

## Practical No : 1

### AIM – Introduction to R

R and RStudio are separate downloads and installations. R is the underlying statistical computing environment, but using R alone is no fun. RStudio is a graphical integrated development environment (IDE) that makes using R much easier and more interactive. You need to install R before you install RStudio. After installing both programs, you will need to install the tidyverse package from within RStudio. Follow the instructions below for your operating system, and then follow the instructions to install tidyverse.

#### Windows

If you already have R and RStudio installed

- Open RStudio, and click on 'Help' > 'Check for updates'. If a new version is available, quit RStudio, and download the latest version for RStudio.
- To check which version of R you are using, start RStudio and the first thing that appears in the console indicates the version of R you are running. Alternatively, you can type `sessionInfo()`, which will also display which version of R you are running. Go to the [CRAN website](#) and check whether a more recent version is available. If so, please download and install it. You can [check here](#) for more information on how to remove old versions from your system if you wish to do so.

If you don't have R and RStudio installed

- Download R from the [CRAN website](#).
- Run the .exe file that was just downloaded
- Go to the [RStudio download page](#)
- Under *Installers* select **RStudio x.yy.zzz - Windows Vista/7/8/10** (where x, y, and z represent version numbers)
- Double click the file to install it
- Once it's installed, open RStudio to make sure it works and you don't get any error messages.

#### What is R? What is RStudio?

The term "R" is used to refer to both the programming language and the software that interprets the scripts written using it.

[RStudio](#) is currently a very popular way to not only write your R scripts but also to interact with the R software. To function correctly, RStudio needs R and therefore both need to be installed on your computer.

To make it easier to interact with R, we will use RStudio. RStudio is the most popular IDE (Integrated Development Environment) for R. An IDE is a piece of software that provides tools to make programming easier.

## **Why learn R?**

R does not involve lots of pointing and clicking, and that's a good thing. The learning curve might be steeper than with other software, but with R, the results of your analysis do not rely on remembering a succession of pointing and clicking, but instead on a series of written commands, and that's a good thing! So, if you want to redo your analysis because you collected more data, you don't have to remember which button you clicked in which order to obtain your results; you just have to run your script again.

Working with scripts makes the steps you used in your analysis clear, and the code you write can be inspected by someone else who can give you feedback and spot mistakes.

Working with scripts forces you to have a deeper understanding of what you are doing and facilitates your learning and comprehension of the methods you use.

## **R code is great for reproducibility**

Reproducibility is when someone else (including your future self) can obtain the same results from the same dataset when using the same analysis.

R integrates with other tools to generate manuscripts from your code. If you collect more data, or fix a mistake in your dataset, the figures and the statistical tests in your manuscript are updated automatically.

An increasing number of journals and funding agencies expect analyses to be reproducible, so knowing R will give you an edge with these requirements.

## **R is interdisciplinary and extensible**

With 10,000+ packages that can be installed to extend its capabilities, R provides a framework that allows you to combine statistical approaches from many scientific disciplines to best suit the analytical framework you need to analyze your data. For instance, R has packages for image analysis, GIS, time series, population genetics, and a lot more.

## **R works on data of all shapes and sizes**

The skills you learn with R scale easily with the size of your dataset. Whether your dataset has hundreds or millions of lines, it won't make much difference to you.

R is designed for data analysis. It comes with special data structures and data types that make handling of missing data and statistical factors convenient.

R can connect to spreadsheets, databases, and many other data formats, on your computer or on the web.

## Practical No : 2

### AIM – Vectors, Lists and Vectors

#### Vectors:

A vector is a basic data structure in R that contains elements of the same type.

Types of vectors include

- numeric
- character
- logical vectors.

Operations on vectors include:

- Indexing
- Slicing
- Performing arithmetic operations

#### Lists:

A list is a data structure in R that can contain elements of different types.

Lists can contain vectors, other lists, and other objects.

