

**Exp-1****Downloading and installing Hadoop on Ubuntu, Understanding different Hadoop modes, Startup scripts, Configuration files****Aim:**

To successfully install, configure, and run Hadoop on a local system using a single-node setup.

**Procedure:****1. Install Java and SSH:**

- Update your package lists and install OpenJDK 8 and SSH.

```
sudo apt update
```

```
sudo apt install openjdk-8-jdk
```

```
java -version # Verify Java installation
```

```
sudo apt install ssh
```

**2. Create Hadoop User:**

- Add a dedicated user for Hadoop and generate SSH keys for passwordless SSH.

```
sudo adduser hadoop
```

```
su - hadoop # Switch to Hadoop user
```

```
ssh-keygen -t rsa
```

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
chmod 640 ~/.ssh/authorized_keys
```

```
ssh localhost # Test SSH connection to localhost
```

**3. Download and Install Hadoop:**

- Download the latest Hadoop version (3.3.6), extract the tarball, and move it to the desired location.

```
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
```

```
tar -xvzf hadoop-3.3.6.tar.gz
```

```
mv hadoop-3.3.6 hadoop
```

**4. Configure Environment Variables:**

- **Update .bashrc to include Hadoop and Java paths.**

```
nano ~/.bashrc
```

```
# Add the following lines at the end
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

```
export HADOOP_HOME=$HOME/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
source ~/.bashrc # Apply changes
```

## 5. Edit Hadoop Configuration Files:

- Modify configuration files to set up the necessary Hadoop directories and services.
- **core-site.xml:**

```
nano $HADOOP_HOME/etc/hadoop/core-site.xml
```

# Add between <configuration></configuration>:

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
```

- **hdfs-site.xml:**

```
nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

**Add:**

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
</property>
```

- **mapred-site.xml:**

```
cp $HADOOP_HOME/etc/hadoop/mapred-site.xml.template
  $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

```
nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

**Add:**

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

- **yarn-site.xml:**

```
nano $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

**Add:**

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
```

**6. Format the NameNode:**

- Format the HDFS NameNode.

```
hdfs namenode -format
```

**7. Start Hadoop:**

- Start Hadoop services (NameNode, DataNode, ResourceManager, and NodeManager).

```
start-all.sh
```

```
jps # Verify running services
```

**8. Access Web Interfaces:**

- Verify that Hadoop is running by accessing the following URLs:

- **NameNode:** <http://localhost:9870>
- **Resource Manager:** <http://localhost:8088>

**9. Stop Hadoop Cluster:**

- Stop all Hadoop services.

```
stop-all.sh
```

```

ram@Ubuntu: ~
ram@Ubuntu:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as ram in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.

Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [Ubuntu]
2024-10-01 20:50:49,041 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Starting resourcemanager
Starting nodemanagers
ram@Ubuntu:~$
ram@Ubuntu:~$

```

```

ram@Ubuntu:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as ram in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.

Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [Ubuntu]
2024-10-01 20:50:49,041 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Starting resourcemanager
Starting nodemanagers
ram@Ubuntu:~$
ram@Ubuntu:~$ jdc
Command 'jdc' not found, but there are 15 similar ones.
ram@Ubuntu:~$ jps
4817 SecondaryNameNode
5730 Jps
5091 ResourceManager
5205 NodeManager
4502 NameNode
4621 DataNode
ram@Ubuntu:~$

```

```

library for your platform... using builtin-java classes where applicable
Starting resourcemanager
Starting nodemanagers
ram@Ubuntu:~$
ram@Ubuntu:~$ jdc
Command 'jdc' not found, but there are 15 similar ones.
ram@Ubuntu:~$ jps
4817 SecondaryNameNode
5730 Jps
5091 ResourceManager
5205 NodeManager
4502 NameNode
4621 DataNode
ram@Ubuntu:~$ hadoop version
Hadoop 3.4.0
Source code repository git@github.com:apache/hadoop.git -r bd8b77f398f626bb77917
83192ee7a5dfaec760
Compiled by root on 2024-03-04T06:29Z
Compiled on platform linux-aarch_64
Compiled with protoc 3.21.12
From source with checksum f7fe694a3613358b38812ae9c31114e
This command was run using /home/sat/hadoop-3.4.0/share/hadoop/common/hadoop-com
mon-3.4.0.jar
ram@Ubuntu:~$

```

Run (Running) - Oracle VM VirtualBox

File Machine View Help Devices Help  
Activities Firefox Web Browser

Oct 1, 20:54

Homebrew Information All Applications

localhost:8171/health.html#/tab-overview

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

### Overview 'localhost:9000' (✓active)

Started:	Tue Oct 01 20:50:36 +0530 2024
Version:	3.4.0, rbd8b77f398f626bb7791783192ee7a5dfaec760
Compiled:	Mon Mar 04 11:59:00 +0530 2024 by root from (HEAD detached at release-3.4.0-RC3)
Cluster ID:	CID-653f8afa-bc40-4111-9842-8c068261eoad
Block Pool ID:	BP-750355565-127.0.1.1-1724908368015

### Summary

Security is off.  
SafeMode is off.

135 files and directories, 82 blocks (82 replicated blocks, 0 erasure coded block groups) = 217 total filesystem objects).

Heap Memory used 153.93 MB of 473.5 MB Heap Memory. Max Heap Memory is 2.28 GB.

Non Heap Memory used 53.68 MB of 55.16 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	28.87 GB
Configured Remote Capacity:	0 B
DFS Used:	24.1 MB (0.08%)
Non DFS Used:	19.98 GB
DFS Remaining:	7.37 GB (25.55%)

**RESULT:**

The step-by-step installation and configuration of Hadoop on Ubuntu system have been successfully completed.

Expt-2

## Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

### AIM:

To run a basic Word Count MapReduce program using Hadoop.

### PROCEDURE:

#### 1. Create Data File:

```
nano word_count_data.txt
```

#### Example content for word\_count\_data.txt:

Hadoop is a framework that allows for distributed processing of large data sets.

#### 2. Mapper Program (mapper.py):

```
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print(f'{word}\t1')
```

#### 3. Reducer Program (reducer.py):

```
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)

    try:
        count = int(count)
    except ValueError:
        continue
```

```
if current_word == word:

    current_count += count

else:

    if current_word:

        print(f'{current_word}\t{current_count}')

    current_count = count

    current_word = word
```

```
if current_word == word:

    print(f'{current_word}\t{current_count}')
```

**4. Set Hadoop Environment:**

```
hdfs dfs -mkdir /word_count_input

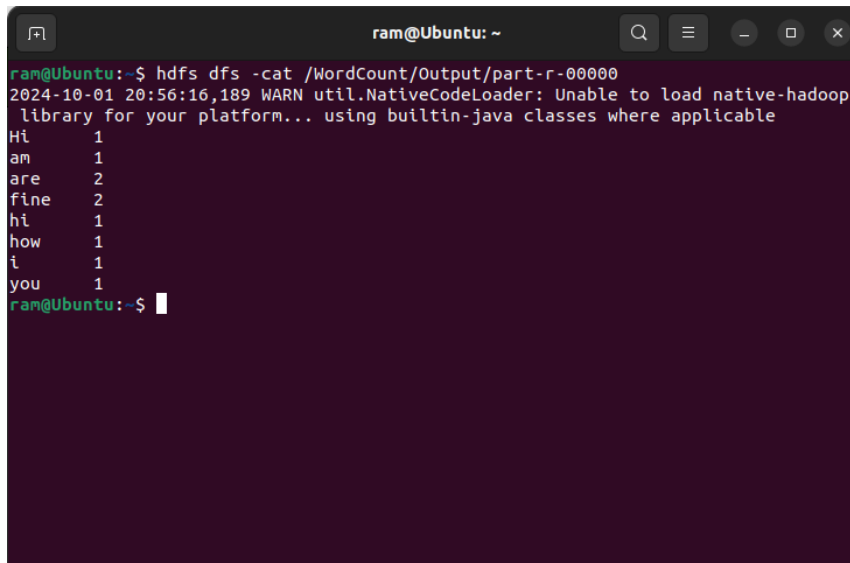
hdfs dfs -copyFromLocal word_count_data.txt /word_count_input
```

**5. Run Word Count Program:**

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
-input /word_count_input/word_count_data.txt \
-output /word_count_output \
-mapper mapper.py \
-reducer reducer.py
```

**6. Check Output:**

```
hdfs dfs -cat /word_count_output/part-00000
```

**OUTPUT:**A terminal window titled 'ram@Ubuntu: ~' with search, menu, and window control icons. It shows the command 'hdfs dfs -cat /WordCount/Output/part-r-00000' being executed. The output lists words and their counts: 'Hi 1', 'am 1', 'are 2', 'fine 2', 'hi 1', 'how 1', 'i 1', and 'you 1'. A warning message from 'util.NativeCodeLoader' is also visible.

```
ram@Ubuntu:~$ hdfs dfs -cat /WordCount/Output/part-r-00000
2024-10-01 20:56:16,189 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Hi      1
am      1
are     2
fine    2
hi      1
how     1
i       1
you     1
ram@Ubuntu:~$
```

**RESULT:**

Thus, the program for basic Word Count Map Reduce has been executed successfully.



