EXCERCISE:6(b)

**Write an R Script to find subset of dataset by using subset(), aggregate() functions on iris dataset.**

```
# Load the iris dataset
data(iris)
# Displaying the structure of the iris dataset69
cat("Structure of iris dataset:\n")
str(iris)
# Subset of the iris dataset where Sepal.Length is greater than 5
subset_iris <- subset(iris, Sepal.Length > 5)
cat("\nSubset of iris dataset where Sepal.Length > 5:\n")
print(subset_iris)
# Aggregate function to calculate mean Petal.Length for each Species
aggregate_result <- aggregate(Petal.Length ~ Species, data = iris, FUN = mean)
cat("\nAggregate result - Mean Petal.Length for each Species:\n")
print(aggregate_result)
```

**Output**

```
Structure of iris dataset:
'data.frame':
150 obs. of 5 variables:
$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
$ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
Subset of iris dataset where Sepal.Length > 5:
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1 5.1 3.5 1.4 0.2 setosa
6 5.4 3.9 1.7 0.4 setosa
11 5.4 3.7 1.5 0.2 setosa15 5.8 4.0 1.2 0.2 setosa
16 5.7 4.4 1.5 0.4 setosa
17 5.4 3.9 1.3 0.4 setosa
18 5.1 3.5 1.4 0.3 setosa
19 5.7 3.8 1.7 0.3 setosa
20 5.1 3.8 1.5 0.3 setosa
21 5.4 3.4 1.7 0.2 setosa
22 5.1 3.7 1.5 0.4 setosa
24 5.1 3.3 1.7 0.5 setosa
28 5.2 3.5 1.5 0.2 setosa
29 5.2 3.4 1.4 0.2 setosa
32 5.4 3.4 1.5 0.4 setosa
33 5.2 4.1 1.5 0.1 setosa
```

| | | | | | |
|---|---|---|---|---|---|
| 34 | 5.5 | 4.2 | 1.4 | 0.2 | setosa |
| 37 | 5.5 | 3.5 | 1.3 | 0.2 | setosa |
| 40 | 5.1 | 3.4 | 1.5 | 0.2 | setosa |
| 45 | 5.1 | 3.8 | 1.9 | 0.4 | setosa |
| 47 | 5.1 | 3.8 | 1.6 | 0.2 | setosa |
| 49 | 5.3 | 3.7 | 1.5 | 0.2 | setosa |
| 51 | 7.0 | 3.2 | 4.7 | 1.4 | versicolor |
| 52 | 6.4 | 3.2 | 4.5 | 1.5 | versicolor |
| 53 | 6.9 | 3.1 | 4.9 | 1.5 | versicolor |
| 54 | 5.5 | 2.3 | 4.0 | 1.3 | versicolor |
| 55 | 6.5 | 2.8 | 4.6 | 1.5 | versicolor |
| 56 | 5.7 | 2.8 | 4.5 | 1.3 | versicolor |
| 57 | 6.3 | 3.3 | 4.7 | 1.6 | versicolor |
| 59 | 6.6 | 2.9 | 4.6 | 1.3 | versicolor |
| 7060 | 5.2 | 2.7 | 3.9 | 1.4 | versicolor |
| 62 | 5.9 | 3.0 | 4.2 | 1.5 | versicolor |
| 63 | 6.0 | 2.2 | 4.0 | 1.0 | versicolor |
| 64 | 6.1 | 2.9 | 4.7 | 1.4 | versicolor |
| 65 | 5.6 | 2.9 | 3.6 | 1.3 | versicolor |
| 66 | 6.7 | 3.1 | 4.4 | 1.4 | versicolor |
| 67 | 5.6 | 3.0 | 4.5 | 1.5 | versicolor |
| 68 | 5.8 | 2.7 | 4.1 | 1.0 | versicolor |
| 69 | 6.2 | 2.2 | 4.5 | 1.5 | versicolor |
| 70 | 5.6 | 2.5 | 3.9 | 1.1 | versicolor |
| 71 | 5.9 | 3.2 | 4.8 | 1.8 | versicolor |
| 72 | 6.1 | 2.8 | 4.0 | 1.3 | versicolor |
| 73 | 6.3 | 2.5 | 4.9 | 1.5 | versicolor |
| 74 | 6.1 | 2.8 | 4.7 | 1.2 | versicolor |
| 75 | 6.4 | 2.9 | 4.3 | 1.3 | versicolor |
| 76 | 6.6 | 3.0 | 4.4 | 1.4 | versicolor |
| 77 | 6.8 | 2.8 | 4.8 | 1.4 | versicolor |
| 78 | 6.7 | 3.0 | 5.0 | 1.7 | versicolor |
| 79 | 6.0 | 2.9 | 4.5 | 1.5 | versicolor |
| 80 | 5.7 | 2.6 | 3.5 | 1.0 | versicolor |
| 81 | 5.5 | 2.4 | 3.8 | 1.1 | versicolor |
| 82 | 5.5 | 2.4 | 3.7 | 1.0 | versicolor |
| 83 | 5.8 | 2.7 | 3.9 | 1.2 | versicolor |
| 84 | 6.0 | 2.7 | 5.1 | 1.6 | versicolor |
| 85 | 5.4 | 3.0 | 4.5 | 1.5 | versicolor |
| 86 | 6.0 | 3.4 | 4.5 | 1.6 | versicolor |
| 87 | 6.7 | 3.1 | 4.7 | 1.5 | versicolor |
| 7188 | 6.3 | 2.3 | 4.4 | 1.3 | versicolor |
| 89 | 5.6 | 3.0 | 4.1 | 1.3 | versicolor |
| 90 | 5.5 | 2.5 | 4.0 | 1.3 | versicolor |
| 91 | 5.5 | 2.6 | 4.4 | 1.2 | versicolor |

