ASSIGNMENT 1

DSCI 6607 – Programmatic Data Analysis Using Python and R
Name: Sahil Khan
Student ID: 202482066
Submission Date: 15-Oct-2024

Question 1

Simulate tossing an unfair coin (probability of heads = 0.63) for 301 trials and compute the proportion of heads.

$$\widehat{p} = \frac{\text{Number of Heads}}{\text{Total number of trials}}$$

```python
import random

probHeads = 0.63
tossTrials = 301

headsCount = 0

for i in range(tossTrials):
    if random.random() < probHeads:
        headsCount += 1

estProbHeads = headsCount / tossTrials


print(f"Number of heads: {headsCount}")
```

```
## Number of heads: 213
```

```python
print(f"Estimated probability of Heads: {estProbHeads}")
```

```
## Estimated probability of Heads: 0.707641196013289
```

Question 2

a) Implementation of the Bisection Method

```python
import math

def bisection_root(a, b, f):
    """Finds the root of a function using the bisection method.

    Args:
        a: Left endpoint of the interval.
        b: Right endpoint of the interval.
        f: The function for which to find the root.

    Returns:
        The root of the function.
    """

    while abs(f((a + b) / 2)) > 1e-5:  # Changed the stop condition to be based on f(m)
        m = (a + b) / 2
        if f(m) > 0:
            a = m
        else:
            b = m
    return (a + b) / 2
```

The bisection method requires an interval [a, b] where f(a) and f(b) have opposite signs (one positive, one negative). This guarantees that the function crosses zero at least once within the interval.

For given interval [-2.3,-0.23]:

```python
def f(x):
    return x**3 + 4 * x**2 - 3

a = -2.3
b = -0.23
print(f"The sign of f({a}) = x^3 + 4x^2 - 3 for a={a} is: {f(a)}")
```

```
## The sign of f(-2.3) = x^3 + 4x^2 - 3 for a=-2.3 is: 5.9929999999999986
```

```python
print(f"The sign of f({b}) = x^3 + 4x^2 - 3 for a={b} is: {f(b)}")
```

```
## The sign of f(-0.23) = x^3 + 4x^2 - 3 for a=-0.23 is: -2.800567
```

Since f(-2.3) and f(-0.23) have opposite signs, the bisection method is guaranteed to find a root in the interval [-2.3, -0.23].

b) Finding the root of $f(x) = x^3 + 4x^2 - 3$ in interval $[-2.3, -0.23]$.

```python
root = bisection_root(a, b, f)
print(f"The root of f(x) = x^3 + 4x^2 - 3 in the interval [-2.3,-0.23] is: {root}")
```

```
## The root of f(x) = x^3 + 4x^2 - 3 in the interval [-2.3,-0.23] is: -0.9999999809265135
```

Question 3

```r
# library(datasets)
head(rivers)
```

```
## [1] 735 320 325 392 524 450
```

a) R Function to Compute Cumulative Sample Means

```r
cumulative_means <- function(x) {
  n <- length(x)
  cumsum(x) / (1:n)
}
```

b) $n = 1000$ observations with replacement from 'rivers' dataset.

```r
n <- 1000
rivers_sample <- sample(rivers, n, replace = TRUE)
head(rivers_sample)
```

```
## [1] 630 618 407 237 230 605
```

