M.Sc. (Five Year Integrated) in Computer Science (Artificial Intelligence & Data Science)

Fifth Semester

Laboratory Record

21-805-0506: R Programming Lab

Submitted in partial fulfillment
of the requirements for the award of degree in
Master of Science (Five Year Integrated)
in Computer Science (Artificial Intelligence & Data Science) of
Cochin University of Science and Technology (CUSAT)
Kochi

Submitted by

AMJAD K P
(80522002)

DEPARTMENT OF COMPUTER SCIENCE
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)
KOCHI-682022

DECEMBER 2024

DEPARTMENT OF COMPUTER SCIENCE

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)

KOCHI, KERALA-682022

This is to certify that the software laboratory record for ***21-805-0506: R Programming Lab*** *is a record of work carried out by* ***AMJAD K P (80522002)****, in partial fulfillment of the requirements for the award of degree in* ***Master of Science (Five Year Integrated)*** *in* ***Computer Science (Artificial Intelligence & Data Science)*** *of Cochin University of Science and Technology (CUSAT), Kochi. The lab record has been approved as it satisfies the academic requirements in respect of the fourth semester laboratory prescribed for the Master of Science (Five Year Integrated) in Computer Science degree.*

Faculty Member in-charge

Dr.Remya K Sasi
Assistant Professor
Department of Computer Science
CUSAT

Dr. Madhu S. Nair
Professor and Head
Department of Computer Science
CUSAT

Table of Content

WORD ANALYSIS

AIM

Develop a program to read a paragraph of text and perform the following tasks: a. Count the total number of words. b. Calculate the average word length. c. Identify and print the longest word. d. Replace all occurrences of a specific word with another word of your choice.

PROGRAM

```
para <- function() {
  cat("Enter your paragraph:\n")
  paragraph <- readline()

  words <- unlist(strsplit(paragraph, "\\s+"))

  # (a)
  words_number <- length(words)
  cat("\nThe total number of words =", words_number, "\n")

  # (b)
  word_len <- nchar(words)
  avg_word_len <- mean(word_len)
  cat("Average word length =", avg_word_len, "\n")

  # (c)
  longest_len <- max(word_len)
  longest_word <- words[word_len == longest_len]
  # longest_word <- longest_words[1]
  # In case there are multiple words with the same length
  cat("The longest word is:", longest_word, "\n")

  # (d)
  cat("Enter the word to be replaced: ")
  replace <- readline()
  cat("Enter the word to replace with: ")
  new <- readline()

  modified_para <- modified(replace, new, paragraph)
  cat("The modified paragraph is:\n", modified_para, "\n")
}
```

```
modified <- function(replace, new, paragraph) {
  words <- unlist(strsplit(paragraph, "\\s+"))

  mod <- sapply(words, function(word) {
    if (word == replace) {
      return(new)
    } else {
      return(word)
    }
  })

  modified_para <- paste(mod, collapse = " ")
  return(modified_para)
}

para()
```

OUTPUT

```
Enter your paragraph:
The park was the perfect place for peace . In the peace of the early morning, the peace of the tall trees swaying in the
 breeze brought peace to her mind. Peace , she thought, was all she needed, and this peaceful spot was her sanctuary of
 peace .

The total number of words = 49
Average word length = 4.061224
The longest word is: sanctuary
Enter the word to be replaced:
peace
Enter the word to replace with:
quite
The modified paragraph is:
 The park was the perfect place for quite . In the quite of the early morning, the quite of the tall trees swaying in th
e breeze brought quite to her mind. Peace , she thought, was all she needed, and this peaceful spot was her sanctuary of
 quite .
```

STRING ENCRYPTION

AIM

Write a program that reads a sentence from the user and encrypts it using a simple Caesar cipher. The user can specify the shift value. Implement the encryption such that only alphabetic characters are shifted, while maintaining their case.

PROGRAM

```
# Caesar Cipher

encrypting <- function(){

  sentence <- readline(prompt = "Enter the sentence: ")

  shift <- as.integer(readline(prompt = "Enter the shift key for
  encryption: "))

  lower <- "abcdefghijklmnopqrstuvwxyz"
  upper <- "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

  encrypted <- ""

  chars <- unlist(strsplit(sentence, NULL))

  for (char in chars){
    # lower case
    if (char %in% unlist(strsplit(lower, NULL))){
      pos <- which(unlist(strsplit(lower, NULL)) == char)
      new_pos <- (pos + shift - 1) %% 26 + 1
      encrypted <- paste0(encrypted, substring(lower, new_pos, new_pos))
    }

    else if (char %in% unlist(strsplit(upper, NULL))){
      pos <-  which(unlist(strsplit(upper, NULL)) == char)
      new_pos <- (pos + shift - 1) %% 26 + 1
      encrypted <- paste0(encrypted, substring(upper, new_pos, new_pos))
    }

    else{
      encrypted <- paste0(encrypted, char)
```

```
    }
  }
  cat("\nThe encrypted sentence is: ", encrypted)
}

encrypted <- encrypting()
```

OUTPUT

```
> encrypted <- encrypting()
Enter the sentence: walking dead
Enter the shift key for
  encryption:  5

The encrypted sentence is:  bfqpnsl ijfi
>
```

DATA VALIDATION AND USER INPUT

AIM

Develop a program to read student records with their names, ages, and grades. Implement validation checks: a. Ensure age is a positive integer. b. Ensure grade is a valid letter grade (A, B, C, D, F). c. Calculate and display the average age of students with valid records.

PROGRAM

```
students <- function(){
  number_students <- as.integer(readline(prompt = "Enter the number of
  students: "))

  ages <- numeric()
  grades <- character()

  grade_list <- c('A', 'B', 'C', 'D', 'F')
  for (i in 1:number_students){
    name <- readline(prompt = sprintf("Enter name of
    student %d: ", i))

    while (TRUE) {
      age <- as.integer(readline(prompt = sprintf("Enter the
      age of student %d: ", i)))
      if (age > 0 ){
        ages <- c(ages, age)
        break
      }
      else{
        cat("Invalid age, please enter a positive integer..")
      }
    }

    while (TRUE) {
      grade <- readline(prompt = sprintf("Enter the grade of
      student %d: ", i))
      if (grade %in% grade_list){
        grades <- c(grades, grade)
        break
      }
      else{
```

```
        cat("Invalid grade, please enter A, B, C, D or F as
        the grade..")
      }
    }
  }

  avg_age = mean(ages)
  cat("The average age is: ", avg_age)
}

students()
```

OUTPUT

```
Enter the number of students: 2
Enter name of student 1: Fe
Enter the age of student 1: 0
Invalid age, please enter a positive integer..
Enter the age of student 1: 9
Enter the grade of student 1: a
Invalid grade, please enter A, B, C, D or F as the grade..
Enter the grade of student 1: A
Enter name of student 2: SO4
Enter the age of student 2: -3
Invalid age, please enter a positive integer..
Enter the age of student 2: 10
Enter the grade of student 2: D
The average age is:  9.5
```

PASSWORD GENERATOR

AIM

Write a program to generate a random password for a user. The password should include a mix of uppercase letters, lowercase letters, digits, and special characters. Allow the user to specify the length of the password.

PROGRAM

```
# password generator

password <- function(){

  len <- as.integer(readline(prompt = "Enter the password length: "))

  upper <- LETTERS
  lower <- letters
  digits <- 0:9
  special <- c('!', '@', '#', '$', '%', '^', '&', '*', '(',
  ')', '-', '_', '=', '+', '~')

  full <- c(upper, lower, digits, special)

  pass <- sample(full, len, replace = TRUE)
  pass <- paste0(pass, collapse="")
  cat(sprintf("The generated password is: %s", pass))
}

password()
```

OUTPUT

```
> password()
Enter the password length: 11
The generated password is: XyfvvkDj++m
>
```

SERIES SUMMATION

AIM

Develop a program to calculate the sum of the series: 1 - 2/3 + 3/5 - 4/7 + ... up to a specified number of terms. Allow the user to input the number of terms in the series.

PROGRAM

```
# series summation 1 - 2/3 + 3/5 - 4/7 + ..
series <- function(){
  number_terms <- as.integer(readline(prompt = "Enter the number
  of terms: "))
  sum_series <- 0
  term_series <- vector('character', number_terms)

  for (i in 2:number_terms){
    numerator <- i
    denominator <- 2*i - 1
    term_ <- numerator/denominator
    if (i %% 2 == 0){
      term_ <- -term_
    }
    sum_series <- sum_series + term_
    if (i %% 2 == 0) {
      term_series[i] <- sprintf(" - %d/%d", numerator, denominator)
    } else {
      term_series[i] <- sprintf(" + %d/%d", numerator, denominator)
    }
  }
  series_string <- paste0(term_series, collapse = "")
  cat("The series is: 1 ", series_string, "\n")
  cat("The sum of the series is: ", sum_series + 1, "\n")
}
series()
```

OUTPUT

```
Enter the number of terms: 8
The series is: 1   - 2/3 + 3/5 - 4/7 + 5/9 - 6/11 + 7/13 - 8/15
The sum of the series is:  0.377134
```

PRIME NUMBER CHECKER

AIM

Write a program to check whether a given number is prime or not. Implement this using both loops and functions. Additionally, allow the user to input a range and identify all prime numbers within that range.

PROGRAM

```
prime <- function(n) {
  if (n <= 1) {
    return(FALSE)
  }
  if (n == 2){
    return(TRUE)
  }
  for (i in 2:floor(sqrt(n))) {
    if (n %% i == 0) {
      return(FALSE)
    }
  }
  return(TRUE)
}

main <- function(){
  while (TRUE) {
    cat("\n\n1. Check if the given number is prime.
    \n2. Print all prime numbers in a given range. \n3. Quit.")
    choice <- as.integer(readline(prompt = "Enter your choice: "))
    if (choice == 1){
      n <- as.integer(readline(prompt = "Enter the integer to check: "))
      if (prime(n))
        cat(n, "is a prime number.")
      else
        cat(n, "is not a prime number.")
    }
    else if (choice == 2){
      initial <- as.integer(readline(prompt = "Enter the
      initial number for the range: "))
      end <- as.integer(readline(prompt = "Enter the ending
      number for the range: "))
```

```
      p <- c()
      for ( i in initial:end){
        if (prime(i))
          p <- c(p, i)
      }
      if (length(p) == 0)
        cat("No prime numbers exist in the given range..")
      else
        cat("The prime numbers in the given range are:\n", p)
    }
    else if (choice == 3)
      break
    else
      cat("Please enter a valid option.")
  }
}
main()
```

OUTPUT

```
1. Check if the given number is prime.
2. Print all prime numbers in a given range.
3. Quit.
Enter your choice: 1
Enter the integer to check: 56
56 is not a prime number.

1. Check if the given number is prime.
2. Print all prime numbers in a given range.
3. Quit.
Enter your choice: 1
Enter the integer to check: 13
13 is a prime number.

1. Check if the given number is prime.
2. Print all prime numbers in a given range.
3. Quit.
Enter your choice: 2
Enter the initial number for the range: 100
Enter the ending number for the range: 180
The prime numbers in the given range are:
 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179

1. Check if the given number is prime.
2. Print all prime numbers in a given range.
3. Quit.
Enter your choice: 3
>
```

FIBONACCI SERIES WITH A TWIST

AIM

Develop a program to generate the Fibonacci series, but with a twist. Allow the user to input the number of terms and generate the series where each term is the sum of the last three terms.

PROGRAM

```
#fibonacci seris with a twist - add last 3 numbers instead of 2
fibonacci_twist <- function(n){
  if (n <= 0){
    cat("Please enter a positive integer..")
    n <- as.integer(readline(prompt = "Enter the number of terms: "))
  }

  if (n >= 1)
    series <- c(0)
  if (n >= 2)
    series <- c(series, 1)
  if (n >= 3)
    series <- c(series, 2)

  for (i in 4:n){
    next_term <- series[i-1] + series[i-2] + series[i-3]
    series <- c(series, next_term)
  }

  cat("The series is: \n", paste(series, collapse = ", "))
}

main <- function(){
  n <- as.integer(readline(prompt = "Enter the number of terms: "))
  fibonacci_twist(n)
}

main()
```

OUTPUT

```
Enter the number of terms: 15
The series is:
 0, 1, 2, 3, 6, 11, 20, 37, 68, 125, 230, 423, 778, 1431, 2632
>
```

PALINDROME CHECKER

AIM

Write a program that reads a string and checks if it's a palindrome. A palindrome is a string that reads the same forwards and backwards, ignoring spaces and punctuation.

PROGRAM

```
#palindrome checker

palindrome <- function(){
  string <- readline(prompt = "Enter the string: ")
  string <- tolower(gsub("[^a-zA-Z]", "", string))
  string_letters <- strsplit(string, NULL)[[1]]
  string_n <- length(string_letters)
  i <- 1
  j <- string_n
  while (i < j) {
    if (string_letters[i] != string_letters[j]) {
      return(FALSE)
    }
    i <- i + 1
    j <- j - 1
  }
  return(TRUE)
}

main <- function(){
  bool <- palindrome()
  if (bool)
    cat("The given string is a palindrome.")
  else
    cat("The given string is not a palindrome.")
}
main()
```

OUTPUT

```
Enter the string: Was it a car or a cat I saw?
The given string is a palindrome.
>
```

DATA COMPRESSION

AIM

Design a program to read a string and compress it using run-length encoding. In run-length encoding, consecutive repeated characters are replaced with a single character followed by the count of occurrences.

PROGRAM

```
#run-length encoding

run_length_encoding <- function(){
  string <- readline(prompt = "Enter the string: ")

  string_letters <- unlist(strsplit(string, NULL))
  n <- length(string_letters)

  i = 1
  encoded <- ''
  while (i <= n) {
    count = 1
    while (i < n && string_letters[i] == string_letters[i+1]) {
      count <- count + 1
      i <- i + 1
    }

    encoded <- paste(encoded, string_letters[i],count, sep = "")
    i <- i + 1
  }
  cat("The encoded string is: ", encoded)
}

run_length_encoding()
```

OUTPUT

```
Enter the string: aaabbc
The encoded string is:  a3b2c1
```

DATA REVERSAL

AIM

Write a program to reverse the order of elements in a given list. Implement this without using any built-in functions or loops

PROGRAM

```
#data reversal wihtout using functions or loops

reversal <- function(input){

  if (length(input) <= 1)
    return(input)

  return (c(reversal(input[-1]), input[1]))
}

main <- function(){

  input <- readline(prompt = "Enter space-separated elements: ")
  input <- unlist(strsplit(input, " "))

  reversed <- reversal(input)

  cat("The reversed data is: ", paste(reversed, collapse = " "))
}

main()
```

OUTPUT

```
Enter space-separated elements: 1 2 3 4 5 hehe cat
The reversed data is:  cat hehe 5 4 3 2 1
>
```

SCATTERPLOT IRIS

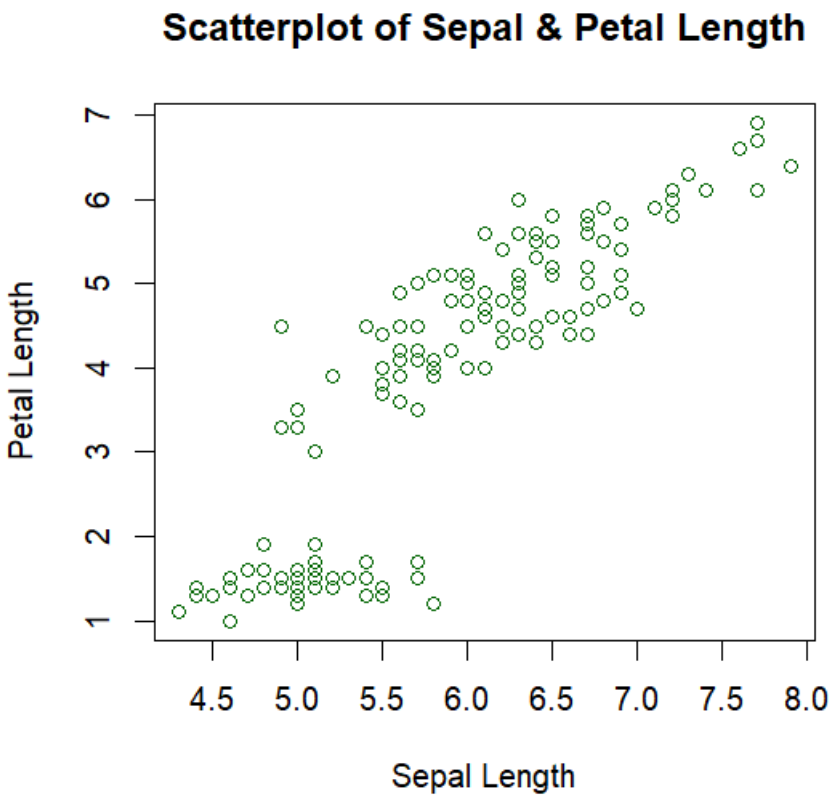
AIM

Create a scatterplot of the Sepal.Length and Petal.Length variables in the iris dataset using the plot function? Add appropriate labels and title to the plot. Save the plot as a high-resolution image file.

PROGRAM

```
# scatterplot of the Sepal.Length and Petal.Length in iris
dataset using the plot function.

data(iris)
png(filename = "scatterplot_hd.png", width = 8, height = 6,
units = "in", res = 300)
plot(iris$Sepal.Length, iris$Petal.Length, xlab = 'Sepal Length',
ylab = 'Petal Length',
     main = 'Scatterplot of Sepal vs Petal Length', col = 'darkgreen')
dev.off()
```

OUTPUT



SCATTERPLOT MTCARS

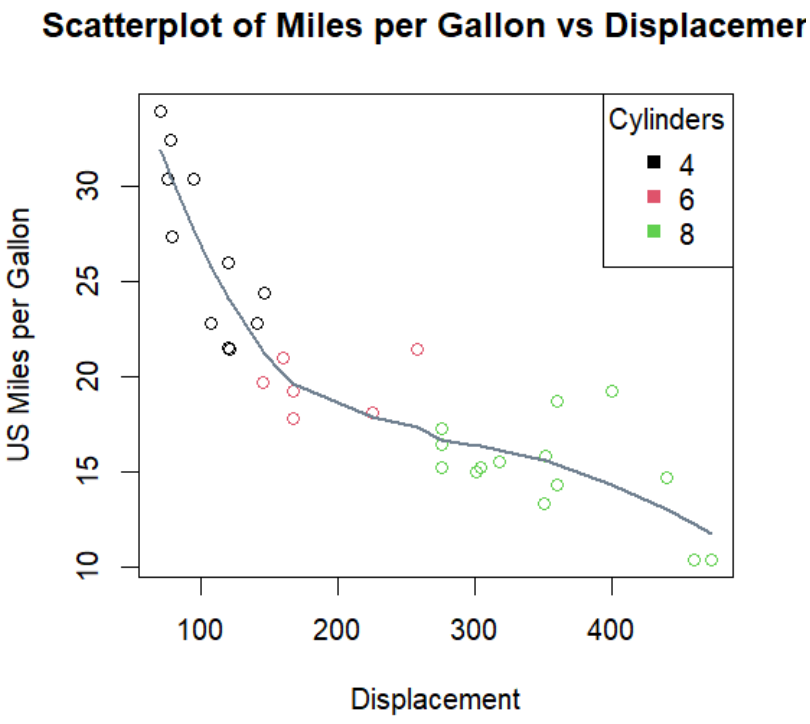
AIM

Create a scatterplot of the mpg and disp variables in the mtcars dataset. Use different colors to represent the cyl variable and add a smooth line to show the trend. Add appropriate labels, title, and legend to the plot

PROGRAM

```
data(mtcars)
cyl_color <- as.factor(mtcars$cyl)
plot(mtcars$disp, mtcars$mpg, ylab = 'US Miles per Gallon',
xlab = 'Displacement', main = 'Scatterplot of Miles per Gallon
vs Displacement', col = cyl_color)
loess_fit <- loess(mpg ~ disp, data = mtcars)
disp_sorted <- sort(mtcars$disp)
lines(disp_sorted, predict(loess_fit, newdata =
data.frame(disp = disp_sorted)),
      col = 'slategrey', lwd = 2)
legend("topright", legend = levels(cyl_color),
       col = 1:length(levels(cyl_color)),
       pch = 15, title = "Cylinders")
```

OUTPUT



BAR PLOT

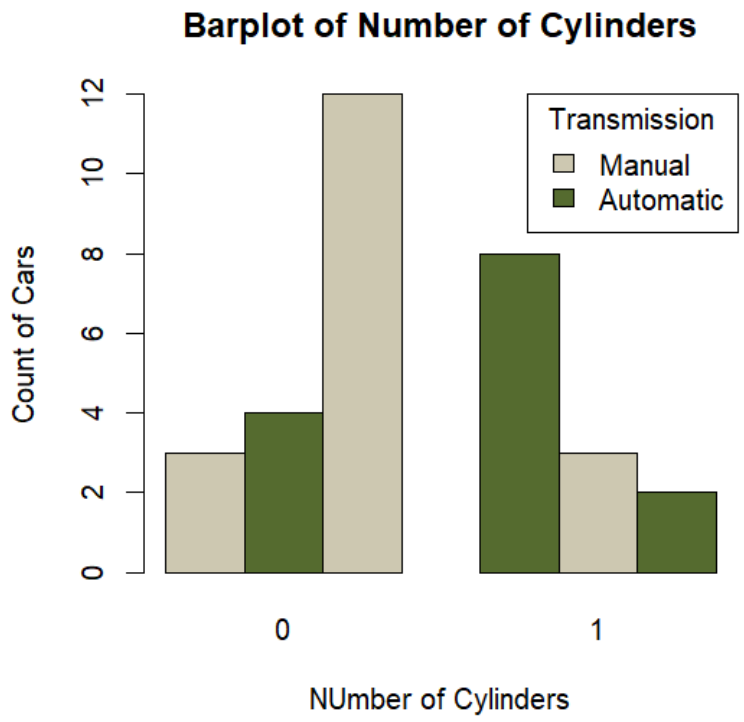
AIM

Create a bar plot of the number of cylinders (cyl) in the mtcars dataset. Use different colors to represent the transmission type (am). Add appropriate title, labels, and legend to the plot.