92 6.1 3.0 4.6 1.4 versicolor
93 5.8 2.6 4.0 1.2 versicolor
95 5.6 2.7 4.2 1.3 versicolor
96 5.7 3.0 4.2 1.2 versicolor
97 5.7 2.9 4.2 1.3 versicolor
98 6.2 2.9 4.3 1.3 versicolor
99 5.1 2.5 3.0 1.1 versicolor
100 5.7 2.8 4.1 1.3 versicolor
101 6.3 3.3 6.0 2.5 virginica
102 5.8 2.7 5.1 1.9 virginica
103 7.1 3.0 5.9 2.1 virginica
104 6.3 2.9 5.6 1.8 virginica
105 6.5 3.0 5.8 2.2 virginica
106 7.6 3.0 6.6 2.1 virginica
108 7.3 2.9 6.3 1.8 virginica
109 6.7 2.5 5.8 1.8 virginica
110 7.2 3.6 6.1 2.5 virginica
111 6.5 3.2 5.1 2.0 virginica
112 6.4 2.7 5.3 1.9 virginica
113 6.8 3.0 5.5 2.1 virginica
114 5.7 2.5 5.0 2.0 virginica
115 5.8 2.8 5.1 2.4 virginica
116 6.4 3.2 5.3 2.3 virginica
72117 6.5 3.0 5.5 1.8 virginica
118 7.7 3.8 6.7 2.2 virginica
119 7.7 2.6 6.9 2.3 virginica
120 6.0 2.2 5.0 1.5 virginica
121 6.9 3.2 5.7 2.3 virginica
122 5.6 2.8 4.9 2.0 virginica
123 7.7 2.8 6.7 2.0 virginica
124 6.3 2.7 4.9 1.8 virginica
125 6.7 3.3 5.7 2.1 virginica
126 7.2 3.2 6.0 1.8 virginica
127 6.2 2.8 4.8 1.8 virginica
128 6.1 3.0 4.9 1.8 virginica
129 6.4 2.8 5.6 2.1 virginica
130 7.2 3.0 5.8 1.6 virginica
131 7.4 2.8 6.1 1.9 virginica
132 7.9 3.8 6.4 2.0 virginica
133 6.4 2.8 5.6 2.2 virginica
134 6.3 2.8 5.1 1.5 virginica
135 6.1 2.6 5.6 1.4 virginica
136 7.7 3.0 6.1 2.3 virginica
137 6.3 3.4 5.6 2.4 virginica
138 6.4 3.1 5.5 1.8 virginica

```
139 6.0 3.0 4.8 1.8 virginica
140 6.9 3.1 5.4 2.1 virginica
141 6.7 3.1 5.6 2.4 virginica
142 6.9 3.1 5.1 2.3 virginica
143 5.8 2.7 5.1 1.9 virginica
73144 6.8 3.2 5.9 2.3 virginica
145 6.7 3.3 5.7 2.5 virginica
146 6.7 3.0 5.2 2.3 virginica
147 6.3 2.5 5.0 1.9 virginica
148 6.5 3.0 5.2 2.0 virginica
149 6.2 3.4 5.4 2.3 virginica
150 5.9 3.0 5.1 1.8 virginica
Aggregate result - Mean Petal.Length for each Species:
Species Petal.Length
1 setosa 1.462
2 versicolor 4.260
3 virginica 5.552
[Execution complete with exit code 0]
```

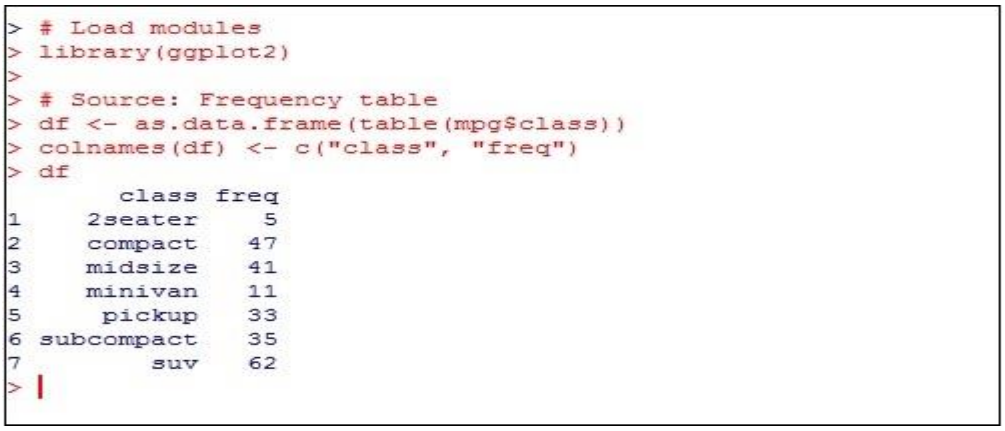
EXCERCISE:8(a)

**a. Implement R Script to create a Pie chart, Bar Chart, scatter plot and Histogram (Introduction to ggplot0 graphics)**

ggplot2 is a popular data visualization package in the R programming language. It was developed by Hadley Wickham and is based on the principles of the "Grammar of Graphics," which provides a systematic and structured approach to creating and understanding data visualizations.

**Create pie chart:**

```
# Load modules
> library(ggplot2)
> # Source: Frequency table
> df <- as.data.frame(table(mpg$class))
> colnames(df) <- c("class", "freq")
```

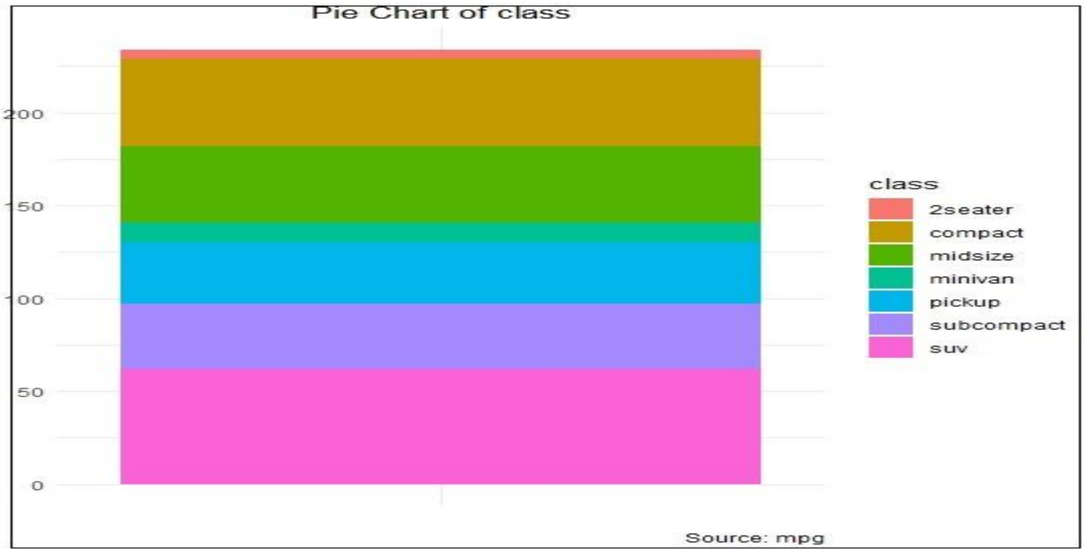


```
> # Load modules
> library(ggplot2)
>
> # Source: Frequency table
> df <- as.data.frame(table(mpg$class))
> colnames(df) <- c("class", "freq")
> df
       class freq
1    2seater    5
2    compact   47
3    midsize   41
4    minivan   11
5     pickup   33
6 subcompact   35
7        suv   62
>
```

