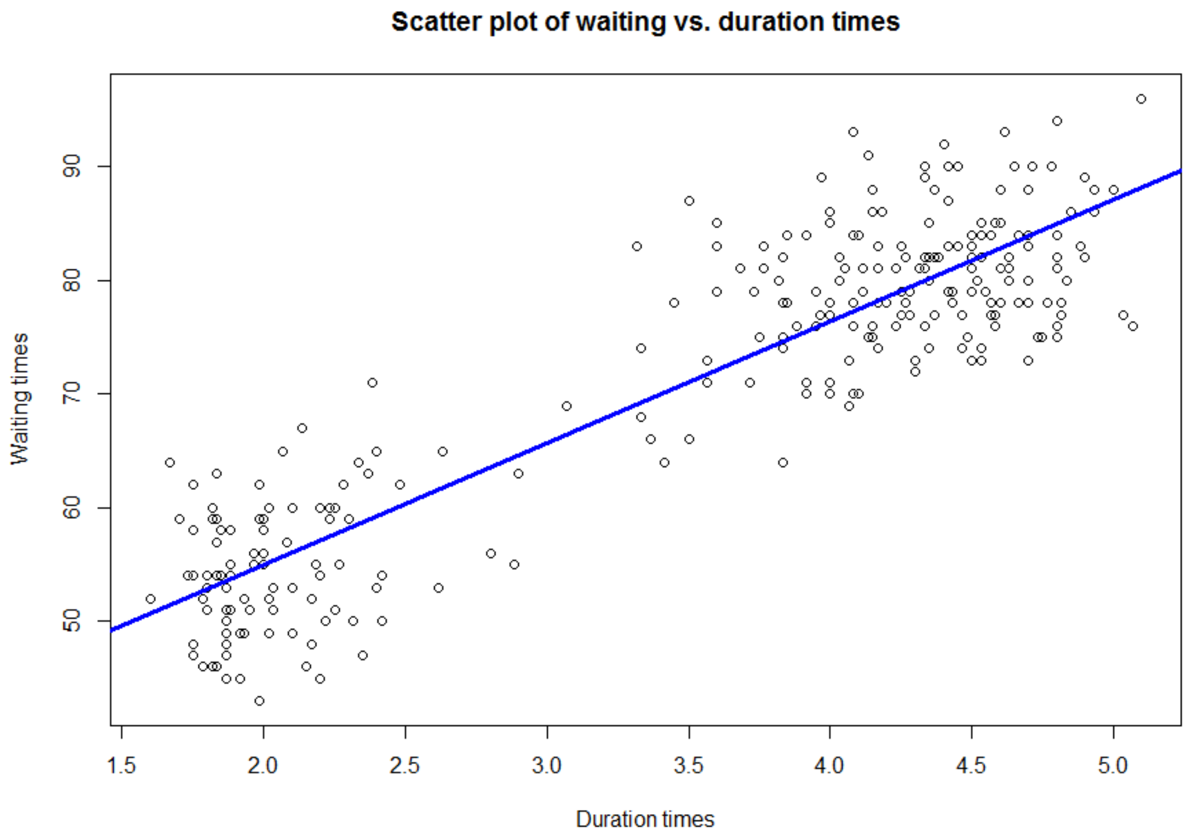
Karan A. Bhandarkar

Assignment 1
CSCI E 63 – Big Data Analytics

```
bp1 <- boxplot(rbinom(60, 60, 0.3), col = "red",
            main = ("Binomial Distribution p = 0.3"),
            xlab = c(paste("p = 0.3, mean", 60 * 0.3, sep = "="),
                  paste("sd", sprintf("%0.2f", sqrt(60 * 0.3 * 0.7)), sep = "=")),
            ylab = "Value")

points(x = 1, y = 60 * 0.3, pch = 8)
text(1, 0.5 + 60 * 0.3, labels = 60 * 0.3)
text(1.3, bp1$stats, labels = bp1$stats)

bp2 <- boxplot(rbinom(60, 60, 0.5), col = "blue",
            main = ("Binomial Distribution of p = 0.5"),
            xlab = c(paste("p = 0.5, mean", 60 * 0.5, sep = "="),
                  paste("sd", sprintf("%0.2f", sqrt(60 * 0.5 * 0.5)), sep = "=")),
            ylab = "Value")

points(x = 1, y = 60 * 0.5, pch = 8)
text(1, 0.5 + 60 * 0.5, labels = 60 * 0.5)
text(1.3, bp2$stats, labels = bp2$stats)

bp3 <- boxplot(rbinom(60, 60, 0.8), col = "green",
            main = ("Binomial Distribution of p = 0.8"),
            xlab = c(paste("p = 0.8, mean", 60 * 0.8, sep = "="),
                  paste("sd", sprintf("%0.2f", sqrt(60 * 0.8 * 0.3)), sep = "=")),
            ylab = "Value")
points(x = 1, y = 60 * 0.8, pch = 8)
text(1, 0.5 + 60 * 0.7, labels = 60 * 0.7)
text(1.3, bp3$stats, labels = bp3$stats)
```

