INDEX

PRACTICAL NO: 01

AIM: REGRESSION MODEL Import a data from web storage. Name the dataset and now do Logistic Regression to find out relation between variables that are affecting the admission of a student in an institute based on his or her GRE score, GPA obtained and rank of the student. Also check the model is fit or not. require (foreign), require(MASS).

To perform Logistic Regression in R, we will use the UCLA Graduate School Admissions dataset, which is a standard dataset for this type of analysis.

Despite its name, **Logistic Regression** is used for **classification** (e.g., Admit vs. Not Admit). In R, we achieve this using the `glm()` function with a "binomial" family.

Step 1: Load Libraries and Import Data

We will use `foreign` to handle data structures and `MASS` for statistical functions.

How to execute: Open RStudio, go to `File > New File > R Script`, and paste the following:

R
```
# Load required libraries
require(foreign)
require(MASS)

# Import dataset from web storage
# This is a CSV file containing admit, gre, gpa, and rank
admission_data <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv")

# View the first few rows
head(admission_data)
```

Step 2: Data Preparation

For a proper logistic regression, the `rank` of the school should be treated as a **factor** (categorical variable), not a continuous number.

R
```
# Convert rank to a factor
admission_data$rank <- factor(admission_data$rank)

# Summary of data to check for missing values
summary(admission_data)
```

Step 3: Run the Logistic Regression

We want to predict `admit` (the dependent variable) based on `gre`, `gpa`, and `rank` (the independent variables).

R

```
# Build the Logit model
# admit ~ . means predict admit using all other columns
logit_model <- glm(admit ~ gre + gpa + rank,
                   data = admission_data,
                   family = "binomial")

# View the results
summary(logit_model)
```

Note for the results:

- **P-values (Pr(>|z|)):** If a p-value is less than **0.05**, that variable significantly affects admission.
- **Coefficients:** A positive estimate means an increase in that score increases the probability of admission.

Step 4: Evaluate Model Fit

In logistic regression, we don't use R^2 like in linear regression. Instead, we check the **Deviance** and use the **Likelihood Ratio Test** to see if the model is a good fit.

R

```
# 1. Check for Overall Model Significance
# Comparing our model to a 'null' model (a model with no predictors)
with(logit_model, pchisq(null.deviance - deviance, df.null - df.residual, lower.tail = FALSE))

# 2. Confusion Matrix to check accuracy
# We predict probabilities; if > 0.5, we classify as 'Admitted'
predictions <- predict(logit_model, type = "response")
predicted_class <- ifelse(predictions > 0.5, 1, 0)

# Create a table of Actual vs Predicted
table(Actual = admission_data$admit, Predicted = predicted_class)
```

Interpreting Fit:

- **Chi-Square Test:** If the result of the `pchisq` code above is very small (less than 0.05), your model fits significantly better than an empty model.
- **Deviance:** Lower Residual Deviance indicates a better fit.

OUT PUT:-

```
head(admission_data)
  admit gre  gpa rank
1     0 380 3.61    3
2     1 660 3.67    3
3     1 800 4.00    1
```

```
   4     1 640 3.19    4


   5     0 520 2.93    4
   6     1 760 3.00    2
   glm(formula = admit ~ gre + gpa + rank, family = "binomial",
       data = admission_data)

   Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.989979   1.139951  -3.500 0.000465 ***
gre          0.002264   0.001094   2.070 0.038465 *
gpa          0.804038   0.331819   2.423 0.015388 *
rank2       -0.675443   0.316490  -2.134 0.032829 *
rank3       -1.340204   0.345306  -3.881 0.000104 ***
rank4       -1.551464   0.417832  -3.713 0.000205 ***
---
   Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

   (Dispersion parameter for binomial family taken to be 1)

       Null deviance: 499.98  on 399  degrees of freedom
   Residual deviance: 458.52  on 394  degrees of freedom
   AIC: 470.52

   Number of Fisher Scoring iterations: 4

    [1] 7.578194e-08
   >
   > # 2. Confusion Matrix to check accuracy
         Predicted
   Actual   0   1
        0 254  19
        1  97  30
```

PRACTICAL NO: 02

AIM : MULTIPLE REGRESSION MODEL Apply multiple regressions, if data have a continuous independent variable. Apply on above dataset.

In Multiple Linear Regression, we predict a **continuous** dependent variable. However, in the admissions dataset used previously, the variable `admit` is binary (0 or 1).

To apply **Multiple Regression** properly to this dataset, we will change our objective: We will predict the **GRE score** (a continuous variable) based on the student's **GPA** and the **Rank** of their school.

Step 1: Prepare the Environment

We will use the same dataset but treat `gre` as our target variable.

R

```
# Ensure libraries are loaded
library(MASS)

# Use the dataset from the previous step
# (Assuming admission_data is already loaded)
# If not: admission_data <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv")

# Ensure rank is a factor
admission_data$rank <- factor(admission_data$rank)
```

Step 2: Build the Multiple Linear Regression Model

We use the `lm()` function (Linear Model) for continuous outcomes.