PROGRAM

```
data(mtcars)

cyl <- as.factor(mtcars$cyl)
am <- as.factor(mtcars$am)
bar_data <- table(cyl, am)
bar_col <- c('cornsilk3', 'darkolivegreen')
barplot(bar_data, col = bar_col, beside = TRUE,
        xlab = 'NUmber of Cylinders', ylab = 'Count of Cars',
        main = 'Barplot of Number of Cylinders',
        legend = TRUE,
        args.legend = list(title = 'Transmission',
                           x = 'topright',
                           legend = c('Manual', 'Automatic')
                           ))
```

OUTPUT



HISTOGRAM OF MPG MTCARS

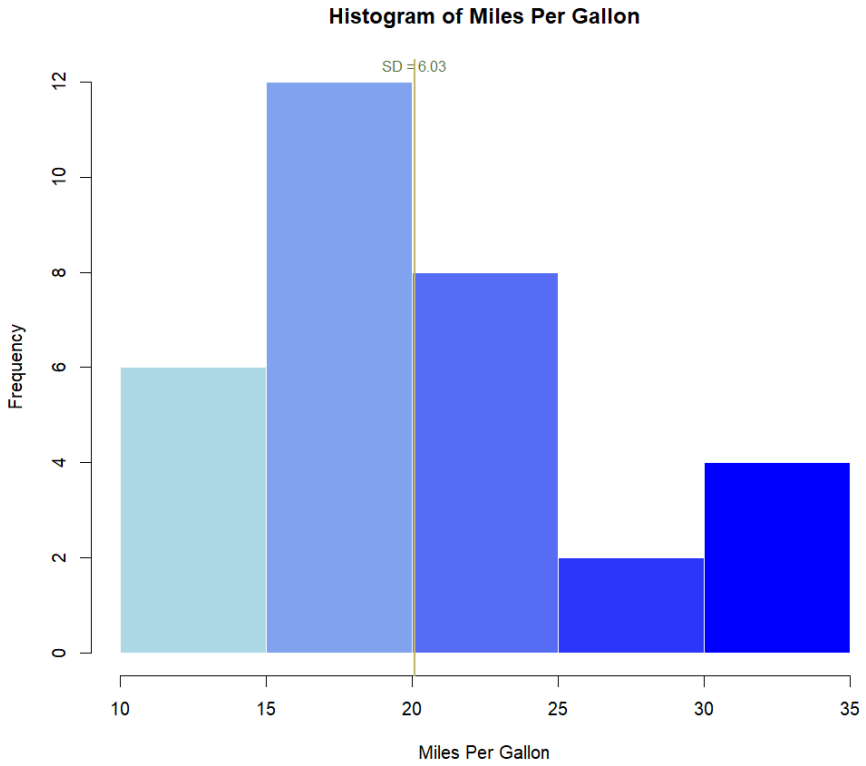
AIM

Create a histogram of the miles per gallon (mpg) in the mtcars dataset. Use different shades of blue to represent the frequency of each bin. Add appropriate title and labels to the plot. Calculate and display the mean and standard deviation of mpg on the plot.

PROGRAM

```
data(mtcars)

mean_mpg <- mean(mtcars$mpg)
sd_mpg <- sd(mtcars$mpg)

hist(mtcars$mpg, breaks = 5, col = colorRampPalette(c('lightblue',
    'blue'))(5), main = 'Histogram of Miles Per Gallon', xlab = 'Miles
    Per Gallon', ylab = 'Frequency', border = 'white')

text(x = mean_mpg, y = max(hist(mtcars$mpg, plot = FALSE)$counts),
    labels = paste('Mean =', round(mean_mpg, 2), '\nSD =',
    round(sd_mpg, 2)), pos = 3, cex = 0.8, col = 'darkolivegreen')

abline(v = mean_mpg, lwd = 2, col = 'darkkhaki')
```

OUTPUT



BOX PLOT

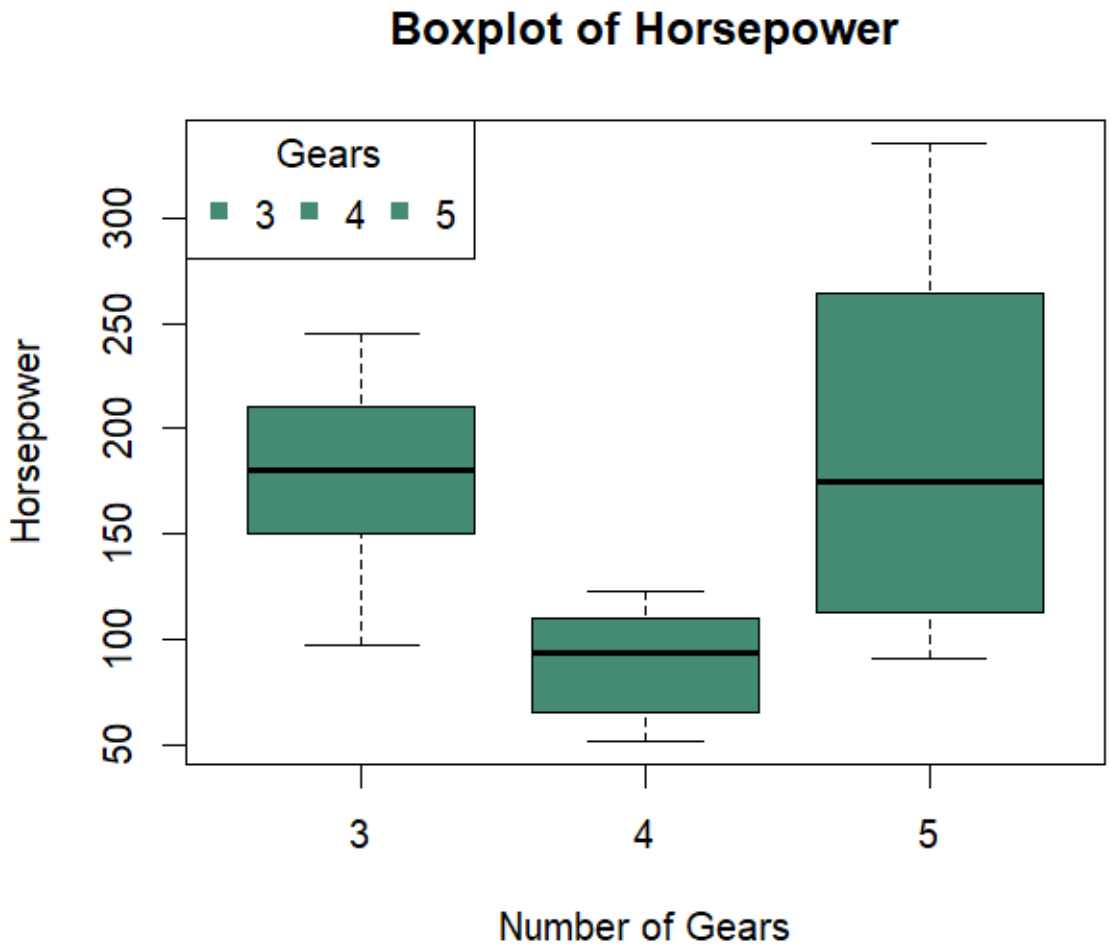
AIM

Create a box plot of the horsepower (hp) in the mtcars dataset. Use different shapes to represent the number of gears (gear). Add appropriate title, labels, and legend to the plot. Identify and label any outliers on the plot.

PROGRAM

```
data(mtcars)

gears <- as.factor(mtcars$gear)

boxplot(hp ~ gear, data = mtcars,
        main = 'Boxplot of Horsepower', pch = 21, cex = 3,
        ylab = 'Horsepower', xlab = 'Number of Gears', col = 'aquamarine4')

outliers <- boxplot.stats(mtcars$hp)$out

for (i in levels(gears)){
  g <- mtcars[mtcars$gear == i, ]   #get the cars if the gear
  number is the same as i..3, 4, or 5..
  outlier_list <- g$hp[g$hp %in% outliers]   # get the gears
  ' hp if its in the outliers vecotr

  if (length(outlier_list > 0)){
    text(x = as.numeric(i),
         y = outlier_list,
         labels = outlier_list,
         pos = 4, col = 'azure')
  }
}

legend('topleft', legend = levels(gears), horiz = TRUE,
       title = 'Gears', col = 'aquamarine4', pch = 15)
```

OUTPUT


Boxplot of Horsepower
Gears
3
4
5
Horsepower
50
100
150
200
250
300
3
4
5
Number of Gears

SCATTERPLOT OF DISP MTCARS

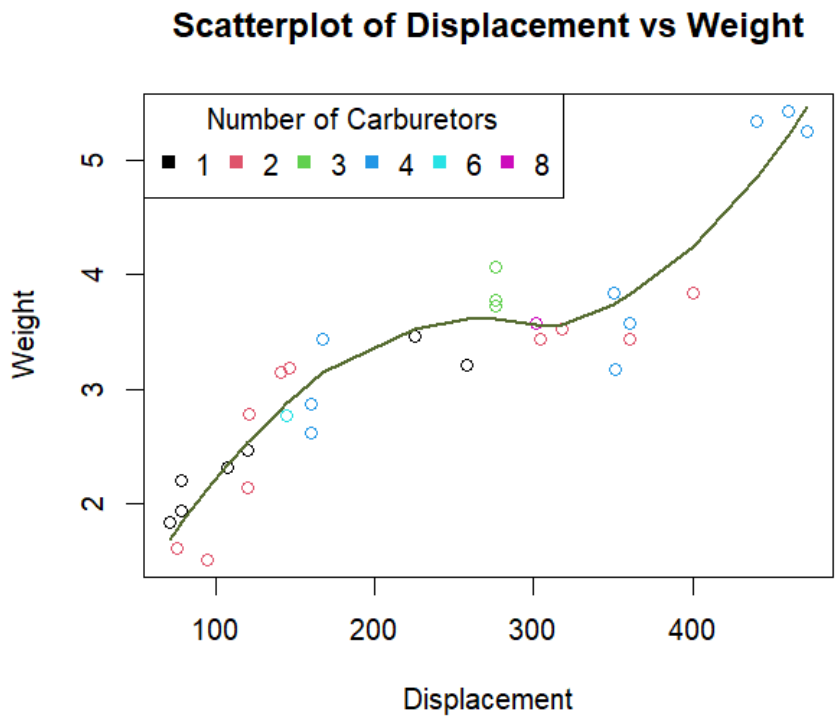
AIM

Create a scatter plot of the displacement (disp) versus the weight (wt) in the mtcars dataset. Use different colors and sizes to represent the number of carburetors (carb). Add appropriate title, labels, and legend to the plot. Add a smooth line to show the trend of the relationship

PROGRAM

```
data(mtcars)

carbs <- as.factor(mtcars$carb)
plot(mtcars$disp, mtcars$wt, col = carbs, main = 'Scatterplot of
Displacement vs Weight', xlab = 'Displacement', ylab = 'Weight', pch = 21 )

legend(x = 'topleft', legend = levels(carbs), horiz = TRUE, col =
1:length(levels(carbs)) , pch = 15, title ='Number of Carburetors')

loess_fit <- loess(wt ~ disp, data = mtcars)
disp_sorted <- sort(mtcars$disp)
lines(disp_sorted, predict(loess_fit, newdata =
    data.frame(disp = disp_sorted)), lwd = 2, col = 'darkolivegreen')
```

OUTPUT



TIME SERIES

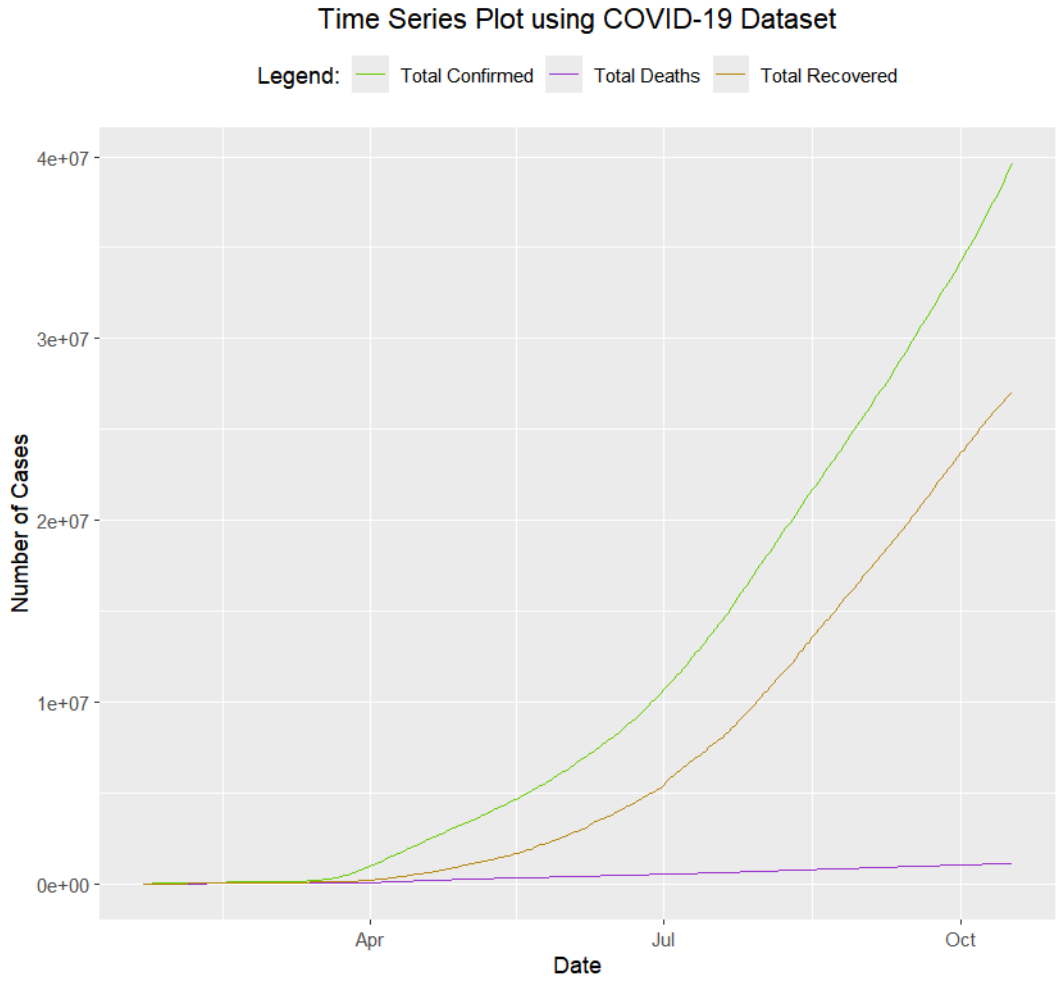
AIM

Develop an R program to create a time series plot using real-world data. (https://www.kaggle.com/datasets/niketchauhan/covid-19-time-series-data)

PROGRAM

```
# create a time series plot using the covid 19 dataset - real world dataset

library(ggplot2)
library(dplyr)

path <- "E:/R/Lab Cycle 2 - Sem 5/time-series-19-covid-combined.csv"
data <- read.csv(path)
data$Date <- as.Date(data$Date)

data_summary <- data %>% group_by(Date) %>%
                summarise(Total_Confirmed = sum(Confirmed, na.rm = TRUE),
                          Total_Death = sum(Deaths, na.rm = TRUE),
                          Total_Recovered = sum(Recovered, na.rm = TRUE))

ggplot(data = data_summary, aes(x = Date)) +
  geom_line(aes(y = Total_Confirmed, col = 'Total Confirmed')) +
  geom_line(aes(y = Total_Death, col = 'Total Deaths')) +
  geom_line(aes(y = Total_Recovered, col = 'Total Recovered')) +
  scale_color_manual(values = c('Total Confirmed' = 'chartreuse3',
                                'Total Deaths' = 'darkorchid',
                                'Total Recovered' = 'darkgoldenrod')) +
  labs(title = 'Time Series Plot using COVID-19 Dataset',
       x = 'Date', y = 'Number of Cases',
       color = 'Legend:') +
  theme(legend.position = 'top', plot.title = element_text(hjust = 0.5))
```

OUTPUT


Time Series Plot using COVID-19 Dataset
Legend:
Total Confirmed
Total Deaths
Total Recovered
4e+07
3e+07
2e+07
1e+07
0e+00
Number of Cases
Apr
Jul
Oct
Date

TITANIC EDA

AIM

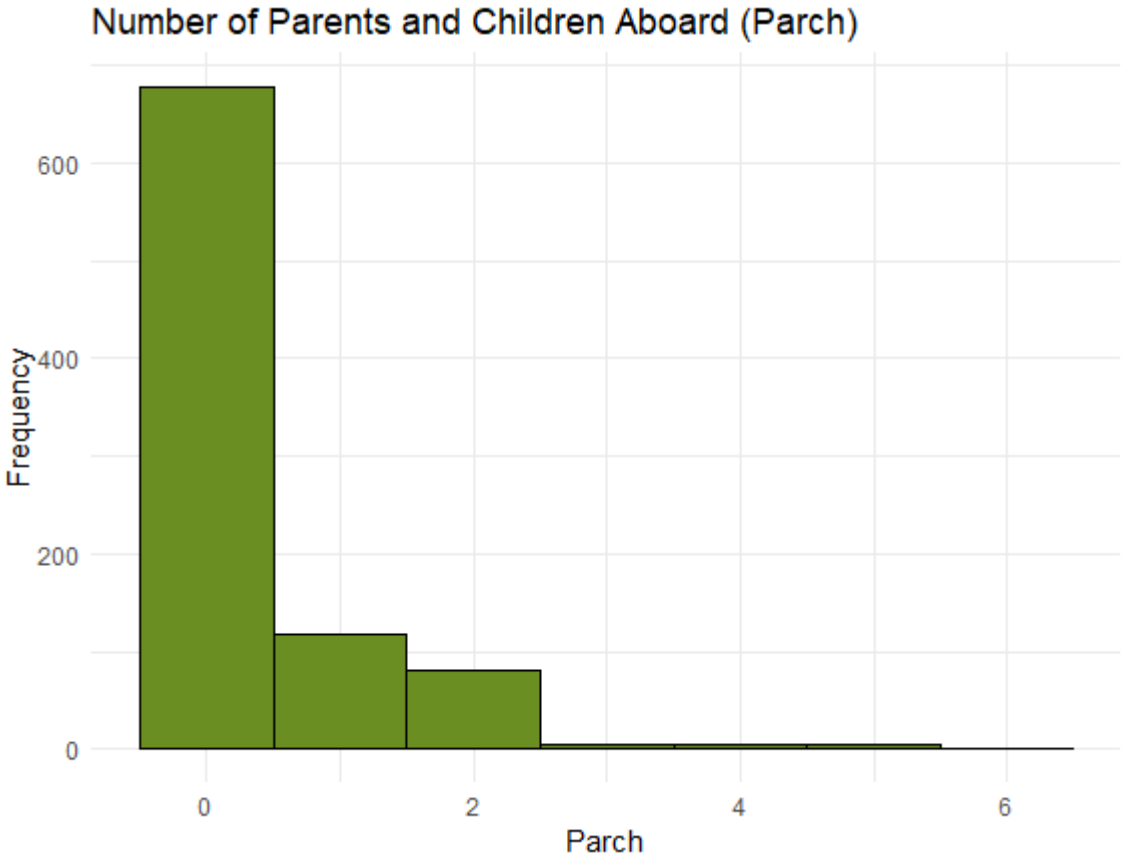
Perform EDA on "Titanic Dataset". You are given the Titanic dataset, which contains information about passengers on the Titanic, including their survival status, age, class, and gender. a)plot the histogram of Number of parents and children of the passenger aboard(parch). b)Perform a detailed EDA, including advanced statistical analysis, to explore factors influencing survival rates. c)Create a customized box plot to visualize the age distribution of survivors and non-survivors.