Assignment 1
CSCI E 63 – Big Data Analytics

Problem 2:
Finish the plot of the correlation between waiting times and durations of Old Faithful data. Recreate the scatter plot of waiting vs. duration times. As we mentioned in class, the best linear assessment in the sense of the least squares fit of a relationship (proportionality) between two or many variables can be achieved with R function lm(). lm stands for the linear model. The first argument of lm() is called formula accepts a model which starts with the response variable, waiting in our case, followed by a tilde (symbol ~, read as "is modeled as") followed by the (so called Wilkinson-Rogers) model on the right. In our case we simply assume that waiting time is proportional to the duration time and that "model" reads: formula = waiting ~ duration. The second argument of function lm() is called data and, in our case, will take value faithful, the data set containing our data. Store the result of function lm() in a variable. The name of that variable is not essential. Call it model. Print the variable. The first component of that variable is the intercept of calculated line with the vertical axis (waiting, here) and the second is the slope of the line. Convince yourself that line with those parameters will truly lie on your graph. Function abline() adds a line to the previously created graph. Next, pass the variable model to the function abline(). Make that line somewhat thicker and blue. Use help(functionName) to find details about invocations of both lm() and abline() functions.
Answer:



Assignment 1
CSCI E 63 – Big Data Analytics

Script:

```
# Recreate the scatter plot of waiting vs. duration times
plot(faithful$eruptions, faithful$waiting,
    main = "Scatter plot of waiting vs. duration times",
    xlab = "Duration times", ylab = "Waiting times")

# The best linear assessment in the sense of the least squares fit
# of a relationship (proportionality) between two or many variables
# can be achieved with R function lm().
# Store the result of function lm() in a variable. Call it model.
model = lm(formula = waiting ~ eruptions, faithful)

# Print the variable.
model

# Pass the variable model to the function abline(). Make that line somewhat thicker and blue.
abline(model, col = "Blue", lwd = 3 )
```

Console:

```
> # Recreate the scatter plot of waiting vs. duration times
> plot(faithful$eruptions, faithful$waiting,
+      main = "Scatter plot of waiting vs. duration times",
+      xlab = "Duration times", ylab = "Waiting times")
>
> # The best linear assessment in the sense of the least squares fit
> # of a relationship (proportionality) between two or many variables
> # can be achieved with R function lm().
> # Store the result of function lm() in a variable. Call it model.
> model = lm(formula = waiting ~ eruptions, faithful)
>
> # Print the variable.
> model

Call:
lm(formula = waiting ~ eruptions, data = faithful)

Coefficients:
(Intercept)    eruptions
      33.47        10.73

>
> # Pass the variable model to the function abline(). Make that line somewhat thick
er and blue.
> abline(model, col = "Blue", lwd = 3 )
```

Assignment 1
CSCI E 63 – Big Data Analytics

Problem 3. Calculate the covariance matrix of the faithful data. Determine the eigenvalues and eigenvectors of that matrix. Demonstrate that two eigenvectors are mutually orthogonal. Examine whether the eigenvector with the larger eigenvalue is parallel with line discovered by lm() function it the previous problem.

Answer:

Script:

```
# Calculate the covariance matrix of the faithful data
# The covariance matrix is a matrix that only concerns the relationships between variables.
# So it will be a 2 x 2 square matrix for our example.
covarianceMatrix <- cov(faithful)

# Determine the eigenvalues and eigenvectors of that matrix.
eigen <- eigen(covarianceMatrix)
eigenValues = eigen$values
eigenVectors = eigen$vectors

# Demonstrate that two eigenvectors are mutually orthogonal.
# Eigenvectors are always orthogonal, V'V =IV =I
crossprod(eigenVectors)

# Examine whether the eigenvector with the larger eigenvalue is parallel with line
# discovered by lm() function it the previous problem.
# eigenVector V1 corresponds to the larger eigenValue
# Slope of this vector is:
slope1 = eigenVectors[1,1]/eigenVectors[2,1]
slope1

# Slope of line discovered by lm() in previous problem is:
model = lm(formula = waiting ~ eruptions, faithful)
slope2 <-  coef(model)[2]
slope2

# The slopes of the two lines are not equal and hence the two lines are not parallel
```