R

```
# Formula: gre ~ gpa + rank
# We are seeing how GPA and Rank together influence the GRE score
multiple_reg_model <- lm(gre ~ gpa + rank, data = admission_data)

# Display the statistical results
summary(multiple_reg_model)
```

Step 3: Interpret the Relation Between Variables

When you run the `summary()`, look for the following:

1. **Coefficients (Estimate):** This tells you the "slope." For example, if the GPA estimate is **50**, it means for every 1-point increase in GPA, the GRE score is expected to increase by 50 points, holding school rank constant.
2. **ZP-values (Pr(>|t|)):** If this is <0.05, the variable is a significant predictor of the GRE score.

3. **Adjusted R-squared:** This tells you what percentage of the variation in GRE scores is explained by your model.

Step 4: Evaluate Model Fit (Diagnostics)

To check if a linear model is "fit" for the data, we must check the residuals (the errors).

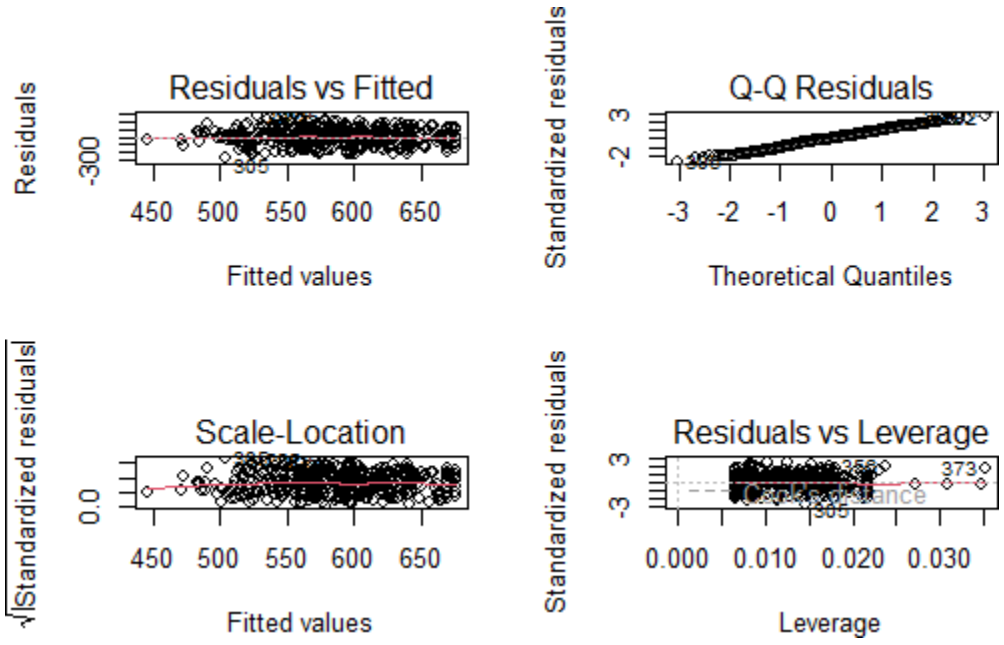
R

```
# Plot diagnostic graphs
# This will show 4 plots; press Enter in the console to see each one
par(mfrow = c(2, 2)) # Arrange plots in a 2x2 grid
plot(multiple_reg_model)
```

How to check if the model is fit:

- **Residuals vs Fitted:** You want to see a random scatter of points. If you see a curve, the relationship might not be linear.
- **Normal Q-Q:** The points should follow the diagonal dashed line. If they do, your errors are normally distributed (which is good).

OUTPUT:



PRACTICAL NO: 03

AIM: CLASSIFICATION MODEL a. Install relevant package for classification. b. Choose classifier for classification problem. c. Evaluate the performance of classifier.

Building a classification model in R is a streamlined process thanks to the `caret` (Classification And REgression Training) package, which provides a consistent interface for hundreds of different algorithms.

Here is a step-by-step guide to building, running, and evaluating a **Random Forest** classifier using the classic `iris` dataset.

Step 1: Install and Load Relevant Packages

First, you need to install the tools. The `caret` package is the industry standard for machine learning in R, and `randomForest` is the specific engine we will use.

How to execute: Type these lines into your R script or the Console and press **Enter**.

```
# Install packages (only need to do this once)
install.packages("caret")
install.packages("randomForest")

# Load the libraries
library(caret)
library(randomForest)
```

Step 2: Prepare the Data

Before choosing a classifier, we must split the data into a **Training set** (to teach the model) and a **Test set** (to see if it actually learned).

```
# Load built-in dataset
data(iris)

# Set a seed for reproducibility (so you get the same results every time)
set.seed(123)

# Create an index to split 80% for training, 20% for testing
trainIndex <- createDataPartition(iris$Species, p = 0.8, list = FALSE)
trainData <- iris[trainIndex, ]
testData  <- iris[-trainIndex, ]
```

Step 3: Choose and Train the Classifier

We will use **Random Forest** because it is robust, handles non-linear data well, and is highly accurate for classification tasks.

