DATA SCIENCE

LAB RECORD FILE

(CBCPE01)

NETAJI SUBHASH UNIVERSITY OF TECHNOLOGY

EAST CAMPUS

SUBMITTED TO:-

Dr. Amita Jain

Assistant Professor

Department of Computer Science

And Engineering

SUBMITTED BY:-

Naman Dureja

2021UCB6017

B.Tech – CSDA

5TH Semester

TABLE OF CONTENT

(TEACHER'S SIGNATURE)

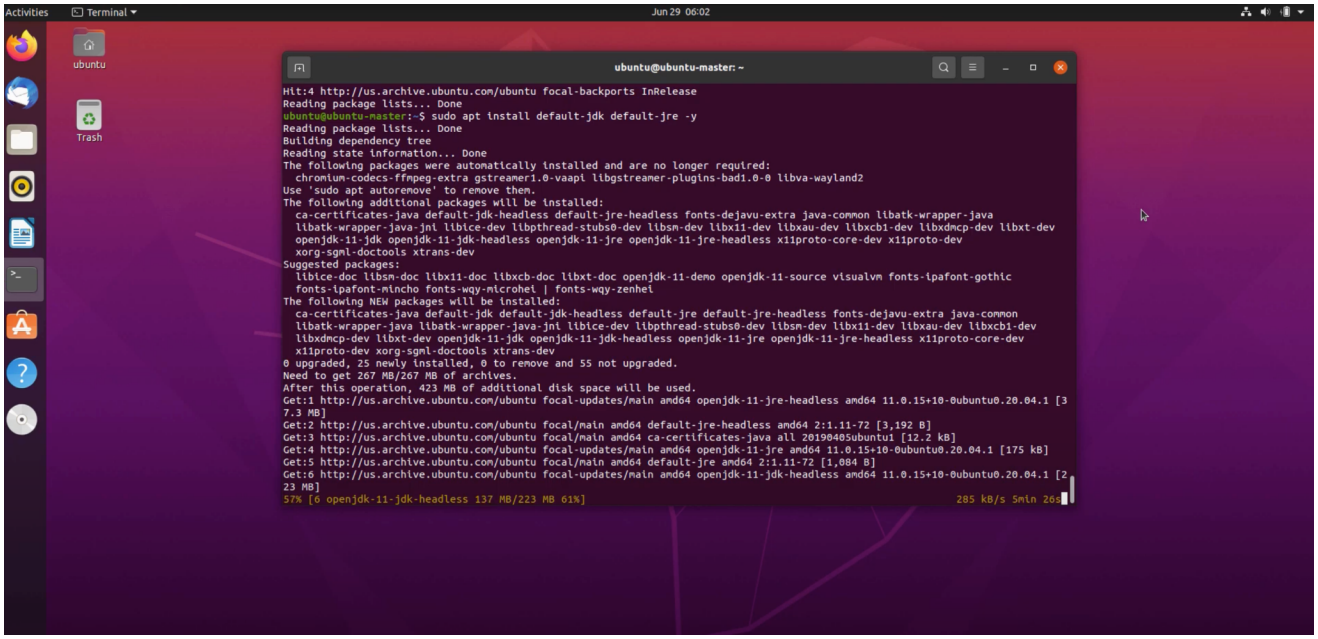
DATA SCIENCE

Laboratory Number - 01

EXPERIMENT:- Install, configure, and run Hadoop and HDFS

OUTPUT -

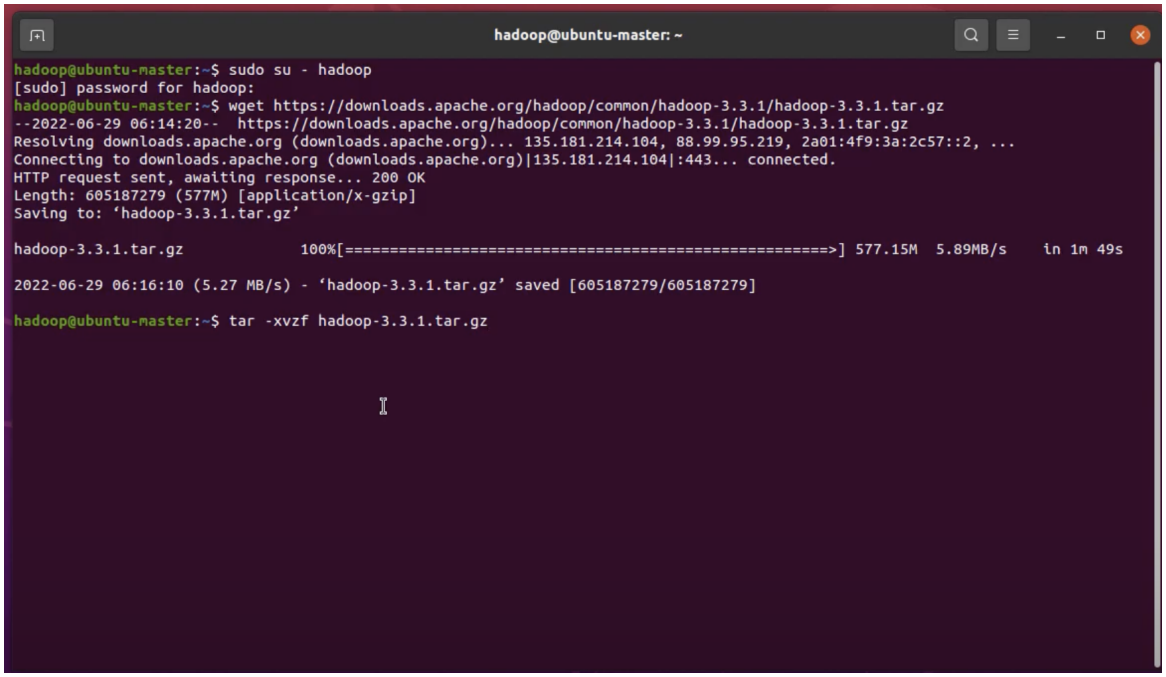
1. Installing Java JDK and JRE