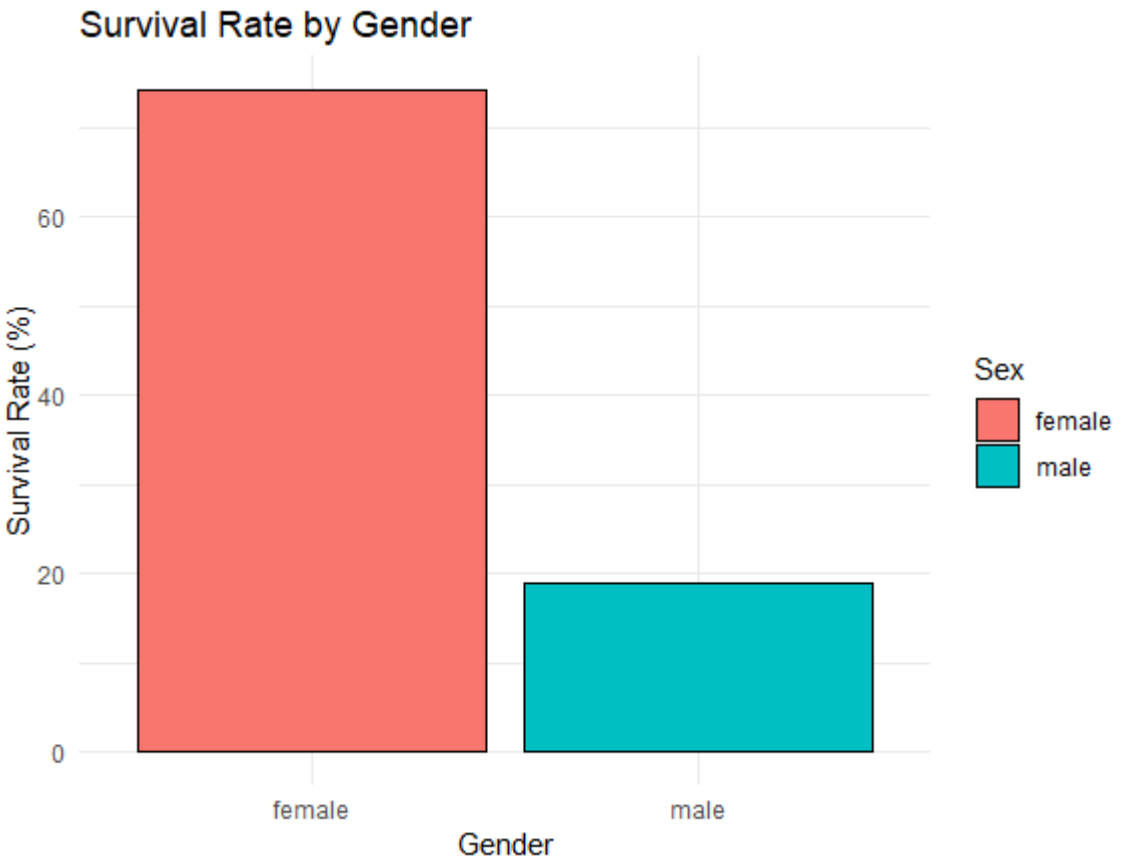
PROGRAM

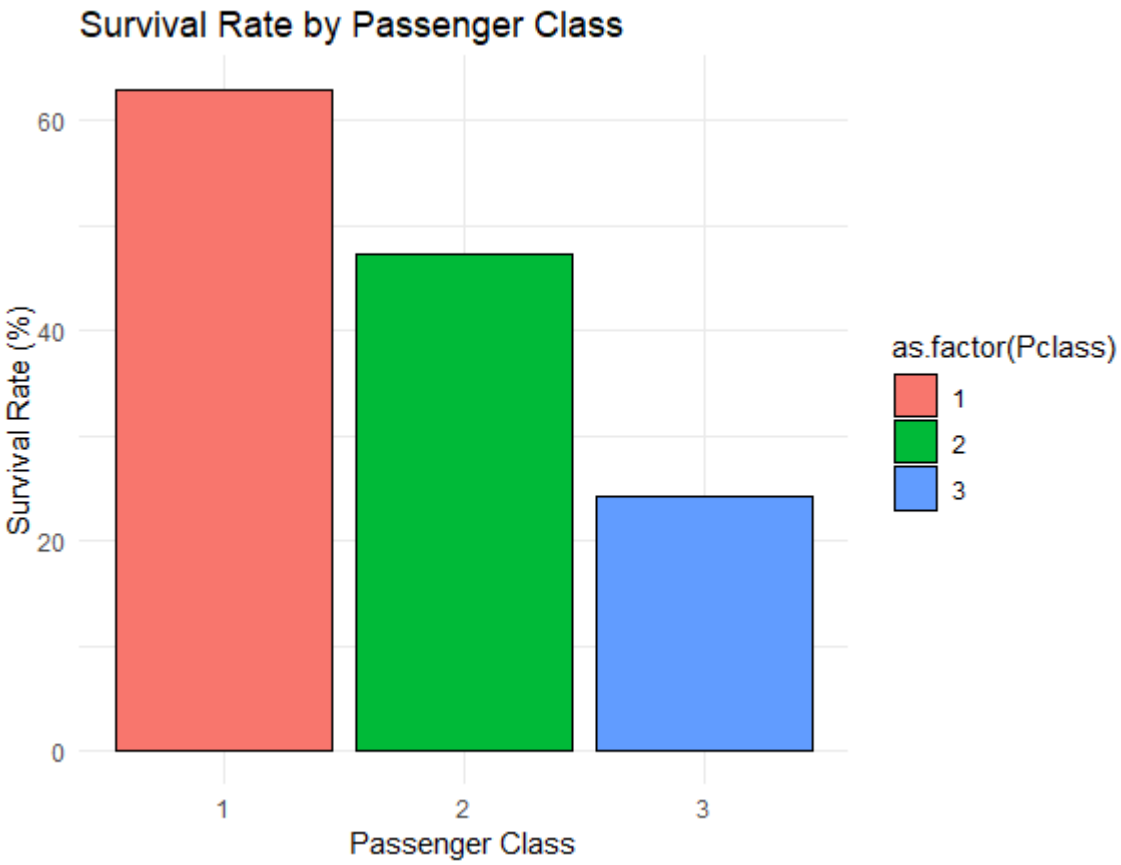
```
library(ggplot2)
library(dplyr)
library(summarytools)

# titanic dataset
titanic <-read.csv("https://raw.githubusercontent.com/
datasciencedojo/datasets/master/titanic.csv")

head(titanic)

# histogram of the number of parents and children aboard (parch)
ggplot(titanic, aes(x = Parch)) +
  geom_histogram(binwidth = 1, fill = "olivedrab", color = "black") +
  labs(title = "Number of Parents and Children Aboard (Parch)",
  x = "Parch", y = "Frequency") + theme_minimal()

# summary of the dataset
summary(titanic)
dfSummary(titanic)

# missing values
colSums(is.na(titanic))

# survival rate by gender
titanic %>%
  group_by(Sex) %>%
  summarise(Survived_Rate = mean(Survived, na.rm = TRUE) * 100) %>%
  ggplot(aes(x = Sex, y = Survived_Rate, fill = Sex)) +
  geom_bar(stat = "identity", color = "black") +
```

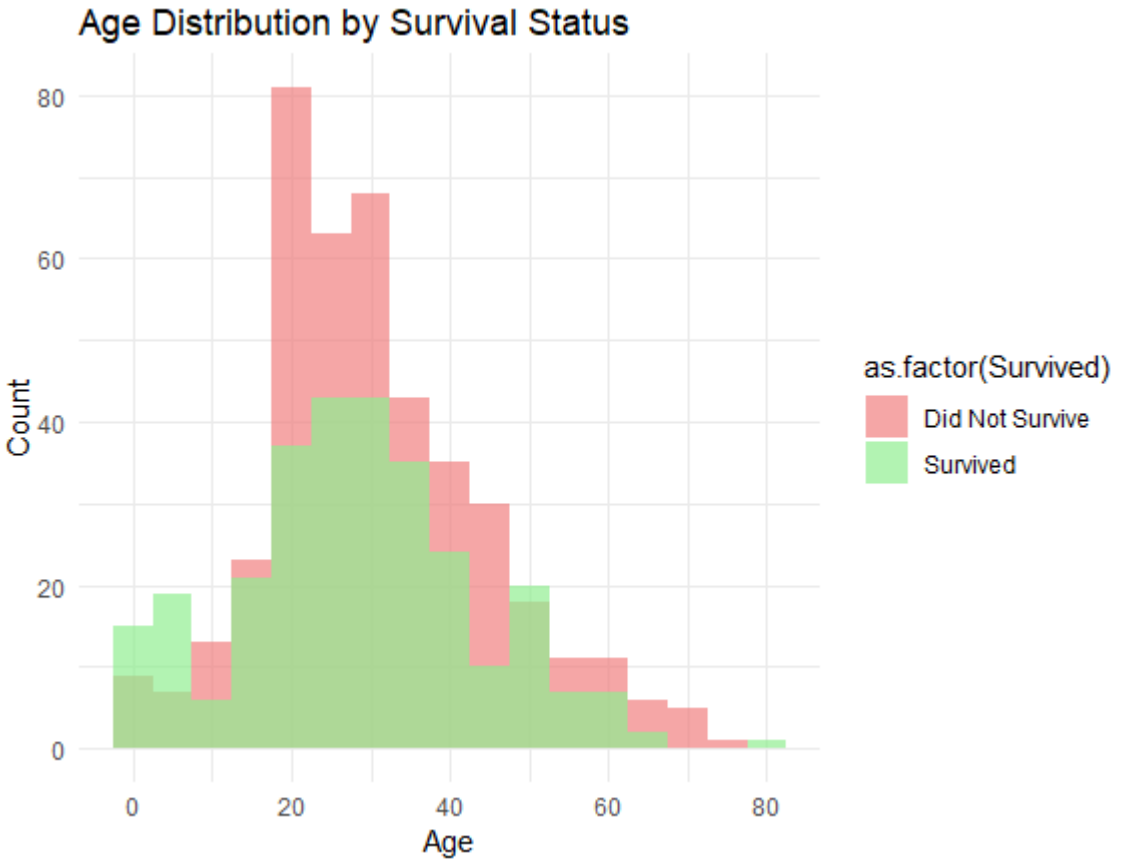
```
  labs(title = "Survival Rate by Gender", x = "Gender",
  y = "Survival Rate (%)") + theme_minimal()

# survival rate by passenger class
titanic %>%
  group_by(Pclass) %>%
  summarise(Survived_Rate = mean(Survived, na.rm = TRUE) * 100) %>%
  ggplot(aes(x = as.factor(Pclass), y = Survived_Rate, fill =
  as.factor(Pclass))) + geom_bar(stat = "identity", color = "black") +
  labs(title = "Survival Rate by Passenger Class", x =
  "Passenger Class", y = "Survival Rate (%)") + theme_minimal()

# age distribution by survival status
ggplot(titanic, aes(x = Age, fill = as.factor(Survived))) +
  geom_histogram(binwidth = 5, alpha = 0.7, position = "identity") +
  scale_fill_manual(values = c("lightcoral", "lightgreen"),
  labels = c("Did Not Survive", "Survived")) + labs(title = "Age
  Distribution by Survival Status", x = "Age", y = "Count") +
  theme_minimal()

# box plot of age by survival status
ggplot(titanic, aes(x = as.factor(Survived), y = Age,
    fill = as.factor(Survived))) +
  geom_boxplot(outlier.color = "peru", outlier.shape = 16,
  outlier.size = 2, notch = TRUE) +
  scale_fill_manual(values = c("lightcoral", "lightgreen"),
  labels = c("Did Not Survive", "Survived")) +
  labs(title = "Age Distribution of Survivors and Non-Survivors",
  x = "Survival Status", y = "Age") +
  theme_minimal() +
  theme(legend.position = "none")
```

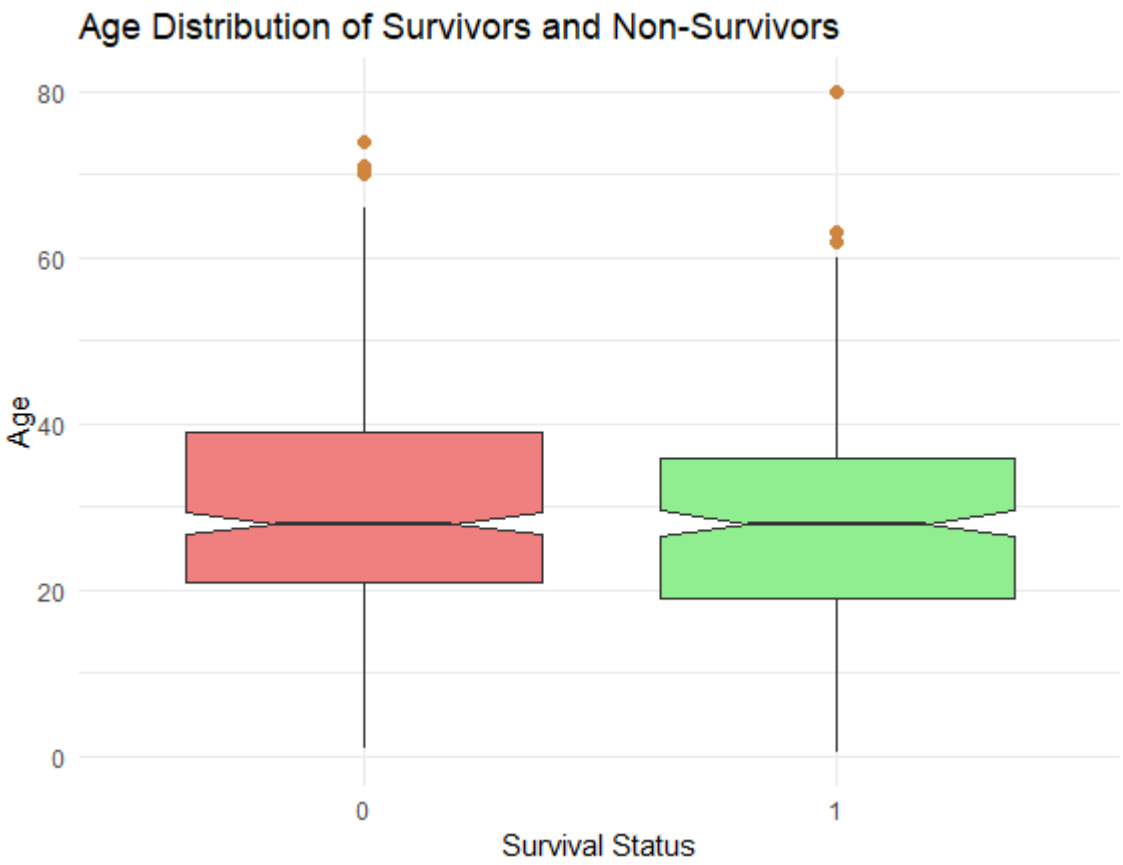
OUTPUT


Number of Parents and Children Aboard (Parch)
Frequency
600
400
200
0
0
2
4
6
Parch


Survival Rate by Gender
Survival Rate (%)
60
40
20
0
female
male
Gender
Sex
female
male


Survival Rate by Passenger Class
Survival Rate (%)
60
40
20
0
1
2
3
Passenger Class
as.factor(Pclass)
1
2
3


Age Distribution by Survival Status
Count
80
60
40
20
0
0
20
40
60
80
Age
as.factor(Survived)
Did Not Survive
Survived


Age Distribution of Survivors and Non-Survivors
80
60
40
20
0
Age
0
1
Survival Status

IRIS EDA

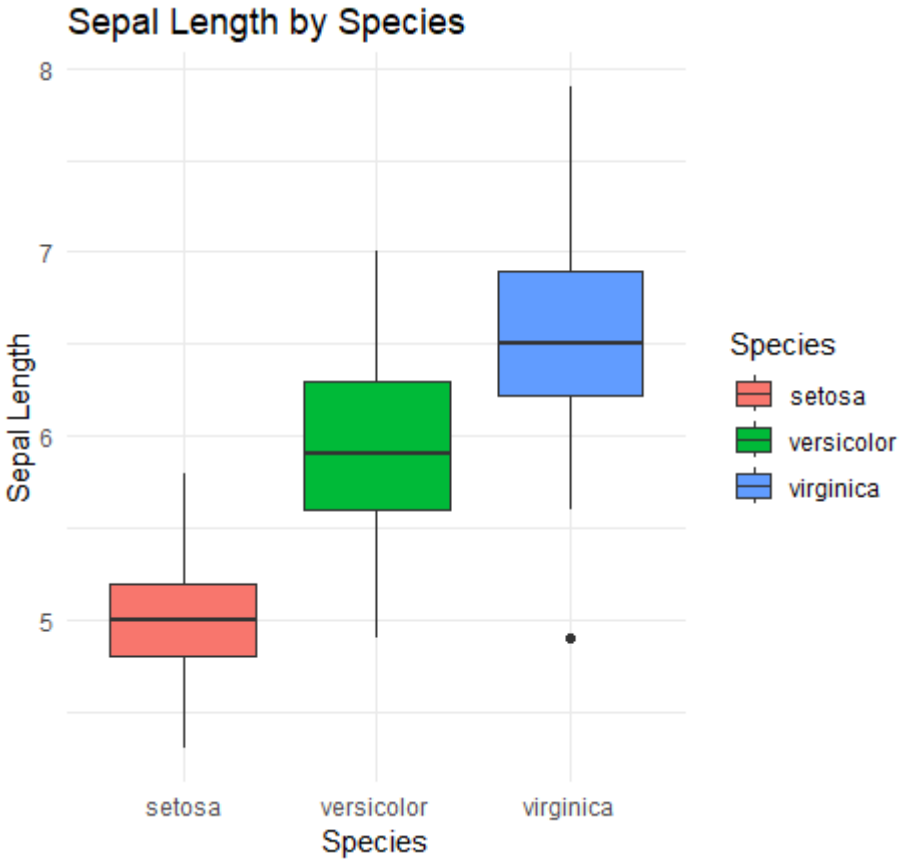
AIM

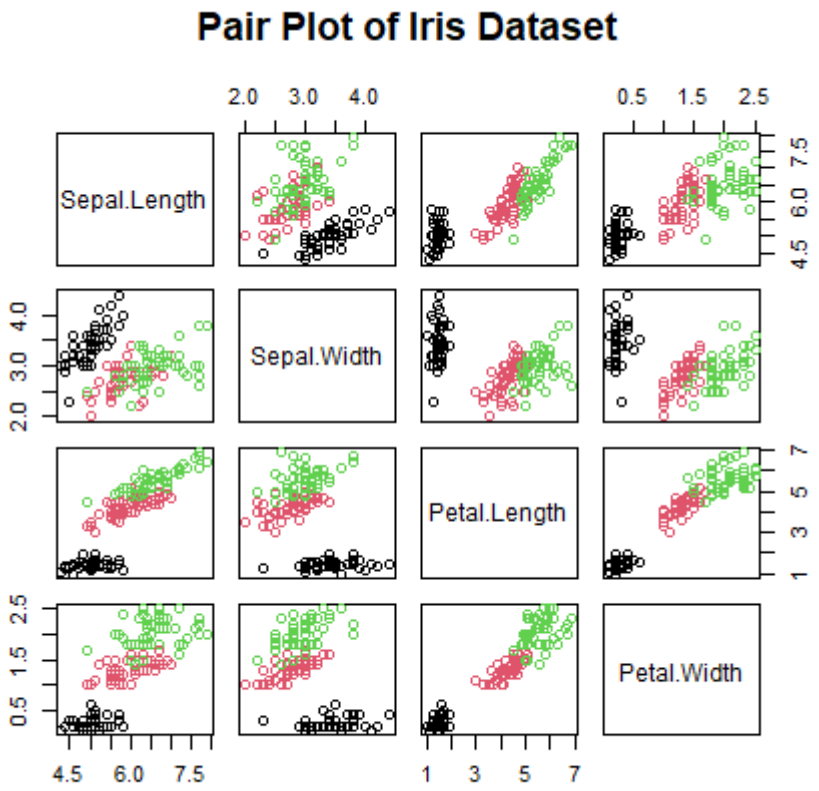
EDA on "Iris Dataset" a) For the Iris dataset, which contains measurements of various iris flowers, conduct an EDA. a. Determine if there are statistically significant differences in sepal lengths between different species using a suitable statistical test. b. Create a pair plot to visualize the relationships between all variables.