Expt-3

## Map Reduce program to process a weather dataset.

### AIM:

To implement MapReduce program to process a weather dataset.

### PROCEDURE:

#### 1. Create Weather Dataset:

```
nano weather_data.txt
```

##### Example content:

```
20220101 30.5
```

```
20220102 29.8
```

#### 2. Mapper Program (mapper.py):

```
#!/usr/bin/env python3

import sys

for line in sys.stdin:

    line = line.strip()

    month = line[4:6] # Extracting month

    temp = line[7:11] # Extracting temperature

    print(f'{month}\t{temp}')
```

#### 3. Reducer Program (reducer.py):

```
#!/usr/bin/env python3

import sys

current_month = None

current_max_temp = -float('inf')

for line in sys.stdin:

    line = line.strip()

    month, temp = line.split('\t')

    try:
```

```
        temp = float(temp)
    except ValueError:
        continue

    if current_month == month:
        current_max_temp = max(current_max_temp, temp)
    else:
        if current_month:
            print(f'{current_month}\t{current_max_temp}')
            current_month = month
            current_max_temp = temp

    if current_month == month:
        print(f'{current_month}\t{current_max_temp}')
```

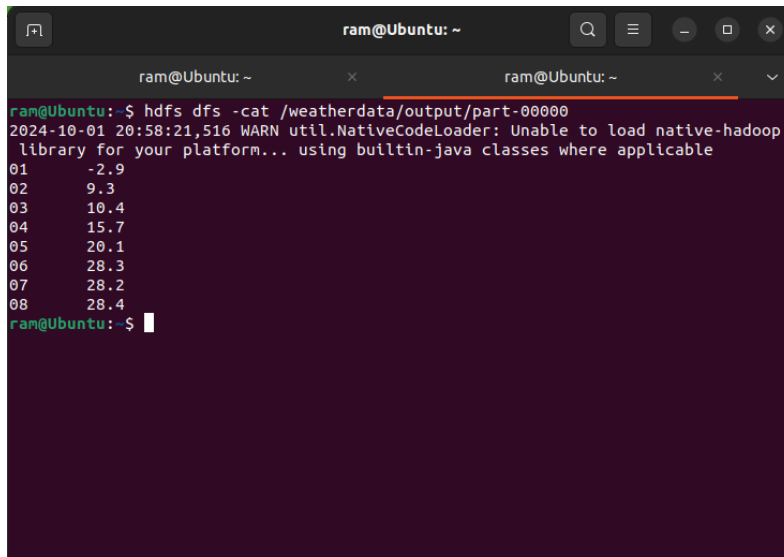
#### 4. Run the Program:

```
hdfs dfs -mkdir /weatherdata
hdfs dfs -copyFromLocal weather_data.txt /weatherdata

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
-input /weatherdata/weather_data.txt \
-output /weatherdata/output \
-mapper mapper.py \
-reducer reducer.py
```

#### 5. Check Output:

```
hdfs dfs -cat /weatherdata/output/part-00000
```

**OUTPUT:**A terminal window titled 'ram@Ubuntu: ~' with search, menu, and window control icons. It shows the command 'hdfs dfs -cat /weatherdata/output/part-00000' being executed. The output consists of a warning message and a list of eight data points. The warning message is '2024-10-01 20:58:21,516 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable'. The data points are: 01 -2.9, 02 9.3, 03 10.4, 04 15.7, 05 20.1, 06 28.3, 07 28.2, and 08 28.4. The prompt 'ram@Ubuntu: ~\$' is visible at the bottom.

```
ram@Ubuntu: ~$ hdfs dfs -cat /weatherdata/output/part-00000
2024-10-01 20:58:21,516 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
01      -2.9
02       9.3
03      10.4
04      15.7
05      20.1
06      28.3
07      28.2
08      28.4
ram@Ubuntu: ~$
```

**RESULT:**

Thus, the program for weather dataset using Map Reduce has been executed successfully.

Expt-4

## **Create UDF (User Defined Functions) in Apache Pig and execute it in MapReduce / HDFS mode**

### **AIM:**

To create UDF in Apache Pig and execute it in MapReduce/HDFS mode.

### **Procedure:**

#### **Step 1: Install and Configure Apache Pig**

##### **1. Download Apache Pig:**

**Download the latest version of Pig from the official website:**

```
wget https://dlcdn.apache.org/pig/pig-0.16.0/pig-0.16.0.tar.gz
```

##### **2. Extract Pig:**

```
tar xvzf pig-0.16.0.tar.gz
```

##### **3. Move Pig Directory:**

**Move the extracted Pig files to a dedicated folder:**

```
sudo mv pig-0.16.0 /usr/local/pig
```

##### **4. Set Environment Variables:**

**Edit the .bashrc file to set up Pig environment variables:**

```
nano ~/.bashrc
```

**Append the following lines:**

```
export PIG_HOME=/usr/local/pig
```

```
export PATH=$PATH:$PIG_HOME/bin
```

```
export PIG_CLASSPATH=$HADOOP_HOME/conf
```

**Apply the changes:**

```
source ~/.bashrc
```

##### **5. Verify Pig Installation:**

**Run the following command to verify if Pig has been installed correctly:**

```
pig -version
```

## Step 2: Create Sample Data for the Pig Job

1. **Create a Sample Data File:** Create a sample text file (sample.txt) with some dummy data:

```
nano sample.txt
```

**Add the following content:**

```
1,John
```

```
2,Jane
```

```
3,Joe
```

```
4,Emma
```

2. **Upload the Data File to HDFS:** Upload the sample file to Hadoop's distributed file system (HDFS):

```
hdfs dfs -mkdir /piginput
```

```
hdfs dfs -put sample.txt /piginput
```

## Step 3: Write Pig Script for the UDF

1. **Create the Pig Script:**

**Create a new Pig script (demo\_pig.pig):**

```
nano demo_pig.pig
```

Write the following code in the script to load and display the data:

```
pig
```

```
-- Load data from HDFS
```

```
data = LOAD '/piginput/sample.txt' USING PigStorage(',') AS (id:int,  
name:chararray);
```

```
-- Display the loaded data
```

```
DUMP data;
```

## Step 4: Write the UDF in Python

1. **Create the Python UDF:**

**Create a Python file (uppercase\_udf.py) to convert text to uppercase:**

```
nano uppercase_udf.py
```

```
def uppercase(text):  
    return text.upper()  
  
if __name__ == "__main__":  
    import sys  
    for line in sys.stdin:  
        line = line.strip()  
        print(uppercase(line))
```

## 2. Upload the Python UDF to HDFS:

### Upload the UDF to HDFS:

```
hdfs dfs -mkdir /udfs  
hdfs dfs -put uppercase_udf.py /udfs
```

## Step 5: Update Pig Script to Use UDF

### 1. Modify the Pig Script to Include UDF:

Edit the `demo_pig.pig` script to register the UDF and process the data:

```
nano demo_pig.pig  
  
Modify the script as follows:  
  
pig  
  
-- Register the Python UDF script  
REGISTER 'udfs/uppercase_udf.py' USING jython AS myudf;  
  
-- Load data from HDFS  
data = LOAD '/piginput/sample.txt' USING PigStorage(',') AS (id:int,  
name:chararray)  
  
-- Apply UDF to convert names to uppercase  
uppercased_data = FOREACH data GENERATE  
myudf.uppercase(name);  
  
-- Display the transformed data  
DUMP uppercased_data;
```