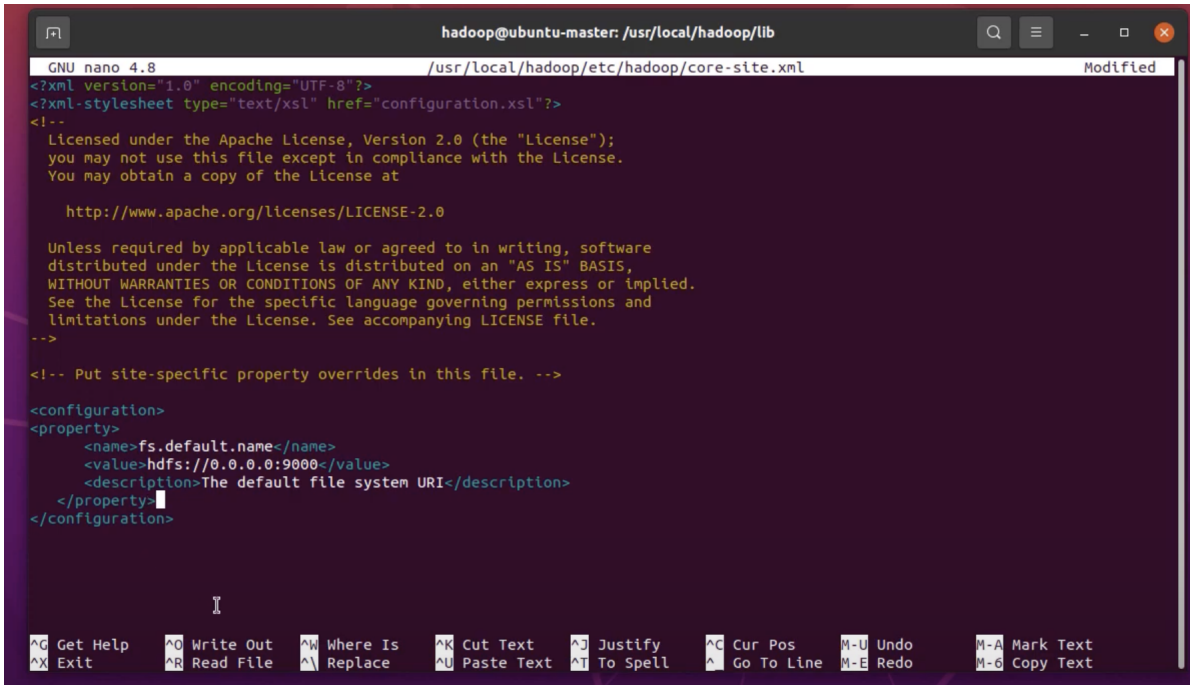
2. Add a new user for hadoop

```
ubuntu@ubuntu-master:~$ sudo adduser hadoop
[sudo] password for ubuntu:
Adding user `hadoop' ...
Adding new group `hadoop' (1001) ...
Adding new user `hadoop' (1001) with group `hadoop' ...
Creating home directory `/home/hadoop' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for hadoop
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
ubuntu@ubuntu-master:~$ sudo usermod -aG sudo hadoop
ubuntu@ubuntu-master:~$
```