PROGRAM

```
library(ggplot2)
library(dplyr)

data(iris)

head(iris)

# box plot to see the distribution of sepal length for each species
ggplot(iris, aes(x = Species, y = Sepal.Length, fill = Species)) +
  geom_boxplot() +
  labs(title = "Sepal Length by Species", x = "Species",
  y = "Sepal Length") + theme_minimal()

# ANOVA to see if there are significant differences in
Sepal.Length across the species
anova_result <- aov(Sepal.Length ~ Species, data = iris)
summary(anova_result)

# if p-value from the ANOVA summary is less than 0.05,
it suggests significant differences
# between species in terms of Sepal.Length.

# if ANOVA is significant, we use a Tukey test to see which
species differ in Sepal Length
if(summary(anova_result)[[1]][["Pr(>F)"]][1] < 0.05) {
  post_hoc <- TukeyHSD(anova_result)
  print(post_hoc)
}

# scatter plots between each pair of variables, colored by species
pairs(iris[1:4], col = iris$Species, main = "Pair Plot of Iris Dataset")
```

OUTPUT


Sepal Length by Species
Sepal Length
8
7
6
5
setosa
versicolor
virginica
Species
Species
setosa
versicolor
virginica


Pair Plot of Iris Dataset
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Sepal.Length ~ Species, data = iris)

$Species
                      diff       lwr       upr p adj
versicolor-setosa    0.930 0.6862273 1.1737727     0
virginica-setosa     1.582 1.3382273 1.8257727     0
virginica-versicolor 0.652 0.4082273 0.8957727     0
```

LINEAR REGRESSION

AIM

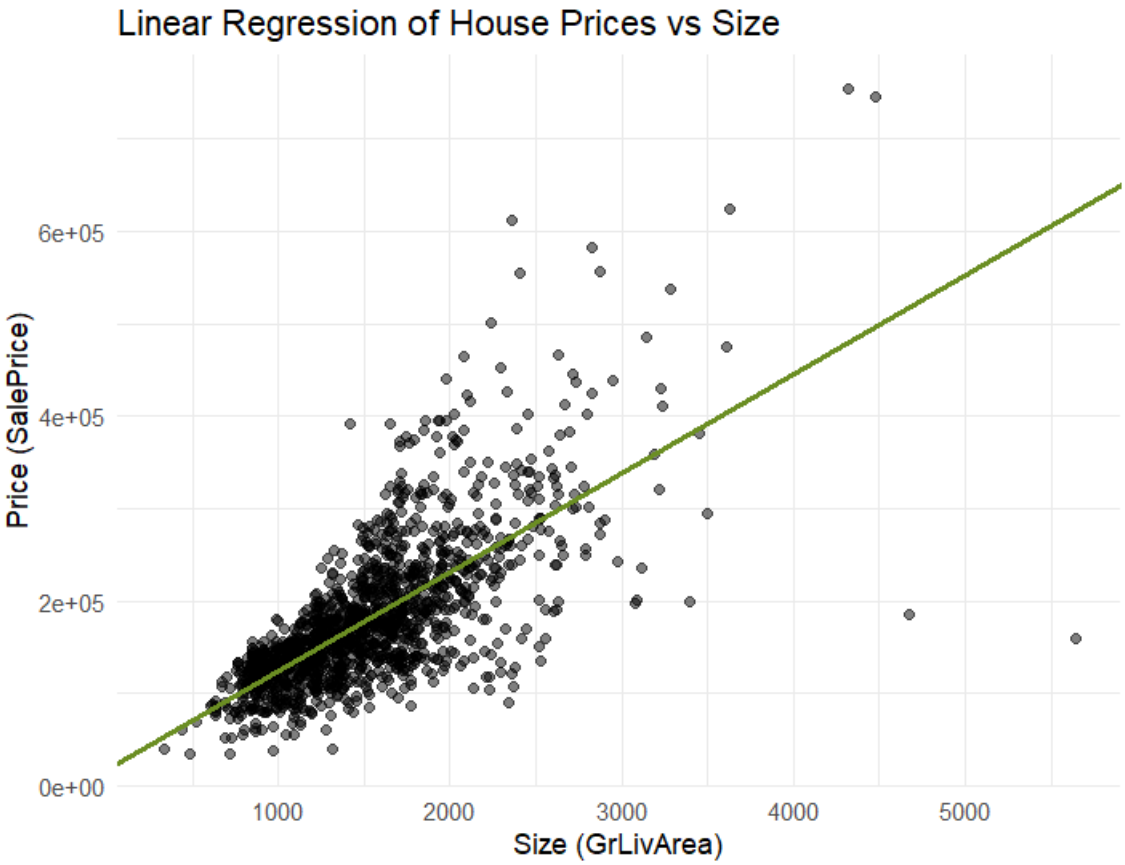
Suppose you have a dataset containing information about house prices (dependent variable, denoted as price) and the size of the houses (in square feet, independent variable, denoted as size). You want to build a linear regression model to predict house prices based on their size. Write an R code snippet to perform the following steps:

1. Load the dataset Download the dataset from https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques
2. Fit a simple linear regression model: Use price as the dependent variable and size as the independent variable.
3. Calculate regression coefficients: Determine the slope and intercept of the regression line.
4. Plot the regression line: Overlay the regression line on the scatter plot of the data points.

PROGRAM

```
#House pricing

library(ggplot2)

path <- 'E:/R/Lab Cycle 3 - Sem 5/house_price/train.csv'
data <- read.csv(path)

model <- lm(SalePrice ~ GrLivArea, data = data)

slope <- coef(model)[2]
intercept <- coef(model)[1]

cat("Slope:", slope, "\n")
cat("Intercept:", intercept, "\n")

ggplot(data, aes(x = GrLivArea, y = SalePrice)) +
  geom_point(alpha = 0.5) +  # Scatter plot of the data points
  geom_abline(slope = slope, intercept = intercept, col = 'olivedrab',
  linewidth = 1) + labs(title = "Linear Regression of House Prices
  vs Size", x = "Size (GrLivArea)",  y = "Price (SalePrice)") +
  theme_minimal()
```

OUTPUT




```
> cat("Slope:", slope, "\n")
Slope: 107.1304
> cat("Intercept:", intercept, "\n")
Intercept: 18569.03
```

GRAPH REPRESENTATION

AIM

Graph

1. Create an adjacency list representation for a given undirected graph
2. Implement a function to add an edge between two vertices in the graph
3. Write an R function to perform DFS traversal on a graph starting from a specific vertex

References https://r.igraph.org/index.html

PROGRAM

```
# Graph

# Create adjacency list representation for an undirected graph
create_adjacency_list <- function(edges) {
  adj_list <- list()

  # Initialize adjacency list for each vertex in edges
  for (edge in edges) {
    v1 <- edge[1]
    v2 <- edge[2]

    # Initialize adjacency lists for each vertex if they don't exist
    if (is.null(adj_list[[as.character(v1)]]))
    adj_list[[as.character(v1)]] <- c()
    if (is.null(adj_list[[as.character(v2)]]))
    adj_list[[as.character(v2)]] <- c()

    # Add edges (undirected)
    adj_list[[as.character(v1)]] <- c(adj_list[[as.character(v1)]], v2)
    adj_list[[as.character(v2)]] <- c(adj_list[[as.character(v2)]], v1)
  }
  return(adj_list)
}

# Function to add an edge between two vertices in the graph
add_edge <- function(adj_list, v1, v2) {
  # Initialize adjacency lists for each vertex if they don't exist
```

```
  if (is.null(adj_list[[as.character(v1)]]))
  adj_list[[as.character(v1)]] <- c()
  if (is.null(adj_list[[as.character(v2)]]))
  adj_list[[as.character(v2)]] <- c()

  adj_list[[as.character(v1)]] <- c(adj_list[[as.character(v1)]], v2)
  adj_list[[as.character(v2)]] <- c(adj_list[[as.character(v2)]], v1)
  return(adj_list)
}

# DFS traversal on a graph starting from a specific vertex
dfs_traversal <- function(adj_list, start, visited = NULL) {
  # Convert start to character to match list indexing
  start <- as.character(start)

  # Initialize visited with enough entries to cover all
  vertices in adj_list
  if (is.null(visited)) visited <- setNames(rep(FALSE,
  length(adj_list)), names(adj_list))

  visited[start] <- TRUE
  cat(start, " ")

  for (neighbor in adj_list[[start]]) {
    neighbor <- as.character(neighbor)
    # Ensure neighbor is a character for indexing
    if (!visited[neighbor]) {
      visited <- dfs_traversal(adj_list, neighbor, visited)
    }
  }
  return(visited)
}

edges <- list(
  c(1, 2), c(1, 5), c(2, 3), c(2, 4),
  c(5, 6), c(6, 7), c(7, 8), c(8, 9),
  c(3, 9), c(3, 10), c(10, 11), c(11, 12),
  c(12, 1))

# Step 1: Create adjacency list
adj_list <- create_adjacency_list(edges)
```

```
print("Initial adjacency list:")
print(adj_list)

# Step 2: Add an additional edge to increase complexity
adj_list <- add_edge(adj_list, 9, 12)
print("Adjacency list after adding edge (9, 12):")
print(adj_list)

# Step 3: Perform DFS traversal starting from vertex 1
cat("DFS traversal starting from vertex 1:\n")
visited <- dfs_traversal(adj_list, 1)
```

OUTPUT

```
[1] "Initial adjacency list:"
> print(adj_list)
$`1`
[1]  2  5 12

$`2`
[1] 1 3 4

$`5`
[1] 1 6

$`3`
[1]  2  9 10

$`4`
[1] 2

$`6`
[1] 5 7

$`7`
[1] 6 8

$`8`
[1] 7 9

$`9`
[1] 8 3

$`10`
[1]  3 11

$`11`
[1] 10 12

$`12`
[1] 11  1
```

```
[1] "Adjacency list after adding edge (9, 12):"
> print(adj_list)
$`1`
[1]  2  5 12

$`2`
[1] 1 3 4

$`5`
[1] 1 6

$`3`
[1]  2  9 10

$`4`
[1] 2

$`6`
[1] 5 7

$`7`
[1] 6 8

$`8`
[1] 7 9

$`9`
[1]  8  3 12

$`10`
[1]  3 11

$`11`
[1] 10 12

$`12`
[1] 11  1  9
```

```
DFS traversal starting from vertex 1:
> visited <- dfs_traversal(adj_list, 1)
1  2  3  9  8  7  6  5  12  11  10  4
```

CORRELATION ANALYSIS

AIM

Suppose we have a dataset of motor trend car road tests (mtcars). The dataset contains information about 32 car brands and 11 attributes. We want to investigate the correlation between the horsepower (hp) and miles per gallon (mpg). Perform a Pearson correlation test to analyze this relationship.

PROGRAM

```
# mtcars - pearson correlation

data("mtcars")

head(mtcars)

correlation <- cor.test(mtcars$hp, mtcars$mpg, method = "pearson")

print(correlation)
```

OUTPUT

```
        Pearson's product-moment correlation

data:  mtcars$hp and mtcars$mpg
t = -6.7424, df = 30, p-value = 1.788e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.8852686 -0.5860994
sample estimates:
       cor
-0.7761684
```

ANOVA TEST

AIM

Suppose we have a dataset of motor trend car road tests (mtcars). The dataset contains information about 32 car brands and 11 attributes. We want to investigate whether there are any significant variations in the average displacement (disp) across different gear types (gear). Perform a one-way ANOVA test to analyze this

PROGRAM

```
# one way ANOVA

data("mtcars")

# Convert gear to a factor as it's a categorical variable
mtcars$gear <- as.factor(mtcars$gear)

# one-way ANOVA
anova_result <- aov(disp ~ gear, data = mtcars)

summary(anova_result)
```

OUTPUT

```
            Df Sum Sq Mean Sq F value   Pr(>F)
gear         2 280221  140110   20.73 2.56e-06 ***
Residuals   29 195964    6757
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

COVID ANALYSIS

AIM

We want to investigate the behavior of the total positive COVID-19 cases weekly from 22 January 2020 to 15 December 2020 in India. Perform the following tasks:
Data set link https://raw.githubusercontent.com/datasets/covid-19/master/data/time-series-19-covid-combined.csv

1. Univariate Time Series Analysis:

- Create a time series object for the total positive COVID-19 cases

- Visualize the time series data using a line chart.

2. Multivariate Time Series Analysis:

- Also, consider the total deaths from COVID-19 during the same period.

- Create a multivariate time series object that includes both the total positive cases and total deaths.

- Plot both series on a single chart.

3. Time Series Forecasting:

- Use the auto.arima() function from the forecast library to fit an ARIMA model to the total positive cases.

- Forecast the next 5 data points.

- Plot the forecasted values.

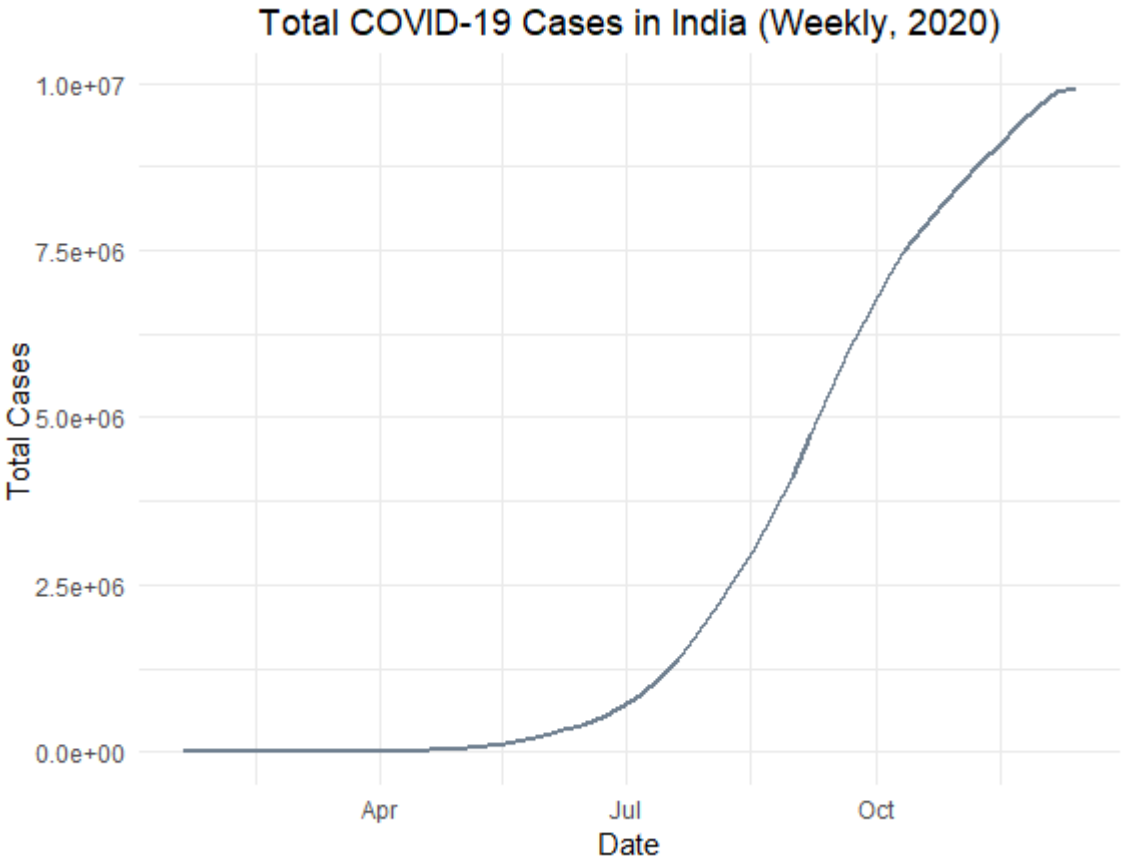
PROGRAM

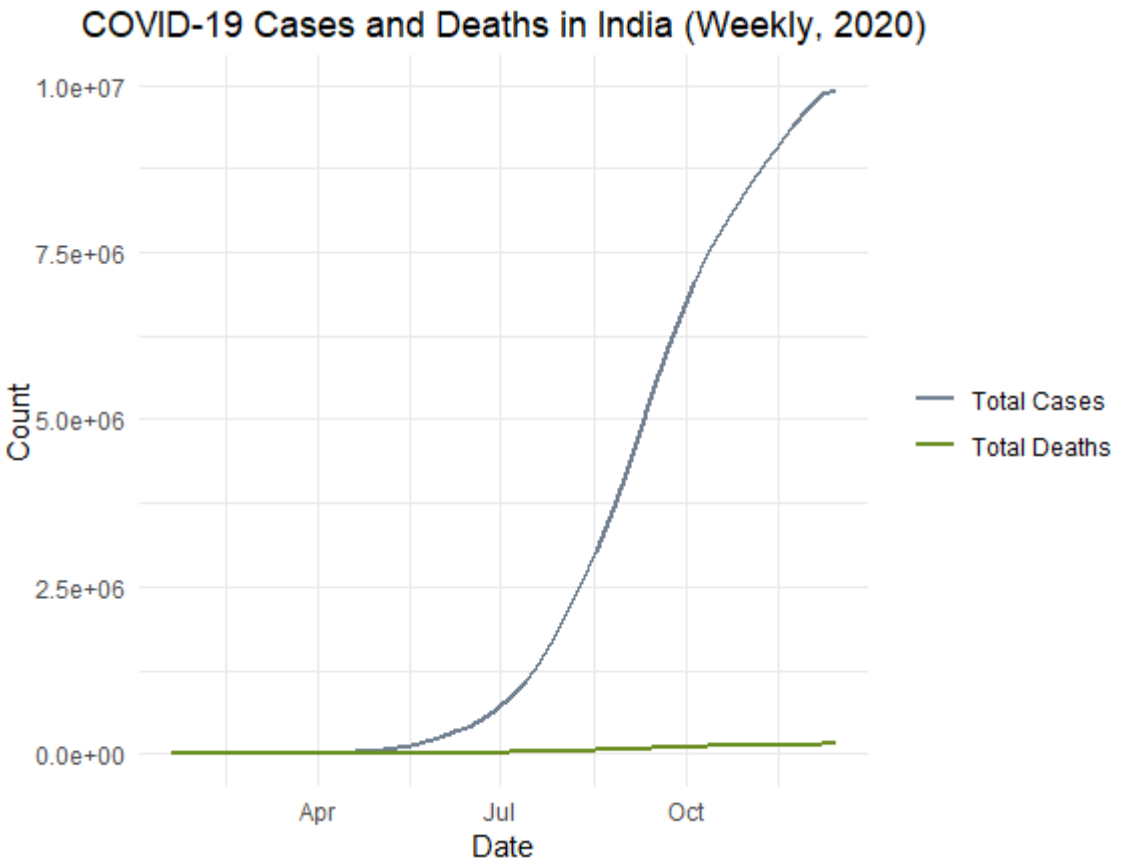
```
library(forecast)
library(ggplot2)
library(dplyr)

covid_data <- read.csv("https://raw.githubusercontent.com/datasets/covid-
19/master/data/time-series-19-covid-combined.csv")

# data range
india_covid <- covid_data %>%
  filter(Country.Region == "India" & Date >= "2020-01-22" & Date
  <= "2020-12-15")
```

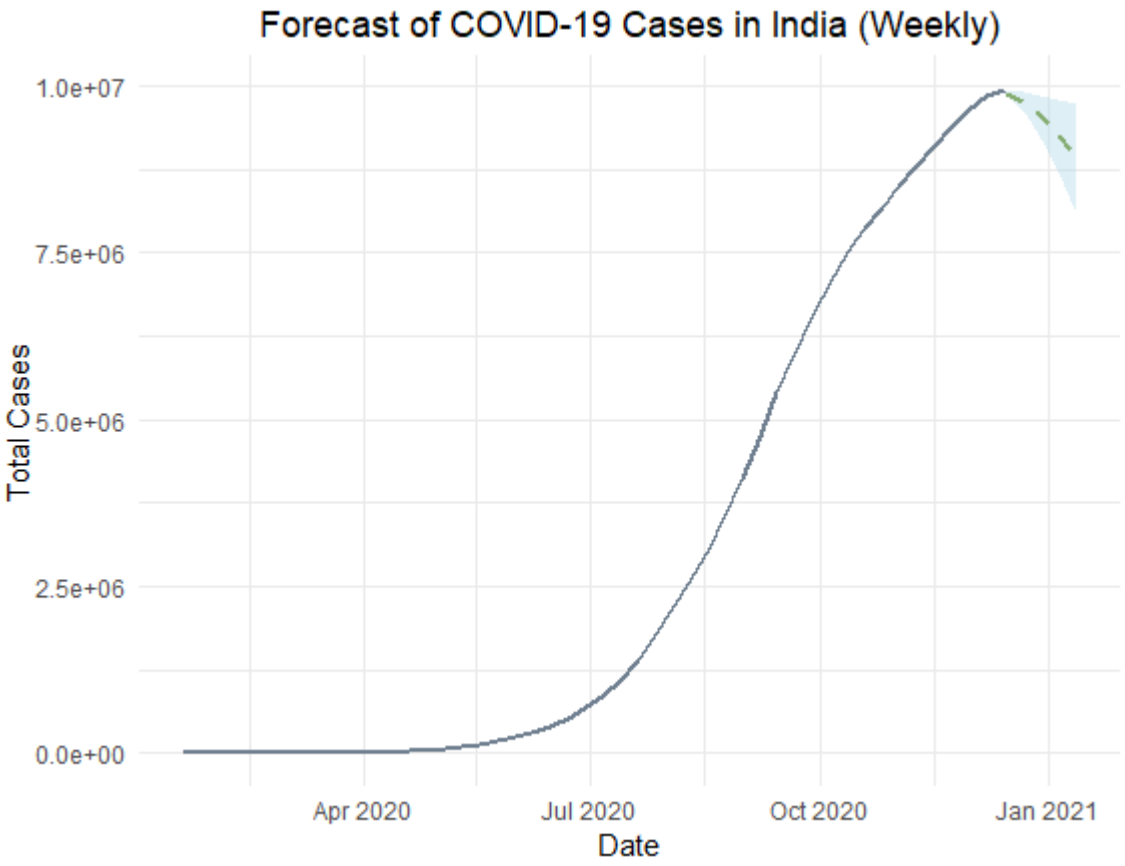
```
india_covid$Date <- as.Date(india_covid$Date)

# weekly data summary
weekly_data <- india_covid %>%
  group_by(Week = format(Date, "%Y-%U")) %>%
  summarise(Confirmed = max(Confirmed), Deaths = max(Deaths)) %>%
  ungroup()

# time series
total_cases_ts <- ts(weekly_data$Confirmed, start = c(2020, 4),
frequency = 52)

# plot total covid cases
ggplot(weekly_data, aes(x = as.Date(paste0(Week, "-1"),
  "%Y-%U-%u"), y = Confirmed)) +
  geom_line(color = "slategrey", size = 1) +
  labs(title = "Total COVID-19 Cases in India (Weekly, 2020)",
  x = "Date", y = "Total Cases") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

# multivariate
total_deaths_ts <- ts(weekly_data$Deaths, start = c(2020, 4)
, frequency = 52)
multivariate_ts <- ts(cbind(total_cases_ts, total_deaths_ts))

multivariate_df <- data.frame(
  Date = as.Date(paste0(weekly_data$Week, "-1"), "%Y-%U-%u"),
  Cases = weekly_data$Confirmed,
  Deaths = weekly_data$Deaths
)

ggplot(multivariate_df, aes(x = Date)) +
  geom_line(aes(y = Cases, color = "Total Cases"), size = 1) +
  geom_line(aes(y = Deaths, color = "Total Deaths"), size = 1) +
  labs(title = "COVID-19 Cases and Deaths in India (Weekly, 2020)",
  x = "Date", y = "Count") +
  scale_color_manual(values = c("Total Cases" = "slategrey",
  "Total Deaths" = "olivedrab")) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
```

```
  legend.title = element_blank())

# forecast - ARIMA
arima_model <- auto.arima(total_cases_ts)

# values for next 5 weeks
forecasted_values <- forecast(arima_model, h = 5)

forecast_df <- data.frame(
  Date = seq(as.Date("2020-12-15"), by = "week", length.out = 5),
  Forecast = forecasted_values$mean,
  Lower = forecasted_values$lower[,2],
  Upper = forecasted_values$upper[,2]
)

ggplot() +
  geom_line(data = multivariate_df, aes(x = Date, y = Cases),
  color = "slategrey", size = 1) +
  geom_line(data = forecast_df, aes(x = Date, y = Forecast),
  color = "olivedrab", size = 1, linetype = "dashed") +
  labs(title = "Forecast of COVID-19 Cases in India (Weekly)",
  x = "Date", y = "Total Cases") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_ribbon(data = forecast_df, aes(x = Date, ymin = Lower,
  ymax = Upper),
              fill = "lightblue", alpha = 0.4)  # Add confidence interval

print(forecasted_values$mean)
```