## Step 6: Run the Pig Script

### 1. Run the Pig Script:

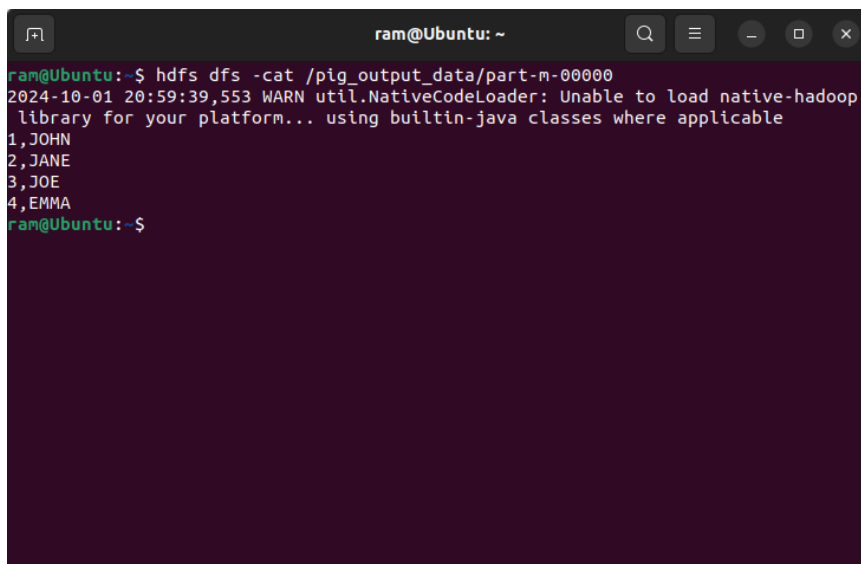
Run the Pig script using the following command:

```
pig -x mapreduce demo_pig.pig
```

### 2. View Output

```
hdfs dfs -cat /pigoutput/part-m-00000
```

## OUTPUT:



```
ram@Ubuntu: ~  
ram@Ubuntu:~$ hdfs dfs -cat /pig_output_data/part-m-00000  
2024-10-01 20:59:39,553 WARN util.NativeCodeLoader: Unable to load native-hadoop  
library for your platform... using builtin-java classes where applicable  
1,JOHN  
2,JANE  
3,JOE  
4,EMMA  
ram@Ubuntu:~$
```

## RESULT:

Thus, UDF in Apache Pig has been created and executed in MapReduce/HDFS mode successfully.

Ex No 5

## Create tables in Hive and write queries to access the data in the table

### AIM:

To create tables in Hive and write queries to access the data in the table.

### PROCEDURE:

#### Step 1: Download and Install Hive

##### 1. Download Hive:

Download Hive from the official website:

```
wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
```

##### 2. Extract Hive:

```
tar -xvf apache-hive-3.1.2-bin.tar.gz
```

##### 3. Move Hive Directory:

```
sudo mv apache-hive-3.1.2-bin /usr/local/hive
```

##### 4. Set Hive Environment Variables:

Edit .bashrc to configure Hive:

```
nano ~/.bashrc
```

Add the following lines:

```
export HIVE_HOME=/usr/local/hive
```

```
export PATH=$PATH:$HIVE_HOME/bin
```

Apply the changes:

```
source ~/.bashrc
```

##### 5. Configure Hive:

Configure Hive to use MySQL as its metastore by editing the Hive configuration file (hive-site.xml):



```
nano $HIVE_HOME/conf/hive-site.xml
```

**Add the following configuration for MySQL connection:**

```
<property>

  <name>javax.jdo.option.ConnectionURL</name>

  <value>jdbc:mysql://localhost/metastore</value>

</property>

<property>

  <name>javax.jdo.option.ConnectionDriverName</name>

  <value>com.mysql.jdbc.Driver</value>

</property>

<property>

  <name>javax.jdo.option.ConnectionUserName</name>

  <value>root</value>

</property>

<property>

  <name>javax.jdo.option.ConnectionPassword</name>

  <value>password</value>

</property>
```

#### 6. Start Hive:

**Once everything is configured, start Hive by simply typing:**

```
hive
```

**Step 2: Create a Database and Table in Hive****1. Create a Database:**

**In the Hive terminal, create a new database:**

```
CREATE DATABASE financials;
```

**2. Use the Database:**

```
USE financials;
```

**3. Create a Table:**

**Create a table to store financial data:**

```
CREATE TABLE finance_table (  
    id INT,  
    name STRING  
)
```

**4. Insert Data into the Table:**

**Insert sample data into the finance\_table:**

```
INSERT INTO TABLE finance_table VALUES (1, 'Alice'), (2, 'Bob'), (3, 'Charlie');
```

**Step 3: Store the Output in HDFS****1. Create a Partitioned Table:**

**For optimized storage, create a partitioned table by year:**

```
CREATE TABLE partitioned_finance_table (  
    id INT,  
    name STRING  
)  
  
PARTITIONED BY (year INT)
```

**2. Insert Data into the Partitioned Table:**

```
INSERT INTO partitioned_finance_table PARTITION (year=2023) VALUES (1, 'Alice'), (2, 'Bob');
```

```
INSERT INTO partitioned_finance_table PARTITION (year=2024) VALUES (3, 'Charlie');
```

### 3. Create a Bucketed Table:

**Create a bucketed table to improve query performance:**

```
CREATE TABLE bucketed_finance_table (  
    id INT,  
    name STRING  
)  
  
CLUSTERED BY (id) INTO 4 BUCKETS
```

### 4. Insert Data into the Bucketed Table:

```
INSERT INTO TABLE bucketed_finance_table VALUES (1, 'Alice'), (2, 'Bob'), (3, 'Charlie');
```

---

## Step 4: View the Output in HDFS

### 1. Create an ORC Table:

**Use ORC (Optimized Row Columnar) format for efficient storage:**

```
CREATE TABLE orc_finance_table (  
    id INT,  
    name STRING  
)
```

### 2. Insert Data into the ORC Table:

```
INSERT INTO TABLE orc_finance_table SELECT * FROM finance_table;
```

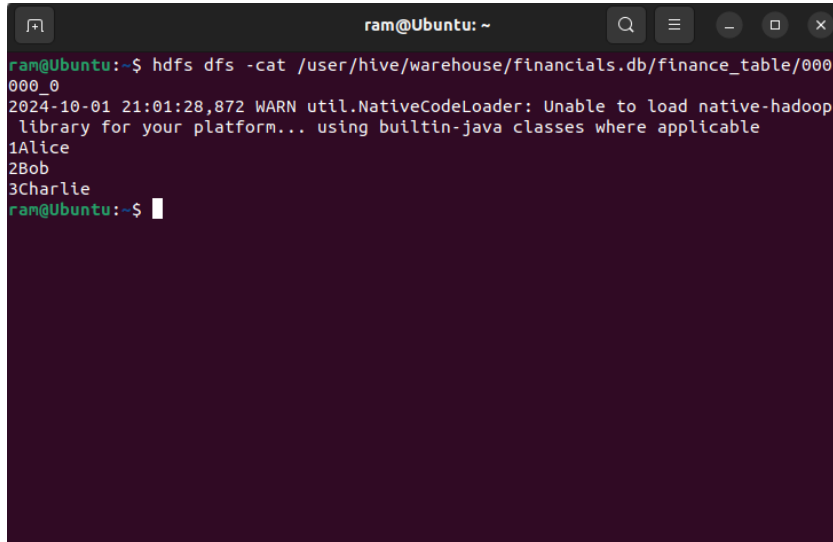
### 3. View the Output in HDFS:

**You can view the output by navigating to the HDFS directory where Hive stores the data. Use the following command to view the stored data:**

```
hdfs dfs -ls /user/hive/warehouse/financials.db/finance_table
```

**To view the contents of the ORC table:**

```
hdfs dfs -cat /user/hive/warehouse/financials.db/orc_finance_table/000000_0
```

**OUTPUT:**A terminal window titled 'ram@Ubuntu: ~' with search, menu, and window control icons. The command 'hdfs dfs -cat /user/hive/warehouse/financials.db/finance\_table/000000\_0' is entered. The output shows a warning message about the native-hadoop library, followed by three lines of data: '1Alice', '2Bob', and '3Charlie'. The prompt 'ram@Ubuntu:~\$' is visible at the bottom.

```
ram@Ubuntu:~$ hdfs dfs -cat /user/hive/warehouse/financials.db/finance_table/000000_0
2024-10-01 21:01:28,872 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
1Alice
2Bob
3Charlie
ram@Ubuntu:~$
```

**RESULT:**

Thus, to create tables in Hive and write queries to access the data in the table was completed successfully.