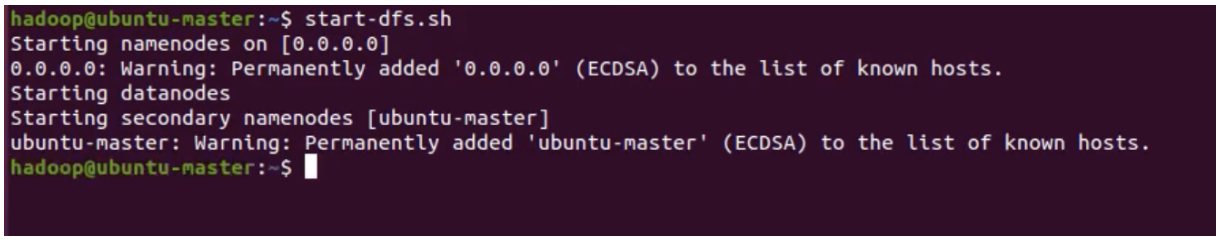
3. Install hadoop using ‘wget’ command



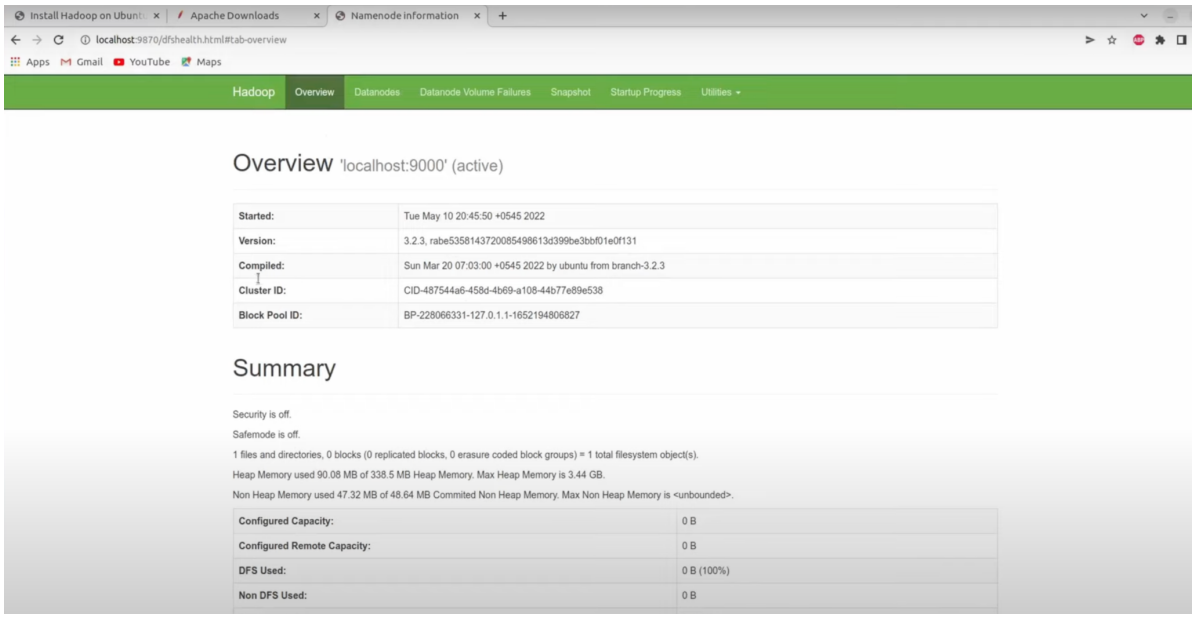
4. Edit the configuration files to add the hostname and port (0.0.0.0)