OUTPUT


Total COVID-19 Cases in India (Weekly, 2020)
1.0e+07
7.5e+06
5.0e+06
2.5e+06
0.0e+00
Total Cases
Apr
Jul
Oct
Date


COVID-19 Cases and Deaths in India (Weekly, 2020)
1.0e+07
7.5e+06
5.0e+06
2.5e+06
0.0e+00
Count
Apr
Jul
Oct
Date
Total Cases
Total Deaths




```
Time Series:
Start = c(2020, 52)
End = c(2021, 4)
Frequency = 52
[1] 9892690 9755875 9537579 9250807 8906490
```

BOSTON DATASET

AIM

The Boston data set comes from the real estate industry in Boston (US). This is a regression problem. The data has 506 rows and 14 columns.

1. Perform data exploration and visualization using R programing.
2. Perform Regression analysis on the dataset.
3. Predict the median value of owner occupied homes.

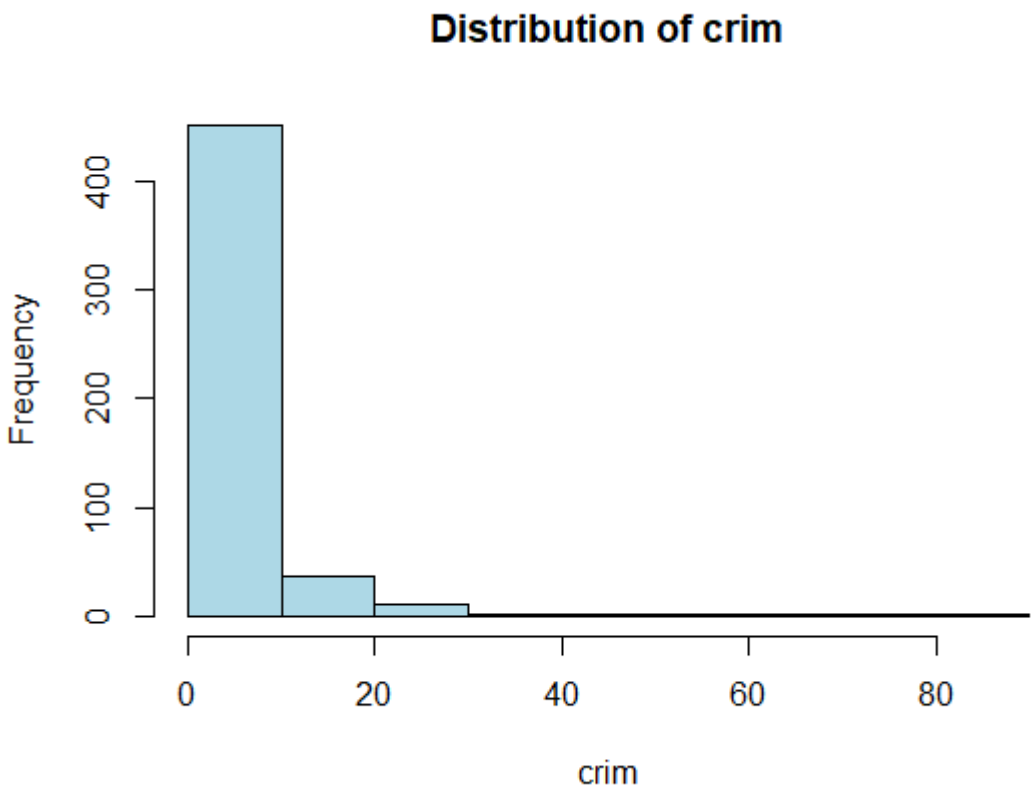
https://www.kaggle.com/code/prasadperera/the-boston-housing-dataset

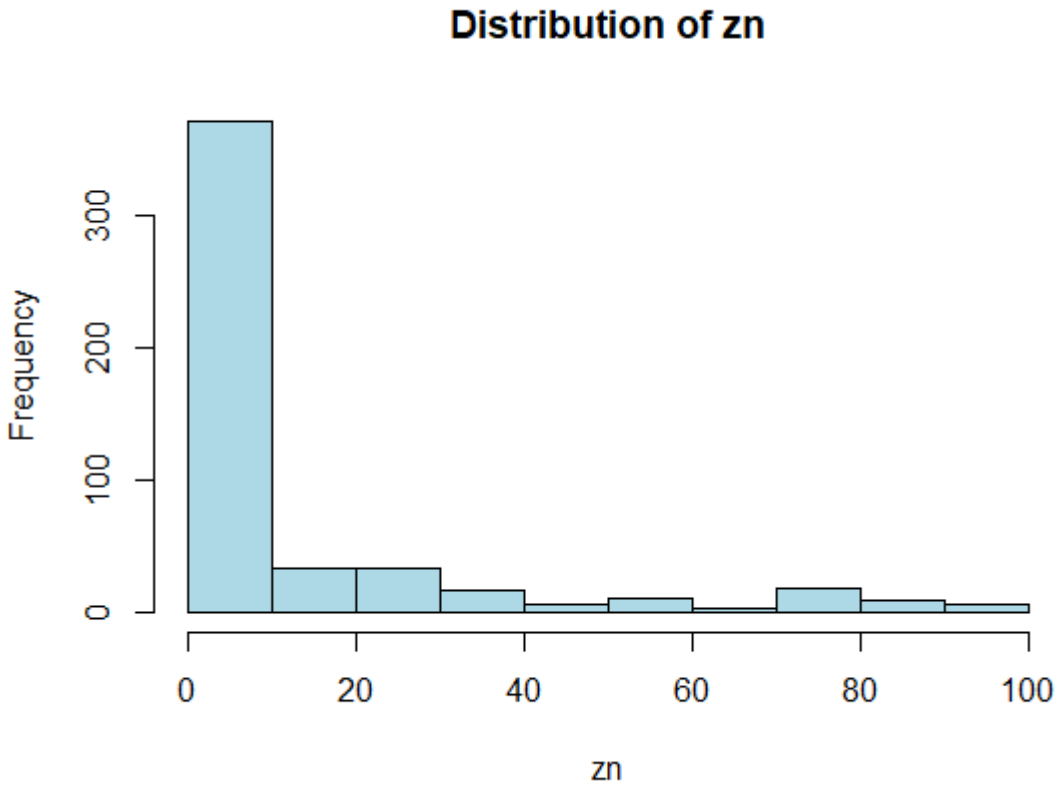
PROGRAM

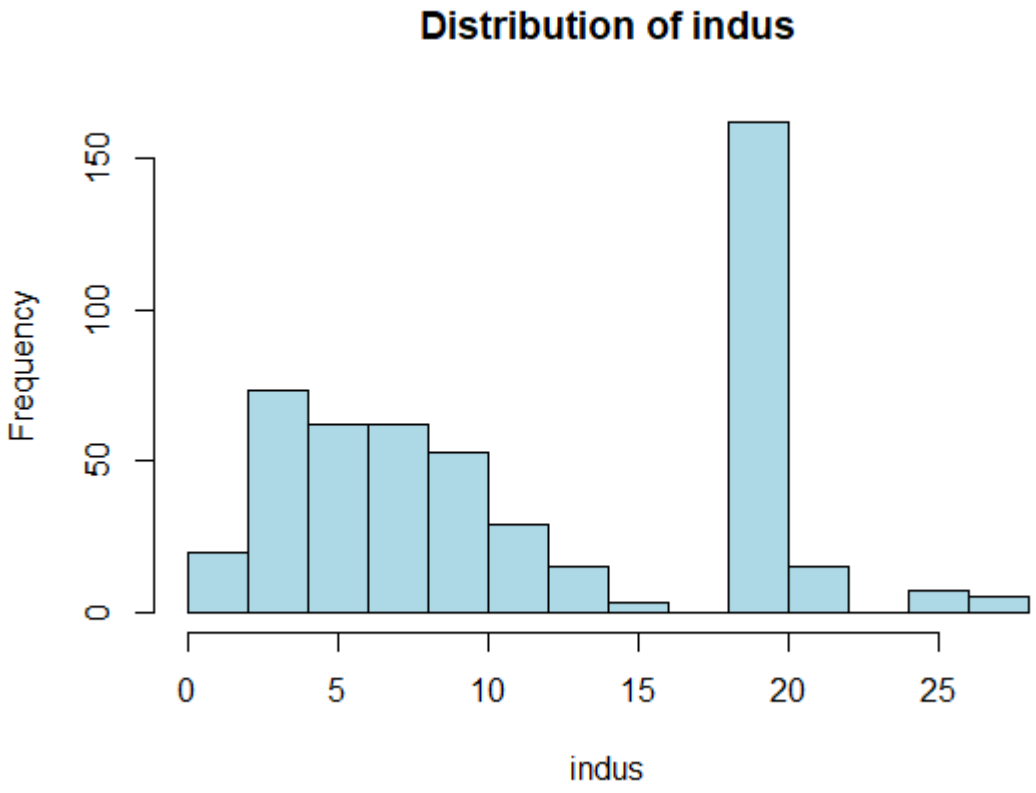
```
install.packages("MASS")
install.packages("ggplot2")
install.packages("corrplot")

library(MASS)
library(ggplot2)
library(corrplot)

# load dataset
data("Boston")
head(Boston)

# data exploration
str(Boston)
summary(Boston)

# missing values
sum(is.na(Boston))

# data visualization - hist for freq dist of all
for (variable in names(Boston)) {
  hist(Boston[[variable]], main = paste("Distribution of", variable),
       col = "lightblue", xlab = variable)
}

# - scatter plot for realtionship: each vs medv
for (variable in names(Boston)[-14]) { # take out medv (14th column)
```

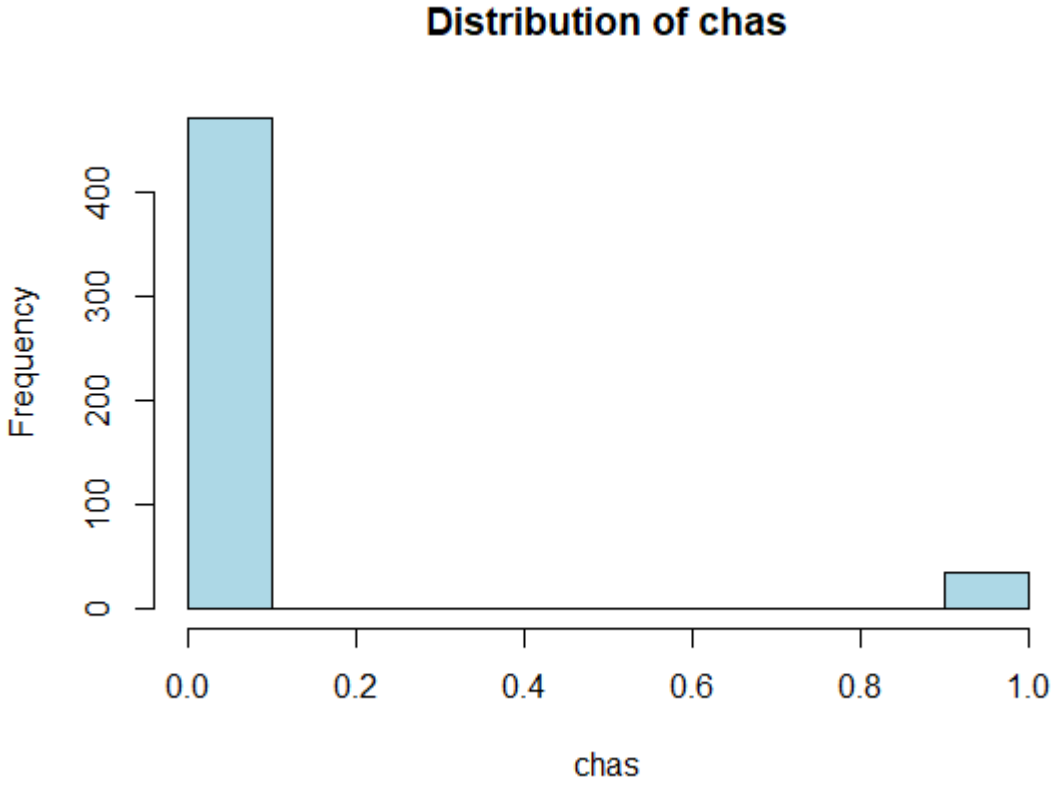
```
  plot(Boston[[variable]], Boston$medv,
       xlab = variable, ylab = "Median Value (medv)",
       main = paste(variable, "vs medv"))
}

# - correlation matrix
correlation_matrix <- cor(Boston)
corrplot(correlation_matrix, method = "color", tl.cex = 0.8)

# regression analysis
linear_model <- lm(medv ~ ., data = Boston)
summary(linear_model)

# Prediction
predicted_values <- predict(linear_model, newdata = Boston)

# actual vs predicted
results <- data.frame(Actual = Boston$medv, Predicted = predicted_values)
head(results)

# model eval
mse <- mean((results$Actual - results$Predicted)^2)
cat("Mean Squared Error (MSE):", mse, "\n")
r_squared <- summary(linear_model)$r.squared
cat("R-squared:", r_squared, "\n")

# plot predictions
ggplot(results, aes(x = Actual, y = Predicted)) +
  geom_point(color = "olivedrab") +
  geom_abline(intercept = 0, slope = 1, color = "peru",
  linetype = "dashed") +
  labs(title = "Actual vs Predicted Median Home Values",
       x = "Actual Median Value",
       y = "Predicted Median Value") +
  theme_minimal()
```