R

```
# Train the model
# Species ~ . means "Predict Species using all other variables"
model <- train(Species ~ .,
               data = trainData,
               method = "rf",
               trControl = trainControl(method = "cv", number = 5)) # 5-fold cross-validation

# View model details
print(model)
```

Step 4: Evaluate Performance

To evaluate the model, we use it to predict the species of the "unseen" test data and compare those predictions to the actual results using a **Confusion Matrix**.

R

```
# Make predictions on the test set
predictions <- predict(model, testData)

# Evaluate using a Confusion Matrix
evaluation <- confusionMatrix(predictions, testData$Species)

# Print the results
print(evaluation)
```

OUTPUT:

```
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         2
  virginica       0          0         8

Overall Statistics

               Accuracy : 0.9333
                 95% CI : (0.7793, 0.9918)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : 8.747e-12

                  Kappa : 0.9

 Mcnemar's Test P-Value : NA

Statistics by Class:
```

	Class: setosa	Class: versicolor	Class: virginica
Sensitivity	1.0000	1.0000	0.8000
Specificity	1.0000	0.9000	1.0000
Pos Pred Value	1.0000	0.8333	1.0000
Neg Pred Value	1.0000	1.0000	0.9091
Prevalence	0.3333	0.3333	0.3333
Detection Rate	0.3333	0.3333	0.2667
Detection Prevalence	0.3333	0.4000	0.2667
Balanced Accuracy	1.0000	0.9500	0.9000

Understanding the Output

- **Accuracy:** The percentage of correct predictions. (e.g., 0.96 means 96% correct).
- **Kappa:** A measure of how well the classifier performed compared to how well it would have performed by simply guessing.
- **Confusion Matrix Table:** Shows exactly which species were misclassified (e.g., if a *Versicolor* was mistaken for a *Virginica*).

PRACTICAL NO: 04

AIM: CLUSTERING MODEL a. Clustering algorithms for unsupervised classification. b. Plot the cluster data using R visualizations.

Clustering is an unsupervised learning technique used to group data points based on similarities without using pre-defined labels. The most common algorithm for this is **K-Means Clustering**.

We will use the `iris` dataset again, but we will hide the "Species" column to see if the algorithm can discover the groupings on its own.

Step 1: Prepare the Data

Clustering requires numeric data. Since the iris dataset has 4 numeric columns and 1 categorical (Species), we will strip the species names so the model works "blind."

How to execute: Copy this into a new R Script and run it.

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

# Remove the Species column (column 5) for unsupervised learning
iris_cluster <- iris[, -5]

# Scale the data (Standardization)
# This ensures variables with larger numbers don't dominate the model
iris_scaled <- scale(iris_cluster)
```

Step 2: Choose the Number of Clusters (The "Elbow" Method)

Before running K-Means, we need to decide on **k** (the number of groups). We use the Elbow Method to find the point where adding more clusters doesn't significantly improve the fit.

```
R
# Determine number of clusters
set.seed(123)
wss <- sapply(1:10, function(k){kmeans(iris_scaled, k, nstart=20 )$tot.withinss})

# Plot the elbow curve
plot(1:10, wss, type="b", pch = 19, frame = FALSE,
    xlab="Number of clusters K",
    ylab="Total within-clusters sum of squares")
```

Look for the "bend" in the arm. For Iris, the bend is usually at ***k = 3****.*

Step 3: Apply the K-Means Algorithm

Now we apply the algorithm using 3 clusters.

R

```
# Apply K-Means
set.seed(123)
km_result <- kmeans(iris_scaled, centers = 3, nstart = 25)

