SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

ACADEMIC YEAR: 2025-2026

ODD SEMESTER

LAB MANUAL

(REGULATION - 2019)

1922706 – STATISTICAL APPROACHES FOR DATA SCIENCE LABORATORY

Prepared By

R. Danu, A.P (O.G) / AI&DS

INDEX

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

1. To afford the necessary background in the field of Information Technology to deal with engineering problems to excel as engineering professionals in industries.
2. To improve the qualities like creativity, leadership, teamwork and skill thus contributing towards the growth and development of society.
3. To develop ability among students towards innovation and entrepreneurship that caters to the needs of Industry and society.
4. To inculcate and attitude for life-long learning process through the use of information technology sources.
5. To prepare then to be innovative and ethical leaders, both in their chosen profession and in other activities.

PROGRAMME OUTCOMES (POs)

After going through the four years of study, Information Technology Graduates will exhibit ability to:

PO#	Graduate Attribute	Programme Outcome
1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2	Problem analysis	Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10	Communication	Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12	Life-long learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAMME SPECIFIC OUTCOMES (PSOs)

After the completion of Bachelor of Technology in Artificial Intelligence and Data Science programme the student will have following Program specific outcomes

1. Design and develop secured database applications with data analytical approaches of data preprocessing, optimization, visualization techniques and maintenance using state of the art methodologies based on ethical values.
2. Design and develop intelligent systems using computational principles, methods and systems for extracting knowledge from data to solve real time problems using advanced technologies and tools.
3. Design, plan and setting up the network that is helpful for contemporary business environments using latest software and hardware.
4. Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.

1922706	STATISTICAL APPROACHES FOR DATA SCIENCE LABORATORY	L T P C 0 0 4 2

OBJECTIVES:

- To know the fundamental concepts of Python Programming
- To learn the usage of R programming in Data structures and Data Visualization
- To explore on various statistical approaches
- To learn and implement machine learning algorithms
- To explore the usage of machine learning algorithms in cloud platform

LIST OF EXPERIMENTS:

1. Write a python program
    a) To enter the elements ‘v’, ’l’, ’a’, ’f’, ’e’ into a list.
    b) To convert the above list into dictionary
2. Creating Data structures in R
3. Creating Data visualization with R
4. Creating Data Analysis with R
5. Statistical approach with Types of variables and Measures of central tendency
6. Statistical approach with Skewness and Kurtosis
7. Creating Student’s T distribution with inferential statistics
8. Creating Missing value analysis with Data Analysis
9. The correction matrix and Outlier detection analysis with Exploratory Data Analysis
10. Design and develop Neural Network and Support Vector Machine using Machine Learning
11. Develop Machine Learning on Cloud Platform.

TOTAL : 60 PERIODS

LAB EQUIPMENT FOR A BATCH OF 30 STUDENTS:

SOFTWARE:

Python Matplotlib, Statistical tools like T-test and ANOVA, Exploratory data analysis tools like Apache Spark, Tableau, etc and Equivalent Open Source tools.

HARDWARE:

Standalone desktops 30 Nos

PYTHON:

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming.

R PROGRAMMING:

R is a programming language for statistical computing and data visualization. It has been adopted in the fields of data mining, bioinformatics, and data analysis.

The core R language is augmented by a large number of extension packages, containing reusable code, documentation, and sample data. R software is open-source and free software.

Course Name: 1922706 – Statistical Approaches for Data Science Laboratory
Year of study: 2024 – 2025 (ODD SEM)

1922706.1 Learn Foundational concepts of python and R programming concepts such as data structures, variables and data types
1922706.2 Import a variety of data formats for Data Visulization
1922706.3 Apply programming language on various statistical approaches.
1922706.4 Learn and implement ML algorithms
1922706.5 Explore the usage of machine learning algorithms in cloud platform

CO- PO-PSO MATRIX

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
1922706.1	3	3	-	-	1	-	-	-	-	-	-	-	2	-	-	-
1922706.2	3	3	3	-	3	-	-	-	-	-	-	-	2	-	-	-
1922706.3	3	3	3	3	-	-	-	-	-	-	-	-	-	3	-	-
1922706.4	-	3	3	3	3	-	-	-	-	-	-	-	-	2	-	-
1922706.5	-	3	3	3	3	-	-	-	-	-	-	-	2	-	-	-

EVALUATION PROCEDURE FOR EACH EXPERIMENT

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	20
2.	Observation	30
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	10
Total		**100**

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Conduction & Execution of Experiment	25
2.	Record	10
3.	Model Test	15
Total		**50**

Ex: 1 List and Dictionary

AIM:

To create a python program for the following"

a. To enter the elements 'v','I','a','f','e' into a list.
b. To convert the above list into dictonary.

PROCEDURE:

Step 1: Create a List of elements

Initialize an Empty List

Prompt used for number of elements

Loop to Collect elements

Point the Resulting List

Check the Result list

Step 2: Convert List to Dictionary

Initialize the List of elements

Initialize an empty Dictionary

Loop to populate the Dictionary

Print the Resulting Dictionary

Pre-Lab Discussion:

a. List

Lists are used to store multiple items in a single variable. Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage. Lists are created using square brackets.

List Items

List items are ordered, changeable, and allow duplicate values. List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change. If we need to add new items to a list, the new items will be placed at the end of the list.

Creating a list with multiple distinct or duplicate elements

A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

Accessing elements from the List

In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list. The index must be an integer. Nested lists are accessed using nested indexing.

Negative indexing

In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

Getting the size of Python list

Python len() is used to get the length of the list.

Adding Elements to a Python List

Method 1: Using append() method

Only one element at a time can be added to the list by using the append() method, for the addition of multiple elements with the append() method, loops are used. Tuples can also be added to the list with the use of the append method because tuples are immutable.

Method 2: Using insert() method

append() method only works for the addition of elements at the end of the List, for the addition of elements at the desired position, insert() method is used. Unlike append() which takes only one argument, the insert() method requires two arguments(position, value).

Method 3: Using extend() method

Other than append() and insert() methods, there's one more method for the Addition of elements, **extend()**, this method is used to add multiple elements at the same time at the end of the list.

Reversing a List

A list can be reversed by using the reverse() method in Python.

Removing Elements from the List

Method 1: Using remove() method

Elements can be removed from the List by using the built-in **remove()** function but an Error arises if the element doesn't exist in the list. Remove() method only removes one element at a time, to remove a range of elements, the iterator is used. The remove() method removes the specified item.

Method 2: Using pop() method

pop() function can also be used to remove and return an element from the list, but by default it removes only the last element of the list, to remove an element from a specific position of the List, the index of the element is passed as an argument to the pop() method.

Slicing of a List

In Python List, there are multiple ways to print the whole list with all the elements, but to print a specific range of elements from the list, we use the Slice operation. Slice operation is performed on Lists with the use of a colon(:).

b. Dictionary

It is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.

Dictionary holds **key:value** pair. Key-Value is provided in the dictionary to make it more optimized.

Creating a Dictionary

In Python, a dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its **Key:value**. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be *immutable*.

Adding elements to a Dictionary

Addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = 'Value'. Updating an existing value in a Dictionary can be done by using the built-in **update()** method. Nested key values can also be added to an existing Dictionary.

Accessing elements of a Dictionary

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets.

Deleting Elements using del Keyword

The items of the dictionary can be deleted by using the del keyword

Program:

a. **To enter the elements 'v','I','a','f','e' into a list.**

```
# Step 1: Initialize an empty list

elements_list = []

# Step 2: Prompt the user to enter the elements

num_elements = int(input("Enter the number of elements: "))

for i in range(num_elements):
```

```
    element = input("Enter an element: ")
    elements_list.append(element)
```

 # Step 3: Print the resulting list

```
print(elements_list)
```

Output:

```
Enter the number of elements: 5
Enter an element: v
Enter an element: l
Enter an element: a
Enter an element: f
Enter an element: e
['v', 'l', 'a', 'f', 'e']
```

b. To convert the above list into dictionary.

#Step 1: Inialize the above list values

```
elements_list = ['v', 'l', 'a', 'f', 'e']
dictionary = {}
for index, element in enumerate(elements_list):
    dictionary[index] = element
```

 # Step 2: Print the resulting dictionary

```
print(dictionary)
```

Output:

```
{0: 'v', 1: 'l', 2: 'a', 3: 'f', 4: 'e'}
```

RESULT

Thus, the creation of list for the given elements and conversion to list to dictionary was executed.

Ex: 2 Data Structures

AIM:

To Create Data structures using R.

PROCEDURE:

Step 1: Vector:

- Create a Vector
- Print the vector

Step 2: List:

- Create Employee ID & Name Vectors
- Create a List
- Print the List

Step 3: Data frame:

- Create Name & Language vector
- Create a Dataframe
- Print the Dataframe

Step 4: Matrices:

- Create a Matrix
- Print the Matrix+++

Step 5: Array :

- Create an Array
- Print the Array

Pre-lab Discussion

Data Structure

A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data structures in R programming are tools for holding multiple values.

R's base data structures are often organized by their dimensionality (1D, 2D, or nD) and whether they're homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types). This gives rise to the six data types which are most frequently utilized in data analysis.

The most essential data structures used in R include:

- **Vectors**

- **Lists**
- **Dataframes**
- **Matrices**
- **Arrays**
- **Factors**

Vectors

A vector is an ordered collection of basic data types of a given length. The only key thing here is all the elements of a vector must be of the identical data type e.g homogeneous data structures. Vectors are one-dimensional data structures.

Lists

A list is a generic object consisting of an ordered collection of objects. Lists are heterogeneous data structures. These are also one-dimensional data structures. A list can be a list of vectors, list of matrices, a list of characters and a list of functions and so on.