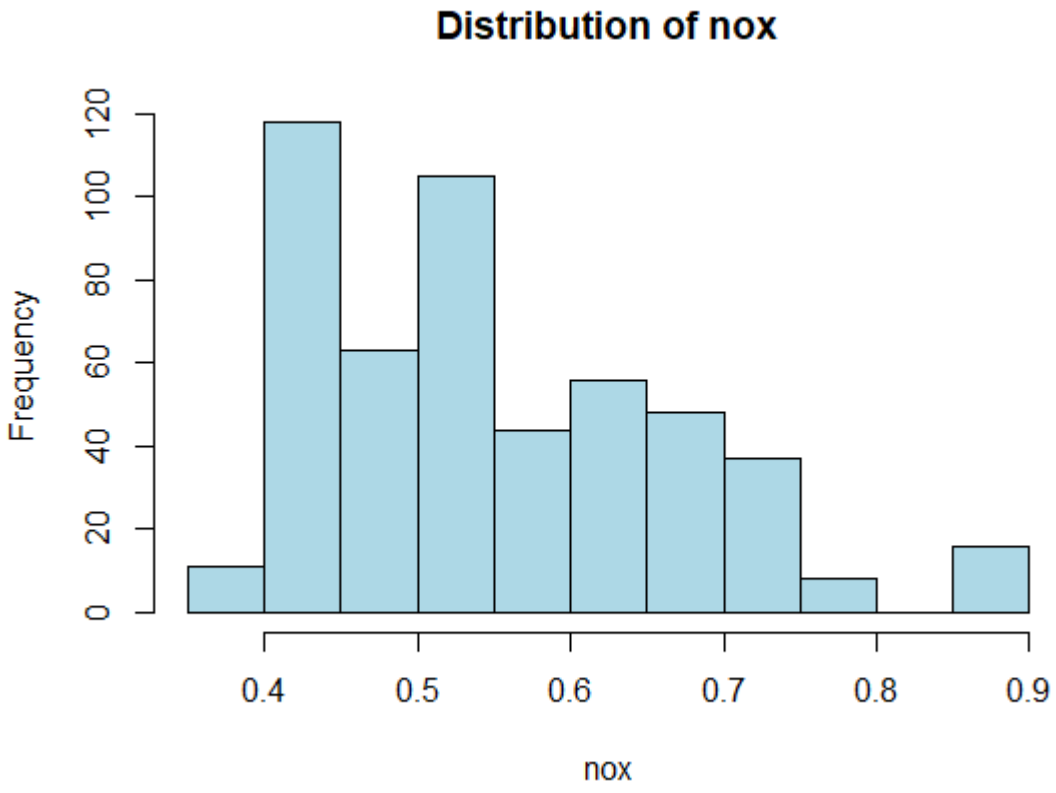
OUTPUT

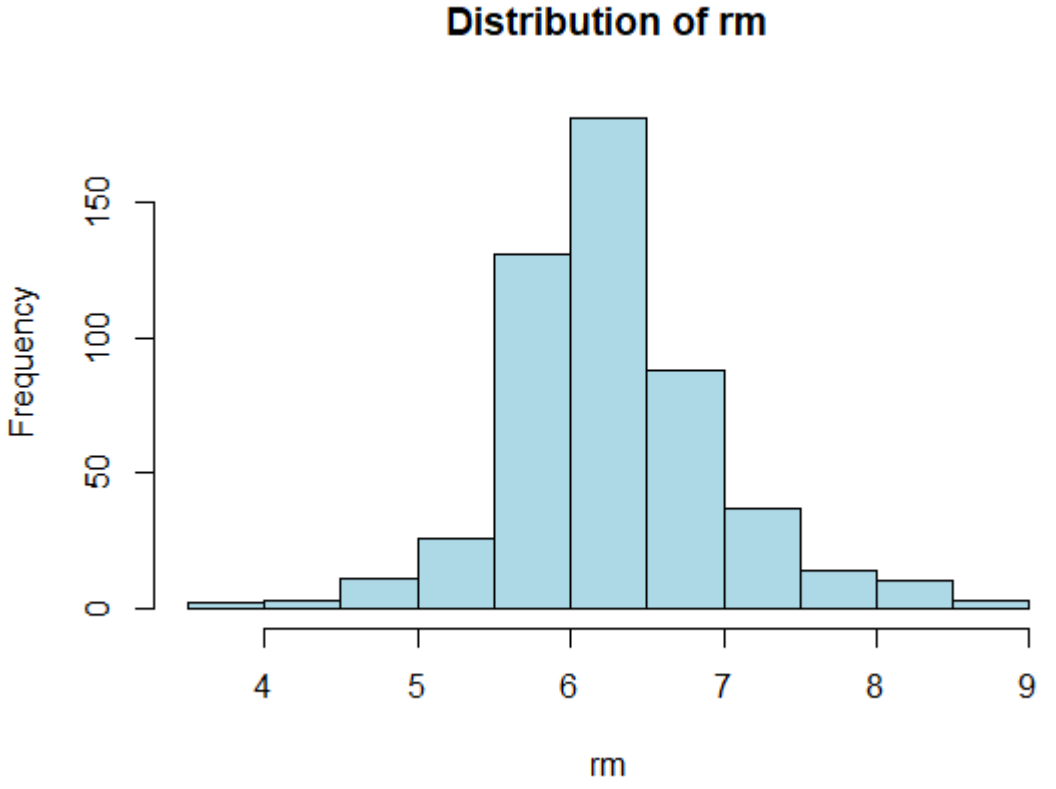

Distribution of crim
Frequency
0
100
200
300
400
0
20
40
60
80
crim

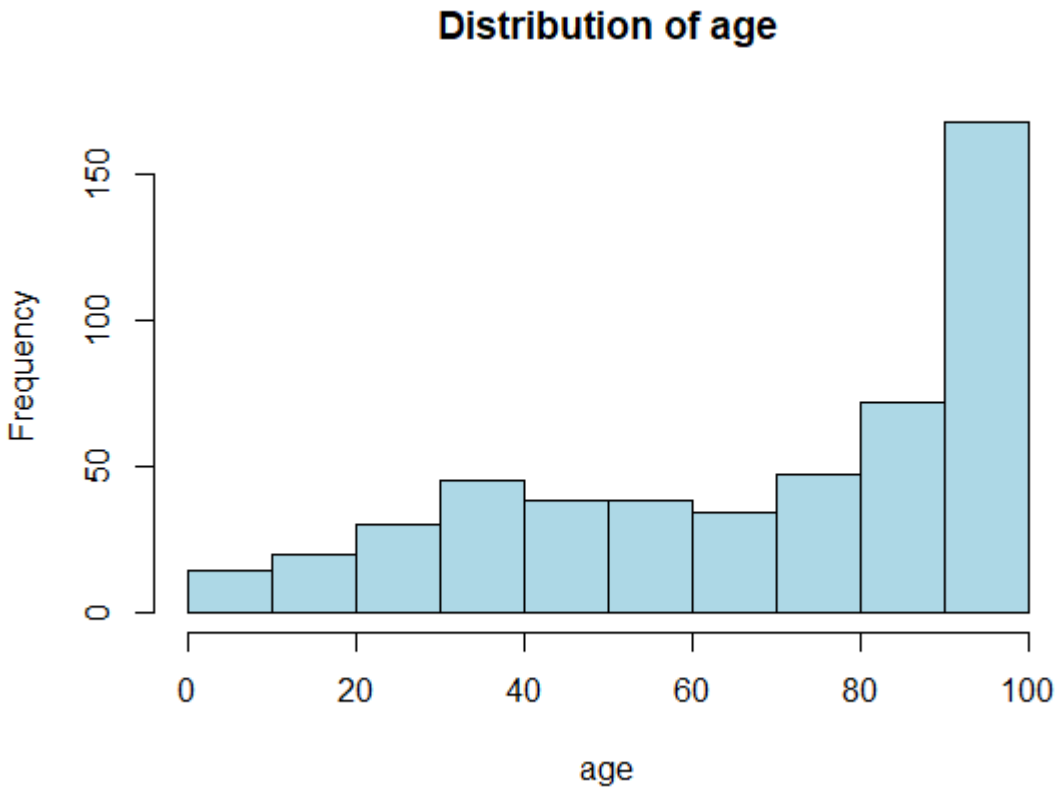

Distribution of zn
Frequency
0
100
200
300
0
20
40
60
80
100
zn

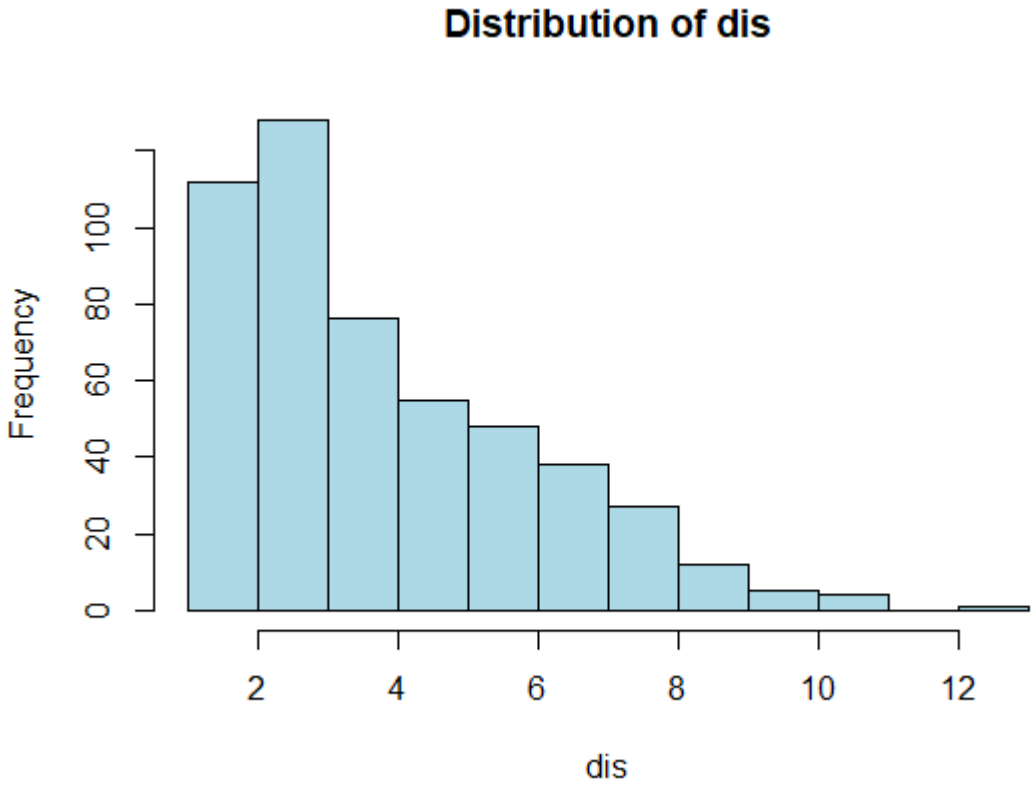

Distribution of indus
Frequency
0
50
100
150
0
5
10
15
20
25
indus

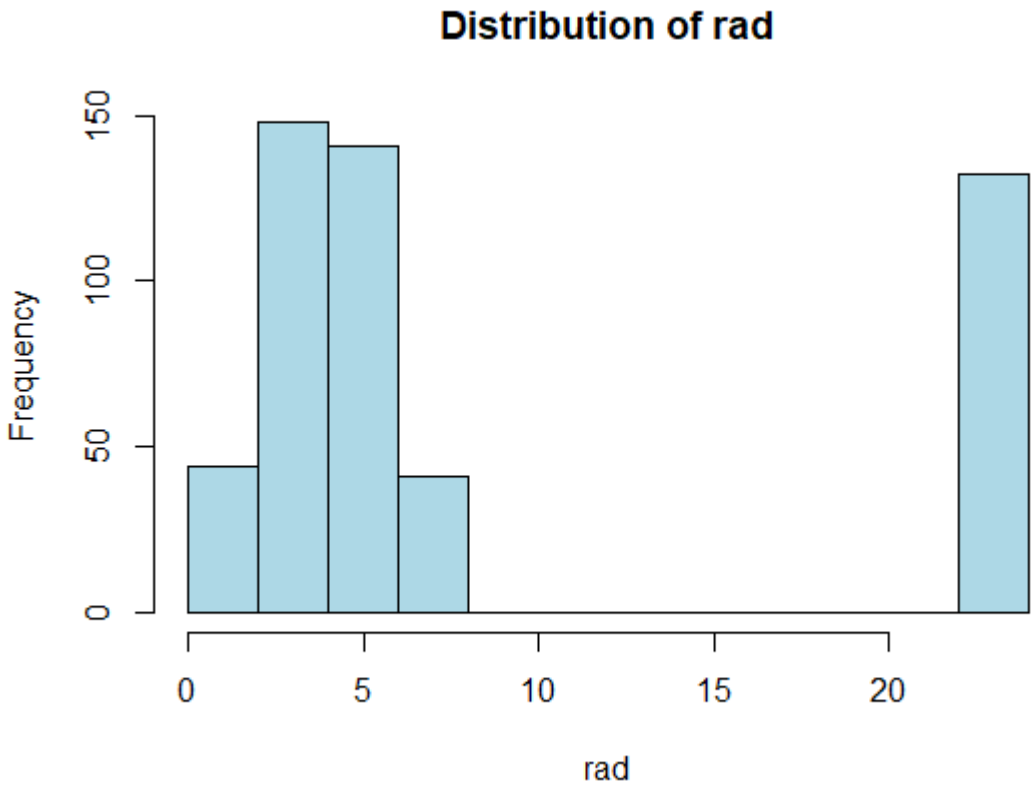

Distribution of chas
Frequency
0
100
200
300
400
0.0
0.2
0.4
0.6
0.8
1.0
chas

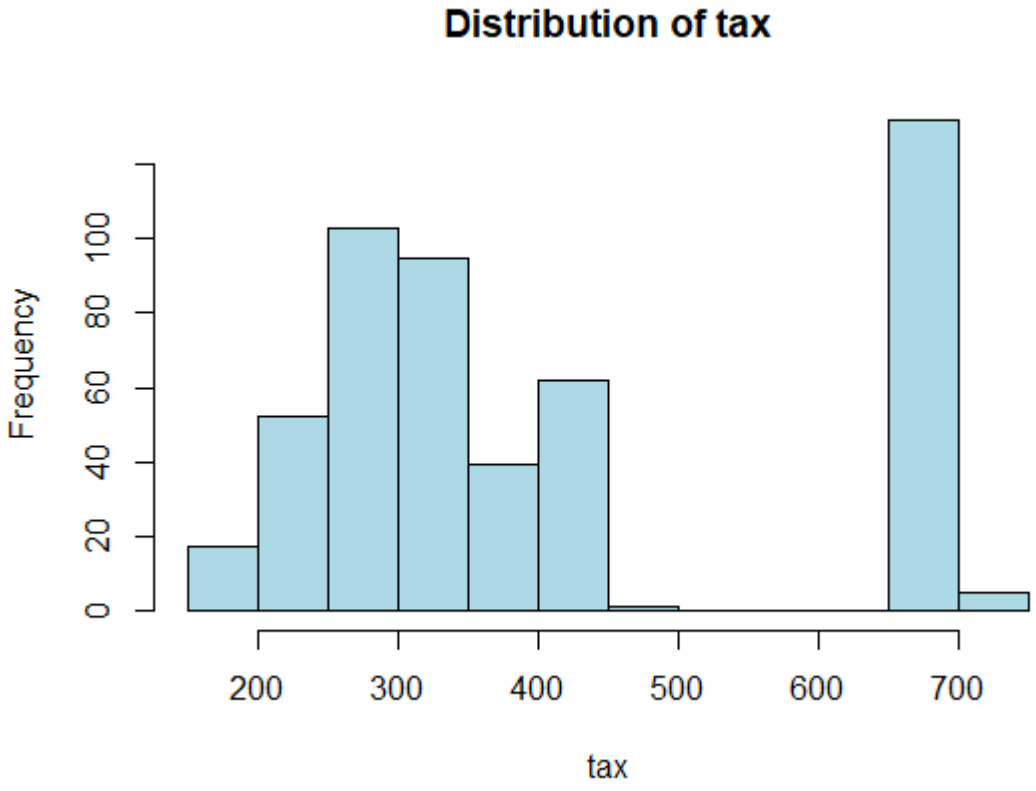

Distribution of nox
Frequency
0
20
40
60
80
100
120
0.4
0.5
0.6
0.7
0.8
0.9
nox

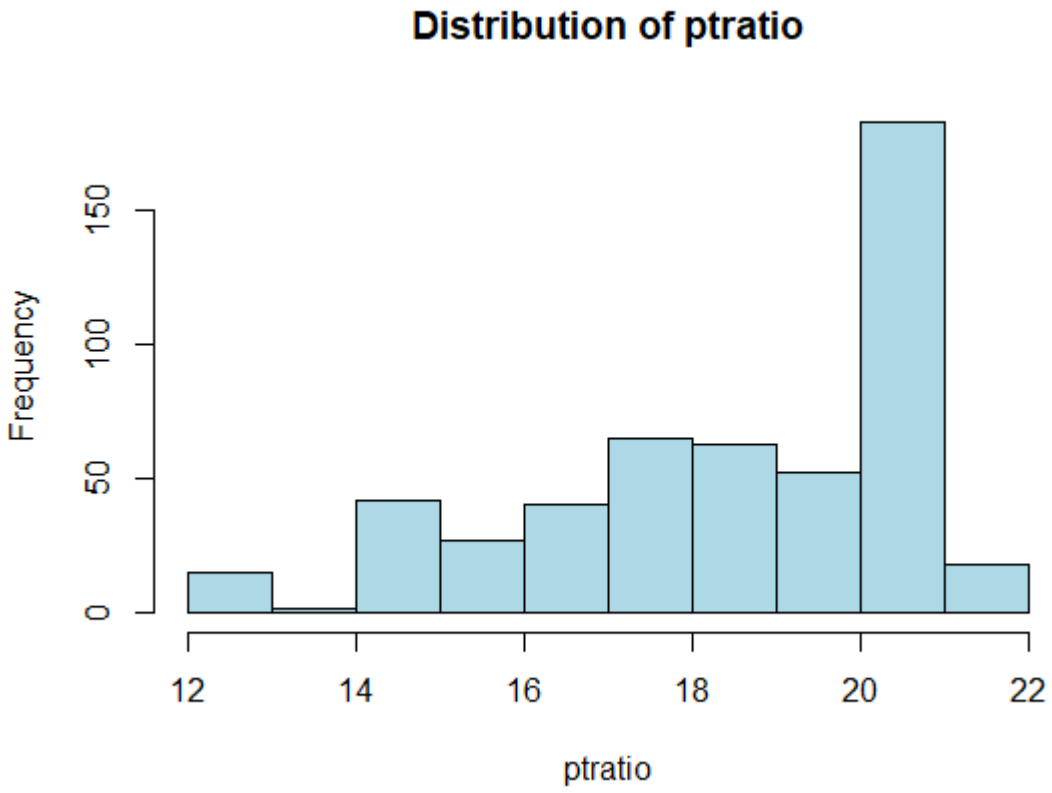

Distribution of rm
Frequency
0
50
100
150
4
5
6
7
8
9
rm

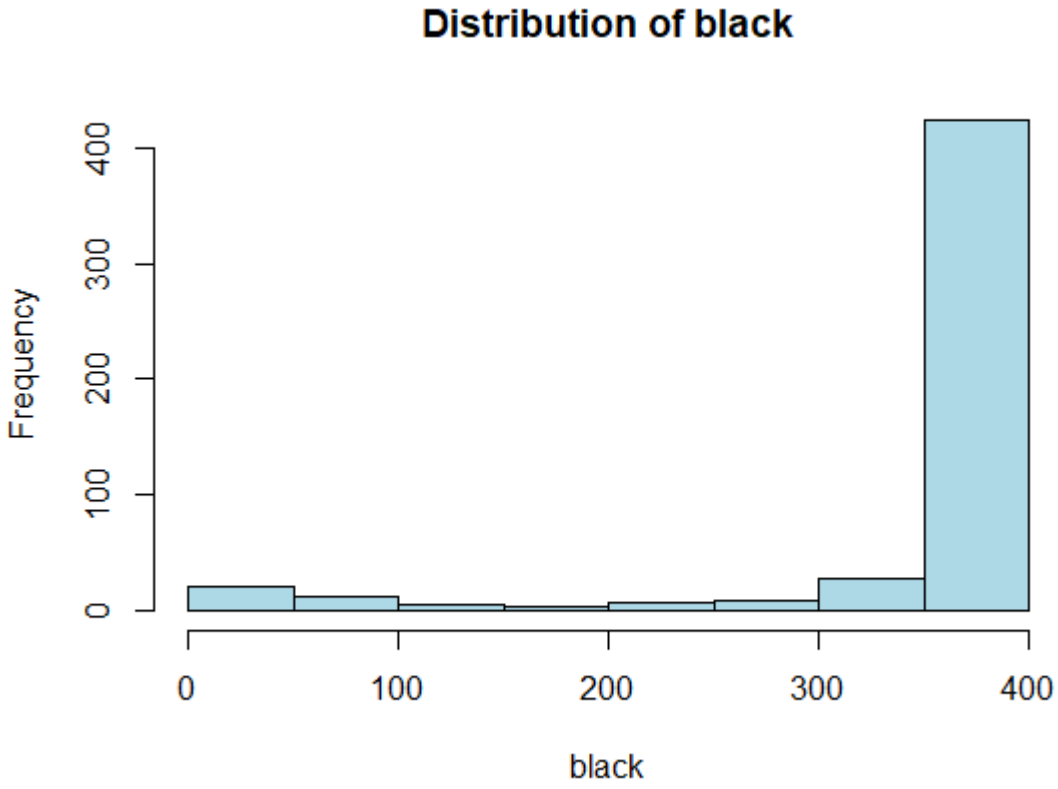

Distribution of age
Frequency
0
50
100
150
0
20
40
60
80
100
age

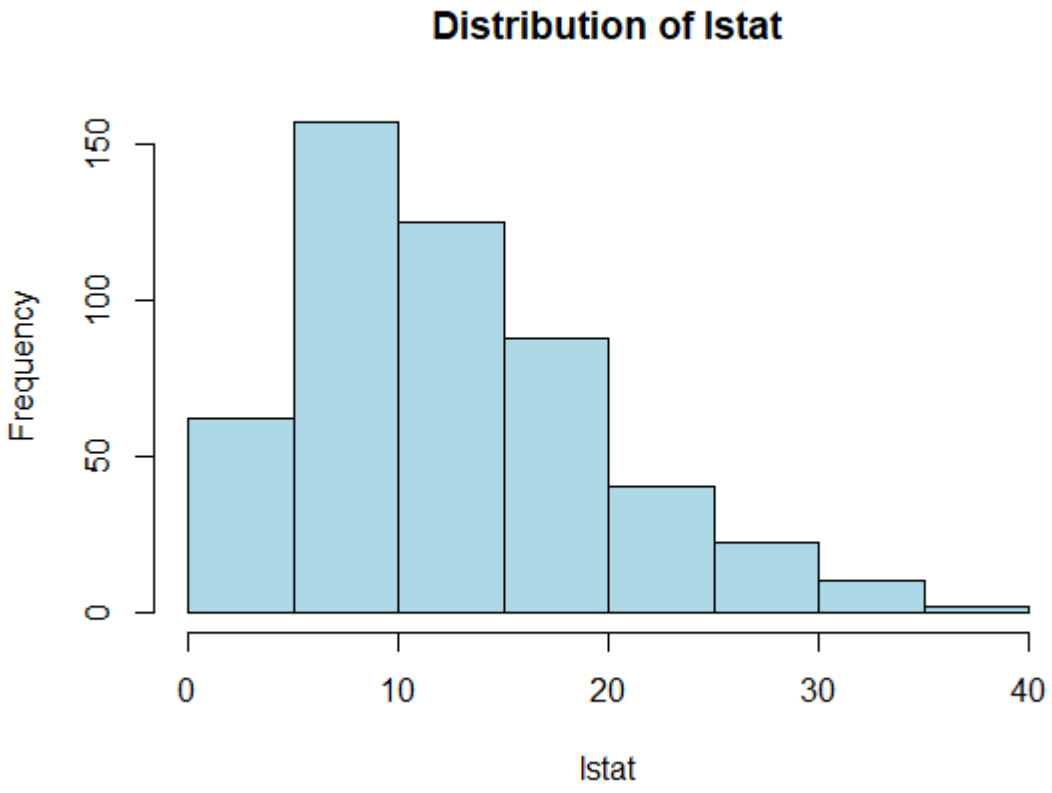

Distribution of dis
Frequency
0
20
40
60
80
100
2
4
6
8
10
12
dis

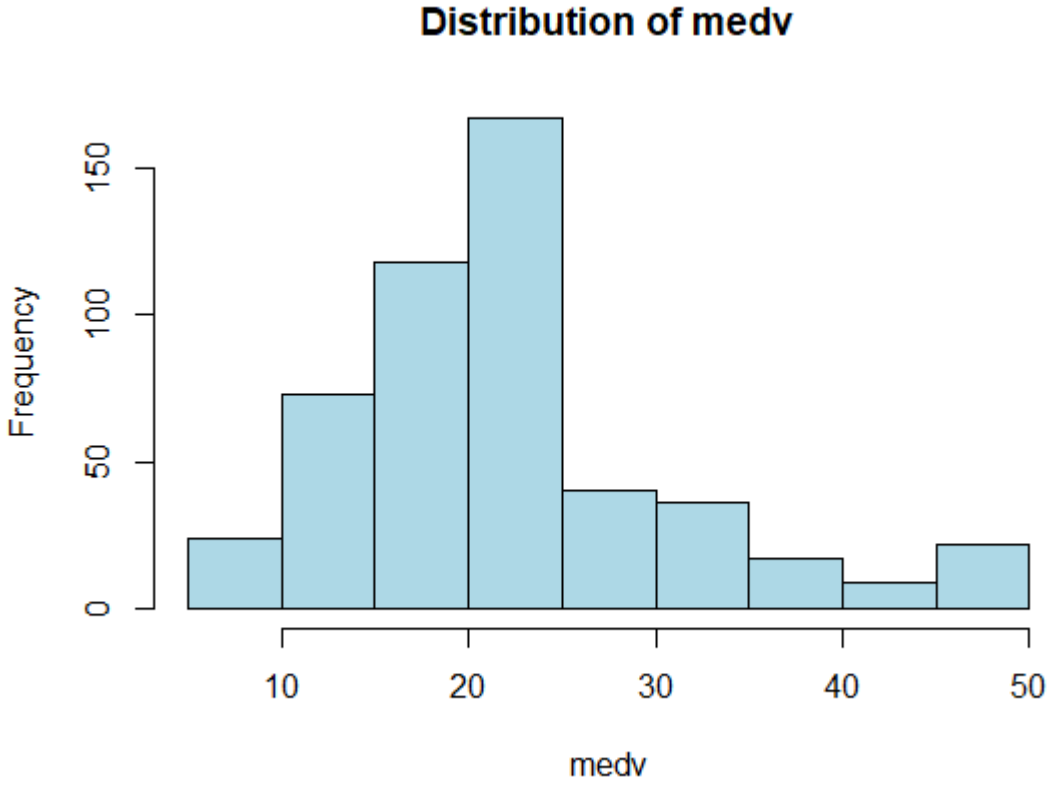

Distribution of rad
Frequency
0
50
100
150
0
5
10
15
20
rad

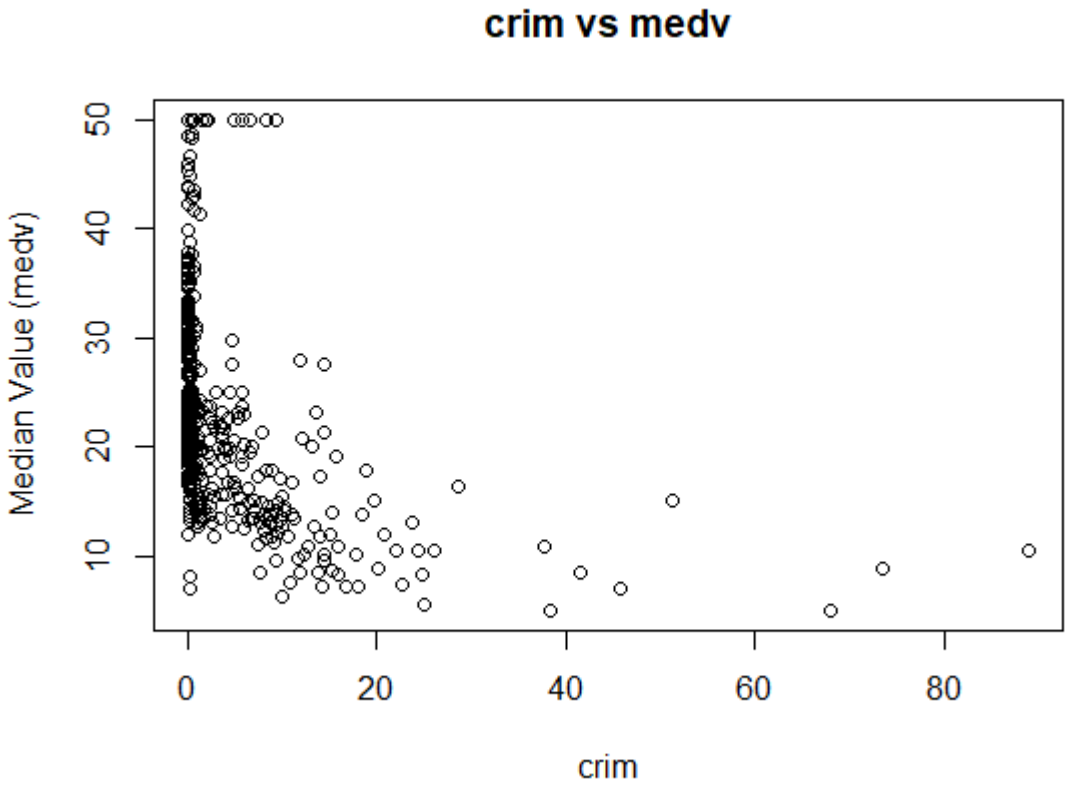

Distribution of tax
Frequency
