

Practical: 01

Aim: Perform K means clustering using R Programming/python.

Description: K-means clustering is a widely used unsupervised machine learning algorithm that partitions a dataset into K distinct clusters. The algorithm aims to minimize the variance within each cluster while maximizing the variance between clusters. Below is a detailed explanation of how to implement K-means clustering using both R and Python, along with a brief overview of its workings.

Key Concepts:

- **Clustering:** Grouping similar data points together.
- **Centroids:** Representative points for each cluster, initially chosen randomly or using a method like k-means++.
- **Distance:** A metric used to measure the similarity between data points and centroids (e.g., Euclidean distance).
- **Iteration:** The process of assigning points to clusters and recalculating centroids, repeated until convergence.
- **K:** The number of clusters to be created, which needs to be determined beforehand.

Code:

```
# install required packages
install.packages("plyr")
install.packages("ggplot2")
install.packages("cluster")
install.packages("lattice")
install.packages("grid")
install.packages("gridExtra")

# Load the package
library(plyr)
library(ggplot2)
library(cluster)
library(lattice)
library(grid)
library(gridExtra)
```

A data frame is a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

```

grade_input=as.data.frame(read.csv("C:/Users/ACER/Desktop/soni/grades_km_input.csv"))
kmdata_orig=as.matrix(grade_input[, c ("Student","English","Math","Science")])
kmdata=kmdata_orig[,2:4]
kmdata[1:10,]

# the k-means algorithm is used to identify clusters for k = 1, 2, ... , 15. For each value of k, the WSS
is calculated.

wss=numeric(15)

# the option n start=25 specifies that the k-means algorithm will be repeated 25 times, each starting
with k random initial centroids

for(k in 1:15)wss[k]=sum(kmeans(kmdata,centers=k,nstart=25)$withinss)

plot(1:15,wss,type="b",xlab="Number of Clusters",ylab="Within sum of square")

#As can be seen, the WSS is greatly reduced when k increases from one to two. Another substantial
reduction in WSS occurs at k = 3. However, the improvement in WSS is fairly linear fork > 3.

km = kmeans(kmdata,3,nstart=25)

km

c( wss[3] , sum(km$withinss))

df=as.data.frame(kmdata_orig[,2:4])
df$cluster=factor(km$cluster)

centers=as.data.frame(km$centers)

g1=ggplot(data=df, aes(x=English, y=Math, color=cluster )) +
  geom_point() + theme(legend.position="right") +
  geom_point(data=centers,aes(x=English,y=Math, color=as.factor(c(1,2,3))),size=10, alpha=.3,
show.legend =FALSE)

g2=ggplot(data=df, aes(x=English, y=Science, color=cluster )) +
  geom_point () +geom_point(data=centers,aes(x=English,y=Science,
color=as.factor(c(1,2,3))),size=10, alpha=.3, show.legend=FALSE)

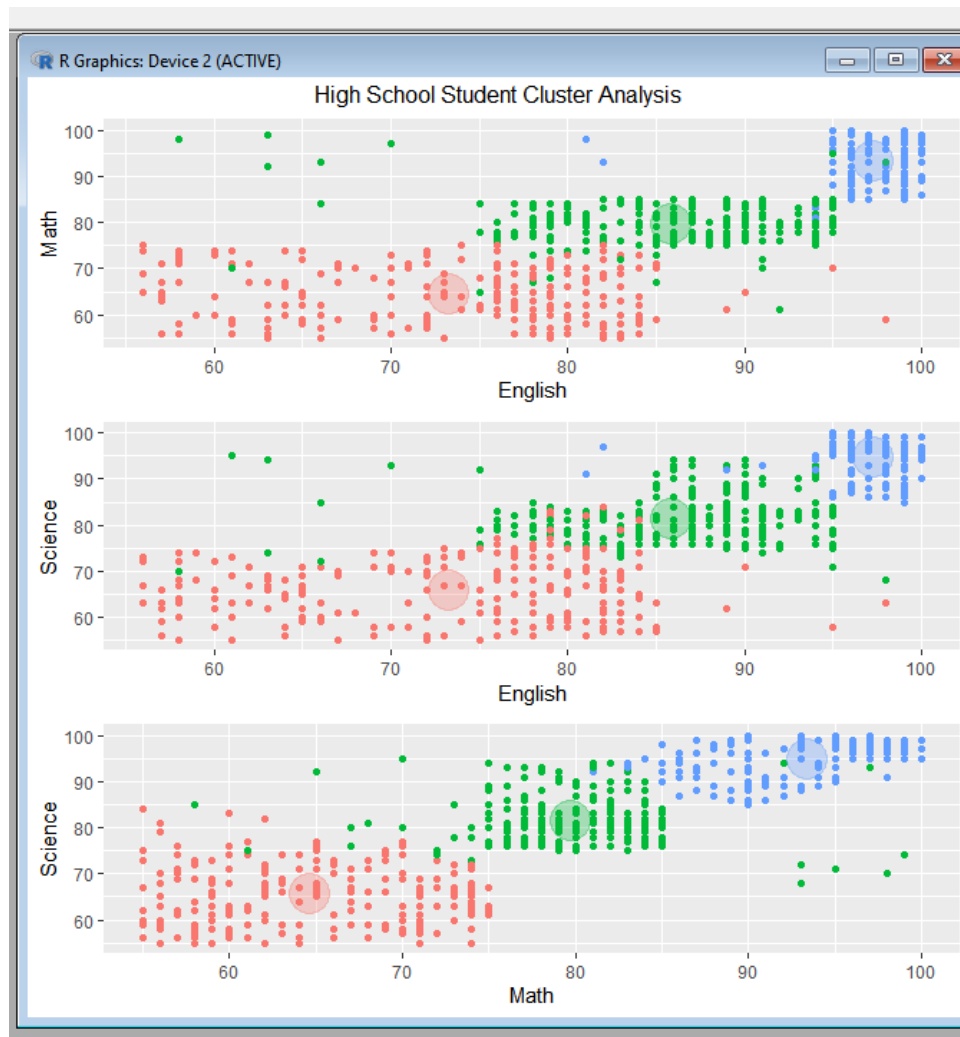
g3 = ggplot(data=df, aes(x=Math, y=Science, color=cluster )) +
  geom_point () + geom_point(data=centers,aes(x=Math,y=Science,
color=as.factor(c(1,2,3))),size=10, alpha=.3, show.legend=FALSE)

tmp=ggplot_gtable(ggplot_build(g1))

grid.arrange(arrangeGrob(g1 + theme(legend.position="none"),g2 +
theme(legend.position="none"),g3 + theme(legend.position="none"),top ="High School Student
Cluster Analysis" ,ncol=1))

```

Output:

**Learning:**

- Understand how to implement K-means clustering in both R and Python.
- Learn how to visualize data and determine an appropriate number of clusters using the elbow method.
- Gain insights into interpreting clustering results and visualizing them effectively.
- K-means clustering serves as a foundational technique in data analysis and machine learning, applicable across various domains such as market segmentation, image compression, and social network analysis.