**Ex No 6**

**Import a JSON file from the command line. Apply the following actions with the data present in the JSON file where, projection, aggregation, remove, count, limit, skip and sort**

**AIM:**

To import a JSON file from the command line and apply the following actions with the data present in the JSON file where, projection, aggregation, remove, count, limit, skip and sort using jq tool.

**PROCEDURE:**

- Create a json file 'employees.json' and provide data in it.
- Open the command prompt.
- Navigate to the folder where employees.json is stored.
- Load and view the JSON data with jq.
- Use the jq commands for projection, aggregation, removal, counting, limiting, and sorting operations.

**employees.json:**

```
[
  {
    "id": 1,
    "name": "Alice Johnson",
    "department": "Engineering",
    "age": 29,
    "salary": 70000
  },
  {
    "id": 2,
    "name": "Bob Smith",
    "department": "Marketing",
```

```
"age": 35,  
  "salary": 55000  
},  
{  
  "id": 3,  
  "name": "Charlie Davis",  
  "department": "Engineering",  
  "age": 25,  
  "salary": 60000  
},  
{  
  "id": 4,  
  "name": "Dana Lee",  
  "department": "Human Resources",  
  "age": 40,  
  "salary": 65000  
},  
{  
  "id": 5,  
  "name": "Eve Martinez",  
  "department": "Finance",  
  "age": 45,  
  "salary": 75000  
}  
]
```

## **OUTPUT:**

### **Running jq queries:**

#### **I. Projection:**

```

ram@Ubuntu: ~
ram@Ubuntu: ~
ram@Ubuntu: ~$ python3 process_data.py
Raw JSON Data: [
{"name": "John Doe", "age": 30, "department": "HR", "salary": 50000},
{"name": "Jane Smith", "age": 25, "department": "IT", "salary": 60000},
{"name": "Alice Johnson", "age": 35, "department": "Finance", "salary": 70000},
{"name": "Bob Brown", "age": 28, "department": "Marketing", "salary": 55000},
{"name": "Charlie Black", "age": 45, "department": "IT", "salary": 80000}
]

```

## II. Aggregation:

```

Aggregation: Calculate total salary
Total Salary: 315000

```

## III. Count:

```

Count: Number of employees earning more than 50000
Number of High Earners (>50000): 4

```

## IV. Remove:

```

Filtered DataFrame (IT department removed):
   name  age department  salary
0  John Doe   30        HR   50000
2  Alice Johnson  35    Finance   70000
3   Bob Brown   28  Marketing   55000

```

## V. Limit:

```
Limit: Top 5 highest salary
      name  age department  salary
4 Charlie Black  45         IT  80000
2 Alice Johnson  35        Finance  70000
1   Jane Smith  25         IT  60000
3   Bob Brown  28        Marketing  55000
0   John Doe   30         HR   50000
```

## VI. Skip:

```
Skipped DataFrame (First 2 rows skipped):
      name  age department  salary
2 Alice Johnson  35        Finance  70000
3   Bob Brown  28        Marketing  55000
4 Charlie Black  45         IT  80000
```

## VII. Sort:

```
Sorted DataFrame by Name:
      name  age department  salary
2 Alice Johnson  35        Finance  70000
3   Bob Brown  28        Marketing  55000
4 Charlie Black  45         IT  80000
1   Jane Smith  25         IT  60000
0   John Doe   30         HR   50000
```

## RESULT:

Thus to import a JSON file from the command line and apply the following actions with the data present in the JSON file where, projection, aggregation, remove, count, limit, skip and sort using jq tool is completed successfully.



**Ex No 7****Implement Linear and Logistic Regression in R****AIM:**

To Implement Linear and Logistic Regression using R

**PROCEDURE:**

- Collect and load the dataset from sources like CSV files or databases.
- Clean and preprocess the data, including handling missing values and encoding categorical variables.
- Split the dataset into training and testing sets to evaluate model performance.
- Normalize or standardize the features to ensure consistent scaling. 5. Choose the appropriate model: Linear Regression for continuous outcomes.
- Train the model on the training data using the `fit` method.
- Make predictions on the testing data using the `predict` method.
- Evaluate the model using metrics like Mean Squared Error (MSE) for Linear Regression or accuracy and confusion matrix for Logistic Regression.
- Visualize the results with plots, such as scatter plots for Linear Regression or decision boundaries for Logistic Regression.
- Fine-tune the model by adjusting hyperparameters or applying regularization Techniques.

**CODE:****LinearRegression.R:**

```
# Sample data
heights <- c(150, 160, 165, 170, 175, 180, 185)
weights <- c(55, 60, 62, 68, 70, 75, 80)

# Create a data frame
data <- data.frame(heights, weights)

# Fit a linear regression model
linear_model <- lm(weights ~ heights, data = data)

# Print the summary of the model
```

```
print(summary(linear_model))  
# Plotting the data and regression line  
plot(data$heights, data$weights,  
      main = "Linear Regression: Weight vs. Height",  
      xlab = "Height (cm)",  
      ylab = "Weight (kg)",  
      pch = 19, col = "blue")  
# Add regression line  
abline(linear_model, col = "red", lwd = 2)
```