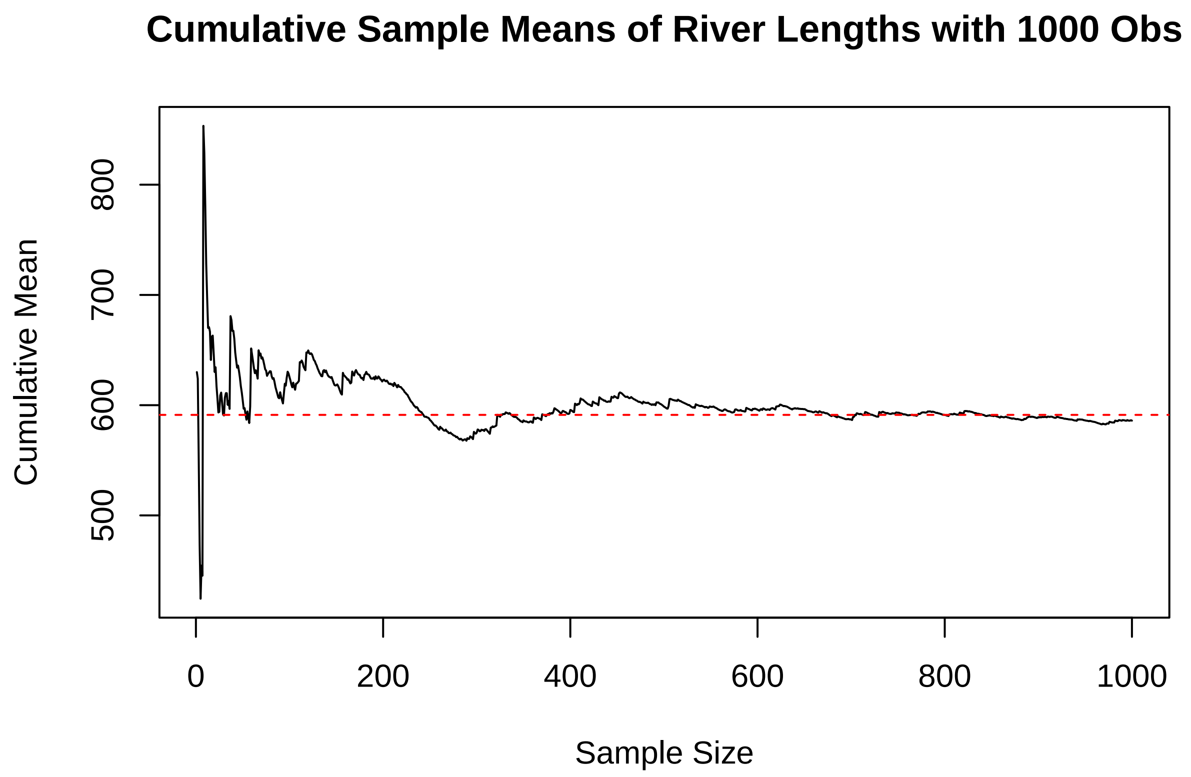
c) Applying function (a) to the data of part (b).

```r
cumulative_sample_means <- cumulative_means(rivers_sample)
head(cumulative_sample_means)
```

```
## [1] 630.0000 624.0000 551.6667 473.0000 424.4000 454.5000
```

d) Plotting the population mean.

```r
plot(cumulative_sample_means, type = "l", xlab = "Sample Size", ylab = "Cumulative Mean",
     main = "Cumulative Sample Means of River Lengths with 1000 Obs")
abline(h=mean(rivers), col="red", lty=2)
```

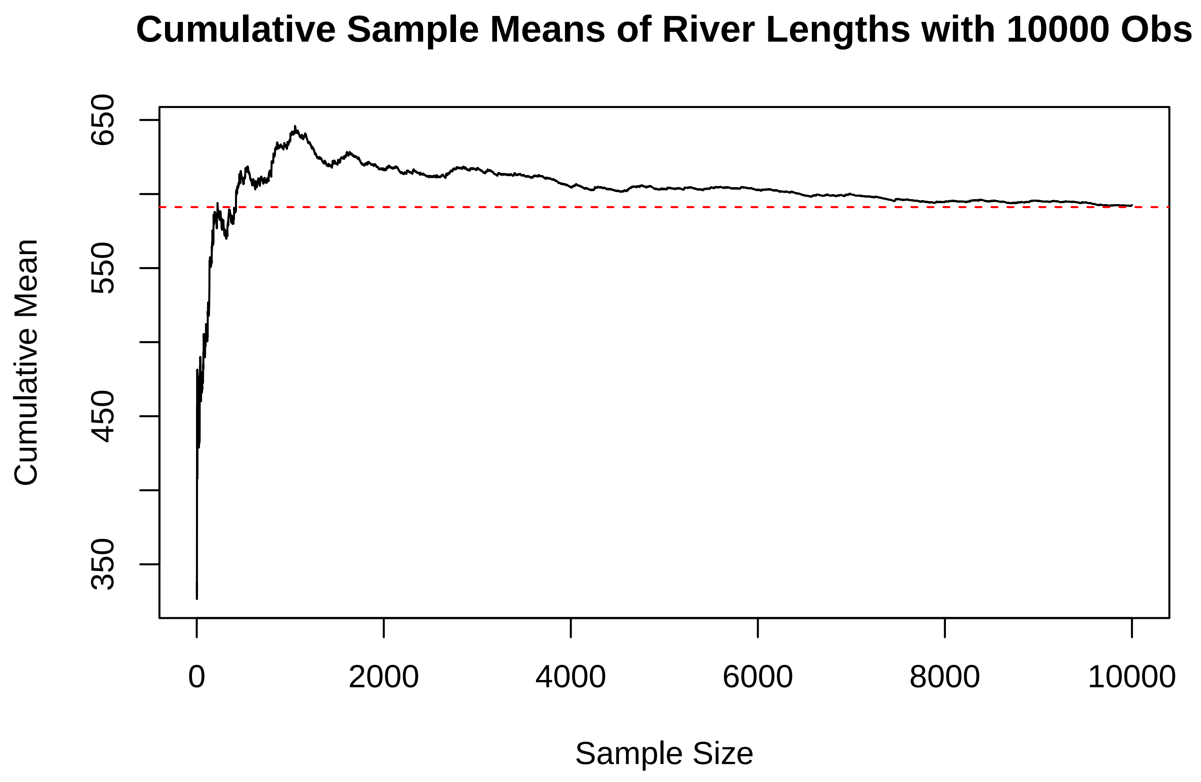


Explanation:

The plot shows how the cumulative sample mean converges towards the population mean (red dashed line) as the sample size increases.
The Central Limit Theorem suggests that as ‘n’ grows larger, the distribution of sample means approaches a normal distribution centered around the population mean. We can observe this convergence visually in the plot.
The more samples we have, the closer the cumulative sample mean gets to the true population mean.