Practical: 02

Aim: Perform classification model using Naive Bayes Algorithm

Description: Naive Bayes is a simple but powerful probabilistic machine learning algorithm used for classification tasks. It is based on **Bayes' Theorem**, which describes the probability of an event occurring based on prior knowledge of conditions that might be related to the event. Despite its simplicity, it performs very well in many real-world situations, especially in text classification (e.g., spam email detection, sentiment analysis) and other problems where features are conditionally independent.

Bayes' Theorem

Bayes' Theorem calculates the probability of a hypothesis (target class) given the data. The formula is:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

Where:

- **P(C|X):** Probability of class **C** given the data **X**.
- **P(X|C):** Likelihood of observing **X** given class **C**.
- **P(C):** Prior probability of class **C**.
- **P(X):** Prior probability of data **X**.

The Naive Bayes classifier applies the "**naive**" assumption that the features (variables) are conditionally independent given the class. This simplifies the calculation of **P(X|C)**, making the model computationally efficient.

Types of Naive Bayes Classifiers

- **Gaussian Naive Bayes:** Assumes that features are normally distributed. This is used when the features are continuous.
- **Multinomial Naive Bayes:** Suitable for discrete count data (e.g., word counts in text classification).
- **Bernoulli Naive Bayes:** Useful when features are binary (true/false, 0/1).

Key Assumptions

- **Conditional Independence:** Given the class label, the features are assumed to be conditionally independent.
- **Feature Distribution:** The distribution of features varies by the type of Naive Bayes used (Gaussian for continuous, multinomial for discrete).

Code: # Prac1:

Prac2: Naive Bayes

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

Importing the dataset

dataset = pd.read_csv('sheet.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Fitting classifier to the Training set

from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X_train, y_train)

Predicting the Test set results

y_pred = classifier.predict(X_test)

Making the Confusion Matrix

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

Visualising the Training set results

from matplotlib.colors import ListedColormap

X_set, y_set = X_train, y_train

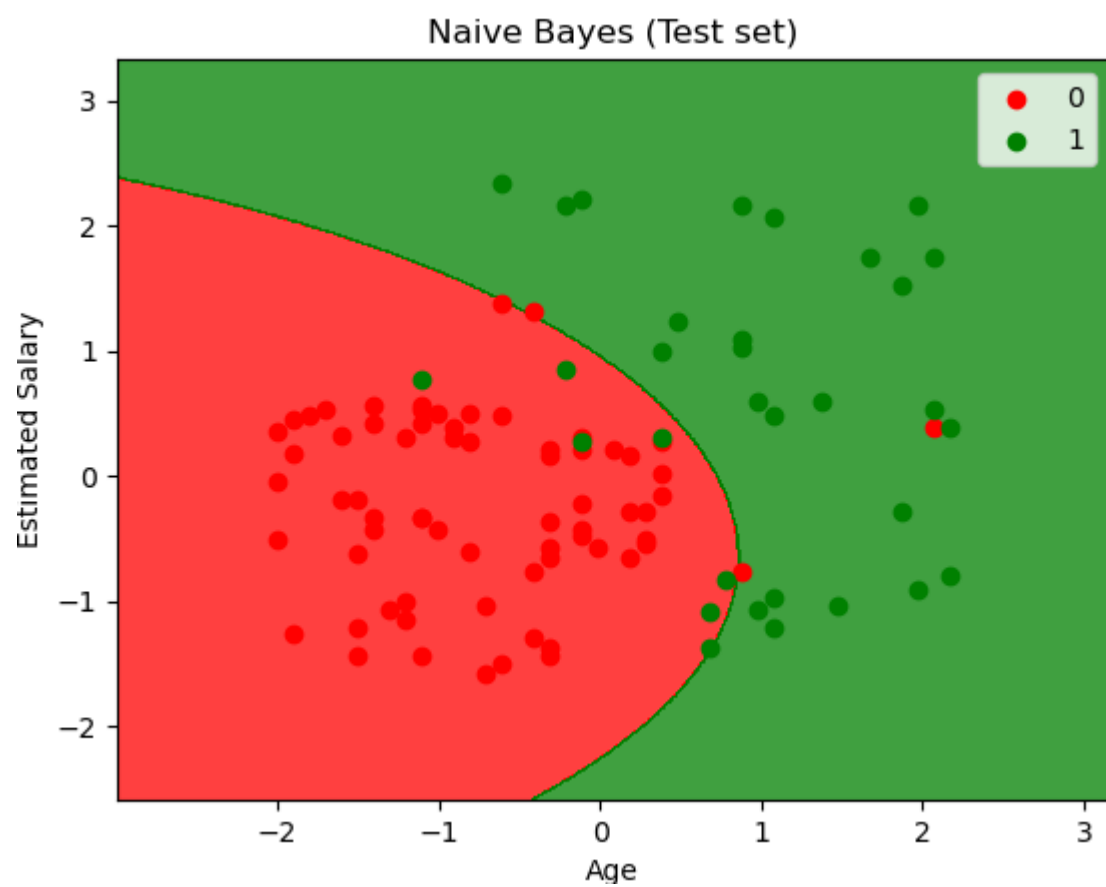
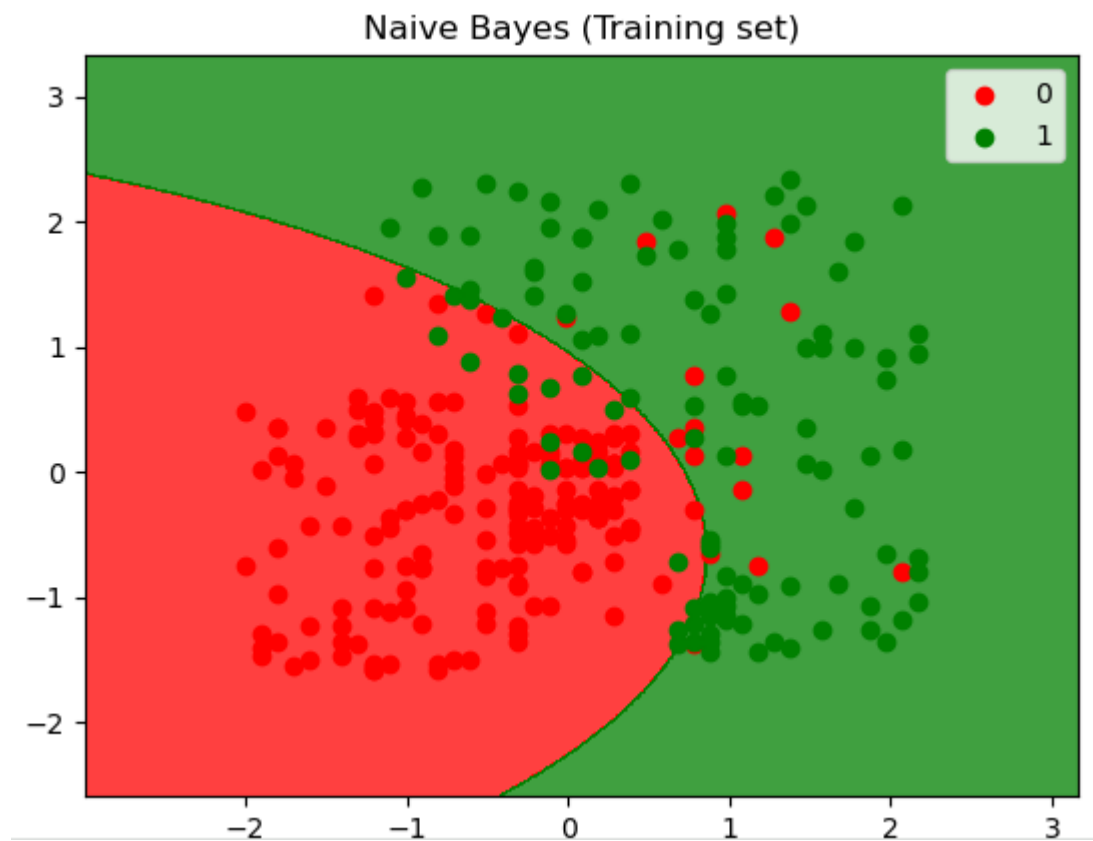
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))

```
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step =
0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Output:



Learnings:

- **Understanding Bayes' Theorem:** It's crucial to grasp Bayes' Theorem as it forms the foundation of Naive Bayes classification.
- **Independence Assumption:** Recognizing that Naive Bayes assumes that features are independent helps understand its limitations.
- **Choosing the Right Naive Bayes Model:** Depending on the type of data you have (continuous or discrete), you should choose the appropriate Naive Bayes model (Gaussian, Multinomial, or Bernoulli).
- **Practical Application:** Naive Bayes is fast and scalable, which makes it a good choice for large datasets. It's commonly used for text classification problems like spam detection and sentiment analysis.
- **Evaluation Metrics:** In addition to accuracy, you should evaluate your model using confusion matrix, precision, recall, and F1-score, especially in cases with imbalanced classes

Practical: 03

Aim: Implement Decision tree classification techniques

Description: This R code performs a decision tree classification on a dataset from a CSV file, splits the data into training and test sets, scales the features, and then visualizes the decision boundaries on both the training and test datasets. Here's a detailed explanation of the steps involved:

1. Loading and Preparing the Dataset