5. Run the the script to start hadoop dfs



6. Access Hadoop UI from Browser



Installation and configuration of Hadoop in its different modes.

Component	Property	Standalone	Pseudo-distributed	Fully distributed
Core	fs.default.name	file:/// (default)	hdfs://localhost/	hdfs://*namenode*/
HDFS	dfs.replication	N/A	1	3 (default)
MapReduce	mapred.job.tracker	local (default)	localhost:8021	*jobtracker*:8021

DATA SCIENCE

Laboratory Number - 02

EXPERIMENT:- Implement word count / frequency programs using MapReduce.

PROGRAM -

Mapper Class

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WC_Mapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable> {

	private final static IntWritable one = new IntWritable(1);
	private Text word = new Text();

	public void map(LongWritable key, Text value, OutputCollector<Text,IntWritable>
	output, Reporter reporter) throws IOException {
		String line = value.toString();
		StringTokenizer tokenizer = new StringTokenizer(line);
		while (tokenizer.hasMoreTokens()) {
			word.set(tokenizer.nextToken()); output.collect(word, one);
		}
	}

}
```

Reducer Class

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WC_Reducer extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {

public voidreduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException {
            int sum=0;
            while (values.hasNext()) {
                  sum+=values.next().get();
            }
            output.collect(key,new IntWritable(sum));
      }
}
```

Runner Class

```
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

public class WC_Runner {
public static void main(String[] args) throws IOException {
            JobConf conf = new JobConf(WC_Runner.class);
            conf.setJobName("WordCount");
            conf.setOutputKeyClass(Text.class);
            conf.setOutputValueClass(IntWritable.class);
            conf.setMapperClass(WC_Mapper.class);
            conf.setCombinerClass(WC_Reducer.class);
            conf.setReducerClass(WC_Reducer.class);
            conf.setInputFormat(TextInputFormat.class);
```

```
            conf.setOutputFormat(TextOutputFormat.class);
            FileInputFormat.setInputPaths(conf, new Path(args[0]));
            FileOutputFormat.setOutputPath(conf, newPath(args[1]));
            JobClient.runJob(conf);
        }
}
```

OUTPUT -

```
ndureja@Namans-MacBook-Air-2
The 2
mame 1
is 4
Naman 1
and 1
I 1
am 1
a 1
good 1
```

DATA SCIENCE

Laboratory Number - 03

EXPERIMENT:- Implement an MR program that processes a weather dataset.