Plotting with 10000 Observations:

```r
n <- 10000
rivers_sample <- sample(rivers, n, replace = TRUE)
cumulative_sample_means <- cumulative_means(rivers_sample)

plot(cumulative_sample_means, type = "l", xlab = "Sample Size", ylab = "Cumulative Mean",
     main = "Cumulative Sample Means of River Lengths with 10000 Obs")
abline(h = mean(rivers), col = "red", lty = 2)
```



The plot clearly shows that as the sample size increases, the cumulative sample mean converges towards the true population mean (represented by the red dashed line). The fluctuations around the population mean decrease with increasing sample size, illustrating the principle of the Central Limit Theorem: with larger samples, the sample mean becomes a more precise estimate of the population mean.

Question 4

The `airquality` data set reports the daily air quality measurements in New York, May to September 1973. The data set includes 153 observations and 6 variables.

```r
# library(datasets)
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

a) Ridge Regression Model Coefficients:

```r
ridge_regression <- function(X, y, lambda) {
  n <- nrow(X)
  p <- ncol(X)
  I <- diag(p)
  beta_ridge <- solve(t(X) %*% X + lambda * I) %*% t(X) %*% y
  return(beta_ridge)
}
```

b) Normalize Variables:

```r
ozone <- airquality$Ozone
solar_r <- airquality$Solar.R
wind <- airquality$Wind
temp <- airquality$Temp

data <- data.frame(Ozone = ozone, Solar.R = solar_r, Wind = wind, Temp = temp)
data <- na.omit(data) # Remove rows with missing values

normalize <- function(x) {
  (x - mean(x)) / sd(x)
}

data_norm <- as.data.frame(lapply(data, normalize))

X_norm <- as.matrix(data_norm[, c("Solar.R", "Wind", "Temp")])
y_norm <- data_norm$Ozone

print("Normalized Explanatory Variables (X_norm):")
```

```
## [1] "Normalized Explanatory Variables (X_norm):"
```

```r
print(head(X_norm, 5))
```

```
##          Solar.R       Wind       Temp
## [1,]  0.05702761 -0.7138405 -1.1325108
## [2,] -0.73285918 -0.5451928 -0.6078501
## [3,] -0.39276904  0.7477726 -0.3979858
## [4,]  1.40641756  0.4385852 -1.6571715
## [5,]  1.25282846 -0.3765451 -1.3423751
```

```r
print("Normalized Response Variable (y_norm):")
```

```
## [1] "Normalized Response Variable (y_norm):"
```

```r
print(head(y_norm, 10))
```

```
##  [1] -0.03302982 -0.18328840 -0.90452961 -0.72421931 -0.57396073 -0.69416759
##  [7] -1.02473648 -0.78432275 -0.93458133 -0.84442618
```

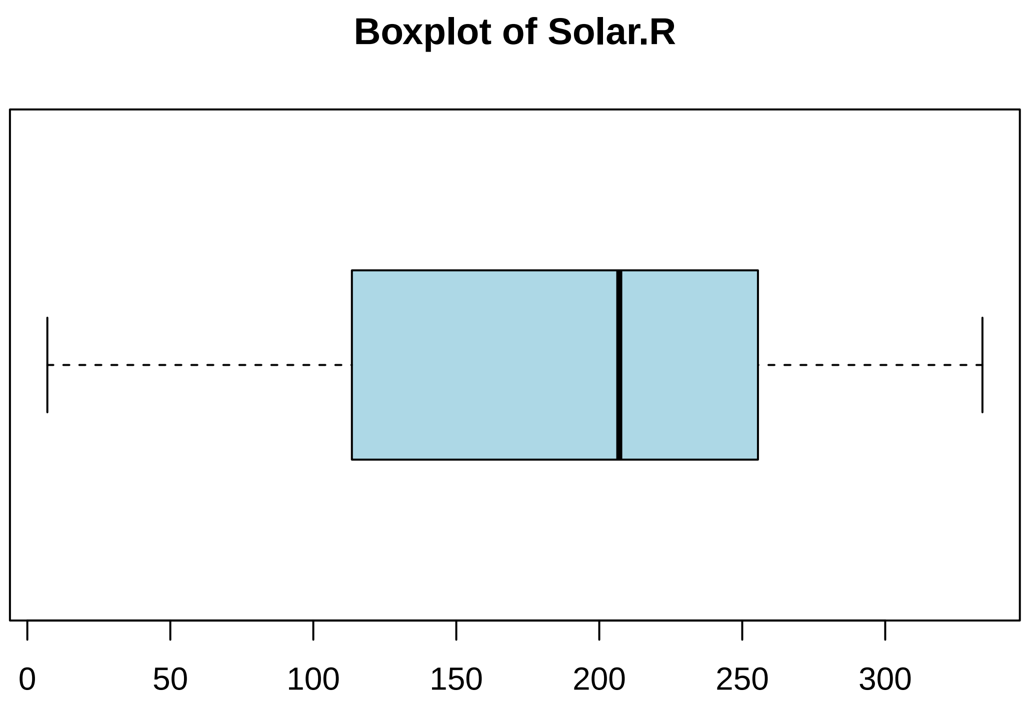
c) computes $\widehat{\beta}_R$ of model for $\lambda = 0.1$:

```r
lambda <- 0.1
beta_r <- ridge_regression(X_norm, y_norm, lambda)

print(beta_r)
```

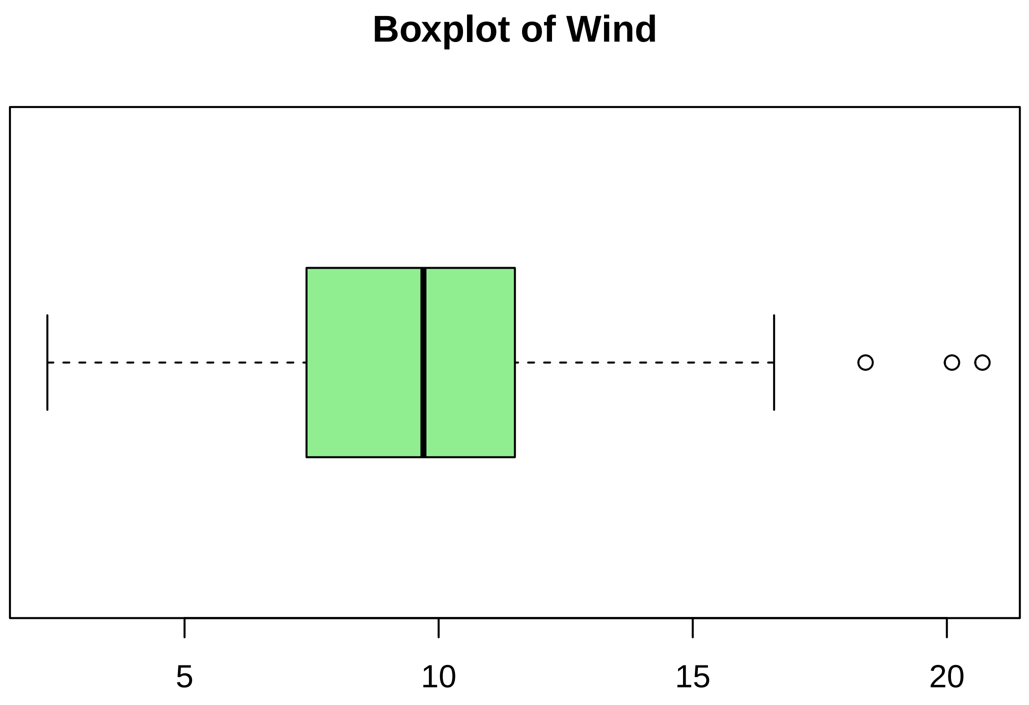
```
##                [,1]
## Solar.R   0.1638378
## Wind     -0.3562652
## Temp      0.4727975
```

d) Box Plot of 'OZONE', 'SOLAR', 'WIND' AND 'TEMP' :

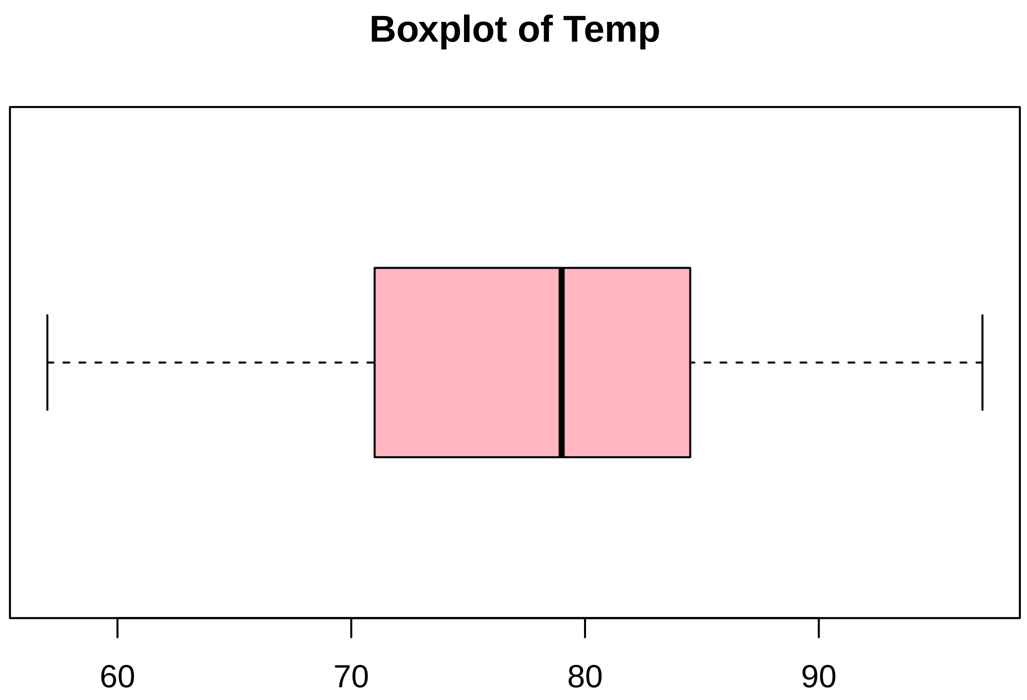
```r
# Load necessary library
library(ggplot2)