Operations on lists include:

- Indexing
- Slicing
- Accessing elements by name or position

#### 1) Create a vector of numbers from 1 to 10

```
numbers <- 1:10  
cat("Numbers vector:", numbers, "\n")
```

```
Numbers vector: 1 2 3 4 5 6 7 8 9 10
```

#### 2) Create a vector of characters

```
characters <- c("a", "b", "c", "d", "e")  
cat("Characters vector:", characters, "\n")
```

```
Characters vector: a b c d e
```

### 3) Create a vector of logical values

```
logical_values <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
cat("Logical values vector:", logical_values, "\n")
```

```
Logical values vector: TRUE FALSE TRUE FALSE TRUE
```

### 4) Create a vector of mixed types

```
mixed_vector <- c(1, "a", TRUE, 3.14, "b")
cat("Mixed vector:", mixed_vector, "\n")
```

```
Mixed vector: 1 a TRUE 3.14 b
```

### 5) Create a list containing different types of elements

```
my_list <- list(numbers, characters, logical_values, mixed_vector)
print("List containing different types of elements:")
print(my_list)
```

```
[1] "List containing different types of elements:"
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
[1] "a" "b" "c" "d" "e"

[[3]]
[1] TRUE FALSE TRUE FALSE TRUE

[[4]]
[1] "1" "a" "TRUE" "3.14" "b"
```

### 6) Access the first element of the list

```
first_element <- my_list[[1]]
cat("First element of the list:", first_element, "\n")
```

```
First element of the list: 1 2 3 4 5 6 7 8 9 10
```

## Practical No : 3

### **AIM – Implementing decision making statements in R**

Decision-making statements are crucial in programming as they allow the execution of code based on certain conditions. In R, the primary decision-making statements are if, else if, else, and switch.

#### 1) if Statement

The if statement evaluates a condition and executes the code block if the condition is TRUE.

```
x <- 10
if (x > 5) {
  print("x is greater than 5")
}
```

In this example, since x is 10, which is greater than 5, the message "x is greater than 5" will be printed.

#### 2) else Statement

The else statement follows an if statement and executes if the if condition is FALSE.

```
x <- 3
if (x > 5) {
  print("x is greater than 5")
} else {
  print("x is not greater than 5")
}
```

Here, since x is 3, which is not greater than 5, the message "x is not greater than 5" will be printed.

#### 3) else if Statement

The else if statement allows you to check multiple conditions sequentially.

```
x <- 5
if (x > 5) {
  print("x is greater than 5")
} else if (x == 5) {
  print("x is equal to 5")
} else {
  print("x is less than 5")
}
```

In this case, since x is 5, the message "x is equal to 5" will be printed.

#### 4) switch Statement

The switch statement evaluates an expression and executes the corresponding code block based on the value of the expression.

```
choice <- "B"
result <- switch(choice,
  "A" = "You chose A",
  "B" = "You chose B",
  "C" = "You chose C",
  "Invalid choice")
print(result)
```

Here, since choice is "B", the message "You chose B" will be printed.

#### 1) if statement

```
x <- 10
if (x > 0) {
  cat("\nx is positive\n\n")
}
```

x is positive

#### 2) if-else statement

```
x <- -5
if (x > 0) {
  cat("\nx is positive\n\n")
} else {
  cat("\nx is negative\n\n")
}
```

x is negative

#### 3) switch statement

```
x <- 2
result <- switch(x,
  "1" = "One",
  "2" = "Two",
  "3" = "Three",
  "4" = "Four",
  "5" = "Five",
  "Unknown"
)
cat("\nResult of switch statement:", result,
  "\n\n")
```

Result of switch statement: Two

## Practical No : 4

### **AIM – R script to generate odd numbers b/w 1 to 20**

#### **Code –**

```
generate_odd_numbers <- function() {  
  odd_numbers <- c()  
  for (i in 1:20) {  
    if (i %% 2 != 0) {  
      odd_numbers <- c(odd_numbers, i)  
    }  
  }  
  return(odd_numbers)  
}  
odd_numbers <- generate_odd_numbers()  
cat("\nOdd numbers between 1 and 20:", odd_numbers, "\n\n")
```

#### **Output –**

```
Odd numbers between 1 and 20: 1 3 5 7 9 11 13 15 17 19
```

## Practical No : 5

### **AIM – R script to check whether given matrix is sparse or not**

- 1) A sparse matrix is a matrix in which most of the elements are zero. In other words, a matrix is considered sparse if the number of zero elements is significantly greater than the number of non-zero elements.
- 2) Typically, a matrix is defined as sparse if more than 50% of its elements are zero.