# View the cluster assignments
print(km_result$cluster)
```

Step 4: Visualize the Clusters

Since we have 4 variables, we can't easily graph them in 2D. We use the `factoextra` package to perform Principal Component Analysis (PCA) automatically and plot the clusters clearly.

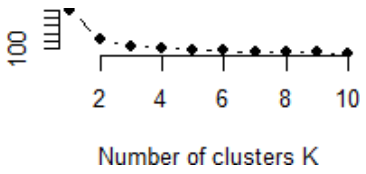
How to execute: You may need to install `factoextra` first.

R

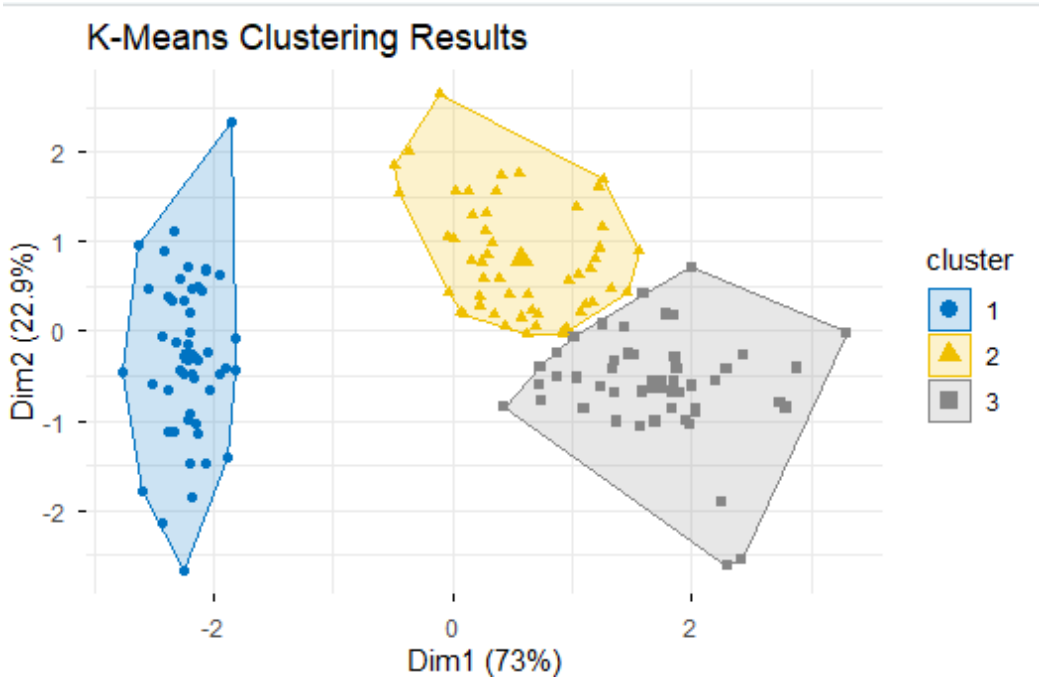
```
# Install and load visualization package
if(!require(factoextra)) install.packages("factoextra")
library(factoextra)

# Plot the clusters
fviz_cluster(km_result, data = iris_scaled,
             palette = "jco",
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_minimal(),
             main = "K-Means Clustering Results")
```

OUTPUT:



```
> print(km_result$cluster)
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [34] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 2 2 2 3 2 2 2 2 2 2 2 2 3
 [67] 2 2 2 2 3 2 2 2 2 3 3 3 2 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2
[100] 2 3 2 3 3 3 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 3 3 3 3 3
[133] 3 2 2 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2
```



PRACTICAL NO: 05

AIM: Install, configure and run Hadoop and HDFS ad explore.

1. Install Java JDK 1.8
2. Download Hadoop and extract and place under C drive
3. Set Path in Environment Variables
4. Config files under Hadoop directory
5. Create folder datanode and namenode under data directory
6. Edit HDFS and YARN files
7. Set Java Home environment in Hadoop environment
8. Setup Complete. Test by executing start-all.cmd

There are two ways to install Hadoop, i.e.

9. Single node
10. Multi node

Here, we use multi node cluster.

1. Install Java

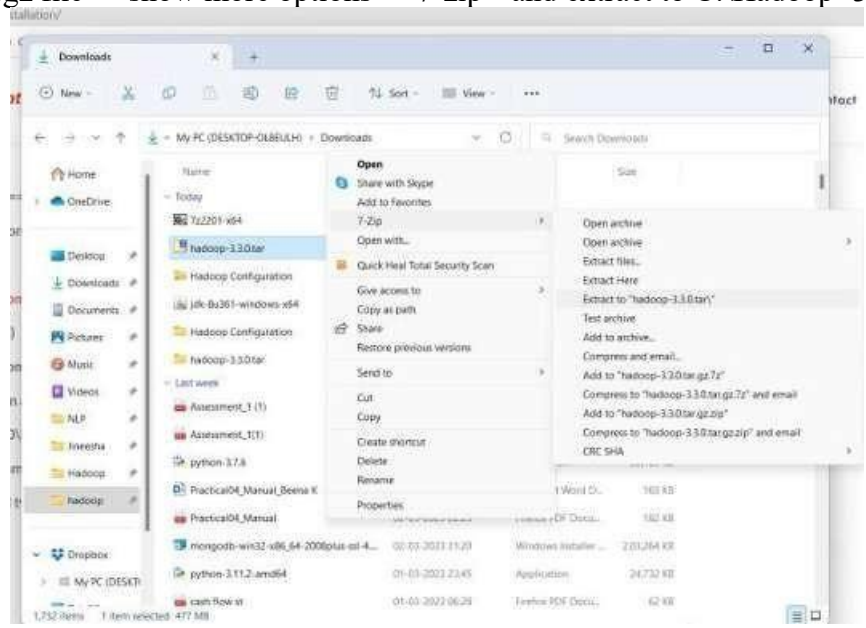
11. – Java JDK Link to download https://www.oracle.com/java/technologies/javase-jdk8-downloads.html
12. – extract and install Java in C:\Java
13. – open cmd and type -> javac -version