# Display boxplots of the variables
boxplot(data$Solar.R, main="Boxplot of Solar.R", col="lightblue", horizontal=TRUE)
```



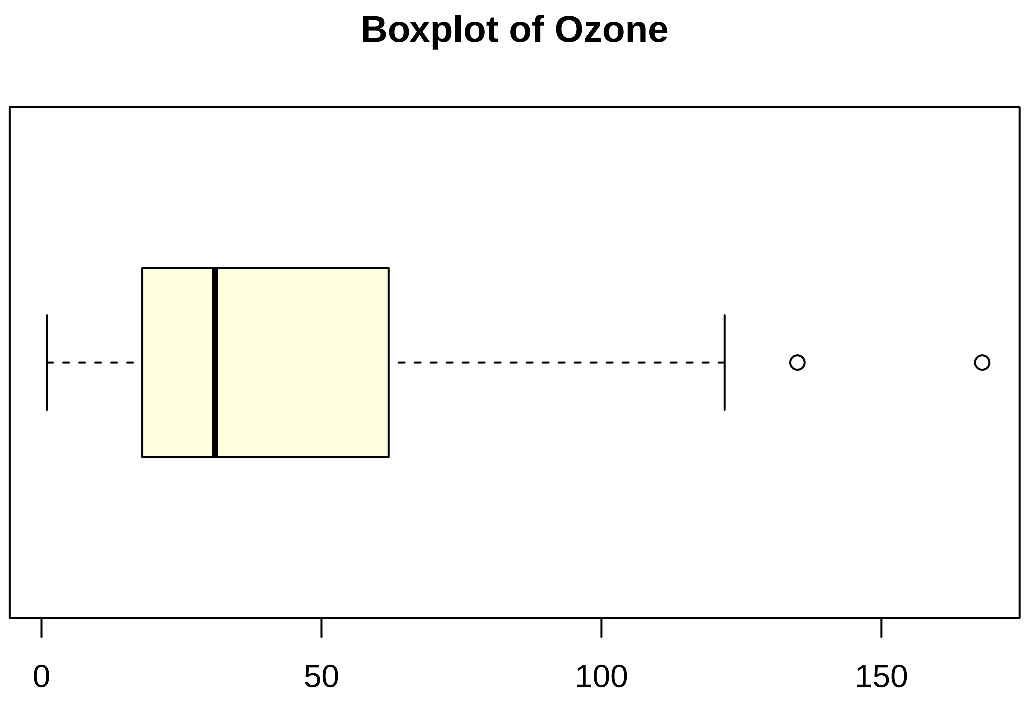
```r
boxplot(data$Wind, main="Boxplot of Wind", col="lightgreen", horizontal=TRUE)
```



```
boxplot(data$Temp, main="Boxplot of Temp", col="lightpink", horizontal=TRUE)
```



```r
boxplot(data$Ozone, main="Boxplot of Ozone", col="lightyellow", horizontal=TRUE)
```



Analysis of BoxPlot:

a) Ozone: The Ozone boxplot shows some outliers to the right (higher values). This indicates a right-skewed distribution. When a distribution is skewed, the mean is pulled in the direction of the skew (in this case, towards the higher values), making it less representative of the typical value. Therefore, the median is a better measure of the center for Ozone.

b) Solar.R: The Solar.R boxplot appears relatively symmetrical, with the median roughly in the center of the box and the whiskers fairly even. There's a possible minor outlier or two, but they don't seem to drastically skew the distribution. In such cases, the mean is a reasonable measure of the center.

c) Wind: Similar to Ozone, the Wind boxplot indicates a right-skewed distribution due to the presence of a few outliers on the right. The skew, while not as pronounced as Ozone, is enough to suggest that the median is a more appropriate measure of central tendency.

d) Temp: The Temp boxplot looks quite symmetrical, similar to Solar.R. The median is in the middle of the box, and the whiskers are relatively even. Therefore, the mean is a suitable measure of central tendency for Temp.

Question 5

a) Divide the AirQuality Dataset [n=100] into training and testing:

```r
set.seed(42)
sample_data <- data_norm[sample(nrow(data_norm), 100), ]
data <- na.omit(sample_data)

# Split the data into training (70%) and testing (30%)
n <- nrow(data)
train_indices <- sample(1:n, 70)  #70% for training
test_indices <- setdiff(1:n, train_indices)
print(n)
```

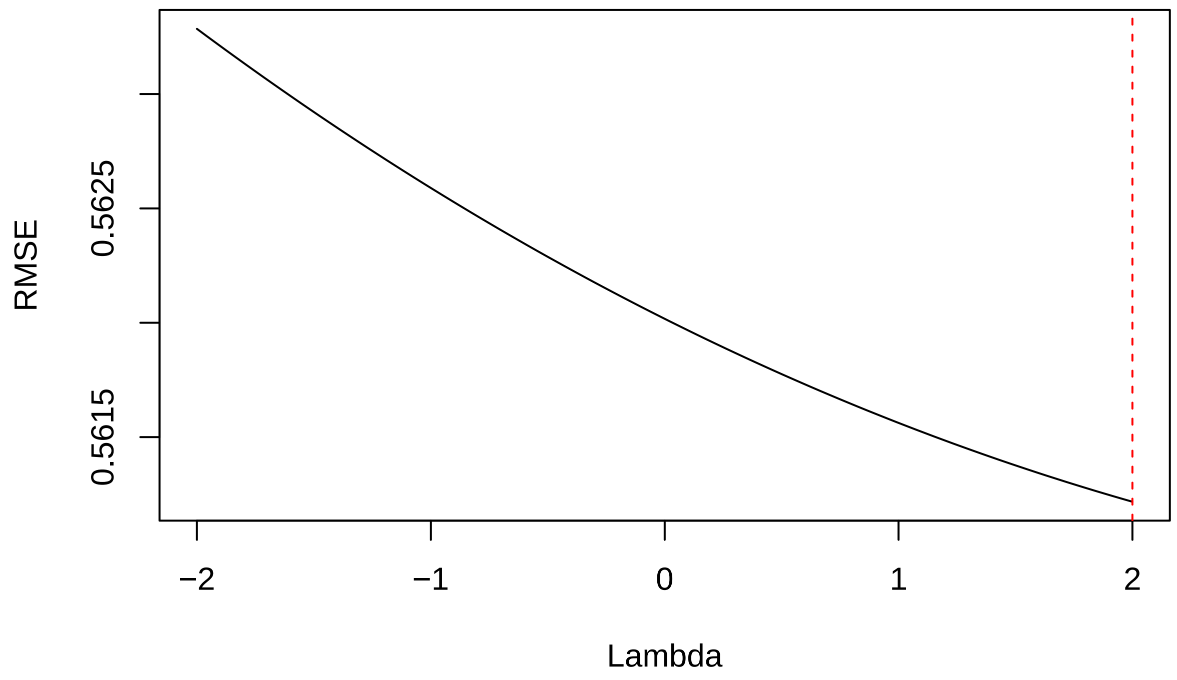
```
## [1] 100
```

b) Function to Calculate Coefficients of Regression Model and RMSE:

```r
ridge_regression <- function(X, y, lambda) {
  n <- nrow(X)
  p <- ncol(X)
  I <- diag(p)
  beta_ridge <- solve(t(X) %*% X + lambda * I) %*% t(X) %*% y
  return(beta_ridge)
}

calculate_rmse <- function(X_train, y_train, X_test, y_test, lambda){
  beta_r <- ridge_regression(X_train, y_train, lambda)
  y_pred <- X_test %*% beta_r
  rmse <- sqrt(mean((y_test - y_pred)^2))
  return(rmse)
}
```

c) Find optimal λ using RMSE:

```r
lambda_values <- seq(-2, 2, length.out = 100)
rmse_values <- numeric(length(lambda_values))

for (i in 1:length(lambda_values)) {
  X_train <- X_norm[train_indices, ]
  y_train <- y_norm[train_indices]
  X_test <- X_norm[test_indices, ]
  y_test <- y_norm[test_indices]


  rmse_values[i] <- calculate_rmse(X_train, y_train, X_test, y_test, lambda_values[i])

}

optimal_lambda <- lambda_values[which.min(rmse_values)]

plot(lambda_values, rmse_values, type = "l", xlab = "Lambda", ylab = "RMSE")
abline(v = optimal_lambda, col = "red", lty = 2)  # Show the optimal lambda
```



```
print(paste("Optimal Lambda:", optimal_lambda))
```

```
## [1] "Optimal Lambda: 2"
```

The plot of RMSE vs. lambda shows the impact of the regularization parameter on model performance. The optimal lambda value ‘2’ (indicated by the vertical red dashed line) is where the RMSE is minimized, indicating that, for this dataset split, this degree of regularization achieves the optimal balance between overfitting and underfitting on unseen data.

Question 6

a) Function to simulate from a standard normal distribution:

```
import math
import random
import matplotlib.pyplot as plt
import numpy as np

def box_muller(u1, u2):
    """Generates standard normal random variables using the Box-Muller transform.
    Args:
        u1: A random number between 0 and 1.
        u2: A random number between 0 and 1.