### **Code –**

```
generate_odd_numbers <- function() {  
  odd_numbers <- c()  
  for (i in 1:20) {  
    if (i %% 2 != 0) {  
      odd_numbers <- c(odd_numbers, i)  
    }  
  }  
  return(odd_numbers)  
}  
odd_numbers <- generate_odd_numbers()  
cat("\nOdd numbers between 1 and 20:", odd_numbers, "\n\n")
```

### **Output –**

```
Odd numbers between 1 and 20: 1 3 5 7 9 11 13 15 17 19
```



## Practical No : 6

### **AIM – R script to generate sum of digits of a number**

- 1) The sum of digits of a number is the sum of all its individual digits.
- 2) For example, the sum of digits of 12345 is  $1 + 2 + 3 + 4 + 5 = 15$ .

#### **Code –**

```
sum_of_digits <- function(n) {  
  digits <- strsplit(as.character(n), "")[[1]]  
  sum(as.numeric(digits))  
}  
  
number <- 12345  
result <- sum_of_digits(number)  
cat("\nThe sum of the digits of", number, "is", result, "\n\n")
```

#### **Output –**

```
The sum of the digits of 12345 is 15
```

## Practical No : 7

### **AIM – R Script to obtain Sum, Mean, Median and Variance of a vector**

- 1) Sum: The sum of all elements in the vector.
- 2) Mean: The average of all elements in the vector.
- 3) Median: The middle value of the vector when it is sorted.
- 4) Variance: A measure of how much the elements of the vector vary from the mean.

#### **Code –**

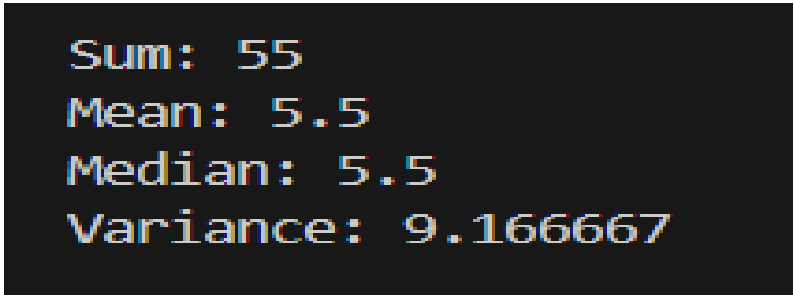
```
cat("\n")
vector <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
vector_sum <- sum(vector)
cat("Sum:", vector_sum, "\n")

vector_mean <- mean(vector)
cat("Mean:", vector_mean, "\n")

vector_median <- median(vector)
cat("Median:", vector_median, "\n")

vector_variance <- var(vector)
cat("Variance:", vector_variance, "\n")
cat("\n")
```

#### **Output –**



```
Sum: 55
Mean: 5.5
Median: 5.5
Variance: 9.166667
```

## Practical No : 8

### AIM – Implement Box-Plot in R

A box plot is a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum. It can also indicate which observations, if any, might be considered outliers.