Dataframes

Dataframes are generic data objects of R which are used to store the tabular data. Dataframes are the foremost popular data objects in R programming because we are comfortable in seeing the data within the tabular form. They are two-dimensional, heterogeneous data structures. These are lists of vectors of equal lengths.

Data frames have the following constraints placed upon them:

- A data-frame must have column names and every row should have a unique name.
- Each column must have the identical number of items.
- Each item in a single column must be of the same data type.
- Different columns may have different data types.

Matrices

A matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. Matrices are two-dimensional, homogeneous data structures.
Now, let's see how to create a matrix in R. To create a matrix in R you need to use the function called matrix. The arguments to this matrix() are the set of elements in the vector. You have to pass how many numbers of rows and how many numbers of columns you want to have in your matrix and this is the important point you have to remember that by default, matrices are in column-wise order.

Arrays

Arrays are the R data objects which store the data in more than two dimensions. Arrays are n-dimensional data structures. For example, if we create an array of dimensions (2, 3, 3) then it creates 3 rectangular matrices each with 2 rows and 3 columns. They are homogeneous data structures.

Factors

Factors are the data objects which are used to categorize the data and store it as levels. They are useful for storing categorical data. They can store both strings and integers. They are useful to categorize unique values in columns like “TRUE” or “FALSE”, or “MALE” or “FEMALE”, etc.. They are useful in data analysis for statistical modeling.

Program:

a. **Vector**

```
# Vectors(ordered collection of same data type)

X = c(1, 3, 5, 7, 8)

# Printing those elements

print(X)
```

Output :

```
[1] 1 3 5 7 8
```

b. **Lists**

```
# empID is created using the 'c' command here

empId = c(1, 2, 3, 4)

empName = c("ABC", "DEF", "GHI", "JKL")

numberOfEmp = 4

empList = list(empId, empName, numberOfEmp)

print(empList)
```

Output :

[[1]]

[1] 1 2 3 4

[[2]]

[1]"ABC" "DEF" "GHI" "JKL"

[[3]]

[1] 4

c. Dataframe

```
# A vector which is a character vector
Name = c("ABC", "DEF", "GHI")
# A vector which is a character vector
Language = c("R", "Python", "Java")
# A vector which is a numeric vector
Age = c(22, 25, 45)

# To create dataframe use data.frame command and then pass each of the vectors
# we have created as arguments to the function data.frame()
df = data.frame(Name, Language, Age)
print(df)
```

Output :

	Name	Language	Age
1	ABC	R	22
2	DEF	Python	25
3	GHI	Java	45

d. Matrix

```
# R program to illustrate a matrix

A = matrix(
```

```
        # Taking sequence of elements
        c(1, 2, 3, 4, 5, 6, 7, 8, 9),

        # No of rows and columns
        nrow = 3, ncol = 3,

        # By default matrices are in column-wise order .So this parameter decides
        # how to arrange the matrix
        byrow = TRUE )

print(A)
```

Output:

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

c. Array

```
# R program to illustrate an array

A = array(
        # Taking sequence of elements
        c(1, 2, 3, 4, 5, 6, 7, 8),

        # Creating two rectangular matrices each with two rows and two columns
        dim = c(2, 2, 2)
)

print(A)
```

Output:

```
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

e. Factors

 # R program to illustrate factors

 # Creating factor using factor()

```
fac = factor(c("Male", "Female", "Male", “Male", "Female", "Male", "Female"))

print(fac)
```

Output:

```
[1] Male   Female Male   Male   Female Male   Female
Levels: Female Male
```

RESULT

Thus, the creating of data structures using R was executed.

Ex: 3 Data Visualization

AIM:

To create data visualization using R .

PROCEDURE:

Step 1: Install the Required packages.

Step 2: Load the Installed packages

Step 3: Create & Visualize Various plots

Step 4: Scatter plot with mtcars dataset

Step 5: Barplot with iris Dataset

Step 6: Line plot with economic dataset

Step 7: Boxplot with Diamond dataset

Step 8: Pie chart with mtcars dataset as well as Heatmap

Step 9: Histogram with iris Dataset.

Pre-lab Discussion:

Data visualization is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it.

Data Visualization in R Programming Language

The popular data visualization tools that are available are Tableau, Plotly, R, Google Charts, Infogram, and Kibana. The various data visualization platforms have different capabilities, functionality, and use cases. They also require a different skill set. This article discusses the use of R for data visualization. R is a language that is designed for statistical computing, graphical data analysis, and scientific research. It is usually preferred for data visualization as it offers flexibility and minimum required coding through its packages.

Types of Data Visualizations

Some of the various types of visualizations offered by R are:

Bar Plot

There are two types of bar plots- horizontal and vertical which represent data points as horizontal or vertical bars of certain lengths proportional to the value of the data item. They are generally used for continuous and categorical variable plotting. By setting the **horiz** parameter to true and false, we can get horizontal and vertical bar plots respectively.

Histogram

A histogram is like a bar chart as it uses bars of varying height to represent data distribution. However, in a histogram values are grouped into consecutive intervals called bins. In a Histogram, continuous values are grouped and displayed in these bins whose size can be varied.

Box Plot

The statistical summary of the given data is presented graphically using a boxplot. A boxplot depicts information like the minimum and maximum data point, the median value, first and third quartile, and interquartile range.

Scatter Plot

A scatter plot is composed of many points on a Cartesian plane. Each point denotes the value taken by two parameters and helps us easily identify the relationship between them.

Heat Map

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. heatmap() function is used to plot heatmap.

Syntax: *heatmap(data)*

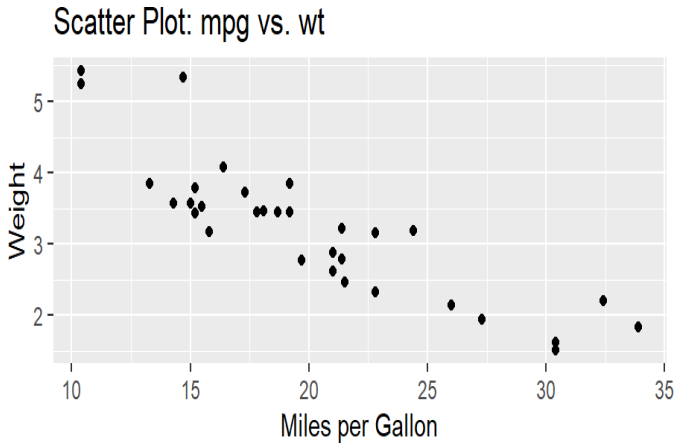
Program:

 # Install and load the necessary packages

```
install.packages("ggplot2")

library(ggplot2)
```

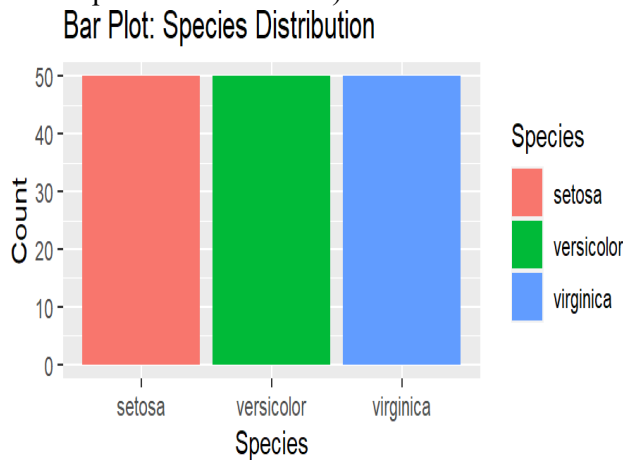
 # Scatter plot with the mtcars dataset