```
C:\Users>cd Beena

C:\Users\Beena>java -version
java version "1.8.0_361"
Java(TM) SE Runtime Environment (build 1.8.0_361-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.361-b09, mixed mode)
```

2. Download Hadoop

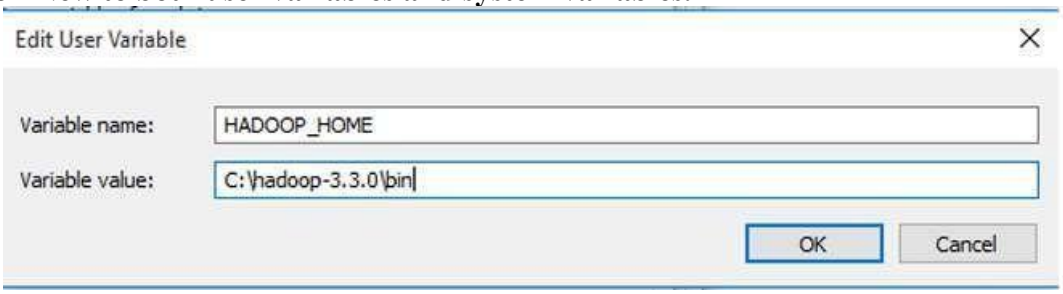
https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.3.0/hadoop- 3.3.0.tar.gz
right click .rar.gz file -> show more options -> 7-zip->and extract to C:\Hadoop- 3.3.0\



3. Set the path JAVA_HOME Environment variable

4. Set the path HADOOP_HOME Environment variable

Click on New to both user variables and system variables.



Click on user variable -> path -> edit-> add path for Hadoop and java upto "bin"
Click Ok, Ok, Ok.

5. Configurations

Edit file C:/Hadoop-3.3.0/etc/hadoop/core-site.xml, paste the xml code in folder and save

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

Rename "mapred-site.xml.template" to "mapred-site.xml" and edit this file C:/Hadoop3.3.0/etc/hadoop/mapred-site.xml, paste xml code and save this file.

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

Create folder "data" under "C:\Hadoop-3.3.0"

Create folder "datanode" under "C:\Hadoop-3.3.0\data"

Create folder "namenode" under "C:\Hadoop-3.3.0\data"

Edit file C:\Hadoop-3.3.0/etc/hadoop/hdfs-site.xml, paste xml code and save this file.

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>/hadoop-3.3.0/data/namenode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>/hadoop-3.3.0/data/datanode</value>
</property>
</configuration>
```

Edit file C:/Hadoop-3.3.0/etc/hadoop/yarn-site.xml, paste xml code and save this file.

```
<configuration>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
<name>yarn.resourcemanager.address</name>
<value>127.0.0.1:8032</value>
<name>yarn.resourcemanager.scheduler.address</name>
<value>127.0.0.1:8030</value>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>127.0.0.1:8031</value>
</property>
</configuration>
```

6. Edit file C:/Hadoop-3.3.0/etc/hadoop/hadoop-env.cmd

Find "JAVA_HOME=%JAVA_HOME%" and replace it as

```
set JAVA_HOME="C:\Java\jdk1.8.0_361"
```

7. Download "redistributable" package

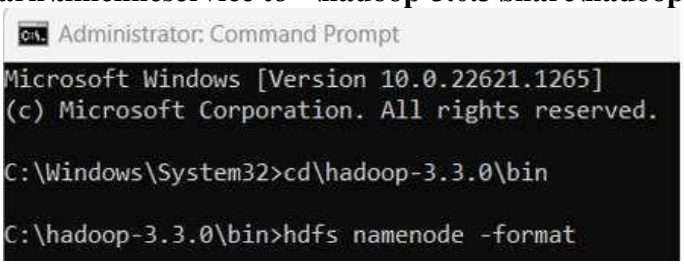
Download and run VC_redist.x64.exe

This is a "redistributable" package of the Visual C runtime code for 64-bit applications, from Microsoft. It contains certain shared code that every application written with Visual C expects to have available on the Windows computer it runs on.

8. Hadoop Configurations

Download bin folder from https://github.com/s911415/apache-hadoop-3.1.0-winutils – Copy the bin folder to c:\hadoop-3.3.0. Replace the existing bin folder.

9. copy "hadoop-yarn-server-timelineservice-3.0.3.jar" from ~\hadoop-3.0.3\share\hadoop\yarn\timelineservice to ~\hadoop-3.0.3\share\hadoop\yarn folder.




```
2023-03-07 21:31:34,685 INFO namenode.FSImageFormatProtobuf: Saving image file C:\hadoop-3.3.0\data\namenode\current\fs
mage.ckpt_0000000000000000000 using no compression
2023-03-07 21:31:34,844 INFO namenode.FSImageFormatProtobuf: Image file C:\hadoop-3.3.0\data\namenode\current\fsimage.c
pt_0000000000000000000 of size 400 bytes saved in 0 seconds .
2023-03-07 21:31:34,860 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-03-07 21:31:34,869 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2023-03-07 21:31:34,870 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at DESKTOP-OL8EULH/192.168.1.19
************************************************************/
```