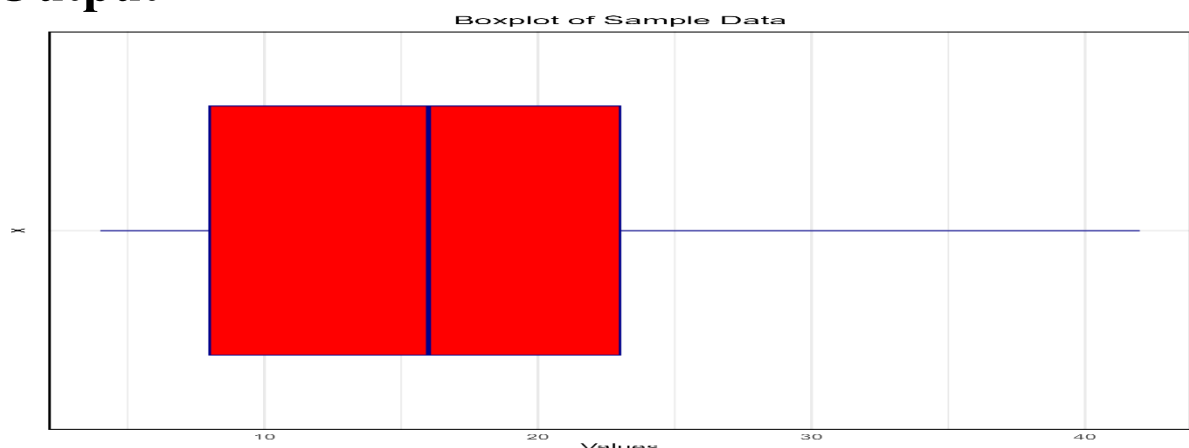
The box plot consists of:

- A box, which spans from Q1 to Q3 (the interquartile range, IQR).
- A line inside the box, which represents the median (Q2).
- Whiskers, which extend from the box to the smallest and largest values within  $1.5 * \text{IQR}$  from Q1 and Q3, respectively.
- Points outside the whiskers, which are considered outliers.

### Code –

```
library(ggplot2)
data <- c(4, 8, 15, 16, 23, 42, 8, 15, 16, 23, 42, 4, 8, 15, 16, 23, 42)
df <- data.frame(values = data)
p <- ggplot(df, aes(x = "", y = values)) +
  geom_boxplot(fill = "red", color = "darkblue") +
  coord_flip() +
  labs(title = "Boxplot of Sample Data", y = "Values") +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5),
    panel.border = element_rect(color = "black", fill = NA, size = 1)
  )
ggsave("prac8_boxplot.png", plot = p)
```

### Output –



## Practical No : 9

### AIM – Implement Stem-Leaf diagram in R

A Stem-Leaf diagram is a way to display quantitative data in a graphical format, similar to a histogram, to retain the original data values. It is useful for small data sets and helps in understanding the shape of the data distribution. In R, the `stem` function is used to create a Stem-Leaf diagram.

Syntax:

```
stem(x, scale = 1, width = 80, atom = 1e-08)
```

Parameters:

x: A numeric vector of data values.

scale: A scaling factor for the plot. Default is 1.

width: The desired width of the plot. Default is 80.

atom: The smallest unit to be used in the plot. Default is 1e-08.

### Code –

```
data <- c(12, 15, 22, 29, 34, 35, 36, 37, 45, 46, 47, 48, 55, 56, 57, 58, 59, 67, 68, 69, 78, 79, 89)
stem(data)
```

### Output –

```
The decimal point is 1 digit(s) to the right of the |
0 | 25
2 | 294567
4 | 567856789
6 | 78989
8 | 9
```

## Practical No : 10

### AIM – Data Frames in R

- 1) Data frames are a type of list in R, where each element of the list is a vector of the same length.
- 2) Data frames are used to store tabular data, where each column can be of a different type (numeric, character, factor, etc.).
- 3) Data frames are similar to spreadsheets or SQL tables, and they are one of the most commonly used data structures in R.
- 4) Data frames can be created using the ``data.frame()`` function.