Output:

```
> # Calculate the covariance matrix of the faithful data
> # The covariance matrix is a matrix that only concerns the relationships between
variables.
> # So it will be a 2 x 2 square matrix for our example.
> covarianceMatrix <- cov(faithful)
>
> # Determine the eigenvalues and eigenvectors of that matrix.
> eigen <- eigen(covarianceMatrix)
> eigenValues = eigen$values
> eigenVectors = eigen$vectors
>
> # Demonstrate that two eigenvectors are mutually orthogonal.
> # Eigenvectors are always orthogonal, V'V =IV =I
> crossprod(eigenVectors)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
>
> # Examine whether the eigenvector with the larger eigenvalue is parallel with lin
e
> # discovered by lm() function it the previous problem.
> # eigenVector V1 corresponds to the larger eigenValue
> # Slope of this vector is:
> slope1 = eigenVectors[1,1]/eigenVectors[2,1]
> slope1
[1] 0.07572801
>
> # Slope of line discovered by lm() in previous problem is:
> model = lm(formula = waiting ~ eruptions, faithful)
> slope2 <-  coef(model)[2]
> slope2
eruptions
 10.72964
>
> # The slopes of the two lines are not equal and hence the two lines are not paral
lel
```

Problem 4. You noticed that eruptions clearly fall into two categories, short and long. Let us say that short eruptions are all which have duration shorter than 3.1 minute. Add a new column to data frame faithful called type, which would have value 'short' for all short eruptions and value 'long' for all long eruptions. Next use boxplot() function to provide your readers with some basic statistical measures for waiting. In a separate plot present the box plot for duration times. Please note that boxplot() function also accepts as its first argument a formula such as waiting ~ type, where waiting is the numeric vector of data values to be split in groups according to the grouping variable type. The second argument of function boxplot() is called data, which in our case will take the name of our dataset, i.e. faithful. Find a way to add meaningful legends to your graphs. Subsequently, present both boxplots on one graph.

Answer:

A] Add a new column to data frame faithful called type, which would have value 'short' for all short eruptions and value 'long' for all long eruptions.

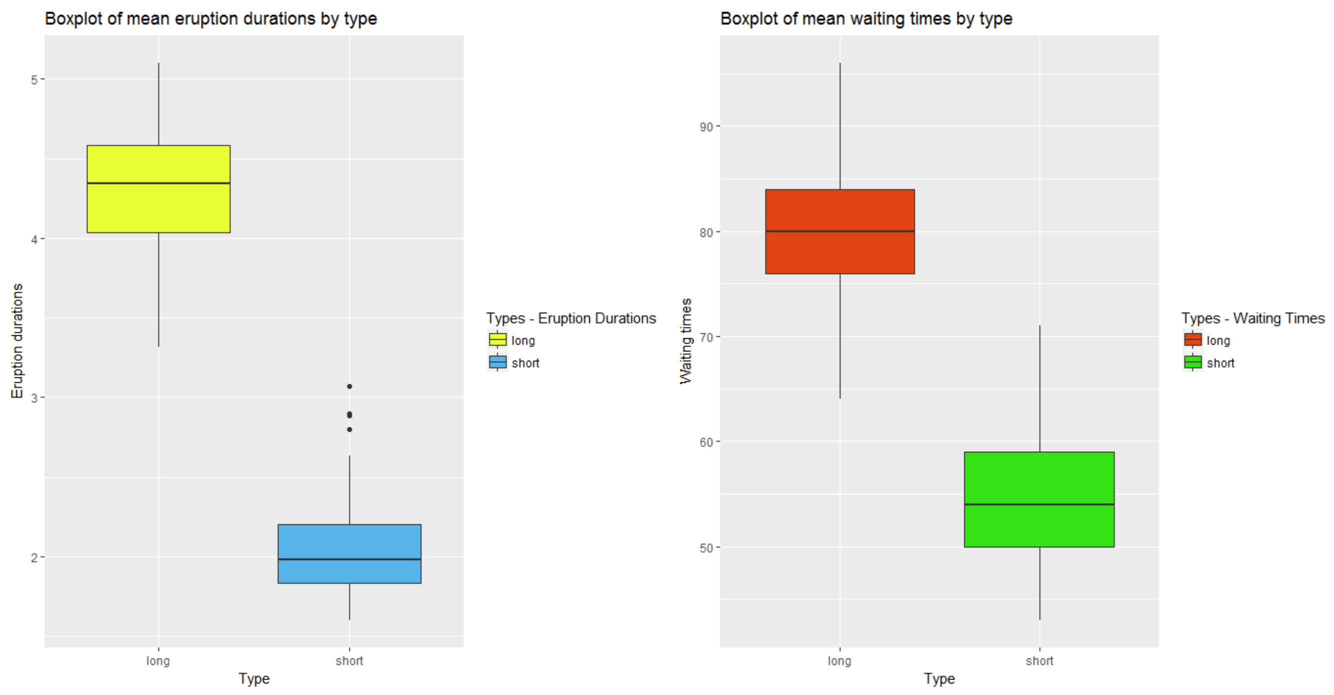
Code:
```
# Separate the eruptions into two intervals
eruptions = faithful$eruptions
# Create a new vector with two factors, 1 and 2 based on eruptions
eruptionIntervals <- findInterval(eruptions, c(0, 3.1, max(eruptions)), left.open = TRUE)
# Add a new column to data frame faithful called type, renaming the two factors to short and long
index <- c(1, 2)
values <- c("short", "long")
faithful$type <- values[match(eruptionIntervals, index)]
```