0
20
40
60
80
100
200
300
400
500
600
700
tax

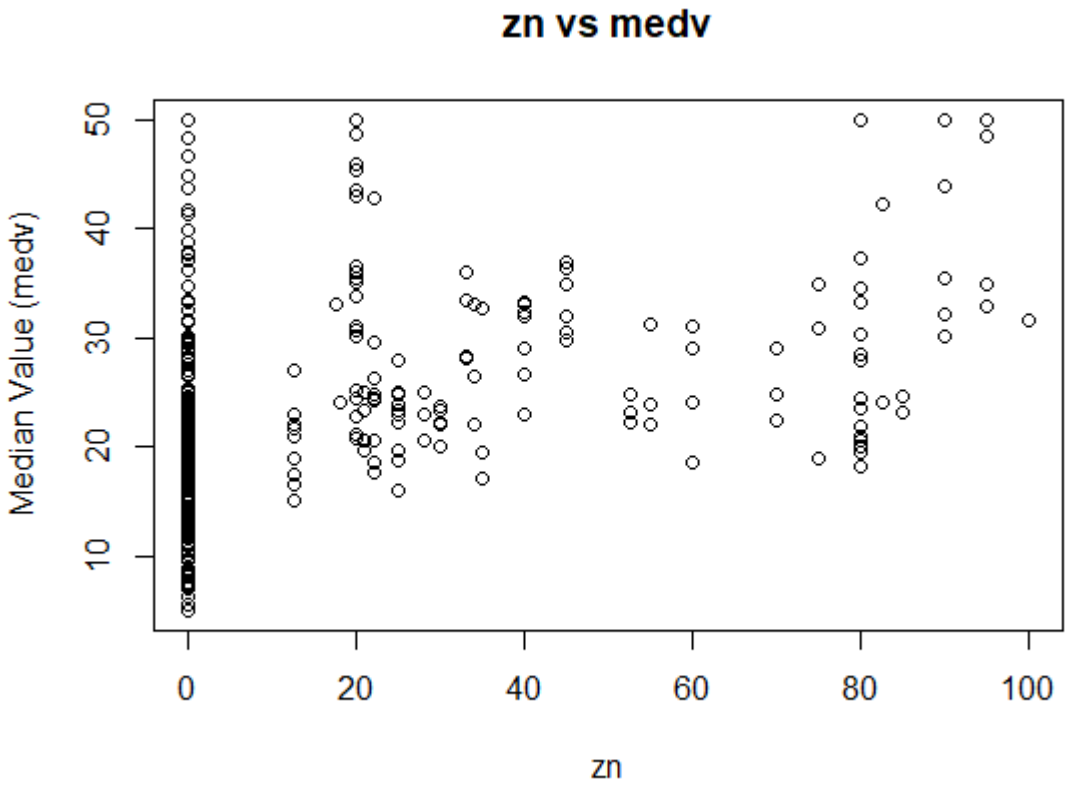

Distribution of ptratio
Frequency
0
50
100
150
12
14
16
18
20
22
ptratio

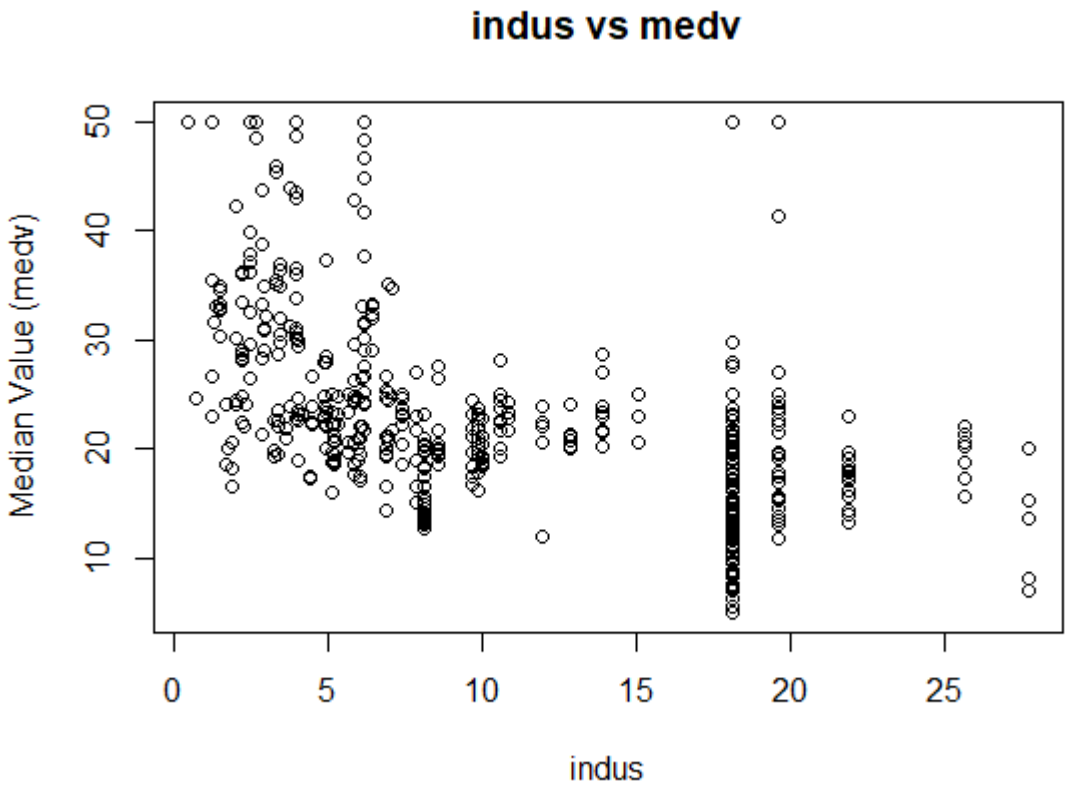

Distribution of black
Frequency
0
100
200
300
400
0
100
200
300
400
black

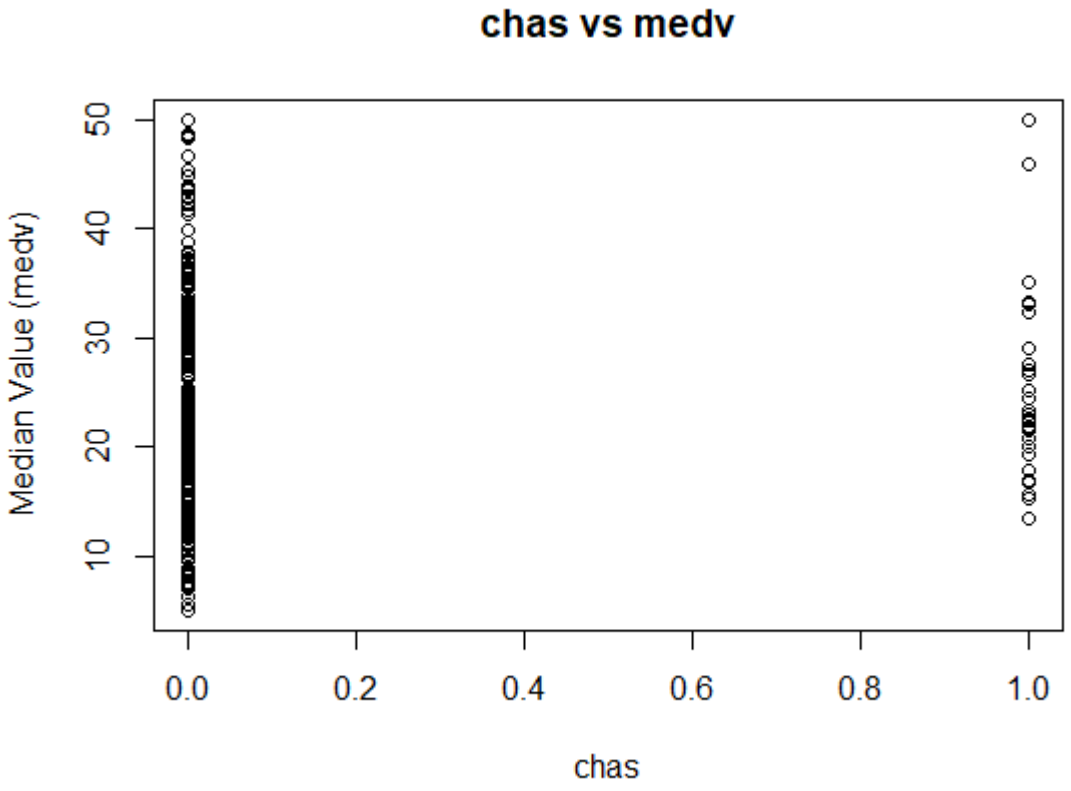

Distribution of lstat
Frequency
0
50
100
150
0
10
20
30
40
lstat

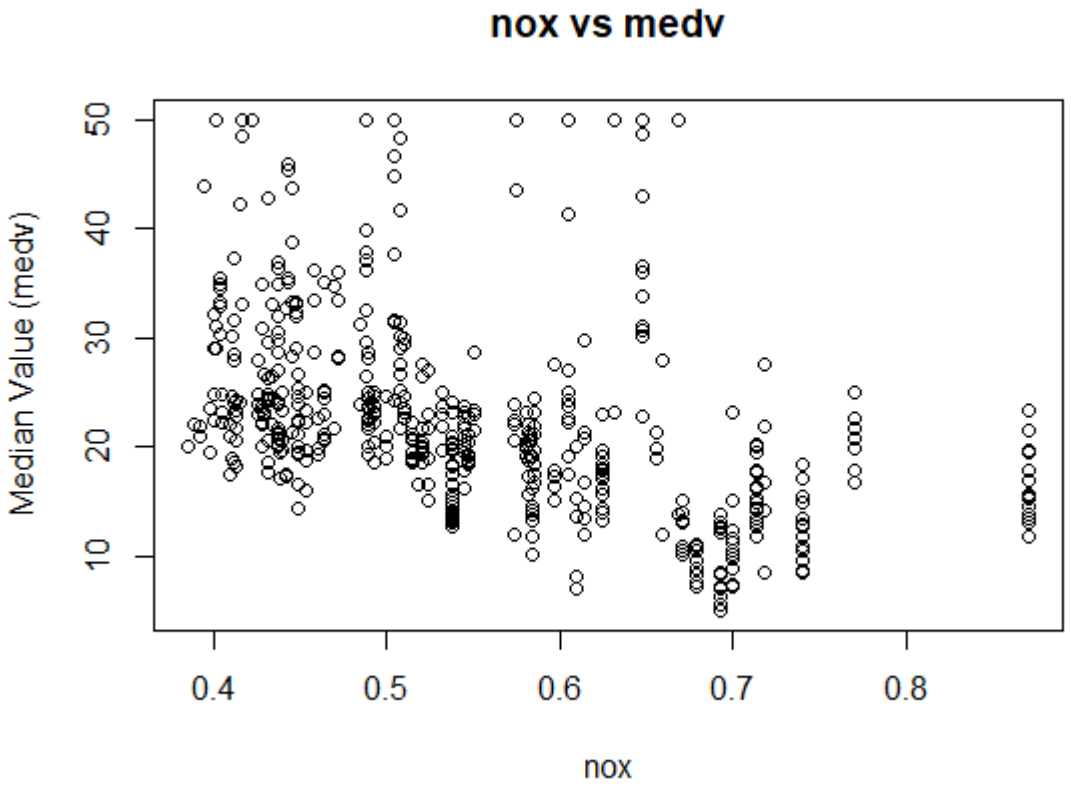

Distribution of medv
Frequency
0
50
100
150
10
20
30
40
50
medv

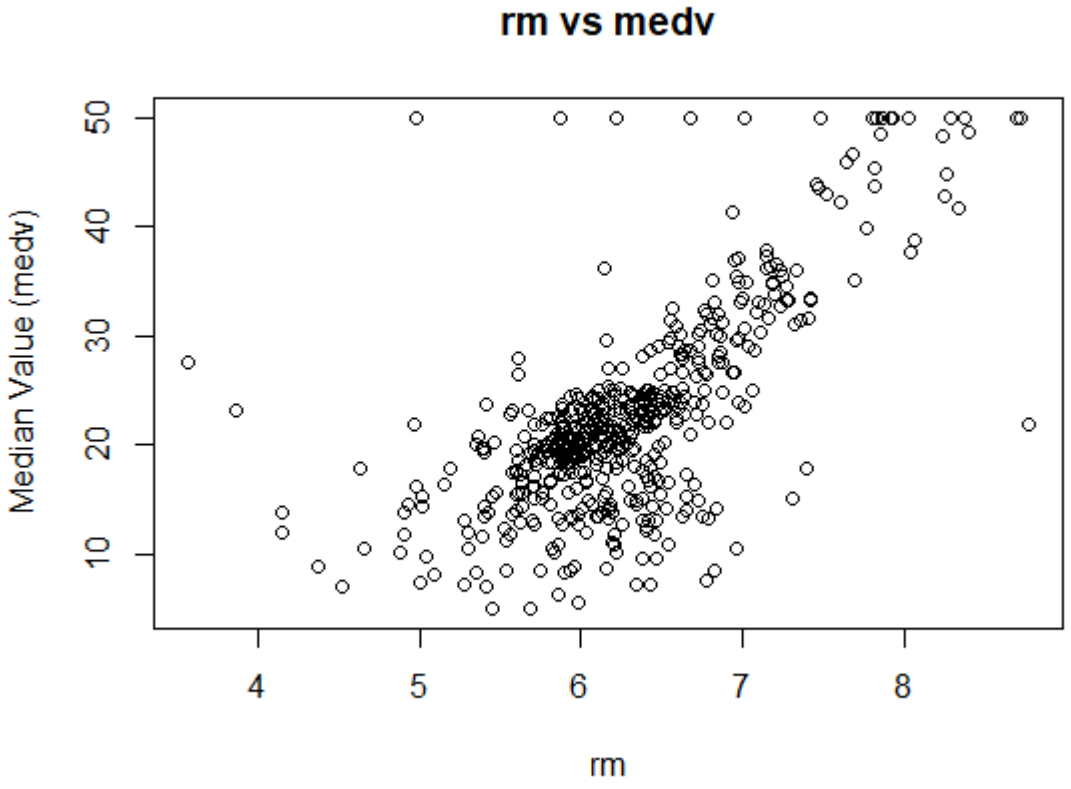

crim vs medv
Median Value (medv)
10
20
30
40
50
0
20
40
60
80
crim

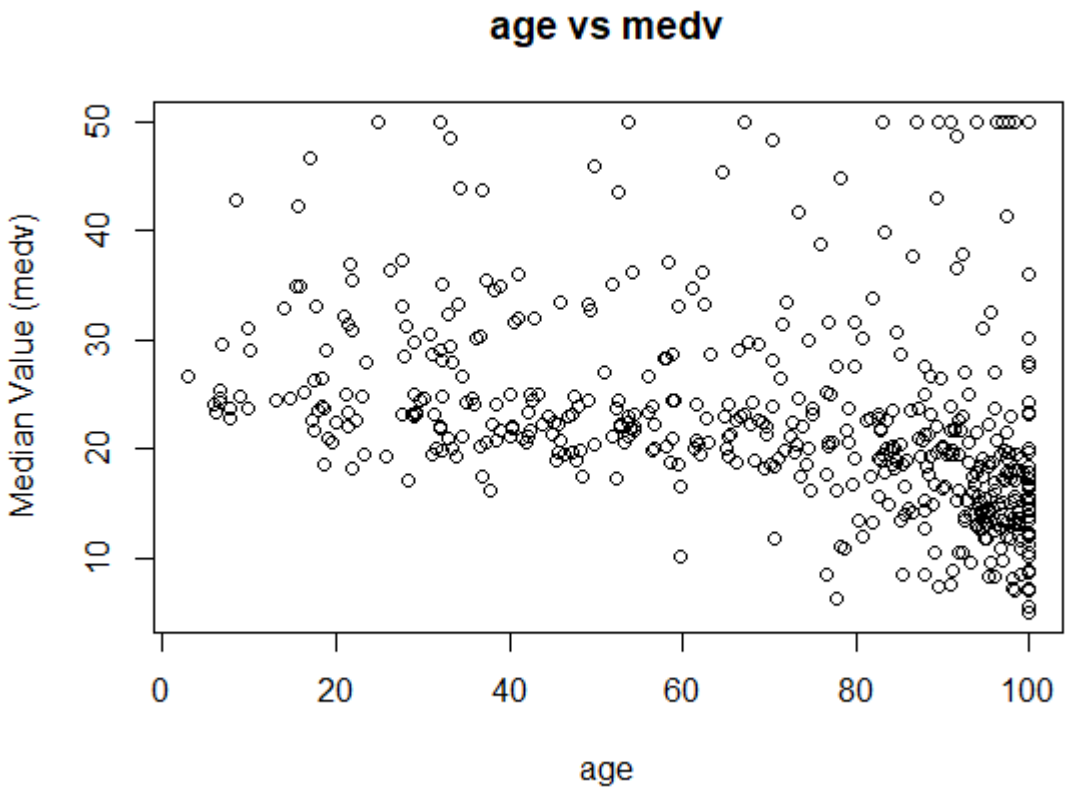

zn vs medv
Median Value (medv)
10
20
30
40
50
0
20
40
60
80
100
zn

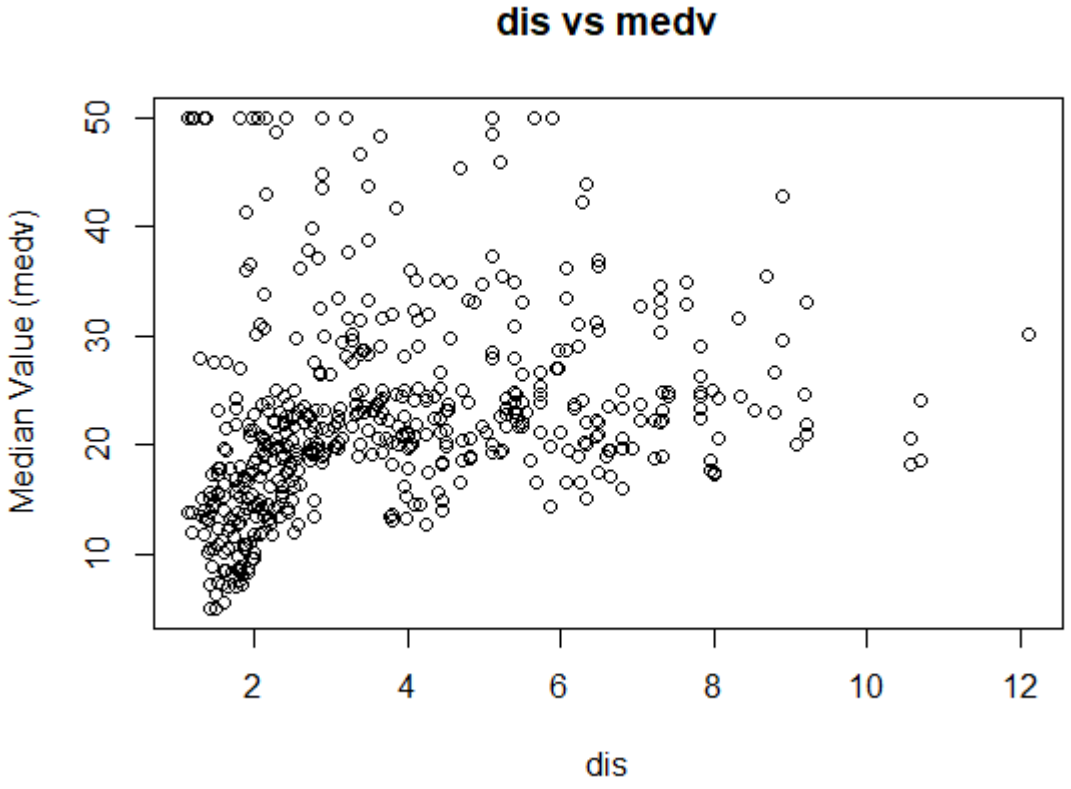

indus vs medv
Median Value (medv)
10
20
30
40
50
0
5
10
15
20
25
indus

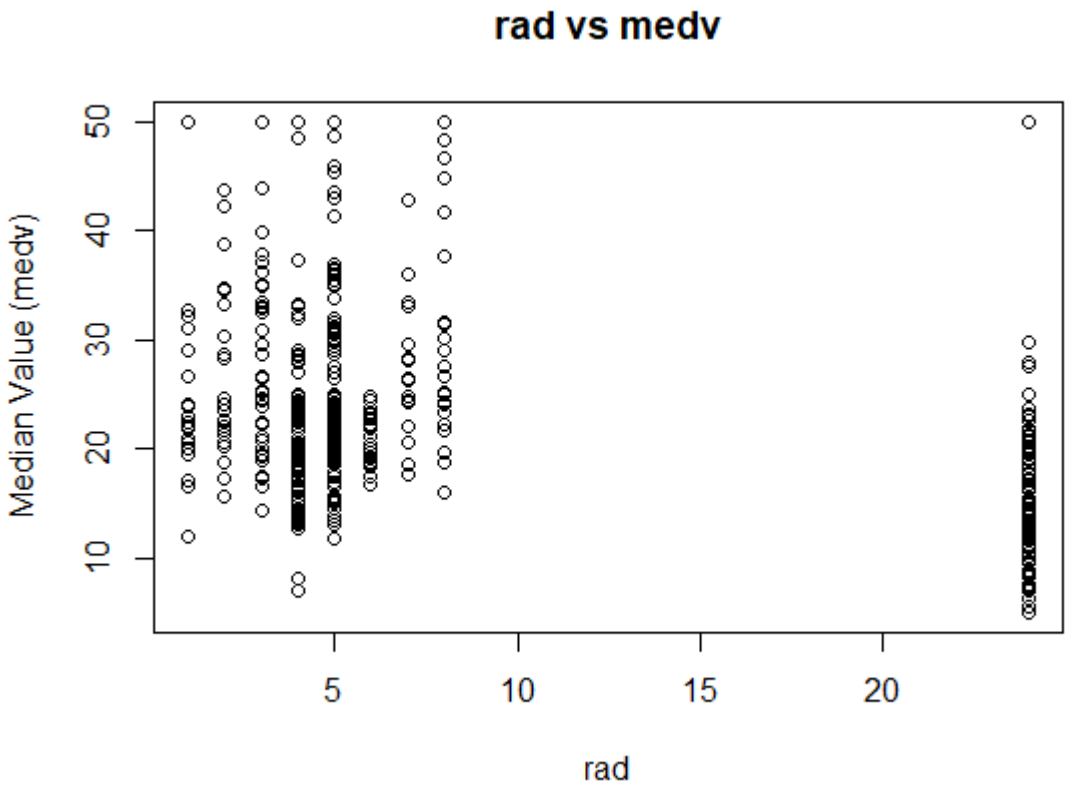

chas vs medv
Median Value (medv)
10
20
30
40
50
0.0
0.2
0.4
0.6
0.8
1.0
chas

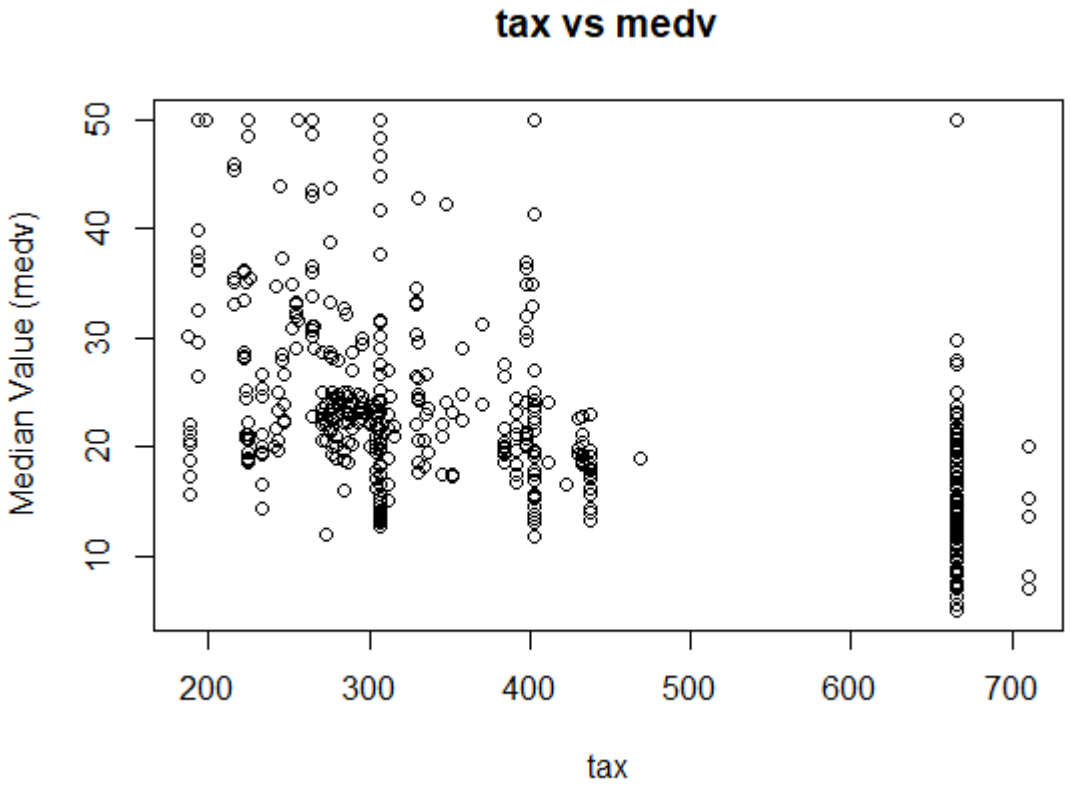

nox vs medv
Median Value (medv)
10
20
30
40
50
0.4
0.5
0.6
0.7
0.8
nox

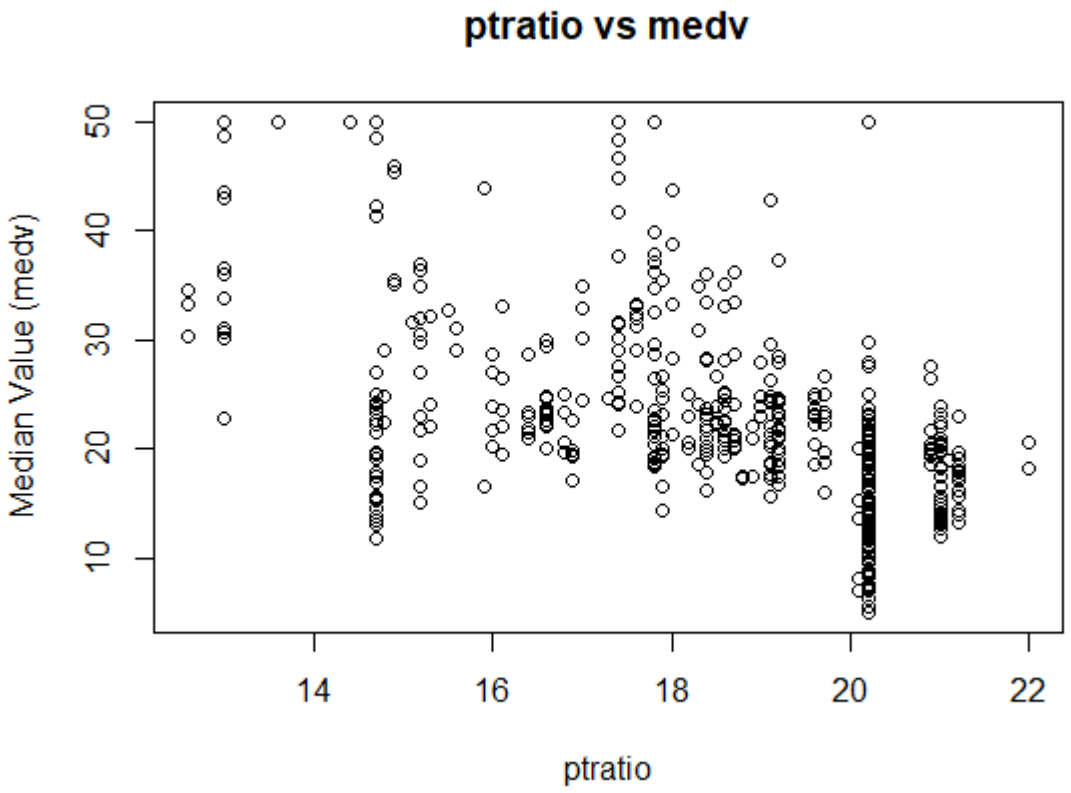

rm vs medv