10. Format the NameNode

– Open cmd „Run as Administrator“ and type command “hdfs namenode –format”

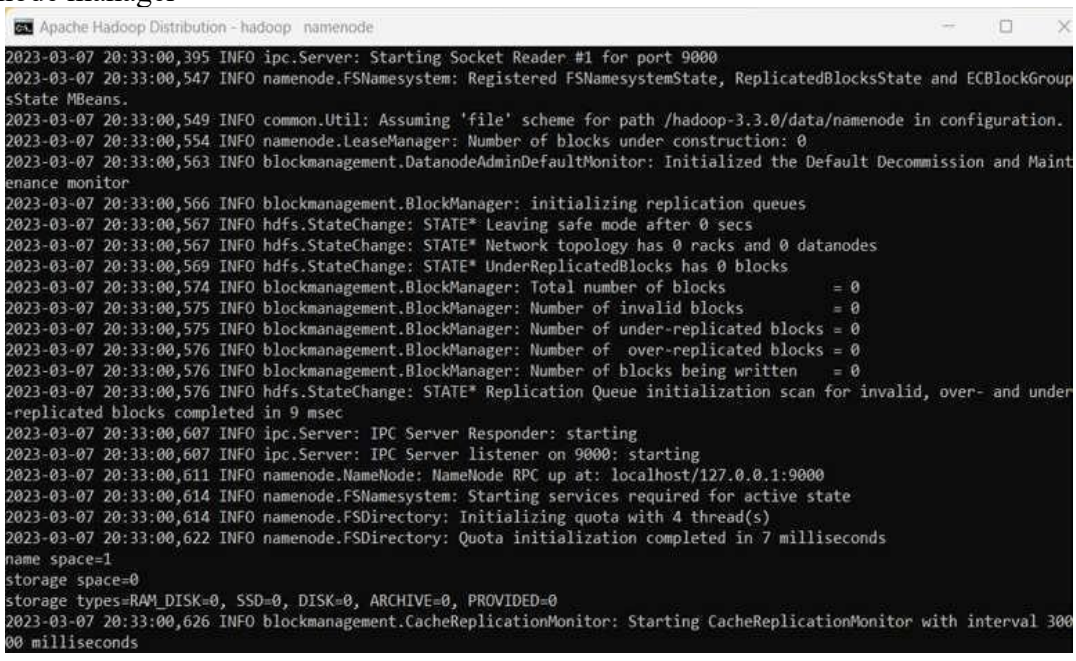
11. Testing

– Open cmd „Run as Administrator“ and change directory to C:\Hadoop-3.3.0\sbin

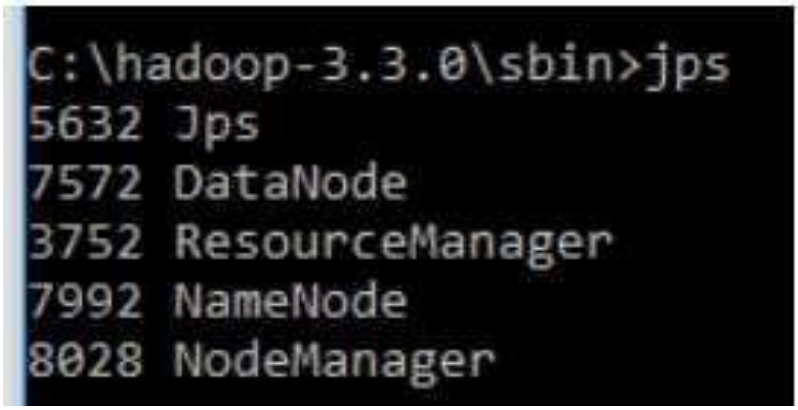
– type start-all.cmd OR type start-dfs.cmd

– type start-yarn.cmd

– You will get 4 more running threads for Datanode, namenode, resouce manager and node manager