B] Use boxplot() function to provide your readers with some basic statistical measures for waiting

Code:

```
library(ggplot2)
library(gridExtra)
p1 <- ggplot(faithful, aes(x = type, y = eruptions, fill = type)) +
  geom_boxplot() +
  scale_x_discrete(name = "Type") +
  scale_y_continuous(name = "Eruption durations") +
  ggtitle("Boxplot of mean eruption durations by type") +
  scale_fill_manual(values = c("#E9FF33", "#56B4E9")) +
  labs(fill = "Types - Eruption Durations")
```

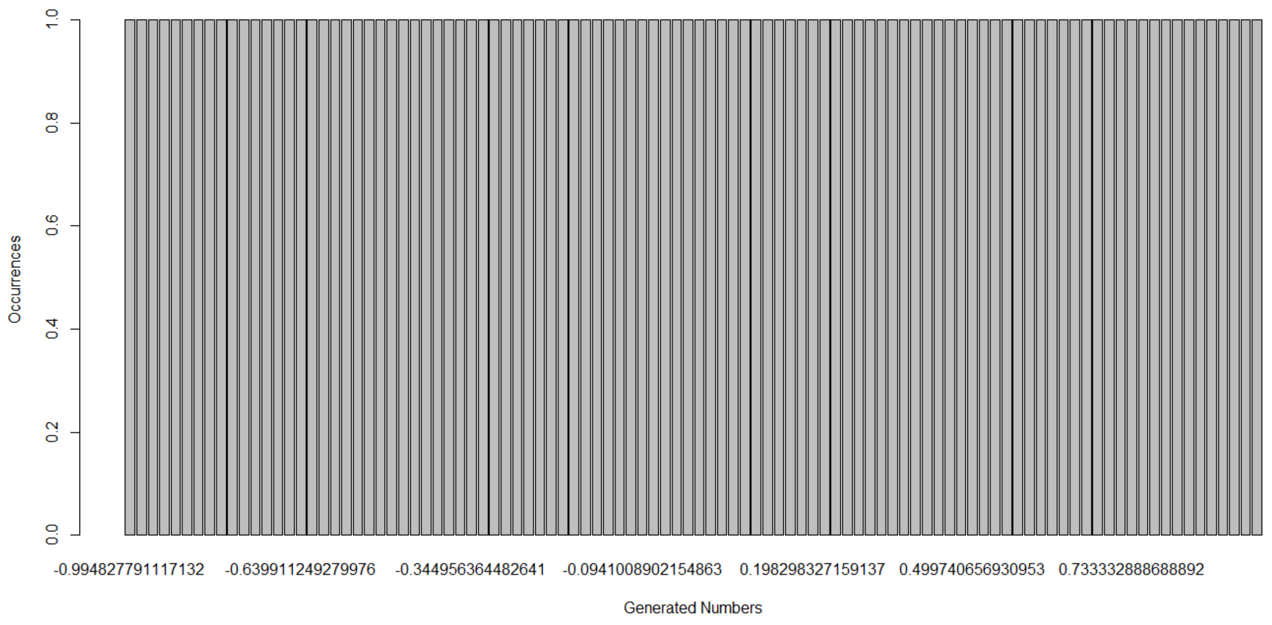
```
p2 <- ggplot(faithful, aes(x = type, y = waiting, fill = type)) +
  geom_boxplot() +
  scale_x_discrete(name = "Type") +
  scale_y_continuous(name = "Waiting times") +
  ggtitle("Boxplot of mean waiting times by type") +
  scale_fill_manual(values = c("#E14513", "#35E216")) +
  labs(fill = "Types - Waiting Times")

grid.arrange(p1, p2, nrow = 1, ncol = 2)
```