```
dataset = read.csv("C:/Users/admin/Downloads/Social_Network_Ads.csv")
dataset = dataset[3:5]
print(dataset)
dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))
```

- The dataset is read from a CSV file (Social_Network_Ads.csv), and only columns 3 to 5 are retained for analysis. These columns likely contain features related to age, estimated salary, and whether the user purchased the product.
- The "Purchased" column is converted to a factor with two levels (0 and 1), representing a binary classification problem (e.g., did the user purchase the product?).

2. Splitting the Data into Training and Test Sets

```
install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

- The caTools library is loaded for splitting the dataset.
- The dataset is split into a training set (75%) and a test set (25%) using the sample.split() function.
- The set.seed(123) ensures reproducibility of the data split.

3. Feature Scaling

```
training_set[-3] = scale(training_set[-3])
test_set[-3] = scale(test_set[-3])
```

- Feature scaling is applied to both the training and test sets. The -3 index excludes the target variable (Purchased) from scaling. Scaling is important for machine learning algorithms like decision trees, even though they are not sensitive to scaling as much as other algorithms like SVMs or logistic regression.

4. Fitting the Decision Tree Classifier

```
install.packages('rpart')
library(rpart)
classifier = rpart(formula = Purchased ~ ., data = training_set)
```

- The rpart library is loaded, which is used for decision tree classification.
- A decision tree classifier is built using the rpart() function, where the target variable is Purchased, and all other variables are used as predictors. The model is fitted using the training dataset (training_set).

5. Predicting Results

```
y_pred = predict(classifier, newdata = test_set[-3], type = 'class')
cm = table(test_set[, 3], y_pred)
```

- The trained decision tree model is used to make predictions (y_pred) on the test set (excluding the third column which is the target variable).
- A confusion matrix (cm) is created to evaluate the performance of the classifier. The table() function compares the predicted values (y_pred) to the actual values from the test set.

6. Visualizing the Decision Boundary for the Training Set

```
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3], main = 'Decision Tree Classification (Training set)', xlab = 'Age', ylab = 'Estimated Salary', xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

- The ElemStatLearn library is used for visualization.
- A grid of points (grid_set) is created for visualizing decision boundaries, with the Age and EstimatedSalary features covering the entire range of the data.
- The decision boundaries are plotted using contour() by predicting the classifier's output (y_grid) for the entire grid.
- The training data points are plotted on top of the grid, with different colors (green and red) based on the actual target variable (Purchased).

7. Visualizing the Decision Boundary for the Test Set

```
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3], main = 'Decision Tree Classification (Test set)', xlab = 'Age', ylab = 'Estimated Salary',
      xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

- This part visualizes the decision boundary for the test dataset, similar to how the training dataset was visualized.
- It helps evaluate how well the decision tree generalizes to unseen data.

8. Visualizing the Tree

```
plot(classifier)
text(classifier)
```

- The plot() function generates a plot of the decision tree model.
- The text() function adds labels to the tree, showing how decisions are made based on the feature values (Age, EstimatedSalary).