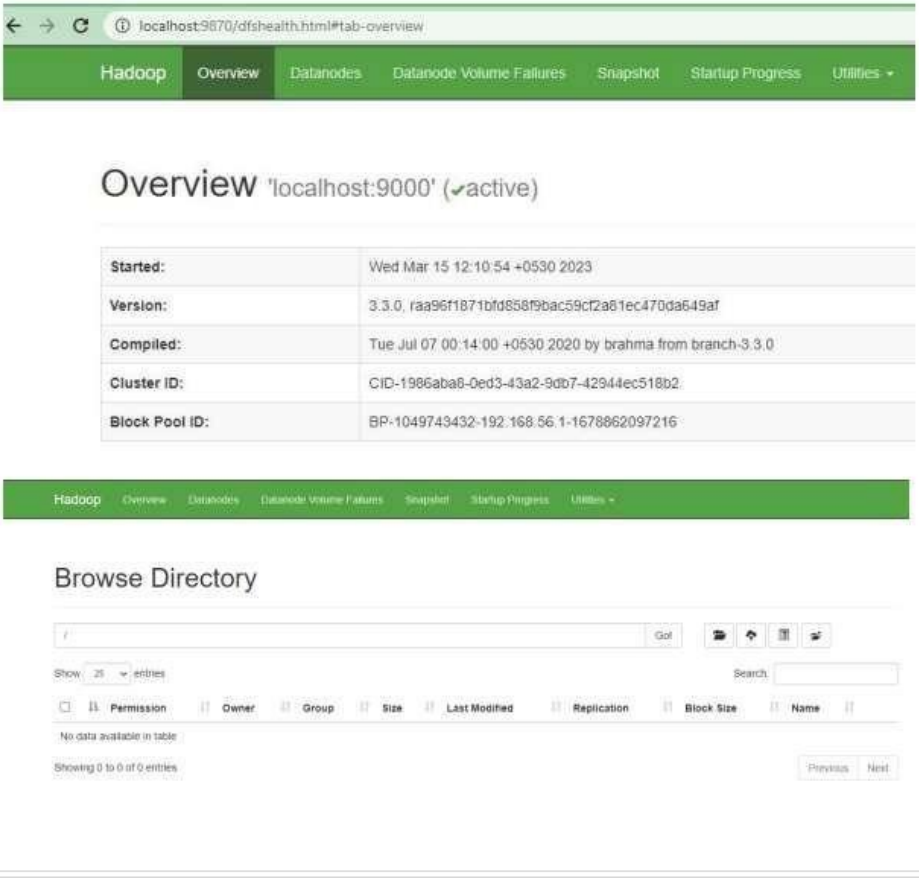


Output :



10. Type JPS command to start-all.cmd command prompt, you will get following output.

11. Run http://localhost:9870/ from any browser



PRACTICAL NO: 06

AIM: Implement word count / frequency programs using MapReduce

Steps:

- C:\hadoop-3.3.0\sbin>start-dfs.cmd
- C:\hadoop-3.3.0\sbin>start-yarn.cmd
- Open a command prompt as administrator and run the following command to create an input and output folder on the Hadoop file system, to which we will be moving the sample.txt file for our analysis.
- C:\hadoop-3.3.0\bin>cd\
- C:\>hadoop dfsadmin -safemode leave
- DEPRECATED: Use of this script to execute hdfs command is deprecated.
- Instead use the hdfs command for it.
- Safe mode is OFF
- C:\>hadoop fs -mkdir /input_dir
- Check it by giving the following URL at browser http://localhost:9870
- Now apply the following command at c:\>
- C:\> hadoop fs -put C:/input_file.txt /input_dir
- Verify input_file.txt available in HDFS input directory (input_dir).
  C:\>Hadoop fs -ls /input_dir/
- C:\>hadoop jar C:/hadoop-3.3.0/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.0.jar wordcount /input_dir /output_dir
- In case, there is some error in executing then copy the file MapReduceClient.jar in C:\ and run the program with the jar file using existing MapReduceClient.jar file as:
- C:\> hadoop jar C:/MapReduceClient.jar wordcount /input_dir /output_dir
- Now, check the output_dir on browser as follows:
- Click on output_dir à part-r-00000 à Head the file (first 32 K) and check the file content as the output.
- Alternatively, you may type the following command on CMD window as:
- C:\> hadoop dfs -cat /output_dir/*

You can get the following output

Output:

```
hadoop dfs -cat /output_dir/*
Hadoop  1
Window  1
version  1
is  1
easy  1
compared  1
to  1
Ubuntu  1
```

PRACTICAL NO: 07

AIM: Implement Decision tree classification techniques

A **Decision Tree** is one of the most intuitive classification techniques because it mimics human decision-making. We will use the `rpart` package to build the tree and `rpart.plot` to visualize it.

For this example, we will use the built-in `iris` dataset, which classifies flowers based on their physical measurements.

Step 1: Install and Load Required Packages

We need `rpart` for the logic and `rpart.plot` for a clean, easy-to-read visualization.

How to execute: Copy these lines into your R script and run them.

R

```
# Install relevant packages
install.packages("rpart")
install.packages("rpart.plot")

# Load the libraries
library(rpart)
library(rpart.plot)
```

Step 2: Prepare and Split the Data

We split the data to ensure our model can generalize to new, unseen information.

R

```
# Load data
data(iris)

# Set seed for reproducibility
set.seed(42)

# Split data: 70% Training, 30% Testing
sample_index <- sample(1:nrow(iris), 0.7 * nrow(iris))
train_data <- iris[sample_index, ]
test_data  <- iris[-sample_index, ]
```

Step 3: Build the Decision Tree Classifier

We use the `rpart()` function. The formula `Species ~ .` tells R to predict the "Species" using all other available variables (Sepal Length, Width, etc.).