The sample chart can be created using the following command –

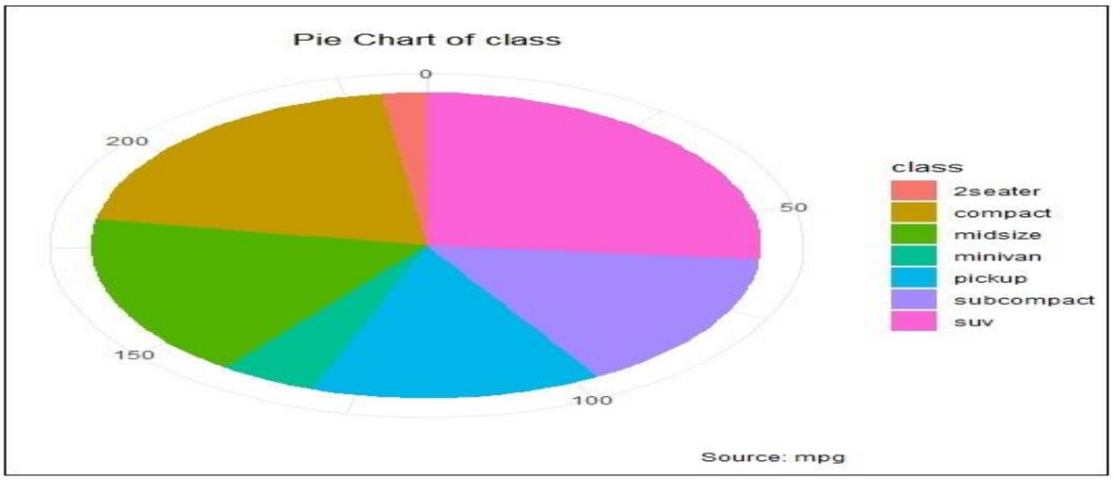
```
> pie <- ggplot(df, aes(x = "", y=freq, fill = factor(class))) +
+ geom_bar(width = 1, stat = "identity") +
+ theme(axis.line = element_blank(),
+    plot.title = element_text(hjust=0.5)) +
+    labs(fill="class",
+       x=NULL,
+       y=NULL,
+       title="Pie Chart of class",
+       caption="Source: mpg")
> pie
```

If you observe the output, the diagram is not created in circular manner as mentioned below –



Creating co-ordinates:

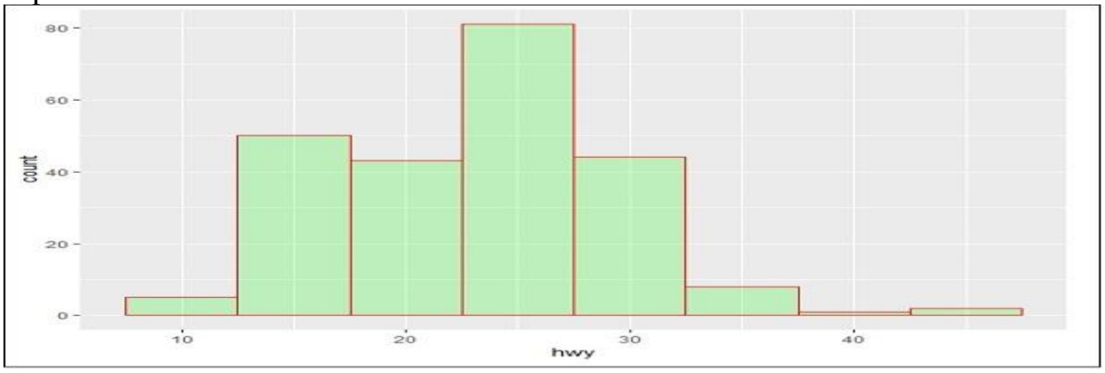
```
> pie + coord_polar(theta = "y", start=0)
```



## Creating Bar Count Plot

The Bar Count plot can be created with below mentioned plot −

```
> # A bar count plot
> p <- ggplot(mpg, aes(x=factor(cyl)))+
+    geom_bar(stat="count")
> p
```



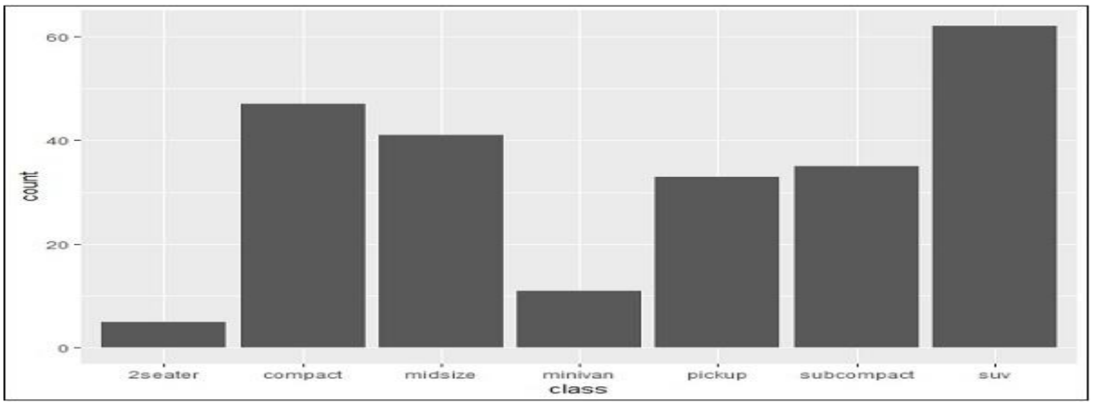
geom_bar() is the function which is used for creating bar plots. It takes the attribute of statistical value called count.