### **LogisticRegression.R:**

```
# Load the dataset  
data(mtcars)  
# Convert 'am' to a factor (categorical variable)  
mtcars$am <- factor(mtcars$am, levels = c(0, 1), labels = c("Automatic", "Manual"))  
# Fit a logistic regression model  
logistic_model <- glm(am ~ mpg, data = mtcars, family = binomial)  
# Print the summary of the model  
print(summary(logistic_model))  
# Predict probabilities for the logistic model  
predicted_probs <- predict(logistic_model, type = "response")  
# Display the predicted probabilities  
print(predicted_probs)  
# Plotting the data and logistic regression curve  
plot(mtcars$mpg, as.numeric(mtcars$am) - 1,  
      main = "Logistic Regression: Transmission vs. MPG",  
      xlab = "Miles Per Gallon (mpg)",  
      ylab = "Probability of Manual Transmission",
```

```

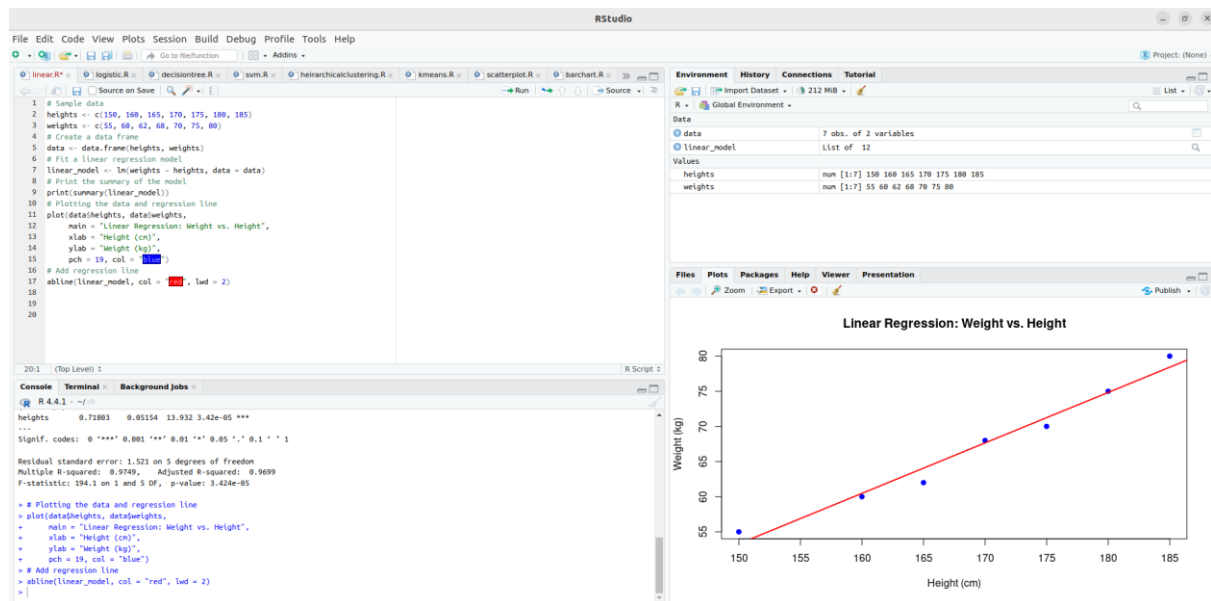
pch = 19, col = "blue")

# Add the logistic regression curve
curve(predict(logistic_model, data.frame(mpg = x), type = "response"),
       add = TRUE, col = "red", lwd = 2)

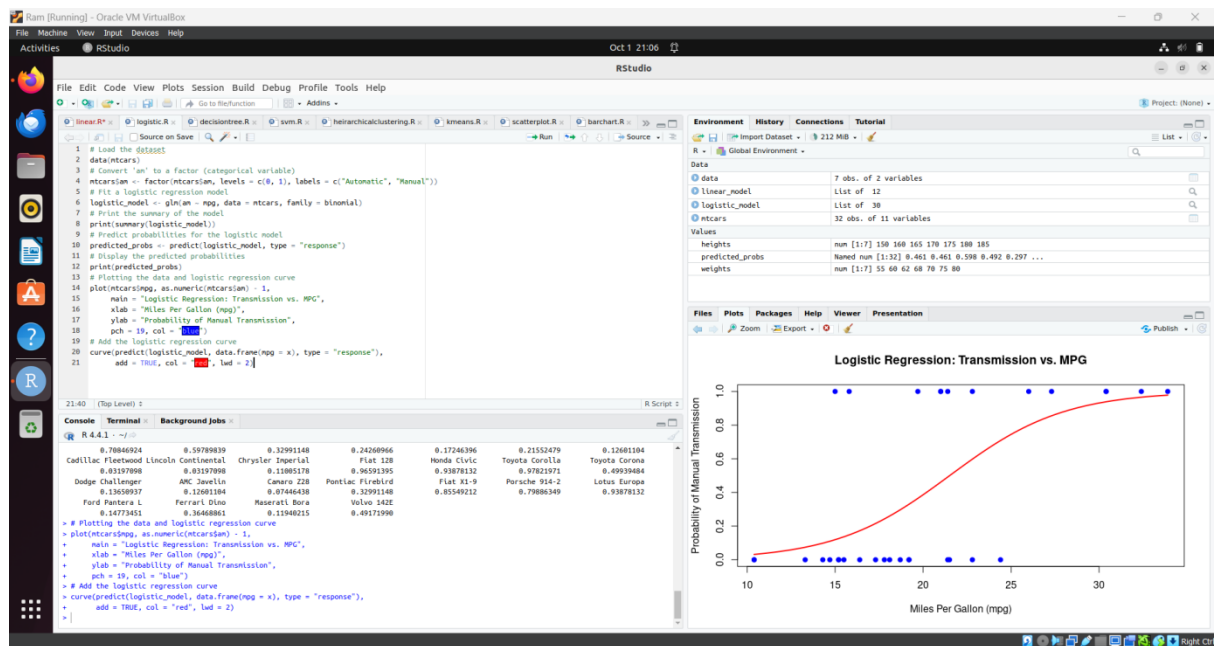
```

## OUTPUT:

### Linear Regression:



## Logistic Regression:



## RESULT:

Thus to Implement Linear and Logistic Regression using R has been successfully executed.

**Ex No 8****Implement SVM/Decision tree classification techniques****AIM:**

To Implement SVM/Decision tree classification techniques using R.

**PROCEDURE:**

- Collect and load the dataset from sources like CSV files or databases.
- Clean and preprocess the data, including handling missing values and encoding categorical variables.
- Split the dataset into training and testing sets to evaluate model performance.
- Normalize or standardize the features, especially for SVM, to ensure consistent scaling.
- Choose the appropriate model: SVM for margin-based classification, Decision Tree for rule-based classification.
- Train the model on the training data using the 'fit' method.
- Make predictions on the testing data using the 'predict' method.
- Evaluate the model using metrics like accuracy, confusion matrix, precision, and recall.
- Visualize the results with plots, such as decision boundaries for SVM or tree structures for Decision Trees.
- Fine-tune the model by adjusting hyperparameters like 'C' for SVM or 'max\_depth' for Decision Trees.

**CODE:****SVM.R:**

```
# Install and load the e1071 package (if not already installed)
install.packages("e1071")
library(e1071)
# Load the iris dataset
data(iris)
```

```
# Inspect the first few rows of the dataset
head(iris)

# Split the data into training (70%) and testing (30%) sets
set.seed(123) # For reproducibility
sample_indices <- sample(1:nrow(iris), 0.7 * nrow(iris))
train_data <- iris[sample_indices, ]
test_data <- iris[-sample_indices, ]

# Fit the SVM model
svm_model <- svm(Species ~ ., data = train_data, kernel = "radial")

# Print the summary of the model
summary(svm_model)

# Predict the test set
predictions <- predict(svm_model, newdata = test_data)

# Evaluate the model's performance
confusion_matrix <- table(Predicted = predictions, Actual = test_data$Species)
print(confusion_matrix)

# Calculate accuracy
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Accuracy:", accuracy * 100, "%\n")
```

### **Decision Tree.R:**

```
# Install and load the rpart package (if not already installed)
install.packages("rpart")
library(rpart)

# Load the iris dataset
data(iris)

# Split the data into training (70%) and testing (30%) sets
set.seed(123) # For reproducibility
```

```
sample_indices <- sample(1:nrow(iris), 0.7 * nrow(iris))
train_data <- iris[sample_indices, ]
test_data <- iris[-sample_indices, ]
# Fit the Decision Tree model
tree_model <- rpart(Species ~ ., data = train_data, method = "class")
# Print the summary of the model
summary(tree_model)
# Plot the Decision Tree
plot(tree_model)
text(tree_model, pretty = 0)
# Predict the test set
predictions <- predict(tree_model, newdata = test_data, type = "class")
# Evaluate the model's performance
confusion_matrix <- table(Predicted = predictions, Actual = test_data$Species)
print(confusion_matrix)
# Calculate accuracy
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Accuracy:", accuracy * 100, "%\n")
```

## OUTPUT:

### SVM in R:

The screenshot displays the RStudio interface with a script editor, console, and environment pane. The script performs the following steps:

- Install and load the `e1071` package.
- Load the `iris` dataset.
- Inspect the first few rows of the dataset.
- Split the data into training (70%) and testing (30%) sets.
- Set a seed for reproducibility.
- Sample indices for training and testing.
- Fit an SVM model using `svm(Species ~ ., data = train_data, kernel = "radial")`.
- Print the summary of the model.
- Predict the test set.
- Evaluate the model's performance using a confusion matrix.
- Calculate the accuracy.
- Print the accuracy.

The console output shows the following results:

```
> # Predict the test set
> predictions <- predict(svm_model, newdata = test_data)
> # Evaluate the model's performance
> confusion_matrix <- table(Predicted = predictions, Actual = test_data$Species)
> print(confusion_matrix)
      Actual
Predicted setosa versicolor virginica
setosa      14         0          0
versicolor  0         17         0
virginica   0          1         13
> # Calculate accuracy
> accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
> cat("Accuracy:", accuracy * 100, "%\n")
Accuracy: 97.77778 %
```

The Environment pane shows the following objects:

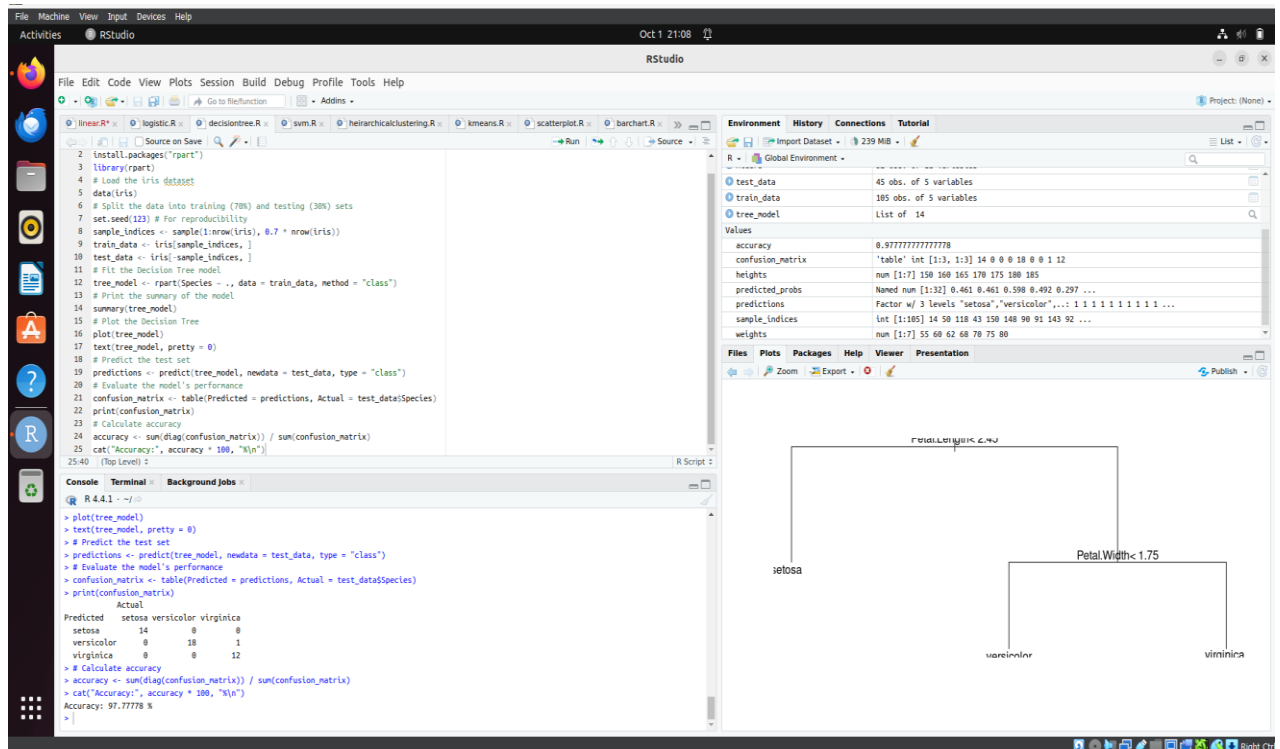
- `data`: 7 obs. of 2 variables
- `iris`: 150 obs. of 5 variables
- `linear_model`: List of 12
- `logistic_model`: List of 30
- `ntcars`: 32 obs. of 11 variables
- `svm_model`: List of 31
- `test_data`: 45 obs. of 5 variables
- `train_data`: 105 obs. of 5 variables
- `tree_model`: List of 14

The Values pane shows the following data:

- `accuracy`: 0.977777777777778
- `confusion_matrix`: 'table' int [1:3, 1:3] 14 0 0 0 17 1 0 0 13
- `heights`: num [1:7] 150 160 165 170 175 180 185
- `predicted_probs`: Named num [1:32] 0.461 0.461 0.598 0.492 0.297 ...
- `predictions`: Factor w/ 3 levels "setosa", "versicolor", ...: 1 1 1 1 1 1 1 1 ...
- `sample_indices`: int [1:185] 14 50 118 43 150 148 90 91 143 92 ...
- `weights`: num [1:7] 55 60 62 68 70 75 80



## Decision tree:



## RESULT:

Thus, Implement SVM and Decision tree classification techniques has been successfully executed.

**Ex No 9****Implement clustering techniques – Hierarchical and K-Means****AIM:**

To Implement clustering techniques – Hierarchical and K-Means using R.

**PROCEDURE:**

- Collect and load the dataset from sources like CSV files or databases.
- Clean and preprocess the data, including handling missing values and scaling features.
- Determine the number of clusters (K) for K-Means, or decide on the stopping criterion for Hierarchical Clustering.
- Choose the appropriate clustering algorithm: K-Means for partitioning, Hierarchical for nested clustering.
- Apply the K-Means algorithm using `fit_predict` to assign data points to clusters.
- Apply the Hierarchical Clustering algorithm using Agglomerative Clustering for hierarchical clusters.
- Visualize the clusters with scatter plots for K-Means, and dendrograms for Hierarchical Clustering.
- Evaluate clustering performance using metrics like silhouette score or inertia (for K-Means).
- Fine-tune the clustering by adjusting the number of clusters or linkage criteria.
- Interpret the results to understand the structure and relationships within the data.

**CODE:****Hierarchical Clustering.R:**

```
# Load the iris dataset
data(iris)

# Use only the numeric columns for clustering (exclude the Species column)
iris_data <- iris[, -5]

# Standardize the data
```

```
iris_scaled <- scale(iris_data)

# Compute the distance matrix
distance_matrix <- dist(iris_scaled, method = "euclidean")

# Perform hierarchical clustering using the "complete" linkage method
hc_complete <- hclust(distance_matrix, method = "complete")

# Plot the dendrogram
plot(hc_complete, main = "Hierarchical Clustering Dendrogram", xlab = "", sub = "", cex = 0.6)

# Cut the tree to form 3 clusters
clusters <- cutree(hc_complete, k = 3)

# Print the cluster memberships
print(clusters)

# Add the clusters to the original dataset
iris$Cluster <- as.factor(clusters)

# Display the first few rows of the updated dataset
head(iris)
```