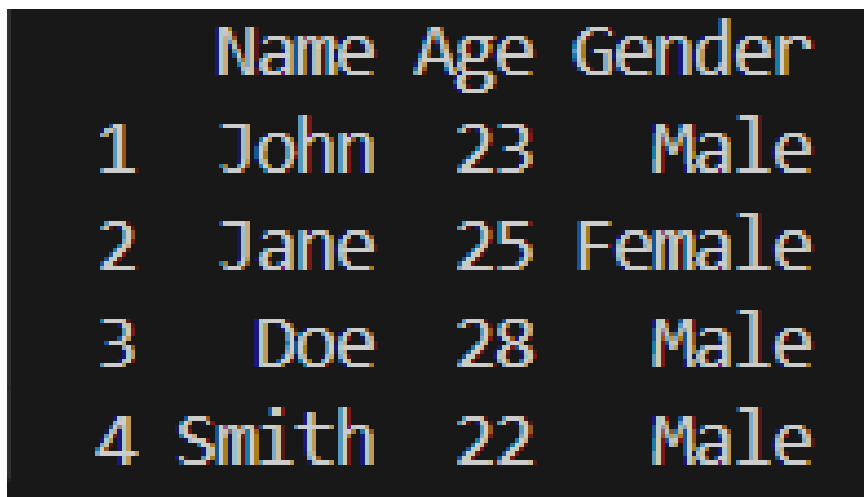
Syntax:

```
data.frame(..., row.names = NULL, check.rows = FALSE,  
check.names = TRUE, fix.empty.names = TRUE, stringsAsFactors =  
default.stringsAsFactors)
```

### Code –

```
names <- c("John", "Jane", "Doe", "Smith")  
ages <- c(23, 25, 28, 22)  
genders <- c("Male", "Female", "Male", "Male")  
df <- data.frame(Name = names, Age = ages, Gender = genders)  
print(df)
```

### Output –



```
  Name Age Gender  
1 John  23  Male  
2 Jane  25 Female  
3 Doe   28  Male  
4 Smith 22  Male
```

**Practical No : 11****AIM – Frequency Distribution plots in R****1) Bar plot**

- a) A bar plot is a graphical representation of data where individual bars represent different categories.
- b) The height of each bar corresponds to the count or frequency of the category it represents.
- c) Bar plots are useful for comparing the frequency or count of different categories.