## Histogram

The histogram count plot can be created with below mentioned plot −

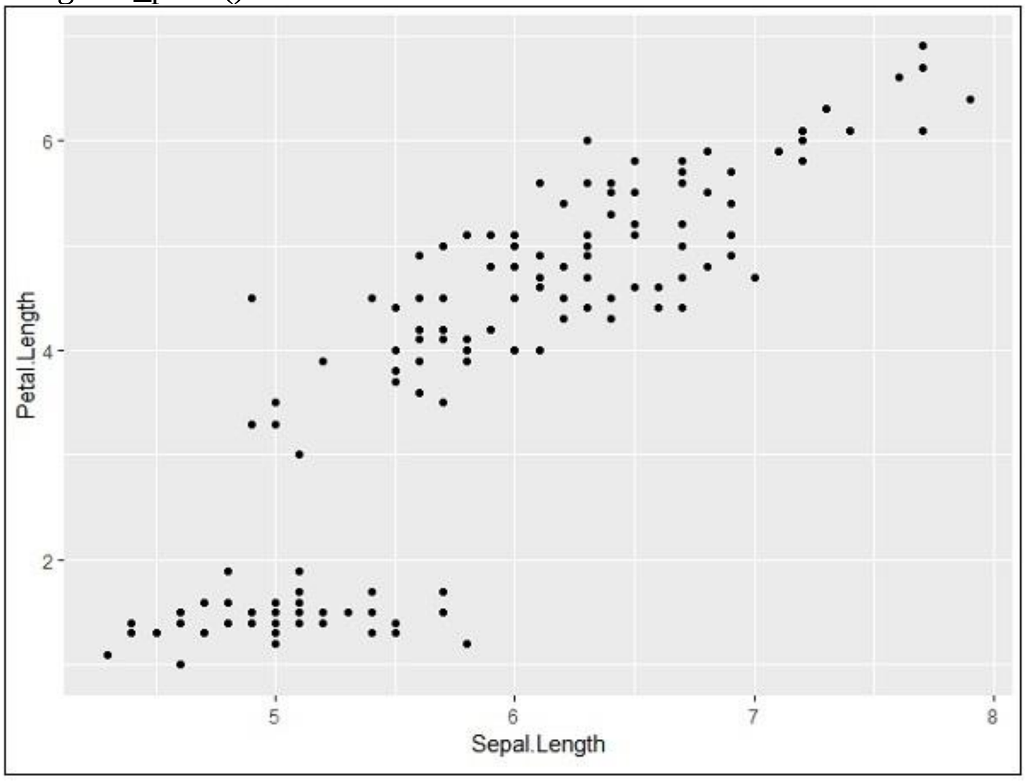
```
> # A historgram count plot
> ggplot(data=mpg, aes(x=hwy)) +
+    geom_histogram( col="red",
+       fill="green",
+       alpha = .2,
+       binwidth = 5)
```

geom_histogram() includes all the necessary attributes for creating a histogram. Here, it takes the attribute of hwy with respective count. The color is taken as per the requirements.



Create Scatter Plot:

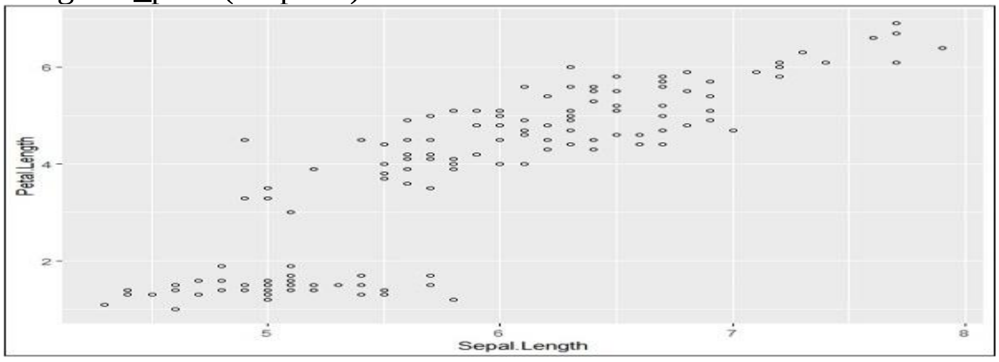
```
> # Basic Scatter Plot
> ggplot(iris, aes(Sepal.Length, Petal.Length)) +
+    geom_point()
```



**Adding attributes**

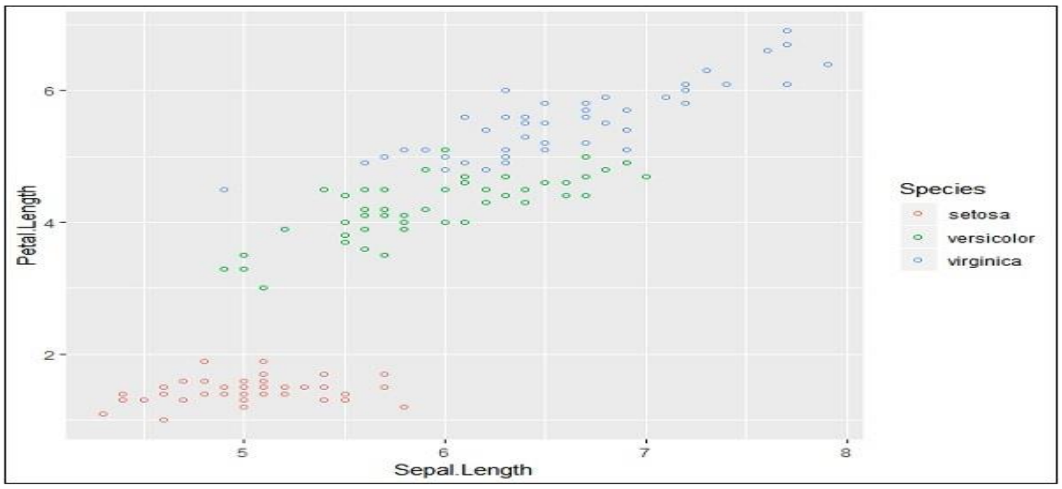
We can change the shape of points with a property called shape in geom_point() function.

```
> # Change the shape of points
> ggplot(iris, aes(Sepal.Length, Petal.Length)) +
+    geom_point(shape=1)
```



We can add color to the points which is added in the required scatter plots.

```
> ggplot(iris, aes(Sepal.Length, Petal.Length, colour=Species)) +
+    geom_point(shape=1)
```



In this example, we have created colors as per species which are mentioned in legends. The three species are uniquely distinguished in the mentioned plot.

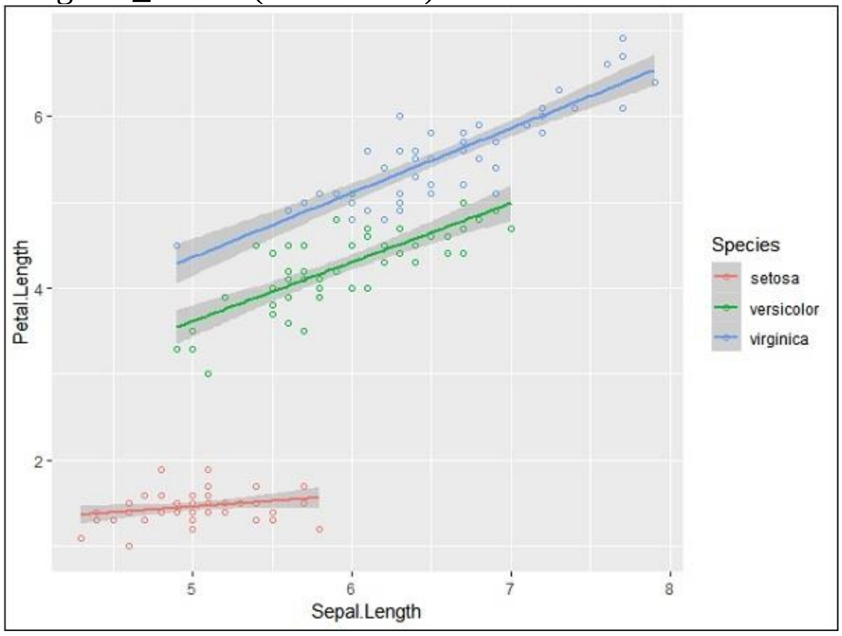
Now we will focus on establishing relationship between the variables.

```
> ggplot(iris, aes(Sepal.Length, Petal.Length, colour=Species)) +
+    geom_point(shape=1) +
+    geom_smooth(method=lm)
```