    Returns:
        A tuple containing two standard normal random variables.
    """
    x = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    y = np.sqrt(-2 * np.log(u1)) * np.sin(2 * np.pi * u2)

    return x, y
```

b) Generate 1000 observations:

```
n_samples = 1000
box_muller_samples = []
for _ in range(n_samples):
    u1 = random.random()
    u2 = random.random()
    x, y = box_muller(u1, u2)
    box_muller_samples.extend([x,y])
```

c) Plot histogram and compare:

```
plt.figure(figsize=(12, 6))
```

```
## <Figure size 1200x600 with 0 Axes>
```

```
plt.subplot(1, 2, 1)
```

```
## <AxesSubplot:>
```

```
plt.hist(box_muller_samples, bins=50, color='b', alpha=0.7, label="Box-Muller Samples")
```

```
## (array([  1.,   1.,   3.,   4.,   4.,   1.,   6.,   7.,  10.,   7.,  15.,
##         20.,  30.,  31.,  36.,  50.,  61.,  84.,  65., 101.,  86.,  91.,
##        108., 109., 101., 112.,  96.,  98.,  86.,  89.,  80.,  72.,  62.,
##         54.,  62.,  35.,  25.,  21.,  15.,  16.,  16.,   7.,   6.,   3.,
##          5.,   4.,   0.,   2.,   0.,   2.]), array([-3.473967  , -3.33397593, -3.19398486, -3.053993
##        -2.77401166, -2.63402059, -2.49402952, -2.35403845, -2.21404739,
##        -2.07405632, -1.93406525, -1.79407418, -1.65408311, -1.51409205,
##        -1.37410098, -1.23410991, -1.09411884, -0.95412778, -0.81413671,
##        -0.67414564, -0.53415457, -0.3941635 , -0.25417244, -0.11418137,
##         0.0258097 ,  0.16580077,  0.30579183,  0.4457829 ,  0.58577397,
##         0.72576504,  0.86575611,  1.00574717,  1.14573824,  1.28572931,
##         1.42572038,  1.56571144,  1.70570251,  1.84569358,  1.98568465,
##         2.12567572,  2.26566678,  2.40565785,  2.54564892,  2.68563999,
##         2.82563105,  2.96562212,  3.10561319,  3.24560426,  3.38559533,
##         3.52558639]), <BarContainer object of 50 artists>)
```

```
plt.title('Histogram of Box-Muller Observations')
```

```
## Text(0.5, 1.0, 'Histogram of Box-Muller Observations')
```

```
plt.xlabel('Value')
```

```
## Text(0.5, 0, 'Value')
```

```
plt.ylabel('Density')
```

```
## Text(0, 0.5, 'Density')
```

```
plt.subplot(1, 2, 2)
```

```
## <AxesSubplot:>
```

```
plt.hist(box_muller_samples, bins=50, alpha=0.5, label="Box-Muller Samples")
```

```
## (array([  1.,   1.,   3.,   4.,   4.,   1.,   6.,   7.,  10.,   7.,  15.,
##          20.,  30.,  31.,  36.,  50.,  61.,  84.,  65., 101.,  86.,  91.,
##         108., 109., 101., 112.,  96.,  98.,  86.,  89.,  80.,  72.,  62.,
##          54.,  62.,  35.,  25.,  21.,  15.,  16.,  16.,   7.,   6.,   3.,
##           5.,   4.,   0.,   2.,   0.,   2.]), array([-3.473967  , -3.33397593, -3.19398486, -3.053993
##         -2.77401166, -2.63402059, -2.49402952, -2.35403845, -2.21404739,
##         -2.07405632, -1.93406525, -1.79407418, -1.65408311, -1.51409205,
##         -1.37410098, -1.23410991, -1.09411884, -0.95412778, -0.81413671,
##         -0.67414564, -0.53415457, -0.3941635 , -0.25417244, -0.11418137,
##          0.0258097 ,  0.16580077,  0.30579183,  0.4457829 ,  0.58577397,
##          0.72576504,  0.86575611,  1.00574717,  1.14573824,  1.28572931,
##          1.42572038,  1.56571144,  1.70570251,  1.84569358,  1.98568465,
##          2.12567572,  2.26566678,  2.40565785,  2.54564892,  2.68563999,
##          2.82563105,  2.96562212,  3.10561319,  3.24560426,  3.38559533,
##          3.52558639]), <BarContainer object of 50 artists>)
```