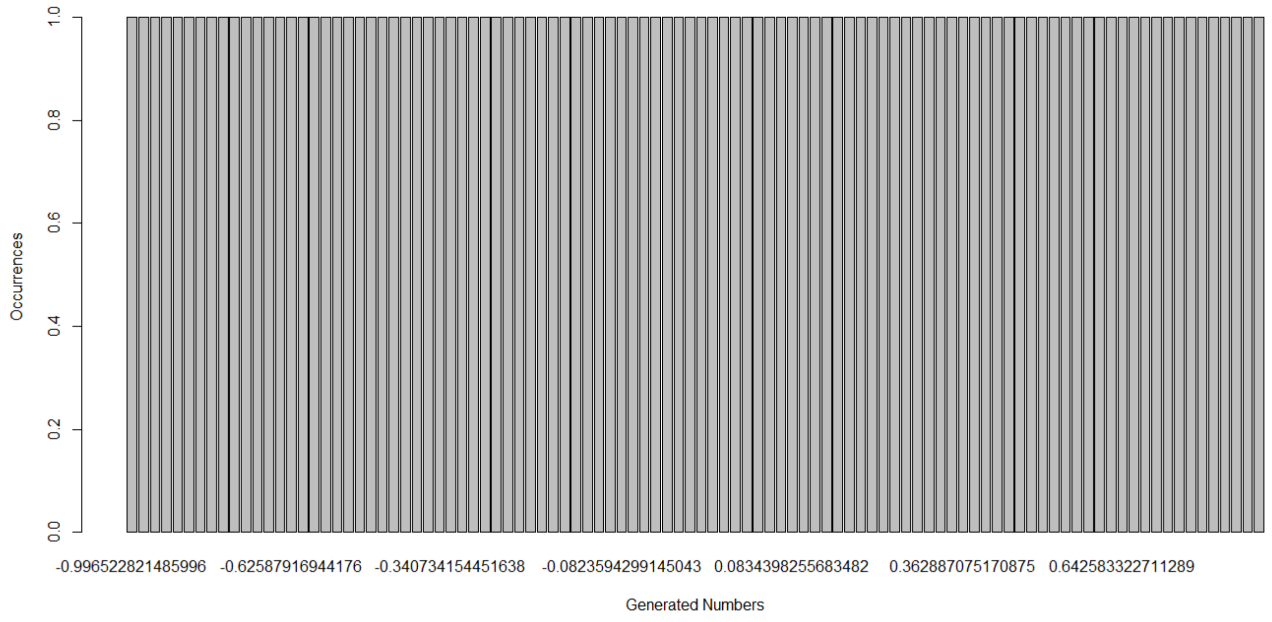
Problem 5. Create a matrix with 40 columns and 100 rows. Populate each column with random variable of the uniform distribution with values between -1 and 1 (symmetric around zero). Let the distribution for each column appear like the one on slide 92 of the lecture note, except centered around zero. Present two distributions contained in any two randomly selected columns of your matrix on two separate plots. Convince yourself that generated distributions are (close to) uniform.