**geom_smooth** function aids the pattern of overlapping and creating the pattern of required variables.

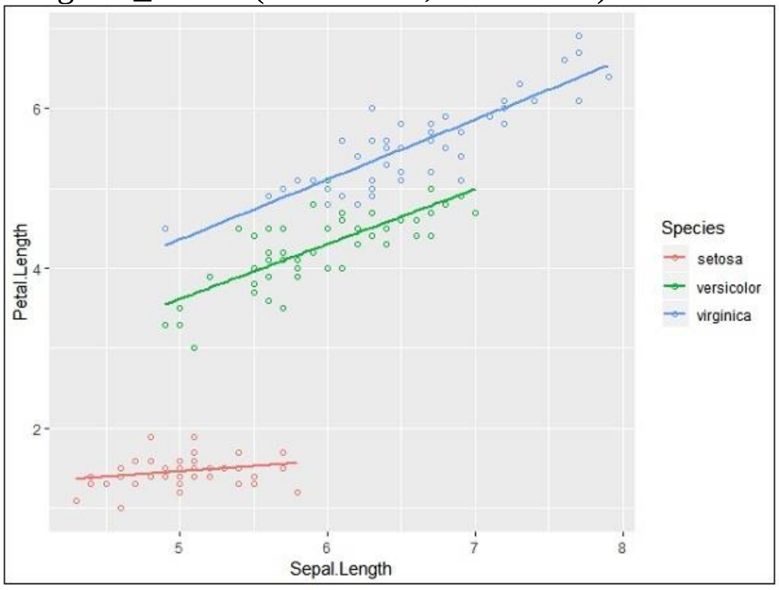
The attribute method “lm” mentions the regression line which needs to be developed.

```
> # Add a regression line
> ggplot(iris, aes(Sepal.Length, Petal.Length, colour=Species)) +
+    geom_point(shape=1) +
+    geom_smooth(method=lm)
```



We can also add a regression line with no shaded confidence region with below mentioned syntax −

```
># Add a regression line but no shaded confidence region
> ggplot(iris, aes(Sepal.Length, Petal.Length, colour=Species)) +
+    geom_point(shape=1) +
+    geom_smooth(method=lm, se=FALSE)
```



Shaded regions represent things other than confidence regions

EXCERCISE:9

**a)Implement R Script to perform Normal,Binomial distributions.**

In a random collection of data from independent sources, it is generally observed that the distribution of data is normal. Which means, on plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shape curve.

R has four in built functions to generate normal distribution. They are described below.

```
dnorm(x, mean, sd)
pnorm(x, mean, sd)
qnorm(p, mean, sd)
rnorm(n, mean, sd)
```

Following is the description of the parameters used in above functions −

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **N** is number of observations(sample size).

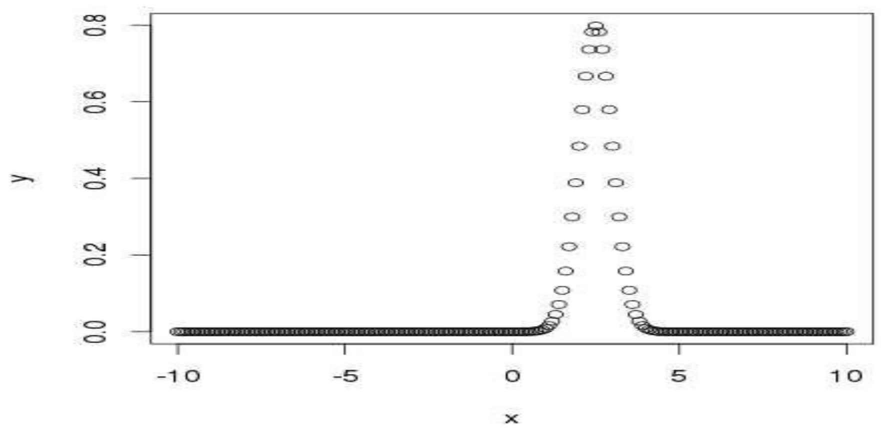
- **Mean** is the mean value of the sample data. It's default value is zero.
- **Sd** is the standard deviation. It's default value is 1.

## dnorm()

- This function gives height of the probability distribution at each point for a given mean and standard deviation.

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.1.
x <- seq(-10, 10, by = .1)

# Choose the mean as 2.5 and standard deviation as 0.5.
y <- dnorm(x, mean = 2.5, sd = 0.5)

# Give the chart file a name.
png(file = "dnorm.png")

plot(x,y)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



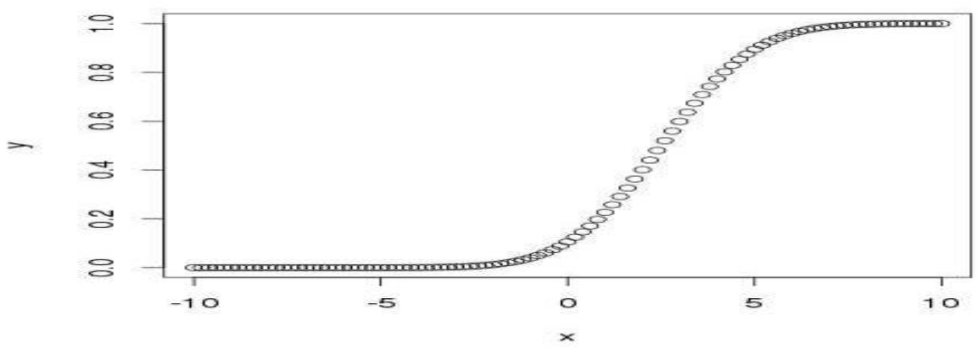
## pnorm()

- This function gives the probability of a normally distributed random number to be less that the value of a given number. It is also called "Cumulative Distribution Function".