R

 # Train the Decision Tree model
dt_model <- rpart(Species ~ ., data = train_data, method = "class")

 # View a text summary of the tree
print(dt_model)

Step 4: Visualize the Tree

The best part of a Decision Tree is the visualization. This makes it "Explainable AI."

R
 # Plot the tree
rpart.plot(dt_model, type = 4, extra = 101, under = TRUE, cex = 0.8, box.palette = "GnBu")

Step 5: Evaluate the Performance

Finally, we test the model on the 30% of data we held back to see its accuracy.

R

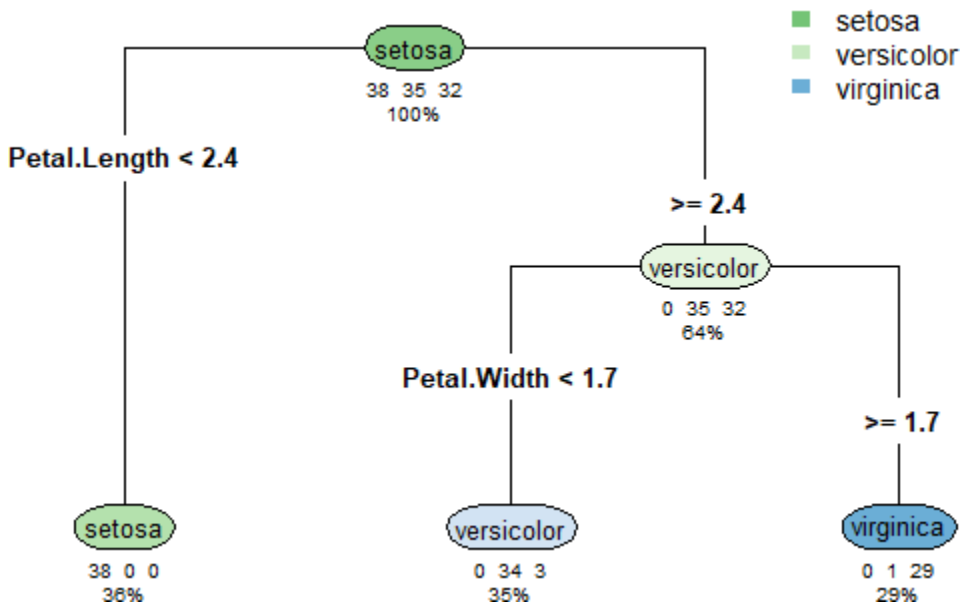
 # Make predictions on the test set
dt_predictions <- predict(dt_model, test_data, type = "class")

 # Create a Confusion Matrix
conf_matrix <- table(test_data$Species, dt_predictions)
print(conf_matrix)

 # Calculate Accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
print(paste("Accuracy:", round(accuracy, 4)))

OUTPUT:-

```
print(conf_matrix)
            dt_predictions
             setosa versicolor virginica
  setosa         12          0         0
  versicolor      0         14         1
  virginica       0          1        17
```




```
print(paste("Accuracy:", round(accuracy, 4)))
[1] "Accuracy: 0.9556"
```

PRACTICAL NO: 08

AIM: Implement SVM classification techniques

Support Vector Machines (SVM) are powerful supervised learning models used for classification. SVM works by finding the **hyperplane** (a boundary) that best separates different classes with the maximum possible margin.

To implement SVM in R, the most popular and reliable package is `e1071`.

---

1. Required Packages

In R, SVM is commonly implemented using the **`e1071`** package.

```
install.packages("e1071")    # run once
library(e1071)
```

---

2. Load Dataset (Iris Dataset)

```
data(iris)
head(iris)
```

- **Features:** Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
- **Target/Class:** Species

---

3. Split Dataset into Training & Testing

```
set.seed(123)

index <- sample(1:nrow(iris), 0.7 * nrow(iris))
train_data <- iris[index, ]
test_data  <- iris[-index, ]
```

---

4. Linear SVM Classification

```
svm_linear <- svm(
  Species ~ .,
  data = train_data,
  kernel = "linear"
)

pred_linear <- predict(svm_linear, test_data)

table(Predicted = pred_linear, Actual = test_data$Species)
mean(pred_linear == test_data$Species)
```

Confusion Matrix

```
library(caret)

confusionMatrix(pred-rbf, test_data$Species)
```

OUTPUT:

```
pred_linear <- predict(svm_linear, test_data)
> table(Predicted = pred_linear, Actual = test_data$Species)
            Actual
Predicted    setosa versicolor virginica
  setosa         14          0         0
  versicolor      0         17         0
  virginica       0          1        13
> mean(pred_linear == test_data$Species)
[1] 0.9777778
```

- **Rows = Predicted class**
- **Columns = Actual class**

Correct Predictions (Diagonal Values)

These are the **correct classifications**:

- **Setosa:** 14 correctly classified
- **Versicolor:** 17 correctly classified
- **Virginica:** 13 correctly classified

Incorrect Predictions (Off-diagonal Values)

- **1 Versicolor was misclassified as Virginica**

All other samples are classified correct