```
ggplot(data = mtcars, aes(x = mpg, y = wt)) +  geom_point() +  labs(x = "Miles per Gallon", y = "Weight", title = "Scatter Plot: mpg vs. wt")
```



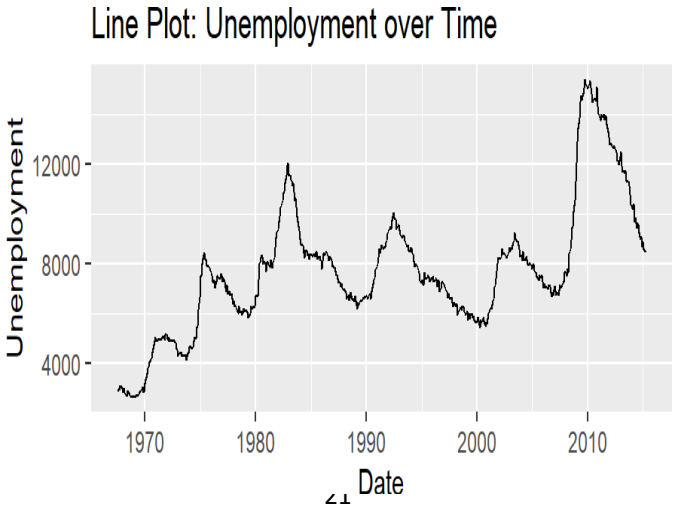
 # Bar plot with the iris dataset

ggplot(data = iris, aes(x = Species, fill = Species)) + geom_bar() + labs(x = "Species", y = "Count", title = "Bar Plot: Species Distribution")



 # Line plot with the economics dataset

ggplot(data = economics, aes(x = date, y = unemploy)) + geom_line() + labs(x = "Date", y = "Unemployment", title = "Line Plot: Unemployment over Time")

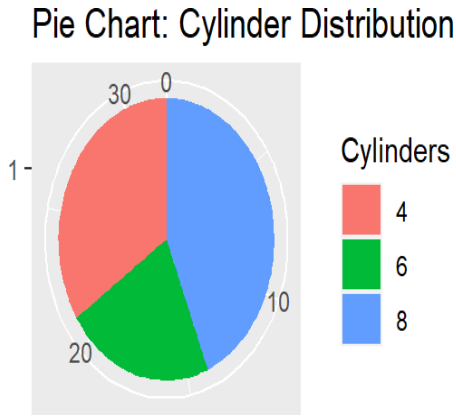


 # Box plot with the diamonds dataset

ggplot(data = diamonds, aes(x = cut, y = price, fill = cut)) + geom_boxplot() + labs(x = "Cut", y = "Price", title = "Box Plot: Price by Cut")

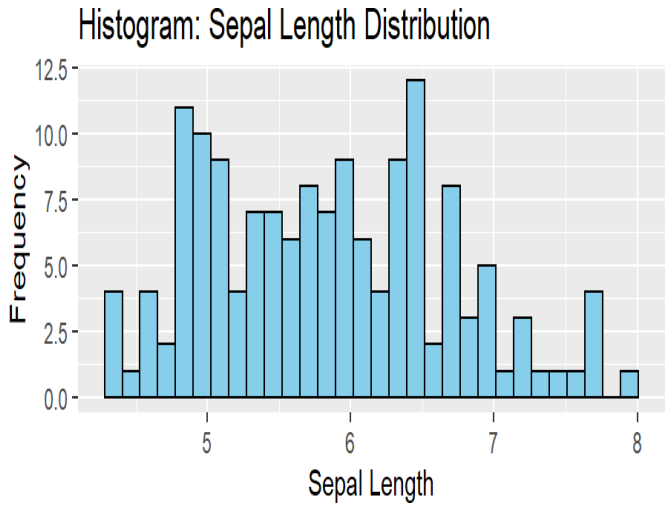
 # Pie chart with the mtcars dataset

ggplot(data = mtcars, aes(x = factor(1), fill = factor(cyl))) + geom_bar(width = 1, stat = "count") + coord_polar("y", start = 0) + labs(x = "", y = "", fill = "Cylinders", title = "Pie Chart: Cylinder Distribution")



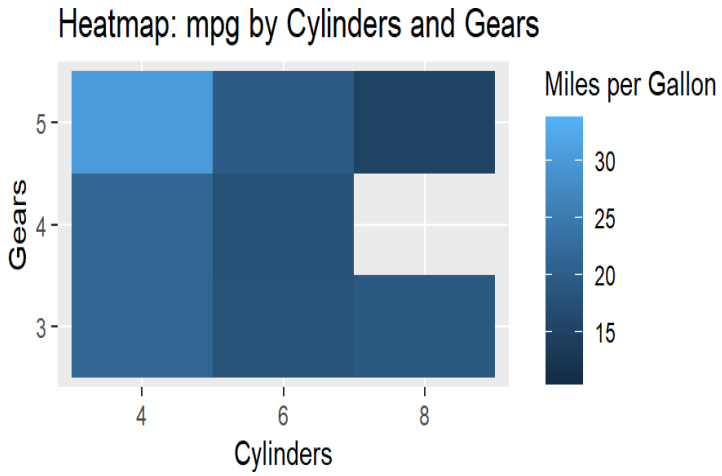
 # Histogram with the iris dataset

ggplot(data = iris, aes(x = Sepal.Length)) + geom_histogram(fill = "skyblue", color = "black") + labs(x = "Sepal Length", y = "Frequency", title = "Histogram: Sepal Length Distribution")



 # Heatmap with the mtcars dataset

ggplot(data = mtcars, aes(x = factor(cyl), y = factor(gear), fill = mpg)) + geom_tile() + labs(x = "Cylinders", y = "Gears", fill = "Miles per Gallon", title = "Heatmap: mpg by Cylinders and Gears")



RESULT

Thus, the data visualization using R was executed.

Ex: 4 Data Analysis

AIM:

To create data analysis using R.

PROCEDURES:

Step 1: Load the Dataset.

Step 2: Check data types

Step 3: Convert the Columns to factors

Step 4: Summarize the statistics of Data

Step 5: Handling the Missing Values

Step 6: Separate Survived & Not Survived Data

Step 7: Visualize the Data

Pre-lab Discussion

Data Analysis is a subset of data analytics, it is a process where the objective has to be made clear, collect the relevant data, preprocess the data, perform analysis(understand the data, explore insights), and then visualize it. The last step visualization is important to make people understand what's happening in the firm.

Steps involved in data analysis:

The process of data analysis would include all these steps for the given problem statement.

Example- Analyze the products that are being rapidly sold out and details of frequent customers of a retail shop.

- **Defining the problem statement** – Understand the goal, and what is needed to be done. In this case, our problem statement is – "The product is mostly sold out and list of customers who often visit the store."
- **Collection of data** – Not all the company's data is necessary, understand the relevant data according to the problem. Here the required columns are product ID, customer ID, and date visited.
- **Preprocessing** – Cleaning the data is mandatory to put it in a structured format before performing analysis.

1. Removing outliers(noisy data).
2. Removing null or irrelevant values in the columns. (Change null values to mean value of that column.)
3. If there is any missing data, either ignore the tuple or fill it with a mean value of the column.

Program

 # Data Analysis for Titanic Dataset

```
titanic=read.csv(train.csv")

head(titanic)
```

Output:

	Passenger Id	Survived	Pclass	Name	Sex
1	892	0	3	Kelly, Mr. James	male
2	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female
3	894	0	2	Myles, Mr. Thomas Francis	male
4	895	0	3	Wirz, Mr. Albert	male
5	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female
6	897	0	3	Svensson, Mr. Johan Cervin	male

	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	34.5	0	0	330911	7.8292		Q
2	47.0	1	0	363272	7.0000		S
3	62.0	0	0	240276	9.6875		Q
4	27.0	0	0	315154	8.6625		S
5	22.0	1	1	3101298	12.2875		S
6	14.0	0	0	7538	9.2250		S

Our dataset contains all the columns like name, age, gender of the passenger and class they have traveled in, whether they have survived or not, etc. To understand the class(data type) of each column **sapply()** method can be used.

```
sapply(train, class)
```

.

Output:

```
PassengerId     Survived       Pclass         Name          Sex          Age
  "integer"    "integer"    "integer" "character" "character"    "numeric"
      SibSp        Parch       Ticket         Fare        Cabin     Embarked
  "integer"    "integer" "character"    "numeric" "character" "character"
```

We can categorize the value **"survived"** into **"dead"** to 0 and **"alive"** to 1 using **factor()** function.

```
train$Survived=as.factor(train$Survived)

train$Sex=as.factor(train$Sex)

sapply(train, class)
```

Output:

```
PassengerId     Survived       Pclass         Name          Sex          Age
  "integer"     "factor"    "integer" "character"      "factor"    "numeric"
      SibSp        Parch       Ticket         Fare        Cabin     Embarked
  "integer"    "integer" "character"    "numeric" "character" "character"
```

To analyze data using a summary of all the columns, their values, and data types. summary() can be used for this purpose.

```
summary(train)
```

Output:

```
 PassengerId      Survived      Pclass          Name              Sex
Min.   : 892.0   0:266    Min.   :1.000   Length:418         female:152
1st Qu.: 996.2   1:152    1st Qu.:1.000   Class :character   male  :266
Median :1100.5            Median :3.000   Mode  :character
Mean   :1100.5            Mean   :2.266
3rd Qu.:1204.8            3rd Qu.:3.000
Max.   :1309.0            Max.   :3.000

     Age           SibSp           Parch           Ticket
```

```
 Min.   : 0.17   Min.   :0.0000   Min.   :0.0000   Length:418
 1st Qu.:21.00   1st Qu.:0.0000   1st Qu.:0.0000   Class :character
 Median :27.00   Median :0.0000   Median :0.0000   Mode  :character
 Mean   :30.27   Mean   :0.4474   Mean   :0.3923
 3rd Qu.:39.00   3rd Qu.:1.0000   3rd Qu.:0.0000
 Max.   :76.00   Max.   :8.0000   Max.   :9.0000
 NA's   :86
     Fare            Cabin             Embarked
 Min.   :  0.000   Length:418         Length:418
 1st Qu.:  7.896   Class :character   Class :character
 Median : 14.454   Mode  :character   Mode  :character
 Mean   : 35.627
 3rd Qu.: 31.500
 Max.   :512.329
 NA's   :1
```

From the above summary we can extract below observations:

- Total passengers: 891
- The number of total people who survived: 342
- Number of total people dead: 549
- Number of males in the titanic: 577
- Number of females in the titanic: 314
- Maximum age among all people in titanic: 80
- Median age: 28

Preprocessing of the data is important before analysis, so null values have to be checked and removed.

```
sum(is.na(train))
```

Output:

```
177
```

```
dropnull_train=train[rowSums(is.na(train))<=0,]
```

- dropnull_train contains only 631 rows because **(total rows in dataset (808) – null value rows (177) = remaining rows (631))**

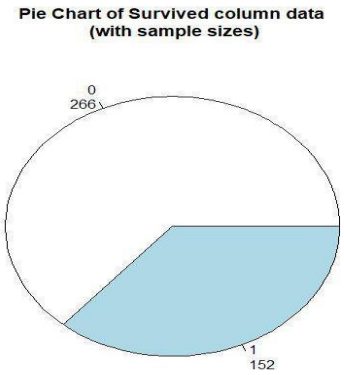
- Now we will divide survived and dead people into a separate list from 631 rows.

```
survivedlist=dropnull_train[dropnull_train$Survived == 1,]

notsurvivedlist=dropnull_train[dropnull_train$Survived == 0,]
```

we can visualize the number of males and females dead and survived using bar plots, histograms, and piecharts.