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.2.
x <- seq(-10,10,by = .2)

# Choose the mean as 2.5 and standard deviation as 2.
y <- pnorm(x, mean = 2.5, sd = 2)

# Give the chart file a name.
png(file = "pnorm.png")
```

```
# Plot the graph.
plot(x,y)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result

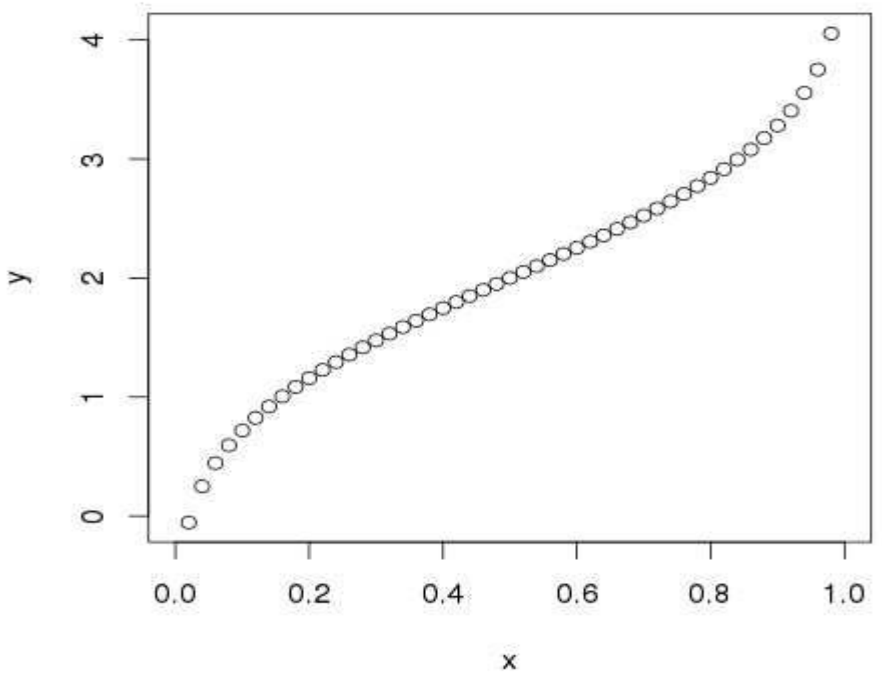


**qnorm()**

- This function takes the probability value and gives a number whose cumulative value matches the probability value.

```
# Create a sequence of probability values incrementing by 0.02.
x <- seq(0, 1, by = 0.02)

# Choose the mean as 2 and standard deviation as 3.
y <- qnorm(x, mean = 2, sd = 1)

# Give the chart file a name.
png(file = "qnorm.png")

# Plot the graph.
plot(x,y)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result

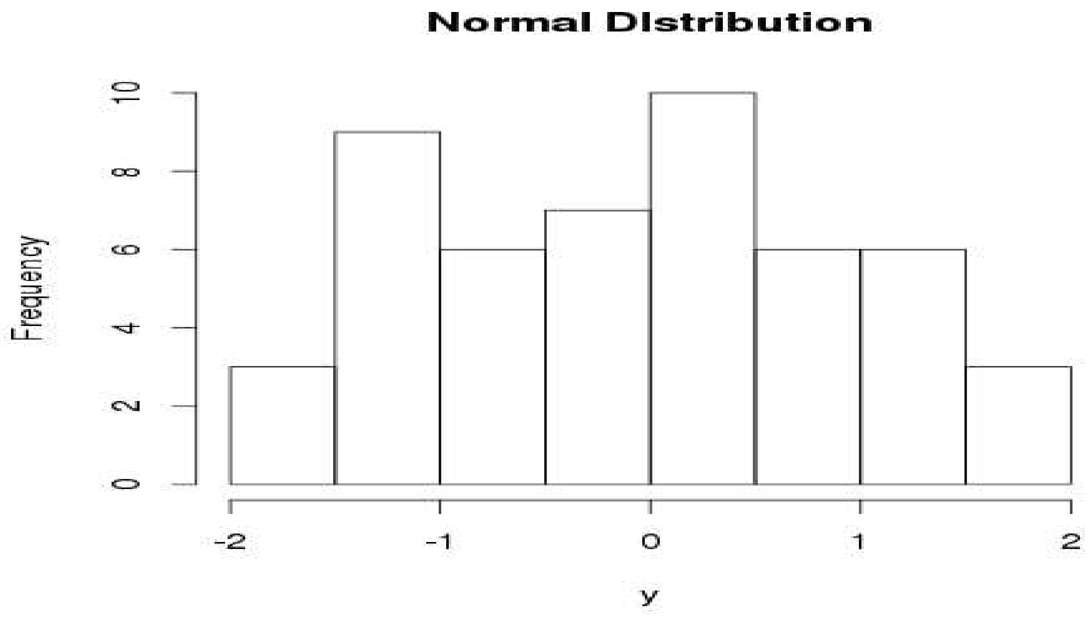


**rnorm()**

- This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

```
# Create a sample of 50 numbers which are normally distributed.
y <- rnorm(50)

# Give the chart file a name.
png(file = "rnorm.png")

# Plot the histogram for this sample.
hist(y, main = "Normal DIstribution")

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result



**R Binomial Distribution:**

- The binomial distribution model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments. For example, tossing of a coin always gives a head or a tail. The probability of finding exactly 3 heads in tossing a coin repeatedly for 10 times is estimated during the binomial distribution.

- R has four in-built functions to generate binomial distribution. They are described below.

- dbinom(x, size, prob)
- pbinom(x, size, prob)
- qbinom(p, size, prob)
- rbinom(n, size, prob)

Following is the description of the parameters used −

- **x** is a vector of numbers.
- **P** is a vector of probabilities.
- **N** is number of observations.
- **Size** is the number of trials.
- **Prob** is the probability of success of each trial.