Code:

```
dataset = read.csv("C:/Users/admin/Downloads/Social_Network_Ads.csv")
dataset = dataset[3:5]
print(dataset)
dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))
install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
training_set[-3] = scale(training_set[-3])
test_set[-3] = scale(test_set[-3])
install.packages('rpart')
library(rpart)
classifier = rpart(formula = Purchased ~ ., data = training_set)
y_pred = predict(classifier, newdata = test_set[-3], type = 'class')
cm = table(test_set[, 3], y_pred)
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
```

```

grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3], main = 'Decision Tree Classification (Training set)', xlab = 'Age', ylab = 'Estimated Salary', xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE) points(grid_set,
pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
library(ElemStatLearn)
set = test_set X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3], main = 'Decision Tree Classification (Test set)', xlab = 'Age', ylab = 'Estimated Salary',
xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE) points(grid_set,
pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
plot(classifier) text(classifier)

```

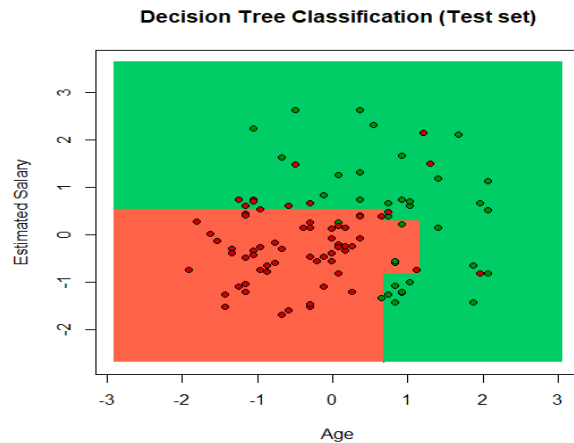
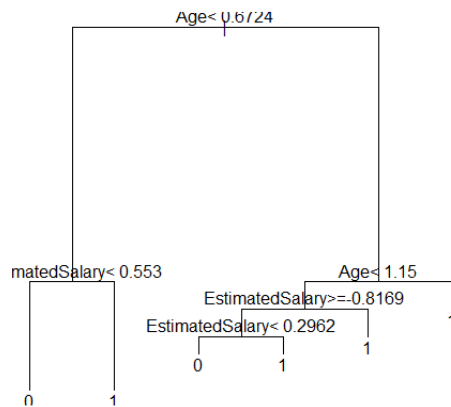
Output:-

```

> print(dataset)
  Age EstimatedSalary Purchased
1   19           19000           0
2   35           20000           0
3   26           43000           0
4   27           57000           0
5   19           76000           0
6   27           58000           0
7   27           84000           0
8   32          150000           1
9   25           33000           0
10  35           65000           0
11  26           80000           0
12  26           52000           0
13  20           86000           0
14  32           18000           0
15  18           82000           0
16  29           80000           0
17  47           25000           1
18  45           26000           1
19  46           28000           1
20  48           29000           1

```





Learnings and Key Insights:

1. **Decision Trees:** They are easy to understand and interpret as they split the data based on feature values. In this case, the decision tree splits data based on Age and EstimatedSalary to classify whether a user will purchase the product or not.
2. **Feature Scaling:** Even though decision trees do not require feature scaling as much as other algorithms, it's still a good practice in general to scale features to ensure consistent behavior when comparing results across models.
3. **Data Splitting:** Splitting the data into training and test sets (using a 75-25 ratio) is crucial for evaluating the model's performance on unseen data, ensuring that the model is not overfitting to the training data.
4. **Visualization:** Visualizing decision boundaries on both training and test sets allows you to see how well the classifier is distinguishing between classes and whether it is overfitting or underfitting.
5. **Confusion Matrix:** The confusion matrix is essential for evaluating the model's accuracy, precision, recall, and F1 score. In this case, it helps identify whether the classifier is correctly predicting the 'Purchased' outcome.

Practical: 04

Aim: Implement Decision tree classification techniques

Description: Decision tree classification is a supervised machine learning algorithm that uses a tree-like structure to make predictions by iteratively splitting data based on features until a class label is reached. It's a popular method for both classification and regression tasks due to its simplicity and interpretability.

1. Loading and Preparing the Data

```
dataset = read.csv("C:/Users/admin/Downloads/Social_Network_Ads.csv")
dataset = dataset[3:5]
```

- Reads the CSV file and selects the 3rd to 5th columns: typically Age, EstimatedSalary, and Purchased.

```
dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))
```

- Converts the Purchased column into a factor for classification (categorical: 0 = No, 1 = Yes).

2. Data Splitting using caTools

```
install.packages('caTools', dependencies=TRUE)
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

- Splits the data: 75% for training, 25% for testing.
- set.seed(123) ensures reproducibility.

3. Feature Scaling

```
training_set[, -3] = scale(training_set[, -3])
test_set[, -3] = scale(test_set[, -3])
```

- Standardizes Age and EstimatedSalary for better model performance, especially for SVMs which are sensitive to feature scales.

4. Training the SVM Model

```
install.packages('e1071', dependencies=TRUE)
library(e1071)
classifier = svm(formula = Purchased ~ ., data = training_set,
                 type = 'C-classification', kernel = 'linear')
```

- Trains an SVM with a **linear kernel** on the training set using the e1071 package.
- C-classification: classification type of SVM.
- Purchased ~ .: Predict Purchased using all other variables.

5. Predictions and Evaluation

```
y_pred = predict(classifier, newdata = test_set[-3])
cm = table(test_set[, 3], y_pred)
print(cm)
```

- Makes predictions on the test set.
- Evaluates performance using a **confusion matrix** showing actual vs predicted labels.

6. Visualizing Decision Boundaries with ggplot2

```
install.packages('ggplot2', dependencies=TRUE)
library(ggplot2)
```

Reusable Function to Plot Decision Boundary

```
plot_decision_boundary <- function(set, title) {
  ...
}
```

- Creates a mesh grid of Age and EstimatedSalary.
- Uses the trained classifier to predict over the grid.
- Plots both:
 - Actual data points (colored by true class).
 - Grid background (colored by predicted class) to visualize decision boundaries.

```
plot_decision_boundary(training_set, "SVM (Training Set)")
plot_decision_boundary(test_set, "SVM (Test Set)")
```

- Visualizes decision boundaries for training and test sets.

Code:

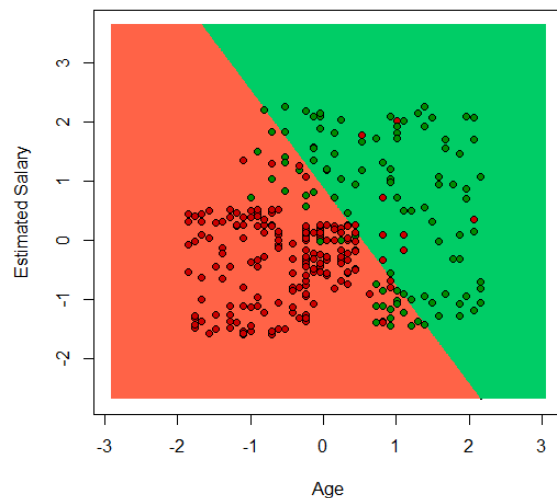
```
dataset = read.csv("C:/Users/admin/Downloads/Social_Network_Ads.csv")
dataset = dataset[3:5]
```