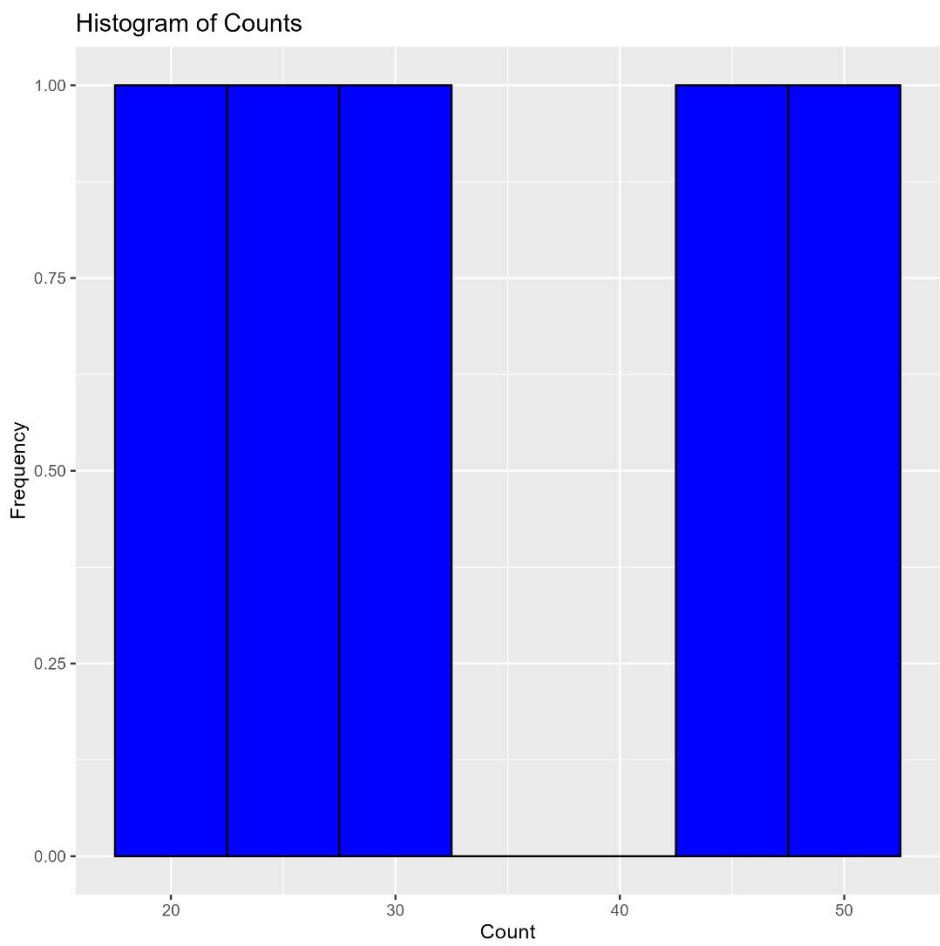
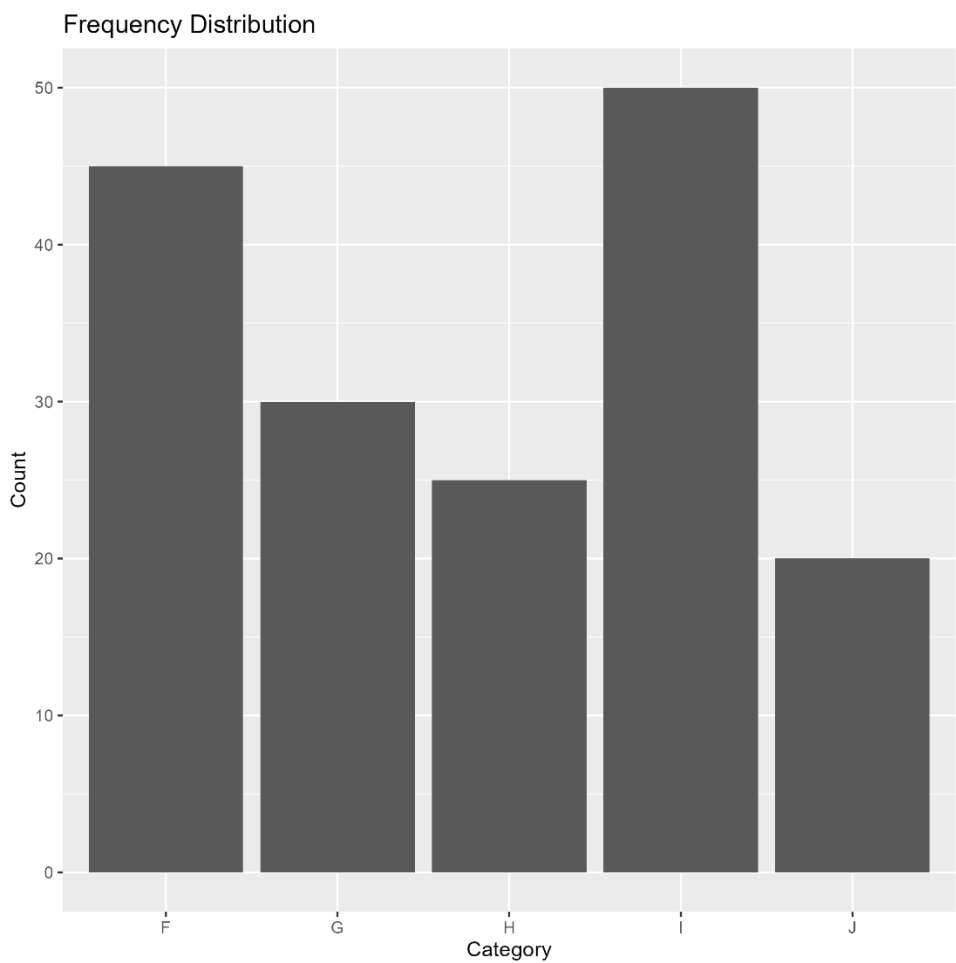
**2) Histogram**

- a) A histogram is a graphical representation of the distribution of numerical data.
- b) It is an estimate of the probability distribution of a continuous variable.
- c) Histograms are useful for understanding the distribution and spread of data.