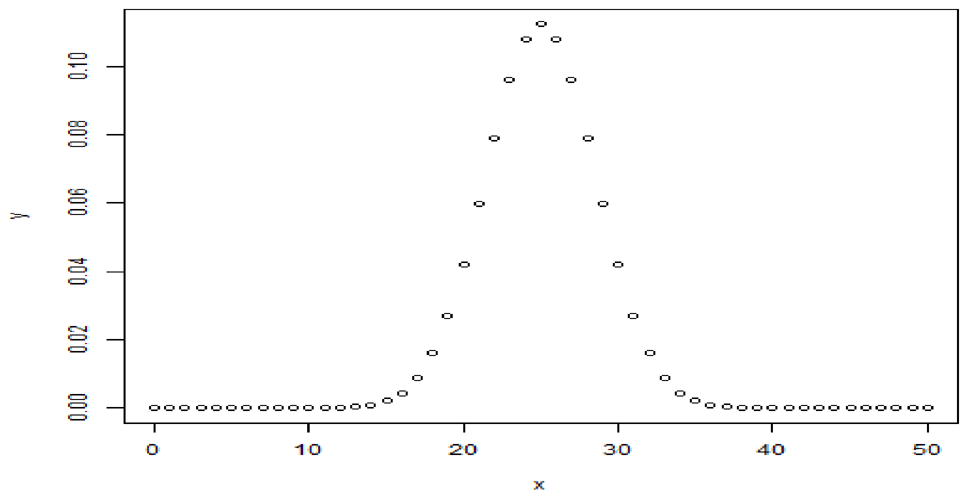
**dbinom()**

- This function gives the probability density distribution at each point.

```
# Create a sample of 50 numbers which are incremented by 1.
x <- seq(0,50,by = 1)

# Create the binomial distribution.
y <- dbinom(x,50,0.5)
```

```
# Give the chart file a name.
png(file = "dbinom.png")

# Plot the graph for this sample.
plot(x,y)

# Save the file.
dev.off()
```

- When we execute the above code, it produces the following result −



**pbinom()**

- This function gives the cumulative probability of an event. It is a single value representing the probability.

```
# Probability of getting 26 or less heads from a 51 tosses of a coin.
x <- pbinom(26,51,0.5)

print(x)
```

- When we execute the above code, it produces the following result

```
[1] 0.610116
```

**qbinom()**

- This function takes the probability value and gives a number whose cumulative value matches the probability value.

```
# How many heads will have a probability of 0.25 will come out when a coin
# is tossed 51 times.
```

```
x <- qbinom(0.25,51,1/2)

print(x)
```

- When we execute the above code, it produces the following result

```
[1] 23
```

**rbinom()**

- This function generates required number of random values of given probability from a given sample.

```
# Find 8 random values from a sample of 150 with probability of 0.4.
x <- rbinom(8,150,.4)

print(x)
```

- When we execute the above code, it produces the following result

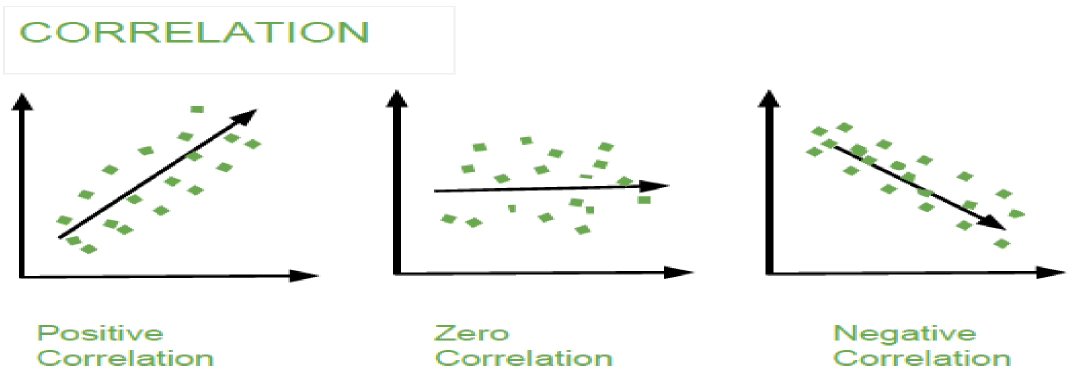
```
[1] 58 61 59 66 55 60 61 67
```

**b)Implement R Script to perform correlation, Linear and multiple regression.**

Correlation and regression analysis are two fundamental statistical techniques used to examine the relationships between variables. R Programming Language is a powerful programming language and environment for statistical computing and graphics, making it an excellent choice for conducting these analyses.

**Correlation Analysis**

Correlation analysis is a statistical technique used to measure the strength and direction of the relationship between two continuous variables.It can take values between -1 (perfect negative correlation) and 1 (perfect positive correlation), with 0 indicating no linear correlation.



```
# Sample data
study_hours <- c(5, 7, 3, 8, 6, 9)
exam_scores <- c(80, 85, 60, 90, 75, 95)
# Calculate Pearson correlation
```

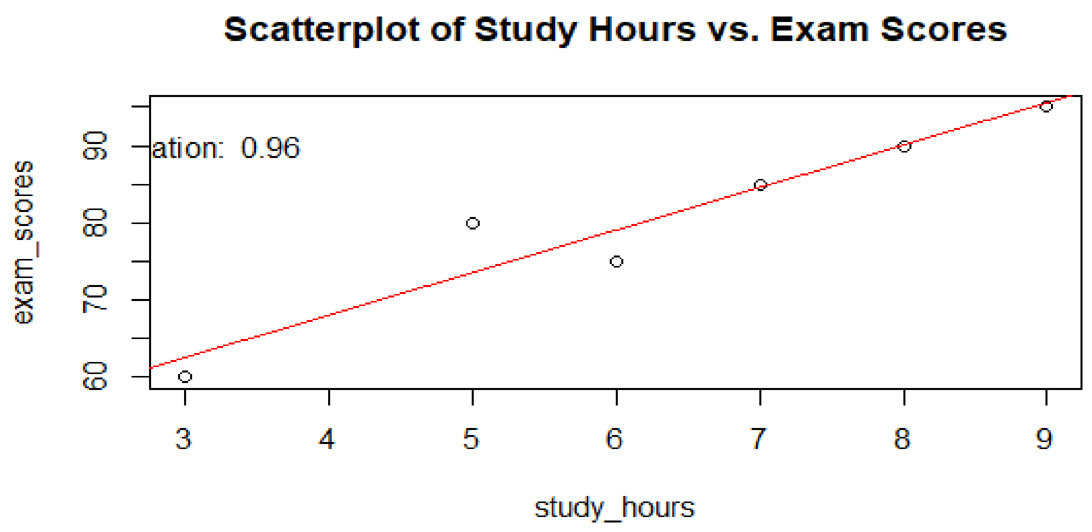
```
correlation <- cor(study_hours, exam_scores)
correlation
```

**Output:**

```
[1] 0.9569094
```

```
# Visualize the data and correlation
plot(study_hours, exam_scores, main = "Scatterplot of Study Hours vs. Exam Scores")
 # Add regression line
abline(lm(exam_scores ~ study_hours), col = "red")
text(3, 90, paste("Correlation: ", round(correlation, 2)))
```

**Output:**



**Simple Linear Regression in R**

Suppose we want to perform a simple linear regression to predict exam scores (exam_scores) based on the number of study hours (study_hours).