```
print(dataset)
dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))
install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
training_set[-3] = scale(training_set[-3])
test_set[-3] = scale(test_set[-3])
install.packages('rpart')
library(rpart) classifier = rpart(formula = Purchased ~ ., data = training_set)
y_pred = predict(classifier, newdata = test_set[-3], type = 'class') cm = table(test_set[, 3], y_pred)
library(ElemStatLearn)
set = training_set X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3], main = 'Decision Tree Classification (Training set)', xlab = 'Age', ylab = 'Estimated Salary', xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE) points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
library(ElemStatLearn)
set = test_set X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set, type = 'class')
plot(set[, -3], main = 'Decision Tree Classification (Test set)', xlab = 'Age', ylab = 'Estimated Salary', xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE) points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
plot(classifier) text(classifier)
```

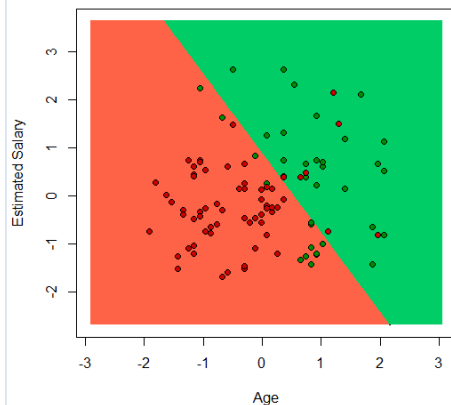
Output:-


```
> print(dataset)
  Age EstimatedSalary Purchased
1   19           19000         0
2   35           20000         0
3   26           43000         0
4   27           57000         0
5   19           76000         0
6   27           58000         0
7   27           84000         0
8   32          150000         1
9   25           33000         0
10  35           65000         0
11  26           80000         0
12  26           52000         0
13  20           86000         0
14  32           18000         0
15  18           82000         0
16  29           80000         0
17  47           25000         1
18  45           26000         1
19  46           28000         1
20  48           29000         1
```

SVM Decision Boundary (Training Set)



SVM Classification (Test set)



```
Call:
svm(formula = Purchased ~ ., data = training_set, type = "C-classification",
     kernel = "linear")
```

```
Parameters:
  SVM-Type: C-classification
  SVM-Kernel: linear
  cost: 1
```

```
Number of Support Vectors: 116
```

```
( 58 58 )
```

```
Number of Classes: 2
```

```
Levels:
 0 1
```

Learnings and Key Insights:

1. **Decision Trees:** They are easy to understand and interpret as they split the data based on feature values. In this case, the decision tree splits data based on Age and EstimatedSalary to classify whether a user will purchase the product or not.
2. **Feature Scaling:** Even though decision trees do not require feature scaling as much as other algorithms, it's still a good practice in general to scale features to ensure consistent behavior when comparing results across models.
3. **Data Splitting:** Splitting the data into training and test sets (using a 75-25 ratio) is crucial for evaluating the model's performance on unseen data, ensuring that the model is not overfitting to the training data.
4. **Visualization:** Visualizing decision boundaries on both training and test sets allows you to see how well the classifier is distinguishing between classes and whether it is overfitting or underfitting.
5. **Confusion Matrix:** The confusion matrix is essential for evaluating the model's accuracy, precision, recall, and F1 score. In this case, it helps identify whether the classifier is correctly predicting the 'Purchased' outcome.

Practical: 05

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).

Description: **Linear regression** is a statistical method used to **predict a continuous outcome** (a number) based on one or more input variables. It finds the **best-fitting straight line** (called the regression line) through the data that minimizes the difference between the actual and predicted values.

◆ **Formula (Simple Linear Regression):**

$$y = mx + b$$

- **y** = predicted value (output)
- **x** = input variable (feature)
- **m** = slope of the line (how much y changes with x)
- **b** = y-intercept (value of y when x = 0)

◆ **Key Points:**

- **Used for:** Predicting numerical values (e.g., house prices, temperature).
- **Assumes:** A linear relationship between input(s) and output.
- **Types:**
 - **Simple Linear Regression:** 1 input feature
 - **Multiple Linear Regression:** 2 or more input features

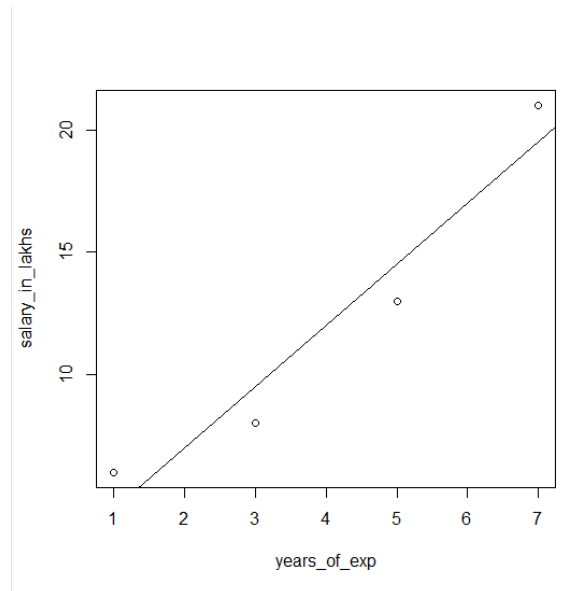
◆ **Example:**

Predictive someone's weight (y) based on their height (x).

Code:

```
years_of_exp = c(7,5,1,3)
salary_in_lakhs = c(21,13,6,8)
#employee.data = data.frame(satisfaction_score, years_of_exp, salary_in_lakhs)
employee.data = data.frame(years_of_exp, salary_in_lakhs)
employee.data
# Estimation of the salary of an employee, based on his year of experience and satisfaction score in his company.
model <- lm(salary_in_lakhs ~ years_of_exp, data = employee.data)
summary(model)
# The formula of Regression becomes
# Y = 2 + 2.5*year_of_Exp
```

```
# Visualization of Regression  
plot(salary_in_lakhs ~ years_of_exp, data = employee.data)  
abline(model)
```

Output:**Learnings:**

- **Simple linear regression** helps predict a numeric outcome based on one predictor.
 - You can **visualize** the regression model to understand the fit.
 - `summary(model)` tells you:
 - How strong and significant the relationship is
 - The model equation
 - This approach can be **expanded** to multiple predictors (e.g., adding satisfaction score).

Logistic regression:**Description:**

Logistic regression is a machine learning algorithm that estimates the probability of a binary outcome based on one or more input features. It uses the **logistic (sigmoid) function** to map predicted values to probabilities between 0 and 1.

Key Points:

- **Used for:** Classification, not regression (despite the name).
- **Output:** Probability of belonging to a class (usually thresholded at 0.5 to decide the final class).
- **Function used:** Sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- **Decision boundary:** Linear (can be extended to non-linear with feature transformations).

Example: Predict if an email is spam (1) or not spam (0) based on features like the number of links, word count, etc.

Code:

```
require(foreign)
require(MASS)
require(ggplot2)
url <- "https://stats.idre.ucla.edu/stat/data/binary.csv"
admission_data <- read.csv(url)
head(admission_data)
summary(admission_data)
str(admission_data)
admission_data$rank <- factor(admission_data$rank)
model <- glm(admit ~ gre + gpa + rank, data = admission_data, family = "binomial")
summary(model)
cat("Null deviance:", model$null.deviance, "\n") cat("Residual deviance:", model$deviance, "\n")
install.packages("pscl")
library(pscl)
pR2(model)
ggplot(admission_data, aes(x = gre, y = gpa, color = factor(admit))) + geom_point() + labs(x = "GRE
Score", y = "GPA", color = "Admission Status") + theme_minimal() + ggtitle("GRE vs GPA with
Admission Status")
```

Output:

```
Call:
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
```

Conclusion:

This exercise in logistic regression and visualization demonstrates how to:

6. **Fit a logistic regression model** to predict a binary outcome (admission).
7. **Visualize** the relationship between features (GRE, GPA, rank) and the predicted probability of admission.
8. Evaluate the model's **fit** and **performance** through model summary statistics and visualizations.

I hope this helps in understanding the process! Let me know if you need further clarifications or would like to dive deeper into any of these topics.

Practical: 06

Aim: Apply multiple regression ,if data have continuous independent variable .Apply on above dataset

Description: A multiple regression model is a mathematical equation used to predict the value of a dependent variable using two or more independent variables.

It models the relationship between the variables based on historical data, and then you can use it to predict future outcomes or understand how each variable contributes.

Key Concepts:

- **Dependent Variable (Y):** The variable you're trying to predict or explain.
- **Independent Variables (X):** The variables used to predict the dependent variable.
- **Regression Equation:** The equation that defines the relationship between the dependent and independent variables.
- **Regression Coefficients (b):** Values that represent the impact of each independent variable on the dependent variable.
- **Constant Term (a):** The value of the dependent variable when all independent variables are zero.

Formula:

The general formula for a multiple regression equation is:

$$Y = b_0 + b_1X_1 + b_2X_2 + \dots + b_nX_n + e$$

Where:

- Y = The dependent variable
- b_0 = The constant term
- b_1, b_2, \dots, b_n = The regression coefficients for each independent variable
- X_1, X_2, \dots, X_n = The independent variables
- e = The error term

```
code: college <- read.csv("C:/Users/admin.VSIT-X121-04/Downloads/binary (1).csv")
head(college)
nrow(college)
```

```
install.packages("caTools")
library(caTools)
split <- sample.split(college, SplitRatio = 0.75)
split
```

```

training_reg <- subset(college, split == "TRUE")
test_reg <- subset(college, split == "FALSE")

fit_MRegressor_model <- lm(formula = admit ~ gre+gpa+rank, data = training_reg)

predict_reg <- predict(fit_MRegressor_model,newdata = test_reg)
predict_reg

cdplot(as.factor(admit)~ gpa, data=college)
cdplot(as.factor(admit)~ gre, data=college)
cdplot(as.factor(admit)~ rank, data=college)

predict_reg <- ifelse(predict_reg > 0.5, 1, 0)
predict_reg
table(test_reg$admit, predict_reg)

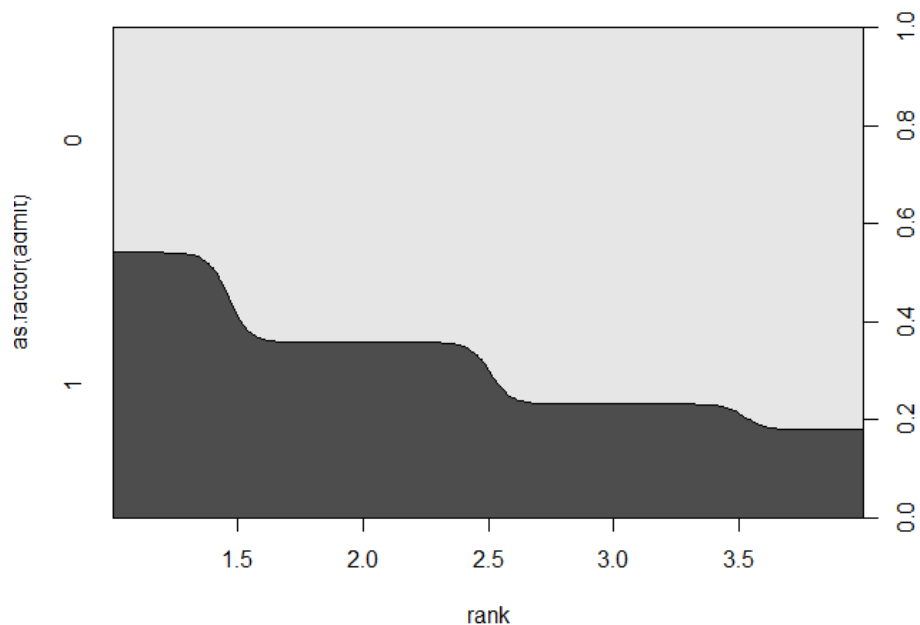
```

output:

```

  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
161 165 169 173 177 181 185 189 193 197 201 205 209 213 217 221 225 229 233 237
  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0
241 245 249 253 257 261 265 269 273 277 281 285 289 293 297 301 305 309 313 317
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
321 325 329 333 337 341 345 349 353 357 361 365 369 373 377 381 385 389 393 397
  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0  0  0
> table(test_reg$admit, predict_reg)
  predict_reg
    0  1
0  70  4
1  23  3
> |

```



Conclusion:

- The multiple regression analysis revealed that [independent variables] collectively have a statistically significant impact on the dependent variable, [dependent variable].
- The model explains approximately [R^2 value]% of the variance in [dependent variable], indicating a [strong/moderate/weak] fit.
- Among the predictors, [list the most significant ones] were found to be significant contributors, suggesting that these variables play a key role in predicting [dependent variable].
- These findings provide useful insights for [brief application or implication], though further research or model refinement may be necessary to improve predictive power and account for other potential influencing factors.

Practical: 07

Aim: HADOOP: Install, configure and run Hadoop and HDFS and explore HDFS.

Description:**Steps to Install Hadoop**

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.

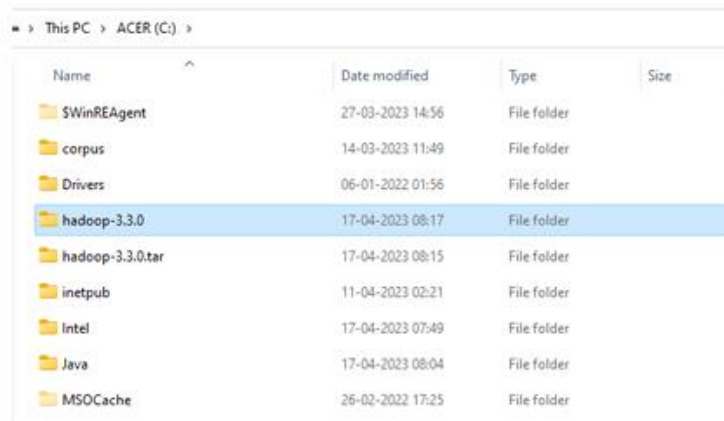
Test by executing start-all.cmd

1. Install Java 1. – Java JDK Link to download
2. <https://www.oracle.com/java/technologies/javase-jdk8-downloads.html>
3. – extract and install Java in C:\Java
4. – open cmd and type -> javac -version

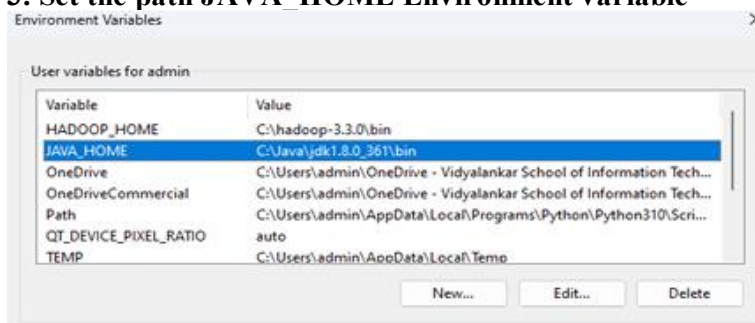
```
C:\Java>cd jdk1.8.0_361  
C:\Java\jdk1.8.0_361>cd bin  
C:\Java\jdk1.8.0_361\bin>javac -version  
javac 1.8.0_361  
C:\Java\jdk1.8.0_361\bin>
```

2. Download Hadoop - <https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz>

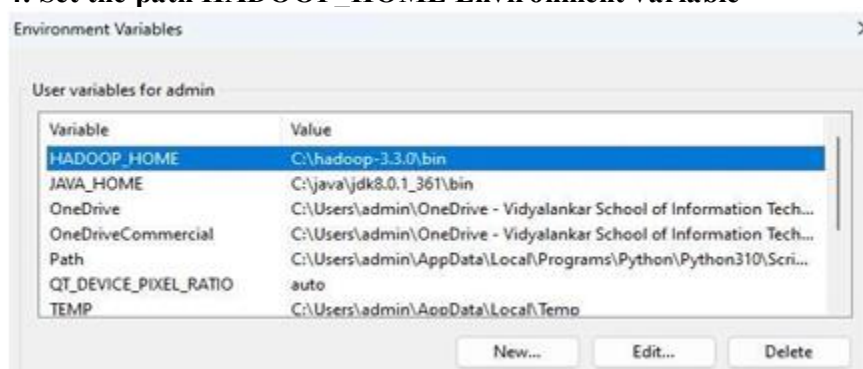
right click .rar.gz file -> show more options -> 7-zip->and extract to C:\Hadoop 3.3.0\



3. Set the path JAVA_HOME Environment variable



4. Set the path HADOOP_HOME Environment variable



Click on user variable -> path -> edit-> add path for Hadoop and java upto 'bin'

paste the

```
<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:/Hadoop-3.3.0/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>
```

code in folder and save

```
<name>yarn.resourcemanager.scheduler.address</name>
<value>127.0.0.1:8030</value>

</property>
<property>
<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 the files

- 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.

10. Format the NameNode

– Open cmd 'Run as Administrator' and type command "hdfs namenode –format"

```
C:\Windows\System32>cmd.exe /format -format
The filename, directory name, or volume label syntax is incorrect.
0023-06-17 09:18:46,777 INFO NameNode.NameNode: STARTUP_MSG:
*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = VST-K121-06/172.16.205.11
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 3.3.0
STARTUP_MSG: classpath = C:\hadoop-3.3.0\etc\hadoop;C:\hadoop-3.3.0\share\hadoop\common;C:\hadoop-3.3.0\share\hadoop\lib\accessors-start-1.2.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\animal-sniffer-annotations-1.17.jar;C:\had
3.3.0\share\hadoop\common\lib\asm-5.0.4.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\audience-annotations-0.8.0.jar;C
hadoop-3.3.0\share\hadoop\common\lib\avro-1.7.7.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\checker-qual-2.5.2.jar;C:h
oop-3.3.0\share\hadoop\common\lib\commons-beanutils-1.9.4.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\commons-cli-1.2
;C:\hadoop-3.3.0\share\hadoop\common\lib\commons-codec-1.11.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\commons-coll
ns-2.2.2.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\commons-compress-1.19.jar;C:\hadoop-3.3.0\share\hadoop\common\
commons-configuration2-2.1.1.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\commons-daemon-1.0.13.jar;C:\hadoop-3.3.0\s
hadoop\common\lib\commons-io-2.5.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\commons-lang3-3.7.jar;C:\hadoop-3.3.0\
hadoop\common\lib\commons-logging-1.1.3.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\commons-math3-3.1.1.jar;C:\ha
d3.3.0\share\hadoop\common\lib\commons-net-3.8.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\commons-text-1.4.jar;C:\ha
d3.3.0\share\hadoop\common\lib\curator-client-4.2.0.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\curator-framework-4
.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\curator-recipes-4.2.0.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\dnsjav
1.7.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\failureaccess-1.0.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\gson-2
1.7.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\guava-27.0-jre.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\hadoop-annt
ns-3.3.0.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\hadoop-auth-3.3.0.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\h
-shaded-protobuf-3_7_1-0.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\htrace-core4-4.1.0-incubating.jar;C:\hadoop
3.3.0\share\hadoop\common\lib\httpclient-4.5.6.jar;C:\hadoop-3.3.0\share\hadoop\common\lib\httpcore-4.4.10.jar;C:\hadoop
C:\Windows\System32>
```