### **K-Means Clustering.R:**

```
# Load the iris dataset
data(iris)

# Use only the numeric columns for clustering (exclude the Species column)
iris_data <- iris[, -5]

# Standardize the data
iris_scaled <- scale(iris_data)

# Set the number of clusters
set.seed(123) # For reproducibility
k <- 3 # Number of clusters

# Perform K-Means clustering
```

```
kmeans_result <- kmeans(iris_scaled, centers = k, nstart = 25)

# Print the K-Means result
print(kmeans_result)

# Print the cluster centers
print(kmeans_result$centers)

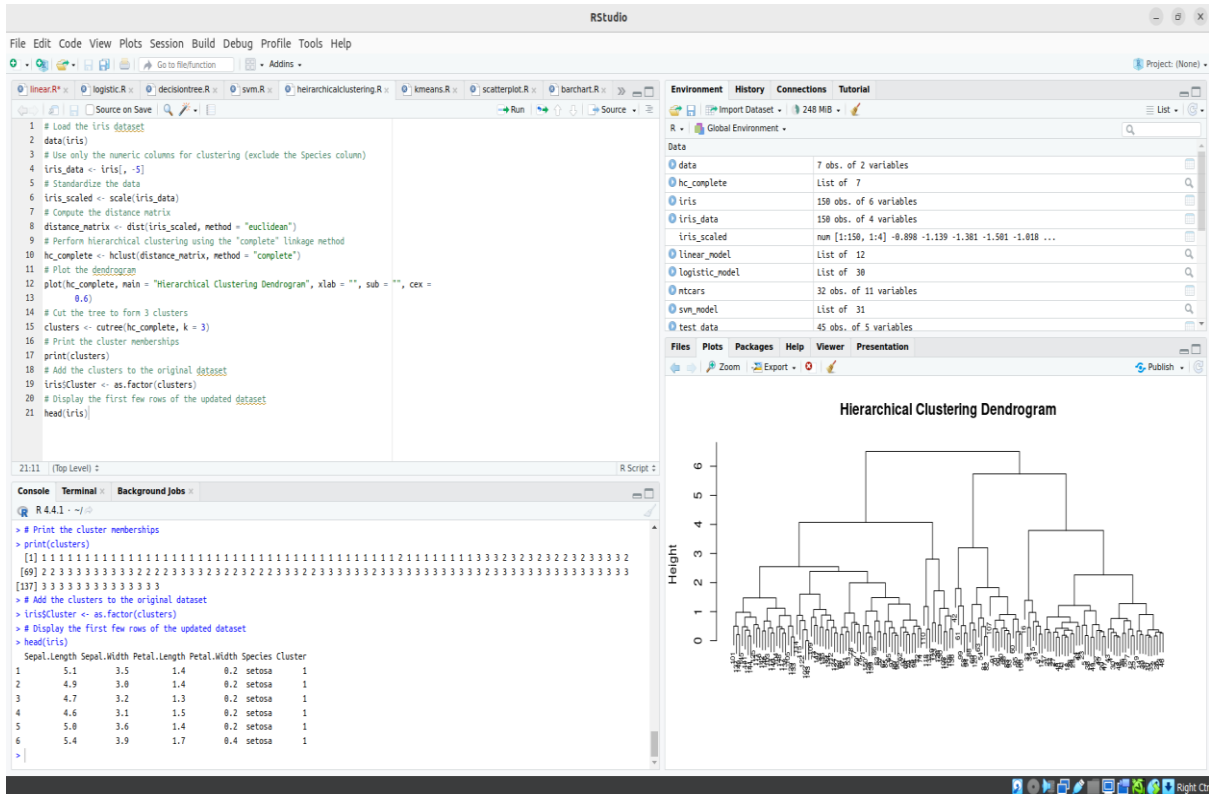
# Add the cluster assignments to the original dataset
iris$Cluster <- as.factor(kmeans_result$cluster)

# Display the first few rows of the updated dataset
head(iris)

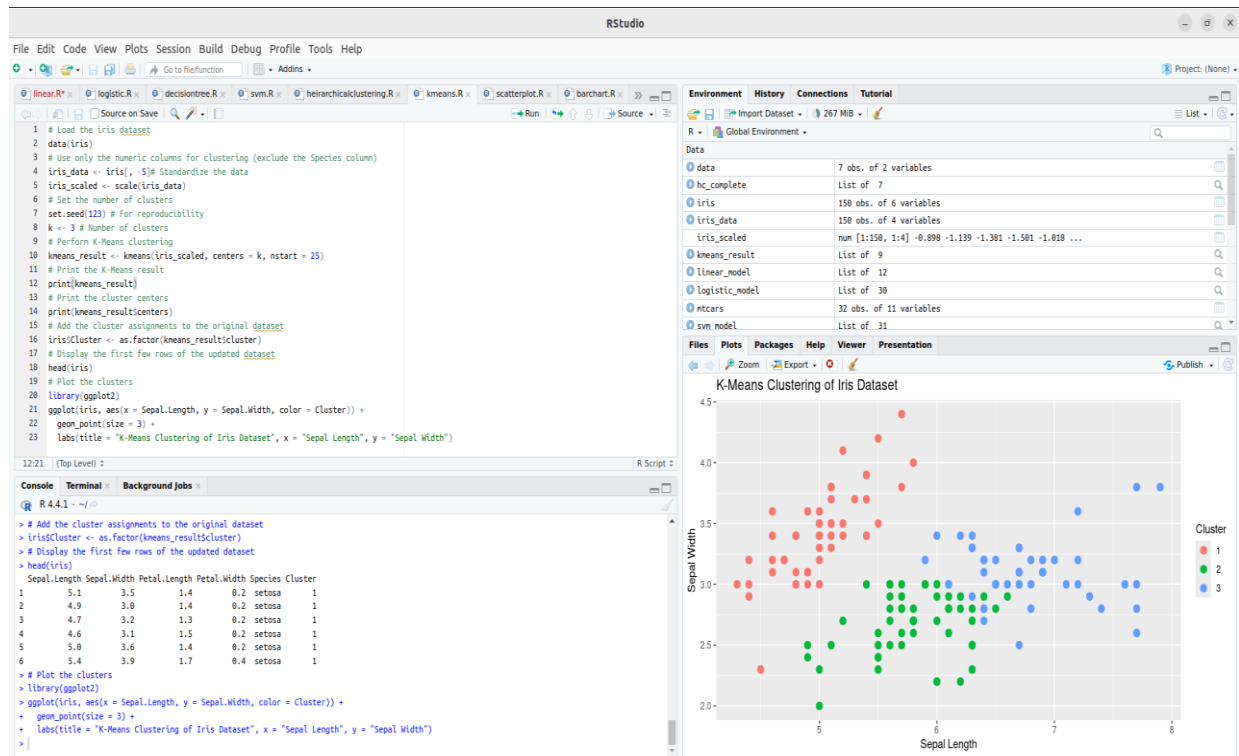
# Plot the clusters
library(ggplot2)

ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Cluster)) +
  geom_point(size = 3) +
  labs(title = "K-Means Clustering of Iris Dataset", x = "Sepal Length", y = "Sepal Width")
```

## Hierarchical Clustering:



## K-Means Clustering:



## RESULT:

Thus, to implement clustering techniques – Hierarchical and K-Means using R has been successfully executed.

**Ex No 10****Visualize Data using Any plotting Framework****AIM:**

To Visualize Data using Any plotting Framework using R programming.

**PROCEDURE:**

- Install Plotly using pip install plotly if it's not already installed.
- Import the necessary libraries: import plotly.express as px and import pandas as pd.
- Load your dataset into a DataFrame using pd.read\_csv() or other data loading methods.
- Explore the dataset to understand its structure, variables, and potential visualizations.
- Choose the appropriate Plotly function (e.g., px.scatter, px.bar, px.line) based on the type of data and the desired plot.
- Define the x and y axes by specifying the columns from the DataFrame.
- Customize the plot by adding titles, labels, color coding, and other plot-specific attributes.
- Add interactive elements like hover data, tooltips, or facet plots for deeper insights.
- Render the plot using fig.show() to display it in a web browser or inline in a notebook.
- Save the plot to an HTML file or as a static image using fig.write\_html() or fig.write\_image().

**CODE:****Scatter Plot.R:**

```
# Install ggplot2 (if not already installed)
install.packages("ggplot2")

# Load the ggplot2 package
library(ggplot2)

# Scatter plot of Sepal.Length vs Sepal.Width, colored by Species
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
```

```
geom_point(size = 3) + # Adds points
labs(title = "Scatter Plot of Sepal Dimensions",
      x = "Sepal Length (cm)",
      y = "Sepal Width (cm)") + # Adds axis labels and title
theme_minimal() # Applies a minimal theme
```

### **Bar Chart.R:**

```
# Install ggplot2 (if not already installed)
install.packages("ggplot2")
# Load the ggplot2 package
library(ggplot2)
# Bar plot of Species counts
ggplot(data = iris, aes(x = Species)) +
  geom_bar(fill = "steelblue") + # Adds bars filled with steel blue color
  labs(title = "Count of Different Species in Iris Dataset",
        x = "Species",
        y = "Count") +
  theme_minimal()
```

### **Histogram.R:**

```
# Install ggplot2 (if not already installed)
install.packages("ggplot2")
# Load the ggplot2 package
library(ggplot2)
# Histogram of Sepal Length
ggplot(data = iris, aes(x = Sepal.Length)) +
```



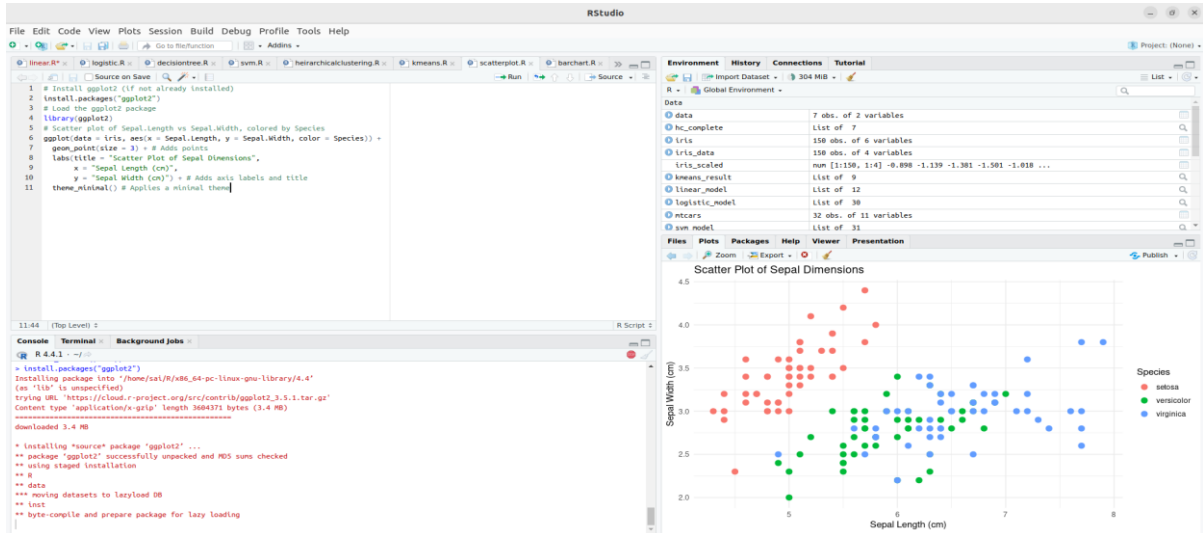
```
geom_histogram(binwidth = 0.3, fill = "orange", color = "black") + # Adds histogram bars
labs(title = "Histogram of Sepal Length",
      x = "Sepal Length (cm)",
      y = "Frequency") +
theme_minimal()
```