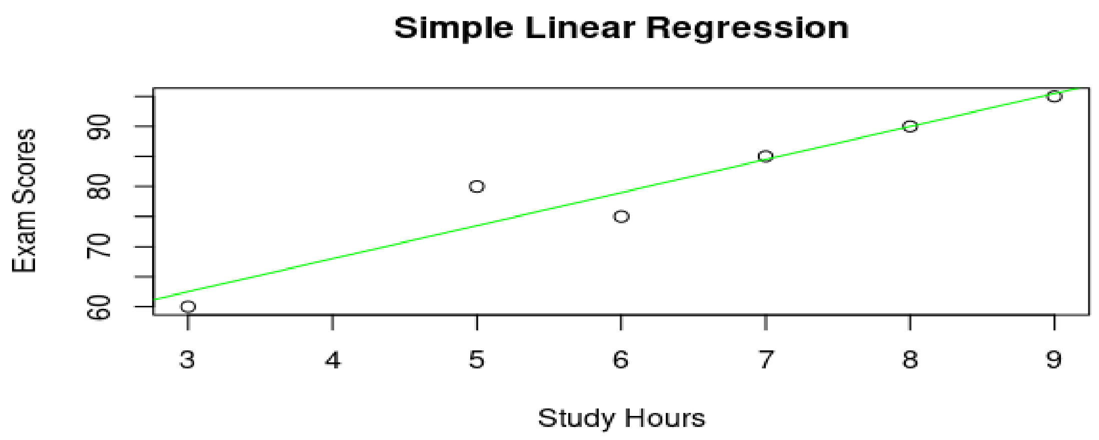
```
# Sample data
study_hours <- c(5, 7, 3, 8, 6, 9)
exam_scores <- c(80, 85, 60, 90, 75, 95)
# Perform simple linear regression
regression_model <- lm(exam_scores ~ study_hours)
# View the summary of the regression results
summary(regression_model)
```

**Output:**

```
Call:
lm(formula            =            exam_scores            ~            study_hours)
Residuals:
       1             2             3             4             5             6
 6.50e+00      5.00e-01     -2.50e+00     -1.11e-15     -4.00e+00     -5.00e-01
Coefficients:
       Estimate        Std.         Error          t          value       Pr(>|t|)
```

```
(Intercept)         46.0000                        5.5356           8.310          0.00115     **
study_hours              5.5000                        0.8345          6.591         0.00275     **
---
Signif.    codes:       0     ‘***’     0.001     ‘**’     0.01     ‘*’     0.05     ‘.’     0.1     ‘     ’     1
Residual        standard        error:        4.031        on        4        degrees        of        freedom
Multiple      R-squared:          0.9157,                  Adjusted      R-squared:          0.8946
F-statistic: 43.44 on 1 and 4 DF,  p-value: 0.002745
```

```
#Visualize the data and regression line:
plot(study_hours, exam_scores, main = "Simple Linear Regression",
    xlab = "Study Hours", ylab = "Exam Scores")
abline(regression_model, col = "Green")
```
output:



**Multiple Linear Regression Example in R**

We’ll use a dataset that contains information about the price of cars based on various attributes like engine size, horsepower, and the number of cylinders. Our goal is to build a multiple linear regression model to predict car prices based on these attributes. We’ll use the mtcars dataset, which is built into R.

```
# Load the mtcars dataset
data(mtcars)
# Perform multiple linear regression
regression_model <- lm(mpg ~ wt + hp + qsec + am, data = mtcars)
# View the summary of the regression results
summary(regression_model)
```
**Output:**

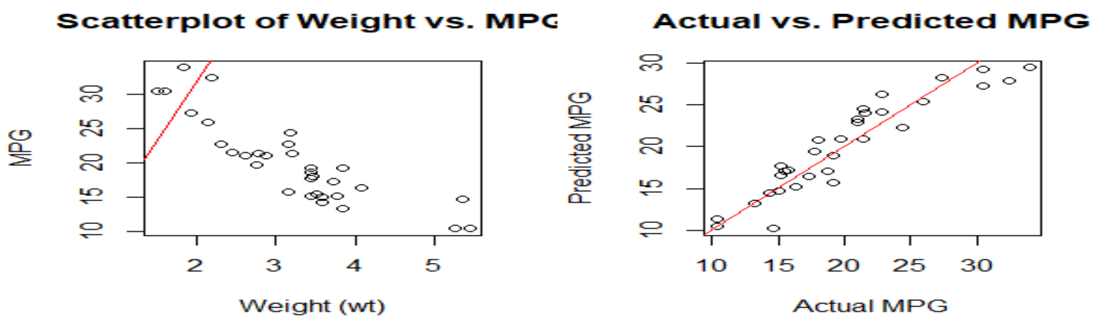
```
Call:
lm(formula    =    mpg    ~    wt    +    hp    +    qsec    +    am,    data    =    mtcars)
Residuals:
   Min                         1Q         Median                          3Q                        Max
-3.4975           -1.5902          -0.1122                        1.1795                        4.5404
Coefficients:
           Estimate            Std.              Error            t               value              Pr(>|t|)
(Intercept)     17.44019                       9.31887                1.871             0.07215       .
wt                                       -3.23810              0.88990          -3.639            0.00114     **
hp                                          -0.01765              0.01415          -1.247             0.22309
```

```
qsec        0.81060    0.43887   1.847  0.07573 .
am          2.92550    1.39715   2.094  0.04579 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 2.435 on 27 degrees of freedom
Multiple R-squared:  0.8579,	Adjusted R-squared:  0.8368
F-statistic: 40.74 on 4 and 27 DF,  p-value: 4.589e-11
```

Visualize the data and regression line:

```
# Visualize the data and regression line for one variable (wt) and the
#actual vs.predicted values
# Create a 1x2 grid of plots
par(mfrow = c(1, 2))
# Plot 1: Scatterplot of Weight (wt) vs. MPG
plot(mtcars$wt, mtcars$mpg, main = "Scatterplot of Weight vs. MPG",
xlab = "Weight (wt)", ylab = "MPG")
abline(regression_model$coefficients["wt"], regression_model$coefficients["(Intercept)"],
col = "red")
# Plot 2: Actual vs. Predicted MPG
predicted_mpg <- predict(regression_model, newdata = mtcars)
plot(mtcars$mpg, predicted_mpg, main = "Actual vs. Predicted MPG",
xlab = "Actual MPG", ylab = "Predicted MPG")
abline(0, 1, col = "red")
```

**Output**:


Scatterplot of Weight vs. MPG
MPG
Weight (wt)
Actual vs. Predicted MPG
Predicted MPG
Actual MPG