**Code –**

```
library(ggplot2)
data <- data.frame(
  category = c("F", "G", "H", "I", "J"),
  count = c(45, 30, 25, 50, 20)
)
bar_plot <- ggplot(data, aes(x = category, y = count)) +
  geom_bar(stat = "identity") +
  ggtitle("Frequency Distribution") +
  xlab("Category") +
  ylab("Count")
ggsave("prac11_bar_plot.png", plot = bar_plot)
histogram <- ggplot(data, aes(x = count)) +
  geom_histogram(binwidth = 5, fill = "blue", color = "black") +
  ggtitle("Histogram of Counts") +
  xlab("Count") +
  ylab("Frequency")
ggsave("prac11_histogram.png", plot = histogram)
```

# Output –



## Practical No : 12

### AIM – Measure of Dispersion in R

- 1) A measure of dispersion is a statistical term that describes the spread or variability of a set of data points.
- 2) It provides insights into how much the data points differ from each other and from the central tendency (mean or median).

Common measures of dispersion include:

1. Range: The difference between the maximum and minimum values in a dataset.
2. Variance: The average of the squared differences from the mean.
3. Standard Deviation: The square root of the variance, representing the average distance of each data point from the mean.
4. Interquartile Range (IQR): The difference between the first quartile (Q1) and the third quartile (Q3), representing the middle 50% of the data.
5. Mean Absolute Deviation (MAD): The average of the absolute differences from the mean.

#### Code –

```
data <- c(10, 12, 23, 23, 16, 23, 21, 16)
cat("\n")
range_value <- range(data)
range_diff <- diff(range_value)
cat("Range:", range_diff, "\n")
variance_value <- var(data)
cat("Variance:", variance_value, "\n")
sd_value <- sd(data)
cat("Standard Deviation:", sd_value, "\n")
iqr_value <- IQR(data)
cat("Interquartile Range (IQR):", iqr_value, "\n")
mad_value <- mean(abs(data - mean(data)))
cat("Mean Absolute Deviation (MAD):", mad_value, "\n")
cat("\n")
```

#### Output –

```
Range: 13
Variance: 27.42857
Standard Deviation: 5.237229
Interquartile Range (IQR): 8
Mean Absolute Deviation (MAD): 4.5
Coefficient of Variation (CV): 29.09572 %
Median Absolute Deviation (MAD): 4.5
```



## Practical No : 13

### AIM – Sampling Distribution in R

- 1) A sampling distribution is the probability distribution of a statistic obtained by selecting random samples from a population.
- 2) It describes how the statistic varies from sample to sample.
- 3) The Central Limit Theorem states that the sampling distribution of the sample mean approaches a normal distribution as the sample size increases, regardless of the population's distribution.

#### Code –

```
library(ggplot2)
set.seed(123)
population <- rnorm(1000, mean = 50, sd = 10)
sampling_distribution <- function(population, sample_size, num_samples) {
  sample_means <- numeric(num_samples)
  for (i in 1:num_samples) {
    sample <- sample(population, sample_size, replace = TRUE)
    sample_means[i] <- mean(sample)
  }
  return(sample_means)
}
sample_size <- 30
num_samples <- 1000
sample_means <- sampling_distribution(population, sample_size, num_samples)
plot <- ggplot(data.frame(sample_means), aes(x = sample_means)) +
  geom_histogram(bins = 30, fill = "blue", color = "black") +
  ggtitle("Sampling Distribution of the Sample Mean") +
  xlab("Sample Mean") +
  ylab("Frequency")
ggsave("prac13_sampling_distribution_plot.png", plot)
```

#### Output –