```
mytable <- table(titanic$Survived)

lbls <- paste(names(mytable), "\n", mytable, sep="")

pie(mytable,

   labels = lbls,

   main="Pie Chart of Survived column data\n (with sample sizes)")
```

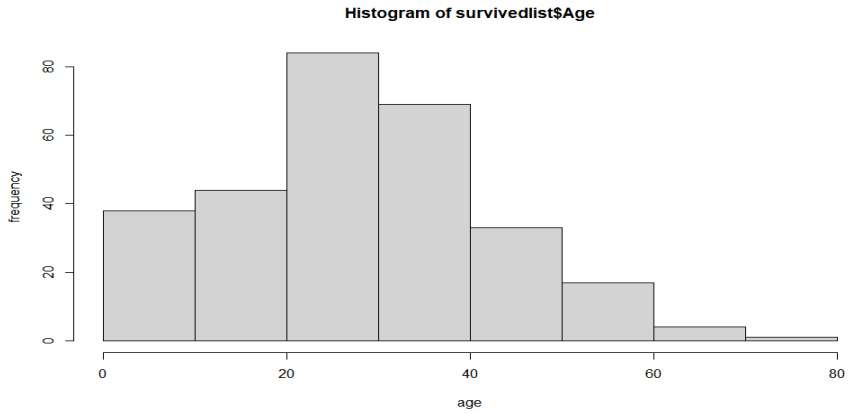
Output:



From the above pie chart, we can certainly say that there is a data imbalance in the target/Survived column.

```
hist(survivedlist$Age,  xlab="gender",   ylab="frequency")
```

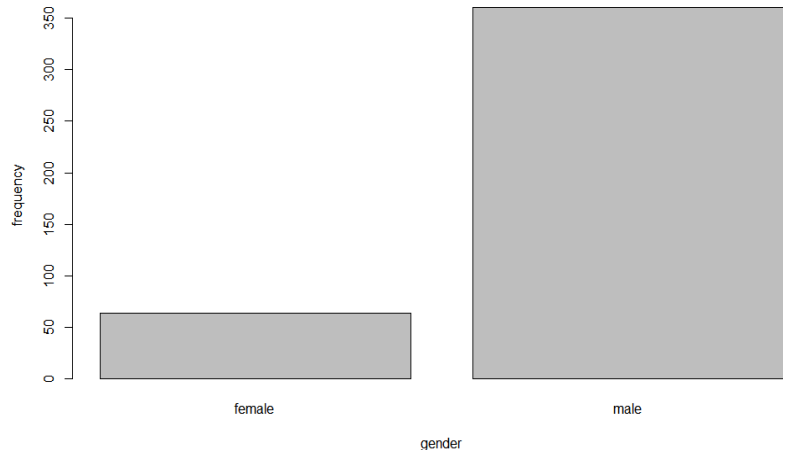
Output:



Bar plot to visualize the number of males and females who were there on the titanic ship.

```
barplot(table(notsurvivedlist$Sex),          xlab="gender",          ylab="frequency")
```

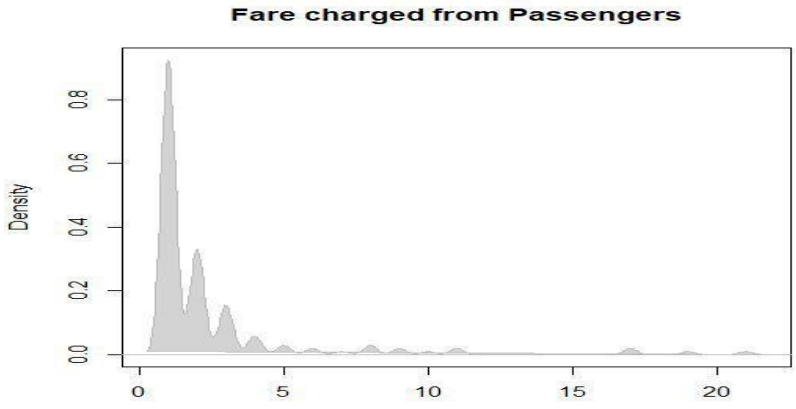
Output:



From the barplot above we can analyze that there are nearly 350 males, and 50 females those are not survived in titanic.

```
temp<-density(table(titanic$Fare))

plot(temp, type="n",      main="Fare charged from Passengers")

polygon(temp, col="lightgray",    border="gray")
```

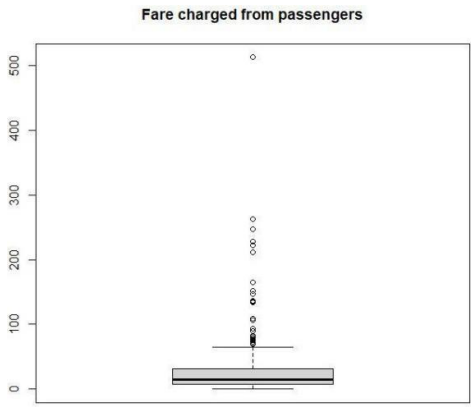
Output:



Here. we can observe that there are some passengers who are charged extremely high. So, these values can affect our analysis as they are outliers. Let's confirm their presence using a boxplot.

```
boxplot(titanic$Fare, main="Fare charged from passengers")
```

Output:



Certainly, there are some extreme outliers present in this dataset.

RESULT

Thus, the design of data analysis was studied.

Ex: 5 Statistical approach with Types of variables and Measures of Central Tendency

AIM:

To design statistical approach with types of variables and Measures of Central tendency.

PROCEDURE:

Step 1: Load the Required Package

Step 2: Load the Dataset with needed Column names (data.csv)

Step 3: Checking Structure and Identifying types of variables

Step 4: Measure the Central Tendancy of vectors (i.e)

1) Mean
2) Median
3) Mode

Step 5: Calculate the Arthimetic mean, Harmonic Mean & Geometric Mean of vector

Step 6: Generate the frequency table

Step 7: Print the frequency table

Pre-lab Discussion:

Quantitative

A quantitative variable is a variable that reflects a notion of magnitude, that is, if the values it can take are numbers. A quantitative variable represents thus a measure and is numerical. Quantitative variables are divided into two types: discrete and continuous. The difference is explained in the following two sections.

Discrete

Quantitative discrete variables are variables for which the values it can take are countable and have a finite number of possibilities. The values are often (but not always) integers. Here are some examples of discrete variables:

- Number of children per family
- Number of students in a class
- Number of citizens of a country

Continuous

On the other hand, quantitative continuous variables are variables for which the values are not countable and have an infinite number of possibilities. For example:

- Age
- Weight
- Height

Qualitative

In opposition to quantitative variables, qualitative variables (also referred as

categorical variables or factors in R) are variables that are not numerical and which values fits into categories.

In other words, a qualitative variable is a variable which takes as its values modalities, categories or even levels, in contrast to quantitative variables which measure a quantity on each individual.

Qualitative variables are divided into two types: nominal and ordinal.

Nominal

A qualitative nominal variable is a qualitative variable where no ordering is possible or implied in the levels.

A nominal variable can have:

two levels (e.g., do you smoke? Yes/No, or are you pregnant? Yes/No), or

Ordinal

On the other hand, a qualitative ordinal variable is a qualitative variable with an order implied in the levels. For instance, if the severity of road accidents has been measured on a scale such as light, moderate and fatal accidents, this variable is a qualitative ordinal variable because there is a clear order in the levels.

Variable transformations

There are two main variable transformations:

- From a continuous to a discrete variable
- From a quantitative to a qualitative variable

Program:

Types of Varaibles

 # Load the required packages

```
library(dplyr)
```

 # Read the data from CSV file

```
data <- read.csv("data.csv")
```

 # Check the structure of the dataset

```
str(data)
```

 # Identify the types of variables

 # Categorical Variables

```
categorical_vars <- c("gender", "ethnicity")

cat_data <- data[, categorical_vars]

print("Categorical Variables:")

print(cat_data)
```

```
# Numerical Variables
numerical_vars <- c("age", "height", "weight")
num_data <- data[, numerical_vars]
print("Numerical Variables:")
print(num_data)
# Discrete Variables
discrete_vars <- c("siblings", "cars_owned")

discrete_data <- data[, discrete_vars]
print("Discrete Variables:")
print(discrete_data)
# Continuous Variables
continuous_vars <- c("income", "savings")
continuous_data <- data[, continuous_vars]
print("Continuous Variables:")
print(continuous_data)
```

Measures of Central Tendency

Central Tendency is one of the features of descriptive statistics. Central tendency tells about how the group of data is clustered around the center value of the distribution. Central tendency performs the following measures:

- Arithmetic Mean
- Geometric Mean
- Harmonic Mean
- Median
- Mode

Arithmetic Mean

The arithmetic mean is simply called the average of the numbers which represents the central value of the data distribution.

Geometric Mean

The geometric mean is a type of mean that is computed by multiplying all the data values and thus, shows the central tendency for given data distribution.

Harmonic Mean

The harmonic mean is another type of mean used as another measure of central tendency.

Median

The median in statistics is another measure of central tendency which represents the middlemost value of a given set of values.

Mode

The mode of a given set of values is the value that is repeated most in the set. There can exist multiple mode values in case there are two or more values with matching maximum frequency.

Program:

Arithmetic Mean

 # Defining vector

```
x <- c(3, 7, 5, 13, 20, 23, 39, 23, 40, 23, 14, 12, 56, 23)
```

 # Print mean

```
print(mean(x))
```

Output:

```
[1] 21.5
```

Geometric Mean

 # Defining vector

```
x <- c(1, 5, 9, 19, 25)
```

 # Print Geometric Mean

```
print(prod(x)^(1 / length(x)))
```

Output:

```
[1] 7.344821
```

Harmonic Mean

 # Defining vector

```
x <- c(1, 5, 8, 10)
```

 # Print Harmonic Mean

```
print(1 / mean(1 / x))
```

Output:

```
[1] 2.807018
```

Median

```
# Defining vector

x <- c(3, 7, 5, 13, 20, 23, 39, 23, 40, 23, 14, 12, 56, 23)

# Print Median

median(x)
```

Output:

```
[1] 21.5
```

Mode

```
# Defining vector

x <- c(3, 7, 5, 13, 20, 23, 39, 23, 40, 23, 14, 12, 56,23, 29, 56, 37, 45, 1, 25, 8)

# Generate frequency table

y <- table(x)

# Print frequency table

print(y)

# Mode of x

m <- names(y)[which(y == max(y))]

# Print mode

print(m)
```

Output:

```
x
 1  3  5  7  8 12 13 14 20 23 25 29 37 39 40 45 56
 1  1  1  1  1  1  1  1  1  4  1  1  1  1  1  1  2
[1] "23"
```

RESULT

Thus, the design of statistical approach with types of variables and measure of central tendency was verified successfully.

Ex: 6 Statistical approach with Skewness and Kurtosis

AIM:

To create Statistical approach with skewness and kurtosis using R.

PROCEDURE:

Step 1: we need to install the library "Moments"

Step 2: Load the package "moments"

Step 3: Define the Vector

Step 4: We need to Print Skewness and histogram of distribution

Step 5: We need to print positive skewness, negative skewness and zero skewness of the taken vector.

Step 6: We need to print kurtosis and Histogram of distribution of the taken vector

Types of kurtosis :-

i) Leptokurtic

ii) Platykurtic

iii) Mesokurtic

Pre-lab Discussion

Skewness

Skewness is a statistical numerical method to measure the asymmetry of the distribution or data set. It tells about the position of the majority of data values in the distribution around the mean value. A fundamental statistical notion called skewness quantifies the asymmetries in data distributions.

There exist 3 types of skewness values on the basis of which the asymmetry of the graph is decided. These are as follows:

Positive Skew

The asymmetry of data distributions where the tail extends towards higher values is known statistically as positive skewness. It is a crucial metric in a number of disciplines, including data analysis, social sciences, finance, and economics.

Zero Skewness or Symmetric

A statistical notion called zero skewness, commonly referred to as symmetry, defines data distributions that are balanced and have equal probability on both sides of the mean.

Negatively skewed

Left-skewed distributions, commonly referred to as negatively skewed distributions, are statistical notions that describe asymmetrical data distributions with a tail that slopes downward.

Kurtosis

A statistical measure known as kurtosis measures the peakedness, flatness, and weight of the tails of data distributions. In a number of disciplines, including finance, economics, social sciences, and data analysis, an understanding of kurtosis is crucial. Kurtosis theory is thoroughly explained in this article, which also covers its definition, computation processes, interpretation, and applications.

Kurtosis is a numerical method in statistics that measures the sharpness of the peak in the data distribution.

There exist 3 types of Kurtosis values on the basis of which the sharpness of the peak is measured. These are as follows:

Platykurtic

Data distributions having flattened tails compared to the normal distribution are referred to statistically as platykurtic distributions

Mesokurtic

Data distributions with tails that are similar in thickness to the normal distribution are known statistically as mesokurtic distributions.

Leptokurtic

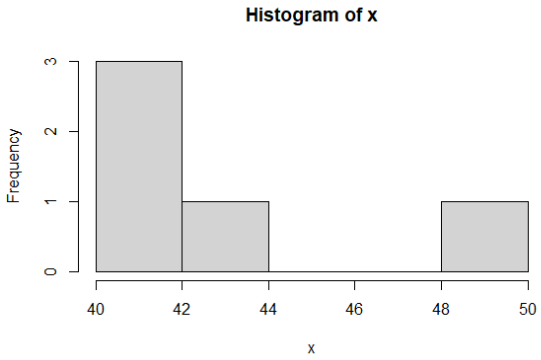
Data distributions having hefty tails compared to the normal distribution are referred to statistically as leptokurtic distributions.

Program:

```
# Required for skewness() function

library(moments)

# Defining data vector

x <- c(40, 41, 42, 43, 50)

# output to be present as PNG file

png(file = "positiveskew.png")

# Print skewness of distribution

print(skewness(x))

# Histogram of distribution

hist(x)

# Saving the file

dev.off()
```

Output:

[1] 1.2099



Zero Skewness or Symmetric

 # Required for skewness() function

library(moments)

 # Defining normally distributed data vector

x <- rnorm(50, 10, 10)

 # output to be present as PNG file

png(file = "zeroskewness.png")

 # Print skewness of distribution

print(skewness(x))

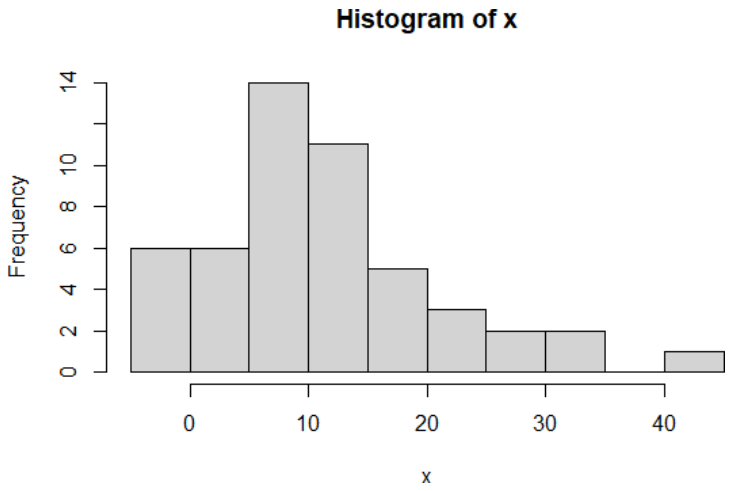
 # Histogram of distribution

hist(x)

 # Saving the file

dev.off()

Output:
[1] -0.02991511



Negatively skewed

 # Required for skewness() function

library(moments)

 # Defining data vector

x <- c(10, 11, 21, 22, 23, 25)

 # output to be present as PNG file

png(file = "negativeskew.png")

 # Print skewness of distribution

print(skewness(x))

 # Histogram of distribution

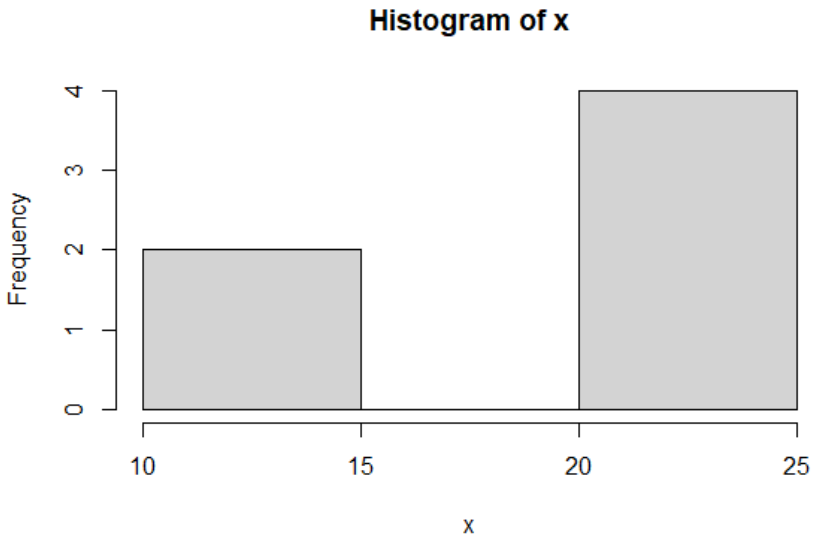
hist(x)

 # Saving the file

dev.off()

Output:

[1] -0.5794294

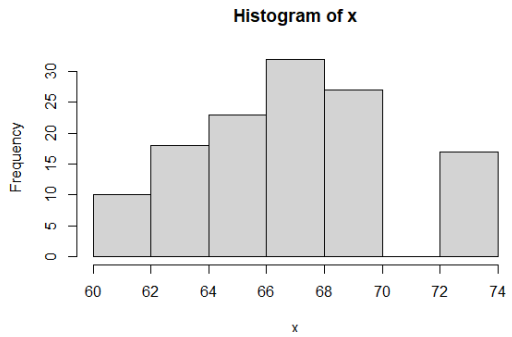

Histogram of x
Frequency
0
1
2
3
4
10
15
20
25
x

Platykurtic

```
# Required for kurtosis() function
library(moments)
# Defining data vector
x <- c(rep(61, each = 10), rep(64, each = 18),
rep(65, each = 23), rep(67, each = 32), rep(70, each = 27),
rep(73, each = 17))
# output to be present as PNG file
png(file = "platykurtic.png")
# Print skewness of distribution
print(kurtosis(x))
# Histogram of distribution
hist(x)
# Saving the file
dev.off()
```

Output:

```
[1] 2.258318
```

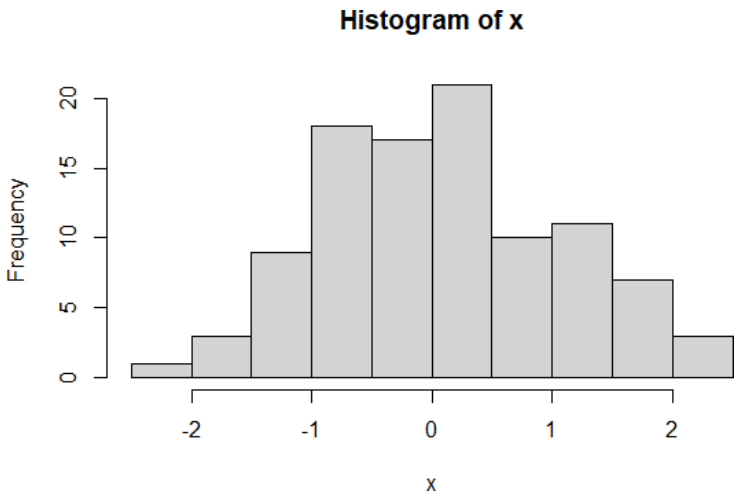


Mesokurtic

```
# Required for kurtosis() function
library(moments)
# Defining data vector
x <- rnorm(100)
# output to be present as PNG file
png(file = "mesokurtic.png")
# Print skewness of distribution
print(kurtosis(x))
# Histogram of distribution
hist(x)
# Saving the file
dev.off()
```

Output:

```
[1] 2.963836
```

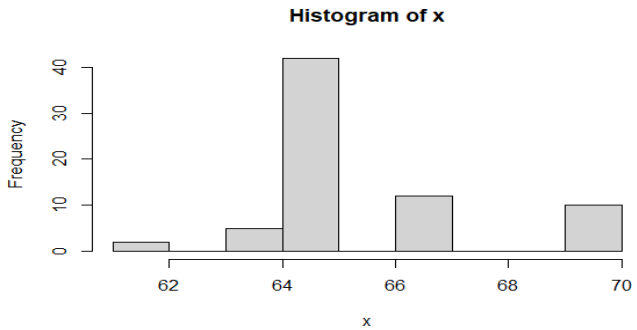


Leptokurtic

```
# Required for kurtosis() function
library(moments)
# Defining data vector
x <- c(rep(61, each = 2), rep(64, each = 5),
rep(65, each = 42), rep(67, each = 12), rep(70, each = 10))
# output to be present as PNG file
png(file = "leptokurtic.png")
# Print skewness of distribution
print(kurtosis(x))
# Histogram of distribution
hist(x)
# Saving the file
dev.off()
```

Output:

```
[1] 3.696788
```



RESULT

Thus, the statistical approach with Skewness and Kurtosis was verified successfully.

Ex: 7 Creating Student's T distribution with inferential statistics

AIM:

To create Student's T Distribution with inferential statistics.

PROCEDURE:

Step 1: we need to find the value of T- distribution at x=0 with 25 degree of freedom

Step 2: we need to Generate a Vector of 100 Values between -6 & 6

Step 3: We to find degree of freedom, plotting normal Distribution, Adding the T-distribution to the plot.

Step 4: Adding the Legend to the plot

Step 5: Finding the P-value and Confidence interval with T-distribution

Step 6: Area to right of a t-statistic with value of 2.1 and 14 degree of freedom.

Pre-lab Discussion

Student's t-distribution, also known as the t-distribution, is a probability distribution that is used in statistics for making inferences about the population mean when the sample size is small or when the population standard deviation is unknown. It is similar to the standard normal distribution (Z-distribution), but it has heavier tails.

Formula

$t = [\bar{x} - \mu] / [s/\sqrt{n}]$

where,

t = The t-score,

$\bar{x}$ = sample mean,

μ = population mean,

s = standard deviation of the sample,

n = sample size

When to Use the t-Distribution?

Student's t Distribution is used when

- The sample size is 30 or less than 30.
- The population standard deviation(σ) is unknown.
- The population distribution must be unimodal and skewed.

Mathematical Derivation of t-Distribution

The t-distribution has been derived mathematically under the assumption of a normally distributed population and the formula for the probability density function will be like this

$$f(t) = \frac{\Gamma\left(\frac{df+1}{2}\right)}{\sqrt{df\,\pi}\,\Gamma\left(\frac{df}{2}\right)}\left(1+\frac{t^2}{df}\right)^{-\frac{df+1}{2}}$$

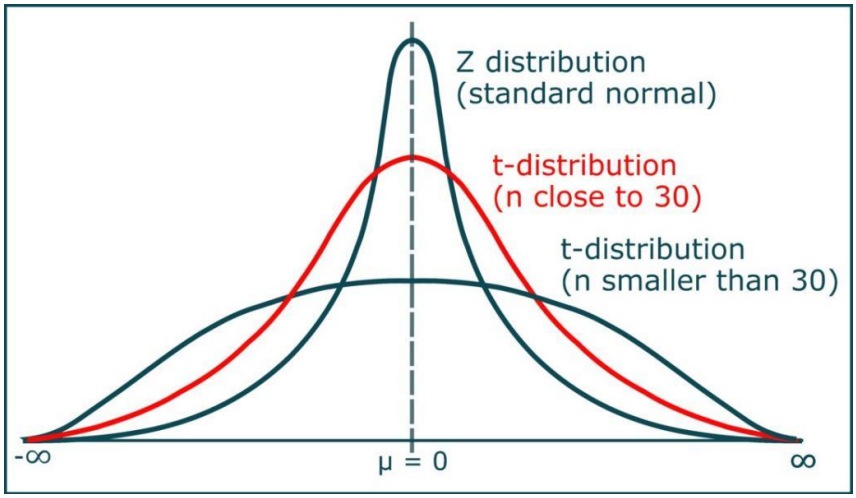
where,

where Γ(.) is the gamma function
df= Degrees of freedom

So, this above equation indicates the probability density function(pdf) of the t-distribution for df degrees of freedom.

Properties of the t-Distribution

- The variable in t-distribution ranges from -∞ to +∞ (**-∞ < t <** +∞).
- t- distribution will be symmetric like the normal distribution if the power of t is even in the probability density function(pdf).
- For large values of ν(i.e. increased sample size n); the t-distribution tends to a standard normal distribution. This implies that for different ν values, the shape of t-distribution also differs.
- The t-distribution is less peaked than the normal distribution at the center and higher peaked in the tails. From the above diagram, one can observe that the red and green curves are less peaked at the center but higher peaked at the tails than the blue curve.
- The value of y(peak height) attains highest at μ = 0 as one can observe the same in the above diagram.
- The mean of the distribution is equal to 0 for ν > 1 where ν = degrees of freedom, otherwise undefined.
- The median and mode of the distribution is equal to 0.
- The variance is equal to ν / ν-2 for ν > 2 and ∞ for 2 < ν ≤ 4 otherwise undefined.



- Degrees of freedom refer to the number of independent observations in a set of data.

- When estimating a mean score or a proportion from a single sample, the number of independent observations is equal to the sample size minus one.
- Hence, the distribution of the t statistic from samples of size 10 would be described by a t distribution having 10 – 1 or 9 degrees of freedom. Similarly, a t- distribution having 15 degrees of freedom would be used with a sample of size 16.

Program:

a. To find a value of t-distribution at x=1, having certain degrees of freedom, say D_f = 25,

 # value of t-distribution pdf at x = 0 with 25 degrees of freedom

```
dt(x = 1, df = 25)
```

Output

```
0.237211
```

b. Comparison of probability density functions having different degrees of freedom.

 # Generate a vector of 100 values between -6 and 6

```
x <- seq(-6, 6, length = 100)
```

 # Degrees of freedom

```
df = c(1,4,10,30)
colour = c("red", "orange", "green", "yellow","black")
```

 # Plot a normal distribution

```
plot(x, dnorm(x), type = "l", lty = 2, xlab = "t-value", ylab = "Density",main = "Comparison of t-distributions", col = "black")
```

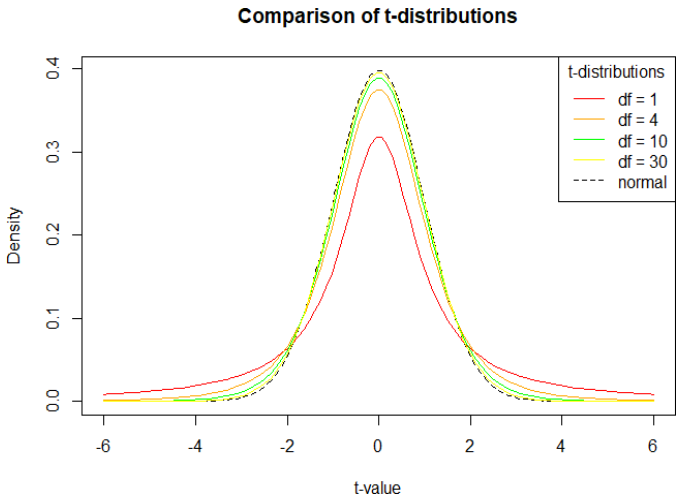
 # Add the t-distributions to the plot

```
for (i in 1:4){
lines(x, dt(x, df[i]), col = colour[i])
}
```

 # Add a legend

```
legend("topright", c("df = 1", "df = 4", "df = 10", "df = 30", "normal"),
       col = colour, title = "t-distributions", lty = c(1,1,1,1,2))
```

Output



c. Finding p-value and confidence interval with t-distribution

 # area to the right of a t-statistic with value of 2.1 and 14 degrees of freedom

pt(q = 2.1, df = 14, lower.tail = FALSE)

Output:

0.02716657

RESULT

Thus, the creation of student's T distribution with inferential statistics was done successfully.

Ex: 8 Missing value analysis with Data Analysis

AIM:

To generate Missing value analysis with Data Analysis techniques.

PROCEDURE:

Step 1: Download & load the Diabetes Missing Value Dataset from web.

Step 2: Summary Statistics of Data

Step 3: Visualize Missing Data Patterns

Step 4: Calculate number of missing for each variable

Step 5: Calculate the percentage of missing values for each variable

Step 6: Impute Missing values with the mean of each column

Step 7: Alternatively, we need to remove rows with missing values.

Pre-lab Discussion:

Missing data is defined as the values or data that is not stored (or not present) for some variables in the given dataset.

How Is a Missing Value Represented in a Dataset?

In the dataset, the blank shows the missing values. In Pandas, usually, missing values are represented by **NaN**. It stands for **Not a Number**.

Why Is Data Missing From the Dataset?

There can be multiple reasons why certain values are missing from the data. Reasons for the missing of data from the dataset affect the approach of handling missing data. So it's necessary to understand why the data could be missing.

Some of the reasons are listed below:

- Past data might get corrupted due to improper maintenance.
- Observations are not recorded for certain fields due to some reasons. There might be a failure in recording the values due to human error.
- The user has not provided the values intentionally
- Item nonresponse: This means the participant refused to respond.

Types of Missing Values

Formally the missing values are categorized as follows:

Figure 1 - Different Types of Missing Values in Datasets

Program:

In R, we can use various packages such as naniar, dplyr, and ggplot2 to perform missing value analysis and visualize missing data patterns. Here's a step-by-step guide:

Step 1: Load Necessary Libraries Start by loading the required libraries for missing value analysis.