**Box Plot.R:**

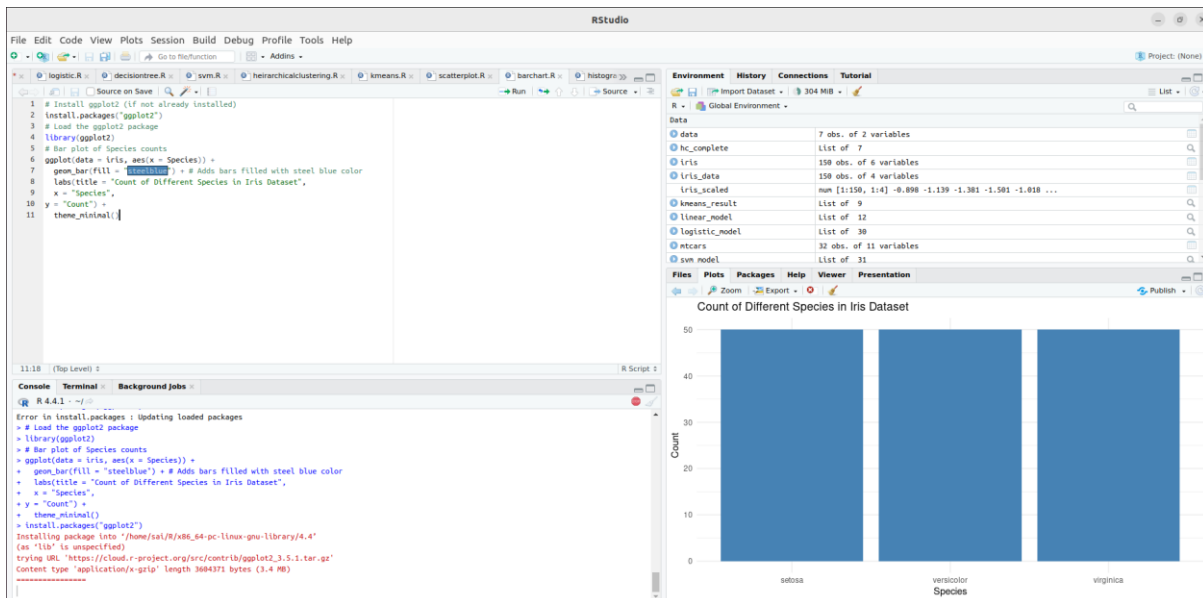
```
# Install ggplot2 (if not already installed)
install.packages("ggplot2")
library(ggplot2)
# Box plot of Sepal Length for each Species
ggplot(data = iris, aes(x = Species, y = Sepal.Length, fill = Species)) +
  geom_boxplot() + # Adds box plot
labs(title = "Box Plot of Sepal Length by Species",
      x = "Species",
      y = "Sepal Length (cm)") +
theme_minimal()
```

## OUTPUT:

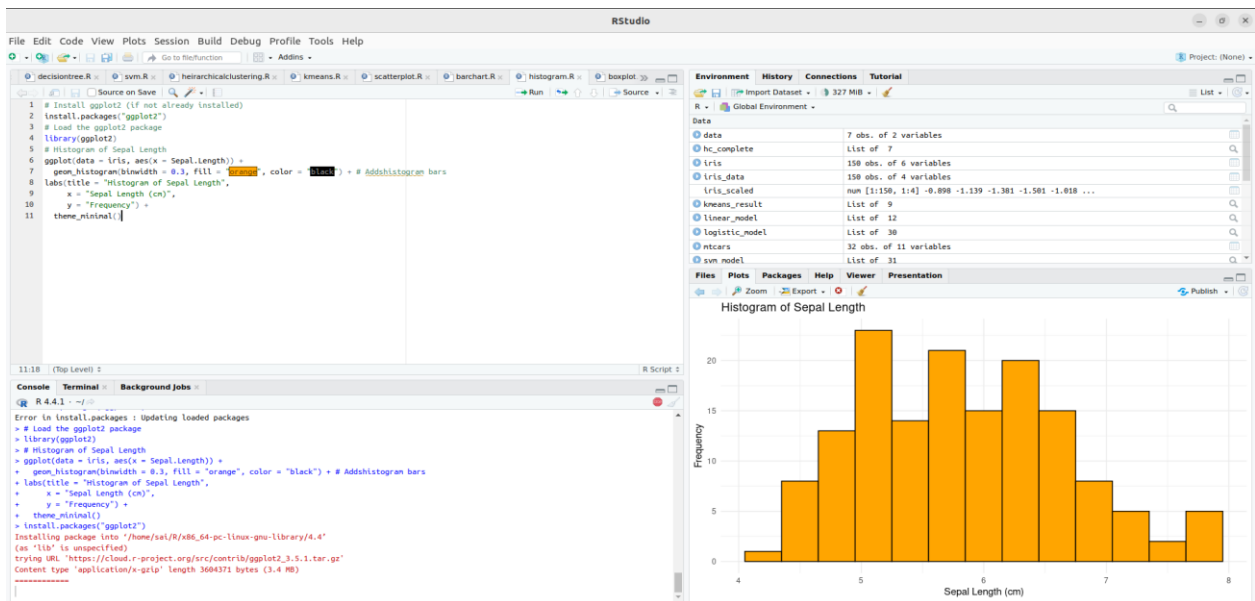
### Scatter Plot:



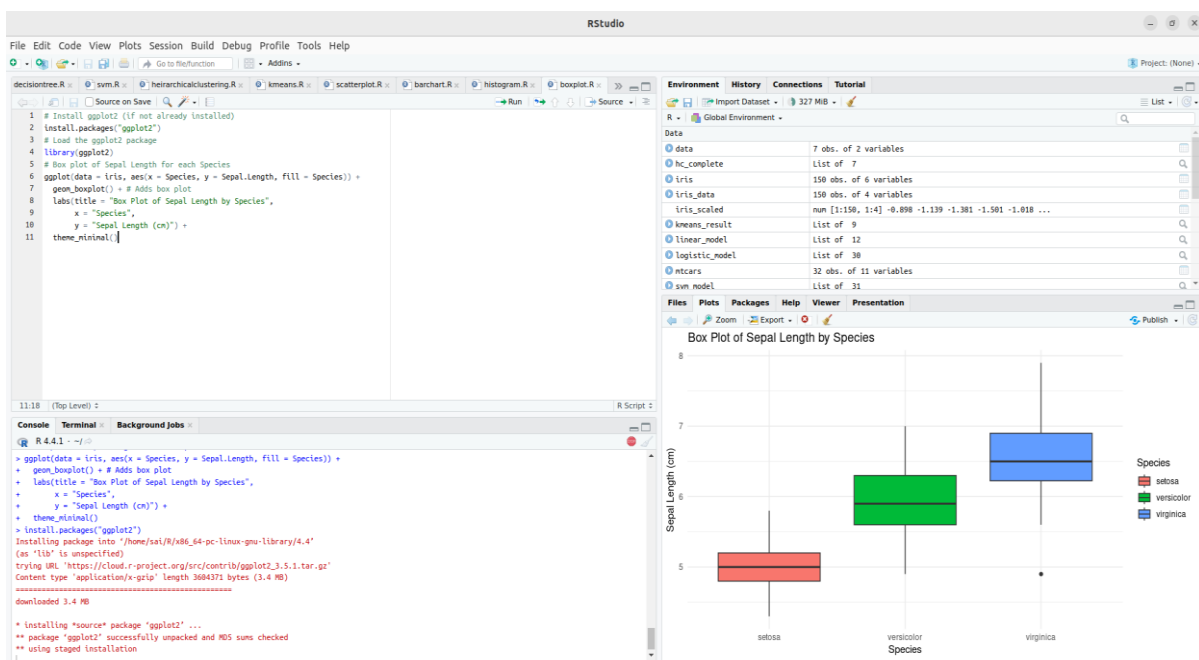
### Bar Chart:



## Histogram:



## Box Plot:



## RESULT:

Thus, Visualizing Data using any plotting framework using R programming has been successfully executed.