```
plt.hist(np.random.randn(1000), bins=50, alpha=0.5, label='True Normal')
```

```
## (array([ 1.,  1.,  0.,  0.,  0.,  0.,  0.,  3.,  2.,  7.,  2.,  5.,  9.,
##          8.,  8., 19., 20., 18., 25., 35., 32., 46., 42., 50., 43., 61.,
##         51., 45., 57., 50., 50., 43., 51., 34., 30., 26., 25., 19., 12.,
##         13., 16., 10.,  6.,  8.,  5.,  7.,  3.,  1.,  0.,  1.]), array([-3.70919817, -3.57297832, -3.
##         -3.0280989 , -2.89187905, -2.75565919, -2.61943934, -2.48321948,
##         -2.34699963, -2.21077977, -2.07455992, -1.93834006, -1.80212021,
##         -1.66590036, -1.5296805 , -1.39346065, -1.25724079, -1.12102094,
##         -0.98480108, -0.84858123, -0.71236137, -0.57614152, -0.43992166,
##         -0.30370181, -0.16748195, -0.0312621 ,  0.10495775,  0.24117761,
##          0.37739746,  0.51361732,  0.64983717,  0.78605703,  0.92227688,
##          1.05849674,  1.19471659,  1.33093645,  1.4671563 ,  1.60337616,
##          1.73959601,  1.87581586,  2.01203572,  2.14825557,  2.28447543,
##          2.42069528,  2.55691514,  2.69313499,  2.82935485,  2.9655747 ,
##          3.10179456]), <BarContainer object of 50 artists>)
```

```
plt.xlabel("Value")
```

```
## Text(0.5, 0, 'Value')
```

```
plt.ylabel("Density")
```

```
## Text(0, 0.5, 'Density')
```

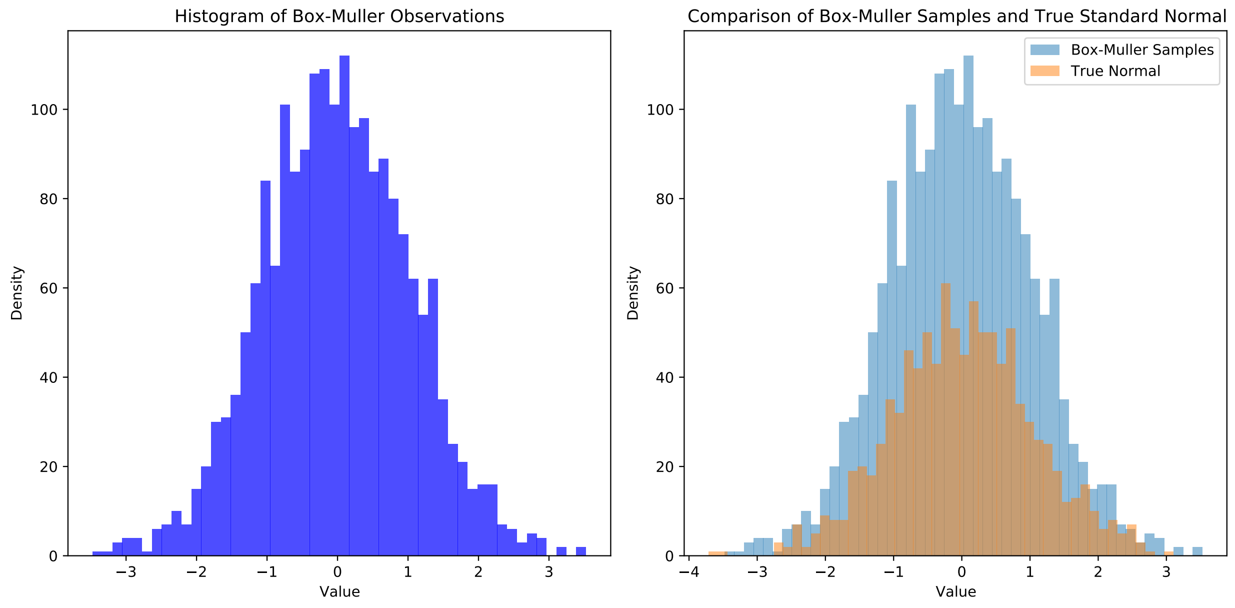
```
plt.title("Comparison of Box-Muller Samples and True Standard Normal")
```

```
## Text(0.5, 1.0, 'Comparison of Box-Muller Samples and True Standard Normal')
```

```
plt.legend()
```

```
## <matplotlib.legend.Legend object at 0x0000017277C43970>
```

```
plt.tight_layout()
plt.show()
```



The histogram of the Box-Muller generated samples closely resembles the histogram of the true standard normal distribution. This visually confirms that the Box-Muller transform effectively generates random numbers that follow a standard normal distribution.

The overlapping histograms indicate that the generated samples mimic the bell-shaped curve, centered around zero, characteristic of a standard normal distribution.