Median Value (medv)
10
20
30
40
50
4
5
6
7
8
rm

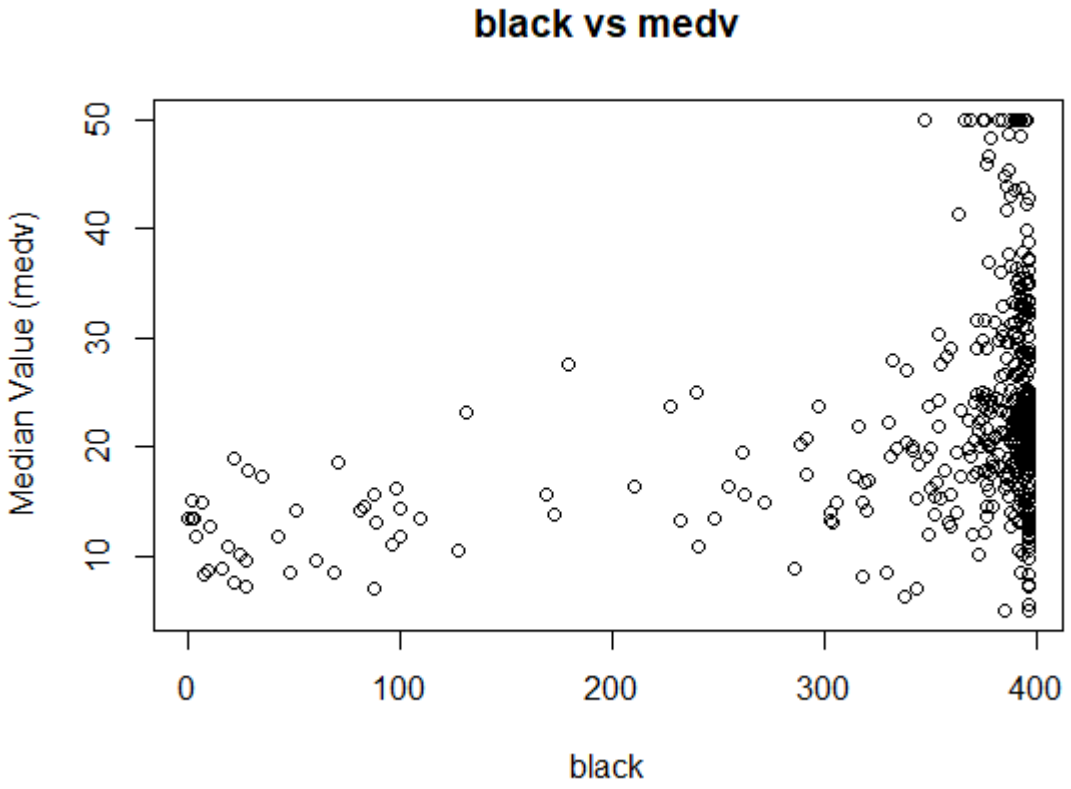

age vs medv
Median Value (medv)
10
20
30
40
50
0
20
40
60
80
100
age

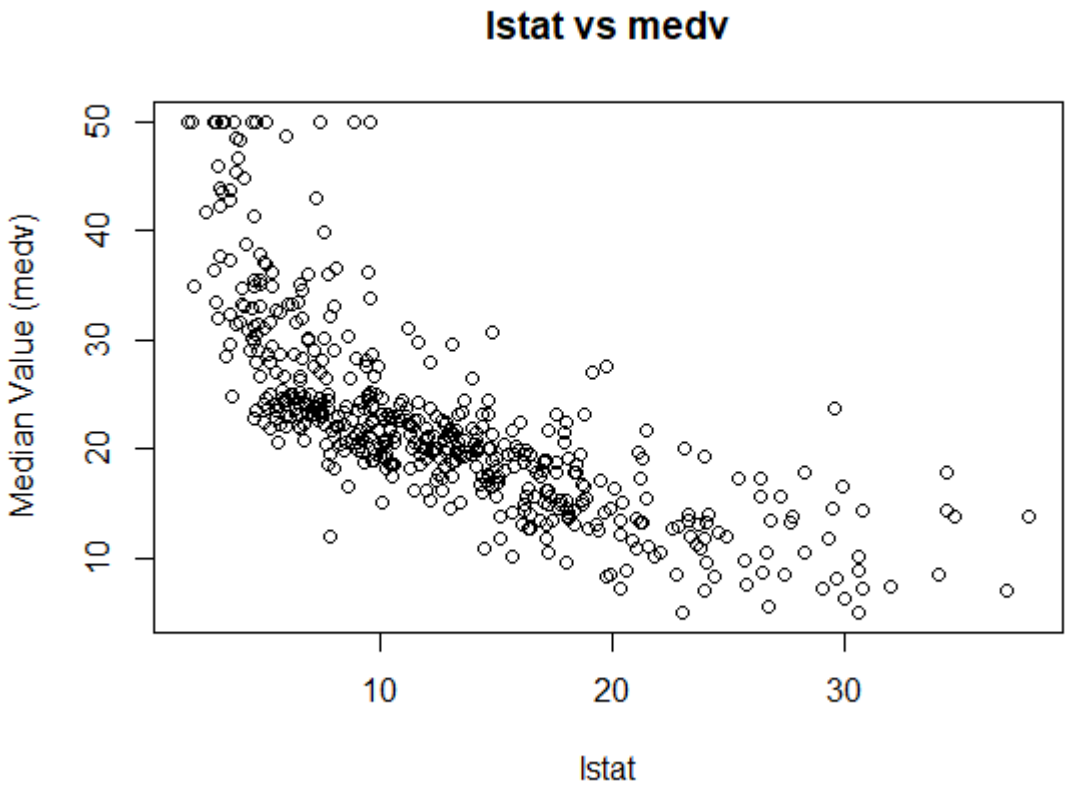

dis vs medv
Median Value (medv)
10
20
30
40
50
2
4
6
8
10
12
dis

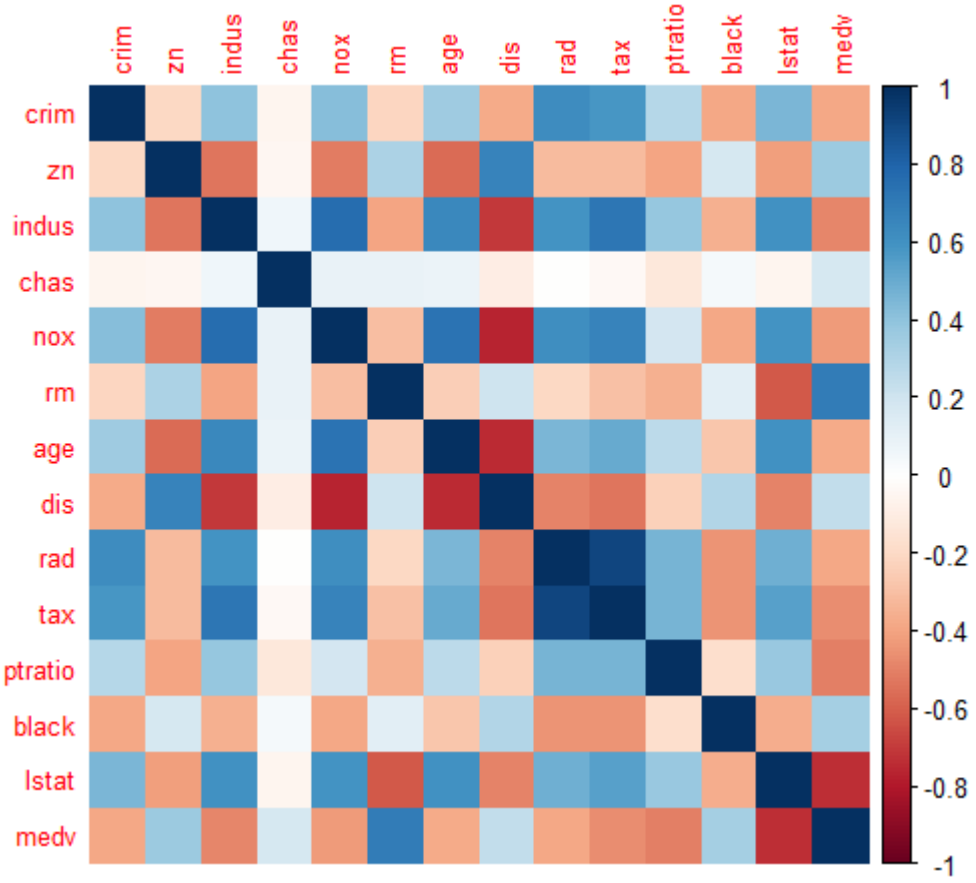

rad vs medv
Median Value (medv)
10
20
30
40
50
5
10
15
20
rad


tax vs medv
Median Value (medv)
10
20
30
40
50
200
300
400
500
600
700
tax


ptratio vs medv
Median Value (medv)
10
20
30
40
50
14
16
18
20
22
ptratio


black vs medv
Median Value (medv)
10
20
30
40
50
0
100
200
300
400
black


lstat vs medv
Median Value (medv)
10
20
30
40
50
10
20
30
lstat


crim
zn
indus
chas
nox
rm
age
dis
rad
tax
ptratio
black
lstat
medv
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

```
Residual standard error: 4.745 on 492 degrees of freedom
Multiple R-squared:  0.7406,    Adjusted R-squared:  0.7338
F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

```
  Actual Predicted
1   24.0  30.00384
2   21.6  25.02556
3   34.7  30.56760
4   33.4  28.60704
5   36.2  27.94352
6   28.7  25.25628
```