PROGRAM -

```
import random


def generate_weather_data(num_records):
	locations = ['Delhi', 'Gurgaon', 'New York', 'London', 'Chennai']
	data = []
	for _ in range(num_records):
		date = f"{random.randint(2000, 2023)}-{random.randint(1,
		12):02d}-{random.randint(1, 28):02d}"
		location = random.choice(locations)
		temperature = round(random.uniform(-20, 40), 1)
		humidity = random.randint(10, 100)
		wind_speed = round(random.uniform(0, 100), 1)
		data.append(f"{date},{location},{temperature},{humidity},{wind_speed}")
	return data


weather_data = generate_weather_data(15)


# The Map function
def map_weather_data(record):
	"""
	Map function that processes a string (weather record),
	and returns a tuple with the format (location, (temperature, 1))
	to calculate the average temperature for that location.
	"""
	_, location, temperature, _, _ = record.split(',')
	return (location, (float(temperature), 1))
```

```python
# The Reduce function
def reduce_weather_data(mapped_data):
	"""
	Reduce function that aggregates the mapped weather data and
	calculates the average temperature for each location.
	"""
	weather_aggregate = {}
	for location, temp_count in mapped_data:
		if location in weather_aggregate:
			weather_aggregate[location][0] += temp_count[0]
			weather_aggregate[location][1] += temp_count[1]
		else:
			weather_aggregate[location] = list(temp_count)
	# Calculate average temperatures
	for location in weather_aggregate:
		total_temp, count = weather_aggregate[location]
		weather_aggregate[location] = total_temp / count
		weather_aggregate[location] = round(weather_aggregate[location], 1)
	return weather_aggregate

# MapReduce process for weather data
mapped_weather = map(map_weather_data, weather_data)
reduced_weather_data = reduce_weather_data(mapped_weather)

for location in reduced_weather_data:
	print(f"{location}: {reduced_weather_data[location]}")
```

OUTPUT -

```
ndureja@Namans-MacBook-Air-2 ML-Lab-FIle % python3 map-reduce.py
Chennai: -0.3
Delhi: 7.3
Gurgaon: 10.3
London: 12.5
New York: -1.5
```

DATA SCIENCE

Laboratory Number - 04

EXPERIMENT:- Implement Linear and logistic Regression.

PROGRAM -

Linear Regression

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables in particular are significant predictors of the outcome variable? The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

```
# import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
```

```
# Create synthetic data for Linear Regression
np.random.seed(0)
X_linear = np.sort(5 * np.random.rand(80, 1), axis=0)
y_linear = 2 * X_linear + 1 + np.random.randn(80, 1)
```

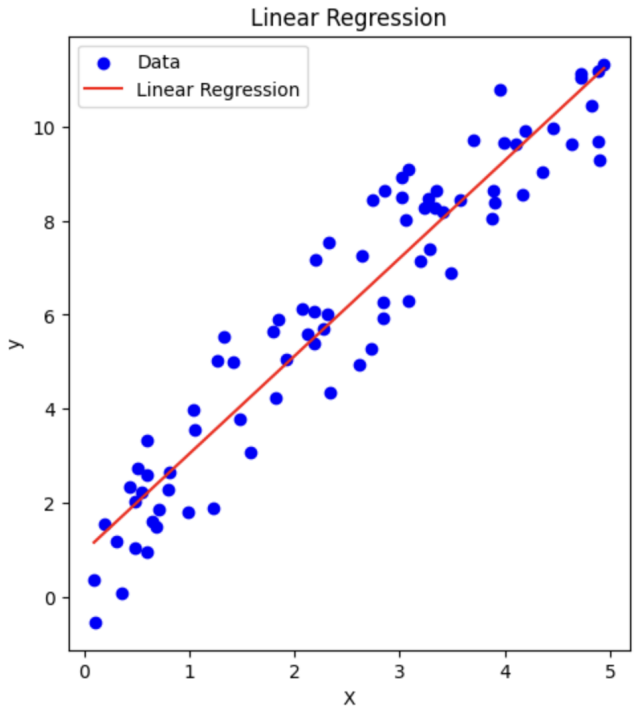
```
linear_model = LinearRegression()
linear_model.fit(X_linear, y_linear)
y_linear_pred = linear_model.predict(X_linear)
```

```
linear_model = LinearRegression()
linear_model.fit(X_linear, y_linear)
y_linear_pred = linear_model.predict(X_linear)
```

✓ 0.0s

```
plt.figure(figsize=(12, 6))

# Linear Regression Plot
plt.subplot(1, 2, 1)
plt.scatter(X_linear, y_linear, color='blue', label='Data')
plt.plot(X_linear, y_linear_pred, color='red', label='Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.show()
```