Code:

```
# Create a matrix with 40 columns and 100 rows.
# Populate each column with random variable of the uniform distribution with values between -1 and 1
ncol = 40
nrow = 100
sampleMatrix = matrix(runif(nrow*ncol, -1, 1), nrow, ncol)

# Select two random columns
column1 = sampleMatrix[, sample(1:40, 1)]
column2 = sampleMatrix[, sample(1:40, 1)]

# Plot each column's distribution
barplot((table(column1)), xlab="Generated Numbers", ylab="Occurrences")
barplot((table(column2)), xlab="Generated Numbers", ylab="Occurrences")

# The bar plots of both the columns shows a uniform distribution between -1 and 1.
```



Assignment 1
CSCI E 63 – Big Data Analytics

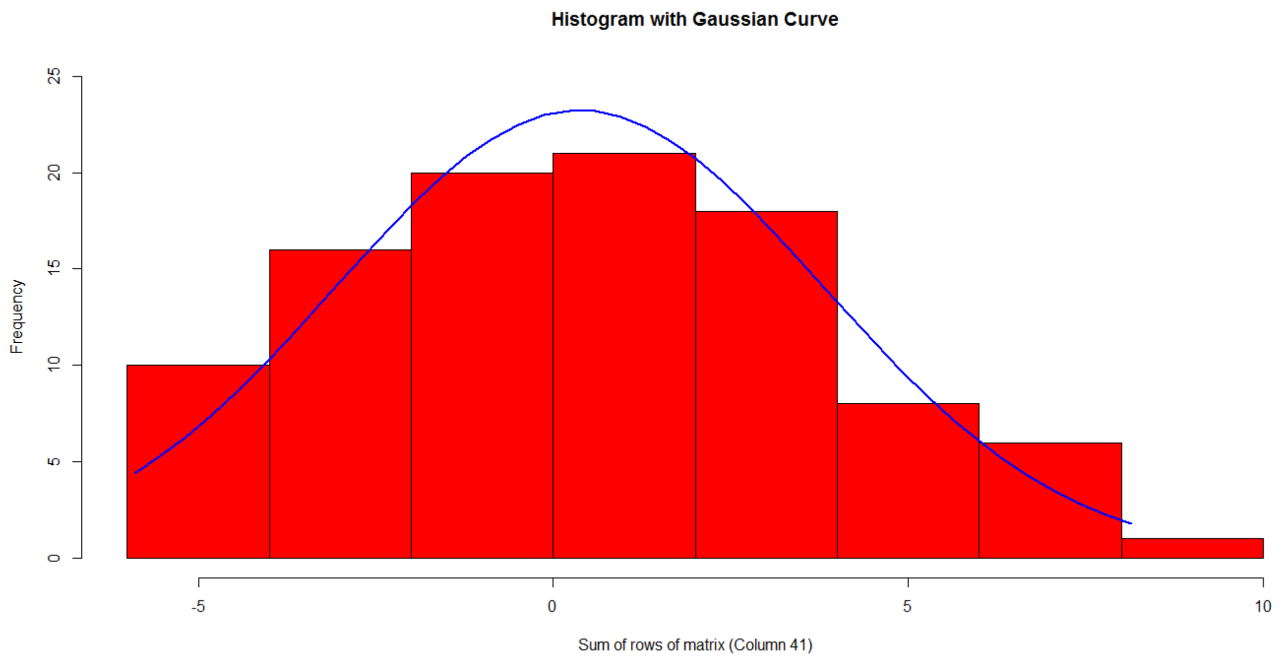


Assignment 1
CSCI E 63 – Big Data Analytics

Problem 6. Start with your matrix from problem 5. Add yet another column to that matrix and populate that column with the sum of original 40 columns. Create a histogram of values in the new column showing that the distribution resembles the Gaussian curve. Add a true, calculated, Gaussian curve to that diagram with the parameters you expect from the sum of 40 random variables of uniform distribution.

Code:

```
# Add yet another column to that matrix and populate that column with the sum of original 40 columns.
updatedSampleMatrix <- cbind (sampleMatrix,rowSums(sampleMatrix));
# Create a histogram of values in the new column showing that the distribution resembles the Gaussian curve.
# Add a true, calculated, Gaussian curve to that diagram with the parameters you expect from the sum of 40 random variables of uniform distribution.
h<-hist(updatedSampleMatrix[,41], col="red", xlab="Sum of rows of matrix (Column 41)", ylim = c(0,25),
    main="Histogram with Gaussian Curve")
xfit<-seq(min(updatedSampleMatrix[,41]),max(updatedSampleMatrix[,41]),length=40)
yfit<-dnorm(xfit,mean=mean(updatedSampleMatrix[,41]),sd=sd(updatedSampleMatrix[,41]))
yfit <- yfit*diff(h$mids[1:2])*length(updatedSampleMatrix[,41])
lines(xfit, yfit, col="blue", lwd=2)
```


Histogram with Gaussian Curve
Frequency
Sum of rows of matrix (Column 41)