```
library(naniar)  # For handling missing data

library(dplyr)    # For data manipulation

library(ggplot2)  # For data visualization
```

Step 2: Load and Inspect the Data Load your dataset into R and examine its structure using functions like str(), head(), and summary().

```
# Load the data (replace 'your_data.csv' with your actual file path)

data <- read.csv("your_data.csv")

# Display the structure of the data

str(data)

# Display the first few rows of the data

head(data)

# Summary statistics of the data

summary(data)
```

Step 3: Check for Missing Values Use the naniar package to visualize and analyze missing values in the dataset.

```
# Visualize missing data patterns
```

```
vis_miss(data)
```

The vis_miss() function creates a matrix plot where missing values are shown as gray cells.

Step 4: Calculate Missing Data Statistics You can use the naniar package in combination with dplyr to calculate statistics related to missing data.

```
# Calculate the number of missing values for each variable

missing_count <- data %>%

  summarise_all(~ sum(is.na(.)))

# Calculate the percentage of missing values for each variable

missing_percentage  <-  data  %>%

 summarise_all(~ mean(is.na(.)) * 100)
```

Step 5: Handle Missing Data Depending on the nature of your data and the extent of missingness, you might choose to handle missing values in various ways, such as imputation or deletion. The appropriate method depends on the specific analysis and the context of your data. For example, if we decide to impute missing values, you can use functions like complete() from the tidyr package or impute() from the naniar package.

```
# Impute missing values with the mean of each column

library(tidyr)

data_imputed <- data %>%

complete()

# Alternatively, you can remove rows with missing values

data_cleaned <- data %>%

drop_na()
```

Result:

Thus, the missing value analysis using data analysis was done successfully.

Ex: 9 Correlation matrix and outlier analysis with EDA

AIM:

To generate correlation matrix and perform outlier analysis using Exploratory Data Analysis techniques.

PROCEDURE:

Step 1: Import the Required libraries Such as Seaborn and pandas

Step 2: Compute the Correlation Matrix

Step 3: Plot the Correlation Matrix Using a Heatmap

Step 4: Calculate the z-score for each numeric column

Step 5: Define a threshold for Outlier detection

Step 6: Find rows with any Column having z-score Greater than the threshold

Step 7: Print the outliers.

Pre-lab Discussion

Exploratory Data Analysis (EDA) is an approach that is used to analyse the data and discover trends, patterns, or check assumptions in data with the help of statistical summaries and graphical representations.

Types of EDA

Depending on the number of columns we are analyzing we can divide EDA into two types.

1. **Univariate Analysis** – In univariate analysis, we analyze or deal with only one variable at a time. The analysis of univariate data is thus the simplest form of analysis since the information deals with only one quantity that changes. It does not deal with causes or relationships and the main purpose of the analysis is to describe the data and find patterns that exist within it.

2. **Bi-Variate analysis –** This type of data involves two different variables. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the relationship between the two variables.

3. **Multivariate Analysis** – When the data involves three or more variables, it is categorized under multivariate.

Depending on the type of analysis we can also subcategorize EDA into two parts.

1. Non-graphical Analysis – In non-graphical analysis, we analyze data using statistical tools like mean median or mode or skewness.

2. Graphical Analysis – In graphical analysis, we use visualizations charts to visualize trends and patterns in the data

Procedure:

- Import the required libraries such as seaborn and pandas
- Load the iris dataset
- Compute the correlation matrix
- Analyze the outliers and print it

PROGRAM:

```
import pandas as pd

import seaborn as sns


# Load the Iris dataset from seaborn

df = sns.load_dataset('iris')


# Compute the correlation matrix

corr_matrix = df.corr()


# Plot the correlation matrix using a heatmap

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')


# Perform outlier detection analysis

# Calculate the Z-score for each numeric column

z_scores = pd.DataFrame((df - df.mean()) / df.std())


# Define a threshold for outlier detection (e.g., Z-score > 3)

threshold = 3
```

 # Find rows with any column having a Z-score greater than the threshold

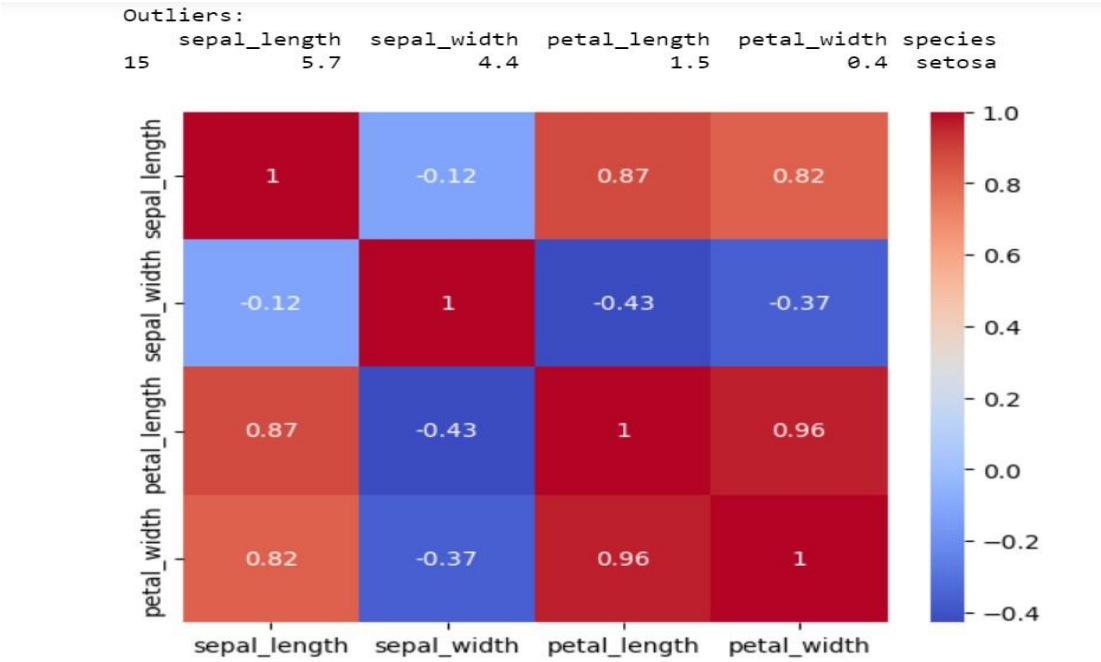
outliers = df[(z_scores > threshold).any(axis=1)]

 # Print the outliers

print("Outliers:")

print(outliers)

OUTPUT:



RESULT:

Thus, the correlation matrix and outlier analysis was done successfully.

Ex: 10 Design and develop Neural Network and SVM

AIM:

To design and develop neural networks and SVM using machine learning.

PROCEDURE:

Step 1: Import the required libraries and load the Dataset

Step 2: The Feature data is assigned to x, and the target variable is assigned to y. split the Dataset into training and testing sets.

Step 3: Design and Develop a Neural Network with a Single hidden Layer Containing 10 neurons and make predictions.

Step 4: Design and Develop a Support vector Machine (SVM) with radial bias function (' rbf ') as kernel and make predictions.

Step 5: Compute and print the accuracies for both Models.

Pre-lab Discussion:

Neural Network

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.

Support Vector Machine

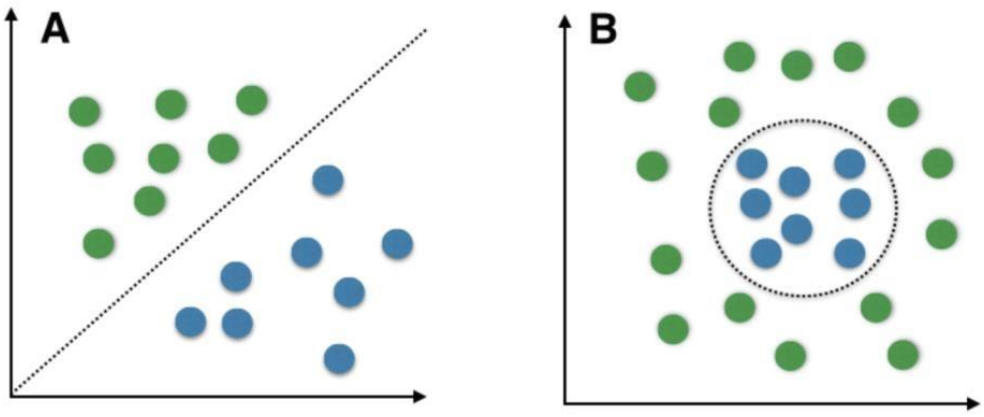
Support Vector Machine (SVM) is a relatively simple **Supervised Machine**

Learning Algorithm used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2-dimensional space, this hyper-plane is nothing but a line. In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyperplane to separate the data. So by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes). However, there are various techniques to use for multi-class problems. **Support Vector Machine for Multi-CLass Problems** To perform SVM on multi-class problems, we can create a binary classifier for each class of the data.

The two results of each classifier will be :

- The data point belongs to that class OR
- The data point does not belong to that class.

For example, in a class of fruits, to perform multi-class classification, we can create a binary classifier for each fruit. For say, the 'mango' class, there will be a binary classifier to predict if it IS a mango OR it is NOT a mango. The classifier with the highest score is chosen as the output of the SVM. **SVM for complex (Non Linearly Separable)** SVM works very well without any modifications for linearly separable data. **Linearly Separable Data** is any data that can be plotted in a graph and can be separated into classes using a straight line.



PROCEDURE:

- Import the required libraries and load the dataset.
- The feature data is assigned to X, and the target variable is assigned to y. Split the dataset into training and testing sets
- Design and develop a neural network with a single hidden layer containing 10 neurons and make predictions.
- Design and develop a Support Vector Machine (SVM) with the radial basis function ('rbf') as the kernel and make predictions.

- Compute and print the accuracies for both models.

PROGRAM:

```
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn import svm
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Design and develop a neural network
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)
mlp.fit(X_train, y_train)

# Make predictions using the neural network
y_pred_nn = mlp.predict(X_test)

# Compute the accuracy of the neural network
accuracy_nn = accuracy_score(y_test, y_pred_nn)
print("Neural Network Accuracy:", accuracy_nn)

# Design and develop a Support Vector Machine (SVM)
svm_model = svm.SVC(kernel='rbf', random_state=42)
svm_model.fit(X_train, y_train)

# Make predictions using the SVM
```

```
y_pred_svm = svm_model.predict(X_test)

# Compute the accuracy of the SVM
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print("SVM Accuracy:", accuracy_svm)
```

OUTPUT:

```
Neural Network Accuracy :0.9666666666666666667
SVM Accuracy : 1.0
```

RESULT

Thus the program to design Neural Nets and SVM was done successfully.

Ex: 11 Develop Machine Learning on Cloud Platform

AIM:

To develop Machine Learning on Cloud Platform.

PROCEDURE:

Step 1: We need to install and Load the Required packages

Step 2: Load the iris dataset

Step 3: Split the Data into features (X) and target (y)

Step 4: Split Data into training and testing sets

Step 5: Train the Support vector Machine Classifier

Step 6: Save and Load the Trained Model

Step 7: Take an example data to classify

Step 8: Make the predictions over Example Data.

Pre-lab Discussion

Developing a machine learning model on a cloud platform using Python typically involves the following steps:

Data Preparation: Prepare your dataset, perform data cleaning, preprocessing, and split it into training and testing sets.

Model Development: Choose a machine learning algorithm, build and train the model on the training data.

Model Evaluation: Evaluate the model's performance on the testing data and make necessary adjustments.

Model Deployment: Deploy the trained model on the cloud platform to make predictions on new data.

Step 1: Data Preparation assume you have the Iris dataset, which is a popular dataset for classification tasks. The dataset contains features of iris flowers (sepal length, sepal width, petal length, petal width) and their corresponding species (Setosa, Versicolor, Virginica).

Step 2: Create a Python Script for Model Training Save the following Python script as train_model.py.

Step 3: Upload Data and Script to GCP Storage Upload the iris.csv and train_model.py files to a bucket on GCP Storage.

Step 4: Create a Training Job on AI Platform

Go to the GCP Console (https://console.cloud.google.com).

Navigate to AI Platform > Jobs.

Click on "New training job."

Provide a job name and select the region where you want the training to take place.

In the "Training package" section, provide the path to the GCP Storage bucket where your train_model.py script is stored.

Specify the Python version (e.g., Python 3.7).

Under "Python module name," enter train_model.

In "Training input arguments," provide any necessary arguments (e.g., paths to input data).

Save the job configuration and start the training job.

Step 5: Monitor the Training Job You can monitor the progress of the training job in the GCP Console under AI Platform > Jobs.

Step 6: Get the Trained Model Once the training job is complete, the trained model will be saved as iris_classifier.joblib in the specified output directory on GCP Storage.

Step 7: Use the Trained Model for Inference Now, you can use the trained model to make predictions on new data. For example, create a new Python script predict.py:

Step 8: Run Inference You can run the predict.py script locally or deploy it to a server to perform real-time predictions.

Program

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import joblib

# Load the Iris dataset
data = pd.read_csv("iris.csv")

# Split data into features (X) and target (y)
X = data.drop("species", axis=1)
y = data["species"]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Support Vector Machine classifier
classifier = SVC(kernel='linear')
classifier.fit(X_train, y_train)

# Save the trained model
joblib.dump(classifier, "iris_classifier.joblib")
```

```
import joblib

# Load the trained model
classifier = joblib.load("gs://your-bucket-name/iris_classifier.joblib")

# Example data to classify
new_data = [[5.1, 3.5, 1.4, 0.2],  # Sample data for Setosa
            [6.1, 3.0, 4.6, 1.4],  # Sample data for Versicolor
            [6.8, 3.2, 5.9, 2.3]]  # Sample data for Virginica

# Make predictions
predictions = classifier.predict(new_data)
print(predictions)
```

RESULT

Thus, the program to develop Machine Learning on Cloud Platform using Python executed successfully.

CONTENT BEYOND SYLLABUS

HYPOTHESIS TESTING

Hypothesis testing is a statistical method that is used in making a statistical decision using experimental data. Hypothesis testing is basically an assumption that we make about a population parameter. It evaluates two mutually exclusive statements about a population to determine which statement is best supported by the sample data.

Example: You say an average student in the class is 30 or a boy is taller than a girl. All of these is an assumption that we are assuming and we need some statistical way to prove these. We need some mathematical conclusion whatever we are assuming is true.

Need for Hypothesis Testing

Hypothesis testing is an important procedure in statistics. Hypothesis testing evaluates two mutually exclusive population statements to determine which statement is most supported by sample data. When we say that the findings are statistically significant, it is thanks to hypothesis testing.

Parameters of hypothesis testing

- **Null hypothesis(H0):** In statistics, the null hypothesis is a general given statement or default position that there is no relationship between two measured cases or no relationship among groups. In other words, it is a basic assumption or made based on the problem knowledge.

  Example: A company production is = 50 units/per day etc.

- **Alternative hypothesis(H1):** The alternative hypothesis is the hypothesis used in hypothesis testing that is contrary to the null hypothesis.

  Example: A company's production is not equal to 50 units/per day etc.

- **Level of significance** It refers to the degree of significance in which we accept or reject the null hypothesis. 100% accuracy is not possible for accepting a hypothesis, so we, therefore, select a level of significance that is usually 5%. This is normally denoted with and generally, it is 0.05 or 5%, which means your output should be 95% confident to give a similar kind of result in each sample.

- **P-value** The P value, or calculated probability, is the probability of finding the observed/extreme results when the null hypothesis(H0) of a study-given problem is true. If your P-value is less than the chosen significance level then you reject the null hypothesis i.e. accept that your sample claims to support the alternative hypothesis.

Steps in Hypothesis Testing

- **Step 1**– We first identify the problem about which we want to make an assumption keeping in mind that our assumption should be contradictory to one another
- **Step 2** – We consider statical assumption such that the data is normal or not, statical independence between the data.
- **Step 3** – We decide our test data on which we will check our hypothesis
- **Step 4** – The data for the tests are evaluated in this step we look for various scores in this step like z-score and mean values.
- **Step 5 –** In this stage, we decide where we should accept the null hypothesis or reject the null hypothesis.

Example: Given a coin and it is not known whether that is fair or tricky so let's decide the null and alternate hypothesis

- Null Hypothesis(H0): a coin is a fair coin.
- Alternative Hypothesis(H1): a coin is a tricky coin.

i.e. our null- hypothesis does not hold good so we need to reject and propose that this coin is a tricky coin which is actually because it gives us 6 consecutive heads.

Formula For Hypothesis Testing

To validate our hypothesis about a population parameter we use statistical functions. we use the z-score, p-value, and, level of significance(alpha) to make evidence for our hypothesis.

$$z = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

Program:

```
import numpy as np

from scipy.stats import norm

def hypothesis_test(sample, pop_mean,

        alpha=0.05, two_tailed=True):

        # len sample dataset

        n = len(sample)

        # mean and stard-deviation of dataset
```

```
sample_mean = np.mean(sample)

sample_std = np.std(sample, ddof=1)

# Calculate the test statistic

z = (sample_mean - pop_mean) / (sample_std / np.sqrt(n))



# Calculate the p-value based on the test type

if two_tailed:
        p_value = 2 * (1 - norm.cdf(abs(z)))
else:
        if z < 0:
                p_value = norm.cdf(z)
      else:
      p_value = 1 - norm.cdf(z)

                # Determine whether

                to reject or fail to #

                reject the null

                hypothesis

                if p_value < alpha:
                        result = "reject"
                else:
                        result = "fail to reject"


              return z, p_value, result
```

VIVA QUESTIONS

1. What is the difference between a population and a sample?

2. What is the difference between inferential and descriptive statistics?

3. What are quantitative and qualitative data?

4. What is the meaning of standard deviation?

5. How do you calculate the needed sample size?

6. Give an example where the median is better measure than mean?

7. What are the types of sampling in statistics?

8. What do you understand by the term Normal Distribution?

9. What is the assumption of normality?

10. How do you convert a normal distribution to standard distribution?

11. What are left-skewed and right-skewed distribution?

12. List some of the properties of a normal distribution?

13. What is a outlier?

14. Mention methods to screen for outliers in a dataset?

15. What is hypothesis testing?

16. What is the p-value in hypothesis testing?

17. When should we use a t-test and z-test?

18. What is the difference between one-tail and two-tail hypothesos testing?

19. What is the difference between type-I and type-II error?

20. What are correlation and covariance in statistics?

21. Define vector.

22. Define list.

23. Define data frame.

24. Give names of those packages which are used for data imputation.

25. Define Data visualization.