✓ 0.2s




```
    mse_linear = mean_squared_error(y_linear, y_linear_pred)
    print(f"Mean Squared Error (Linear Regression): {mse_linear:.2f}")
```

✓ 0.0s

```
Mean Squared Error (Linear Regression): 0.95
```

Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1".

```
# importing the required libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from matplotlib.colors import ListedColormap
```

```
# Create synthetic data for classification
X, y = make_classification(n_samples=200, n_features=2, n_classes=2, n_clusters_per_class=1, n_redundant=0, random_state=42)
```

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

```
# Create and train the Logistic Regression model
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
```

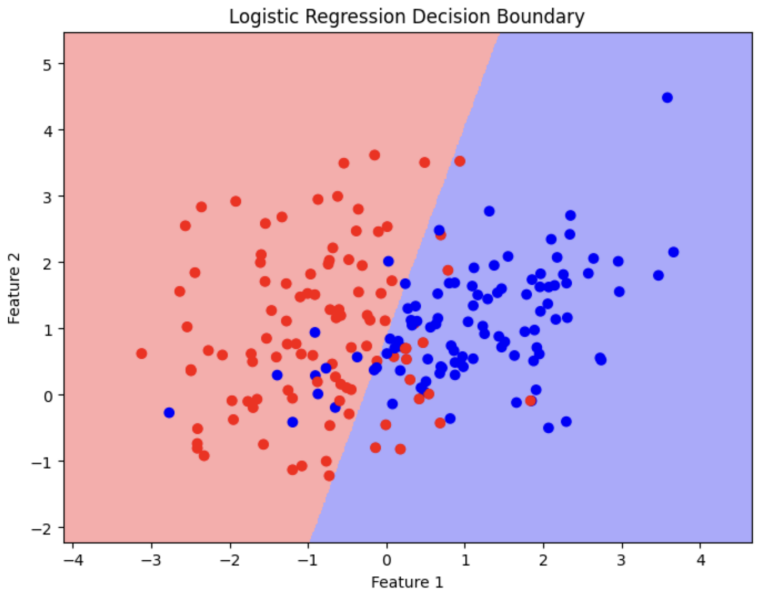
```
# Make predictions on the test data
y_pred = logistic_model.predict(X_test)
```

```
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.88
```

```
# Create a mesh grid for visualization
h = 0.02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict the class for each point in the mesh grid
Z = logistic_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

```
# Plot the decision boundary and data points
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(['#FF0000', '#0000FF']))
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Logistic Regression Decision Boundary')
plt.show()
```



```python
    # Display the confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")

    # Pretty-print the confusion matrix as a Pandas DataFrame
    import pandas as pd
    cm_df = pd.DataFrame(cm, index=['Actual 0', 'Actual 1'], columns=['Predicted 0', 'Predicted 1'])
    print(cm_df)
```

```
Confusion Matrix:
          Predicted 0  Predicted 1
Actual 0           18            5
Actual 1            0           17
```

DATA SCIENCE

Laboratory Number - 05

EXPERIMENT:- Implement SVM / Decision tree classification techniques.

PROGRAM -

SVM

```
import numpy as np
import pandas as pd
from sklearn.svm import SVC

data = pd.read_csv('/content/cell_samples (1).csv')
data
```

[ ]

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2
...	...	...	...	...	...	...	...	...	...	...	...
694	776715	3	1	1	1	3	2	1	1	1	2
695	841769	2	1	1	1	2	1	1	1	1	2
696	888820	5	10	10	3	7	3	8	10	2	4
697	897471	4	8	6	4	3	4	10	6	1	4
698	897471	4	8	8	5	4	5	10	4	1	4

699 rows × 11 columns