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

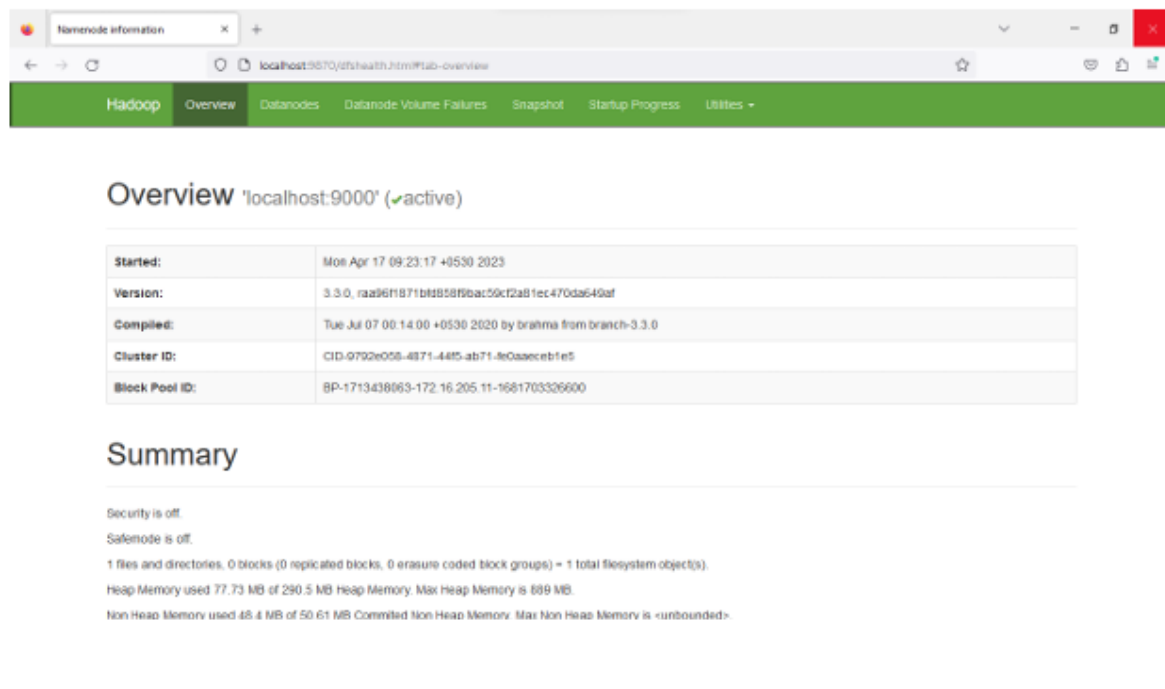
```
C:\Windows\System32>cd C:\hadoop-3.3.0\sbin
C:\hadoop-3.3.0\sbin>start-all.cmd
This script is Deprecated. Instead use start-dfs.cmd and start-yarn.cmd
The filename, directory name, or volume label syntax is incorrect.
The filename, directory name, or volume label syntax is incorrect.
starting yarn daemons
The filename, directory name, or volume label syntax is incorrect.
```

- You will get 4 more running threads for Datanode, namenode, resource manager and node manager**

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

```
C:\hadoop-3.3.0\sbin>jps
8848 NodeManager
11044 DataNode
4548 NameNode
8552 ResourceManager
12044 Jps
```

13. Run <http://localhost:9870/> from any browser

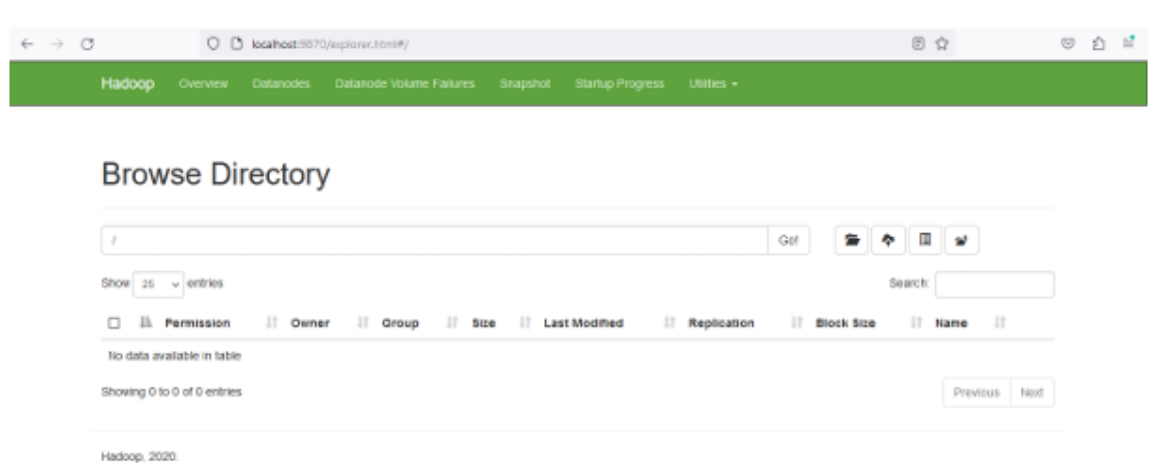


The screenshot shows the Hadoop NameNode Overview page in a web browser. The browser address bar shows `localhost:9870/dfshealth.html#tab-overview`. The page has a green navigation bar with tabs: Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. The main content area is titled "Overview 'localhost:9000' (✓active)". Below the title is a table with the following information:

Started:	Mon Apr 17 09:23:17 +0530 2023
Version:	3.3.0, raa96f1b71b1b858f9bac59cf2a81ec470da649af
Compiled:	Tue Jul 07 00:14:00 +0530 2020 by brahma from branch-3.3.0
Cluster ID:	CID:0792e058-4871-44f5-ab71-4e0aaceb1e5
Block Pool ID:	BP-1713438663-172.16.205.11-1681703326600

Below the table is a "Summary" section with the following text:

Security is off.
Safemode is off.
1 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total filesystem object(s).
Heap Memory used 77.73 MB of 290.5 MB Heap Memory. Max Heap Memory is 689 MB.
Non Heap Memory used 48.4 MB of 50.61 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.



The screenshot shows the Hadoop Browse Directory page in a web browser. The browser address bar shows `localhost:9870/explorer.html#`. The page has a green navigation bar with tabs: Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. The main content area is titled "Browse Directory". Below the title is a search bar with the text `/` and a "Go!" button. To the right of the search bar are icons for file operations. Below the search bar is a "Show" dropdown menu set to "25" and a "Search:" input field. Below the search bar is a table with the following columns: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. The table is currently empty, with the text "No data available in table" displayed below it. Below the table is a "Showing 0 to 0 of 0 entries" message and a "Previous" button. At the bottom of the page is a "Hadoop, 2020." footer.