```
data = data.apply(pd.to_numeric, errors='coerce')
data = data.fillna(data.mean())
data
```

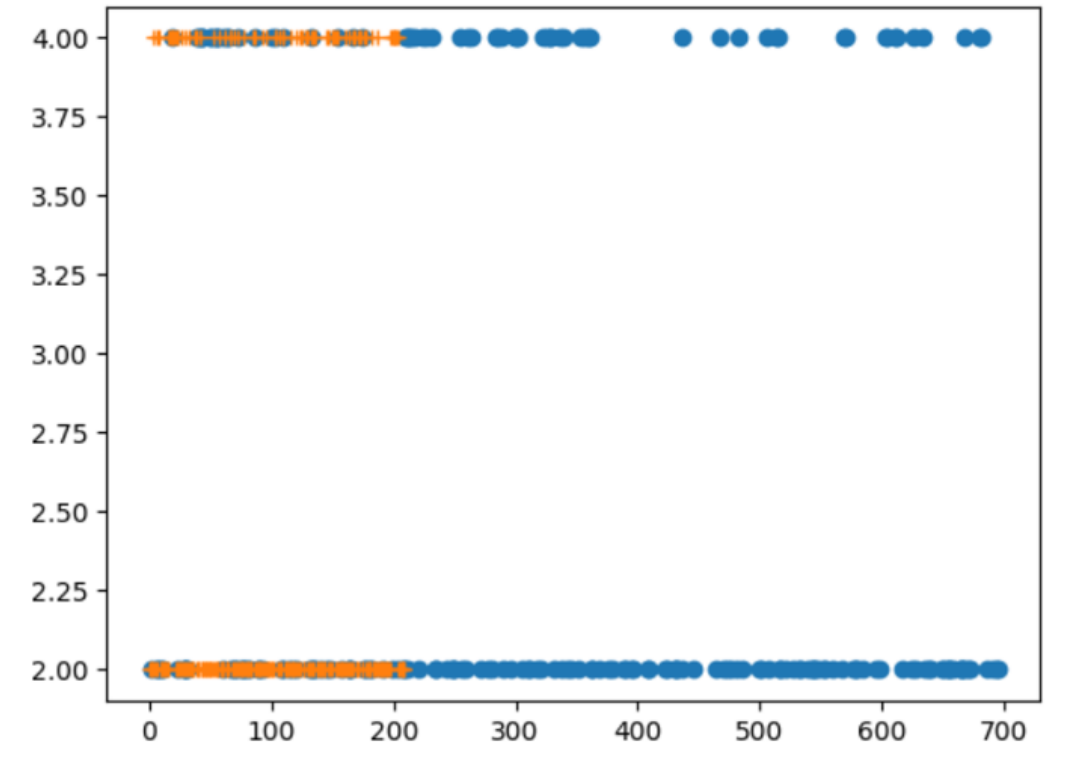
	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
0	1000025	5	1	1	1	2	1.0	3	1	1	2
1	1002945	5	4	4	5	7	10.0	3	2	1	2
2	1015425	3	1	1	1	2	2.0	3	1	1	2
3	1016277	6	8	8	1	3	4.0	3	7	1	2
4	1017023	4	1	1	3	2	1.0	3	1	1	2
...	...	...	...	...	...	...	...	...	...	...	...
694	776715	3	1	1	1	3	2.0	1	1	1	2
695	841769	2	1	1	1	2	1.0	1	1	1	2
696	888820	5	10	10	3	7	3.0	8	10	2	4
697	897471	4	8	6	4	3	4.0	10	6	1	4
698	897471	4	8	8	5	4	5.0	10	4	1	4

699 rows × 11 columns

```
from sklearn.model_selection import train_test_split
tr, te = train_test_split(data, test_size = 0.3, random_state = 42)

# from sklearn.svm import NuSVC
svc = SVC(kernel = 'linear', gamma = 'auto')
svc.fit(tr.iloc[:,1:-1], tr.iloc[:,-1])
SVC(gamma='auto', kernel='linear')
data.iloc[:,-1]
```

```
0 2 1 2 2 2 3 2 4 2 .. 694 2 695 2 696 4 697 4 698 4 Name: Class, Length: 699, dtype: int64
```

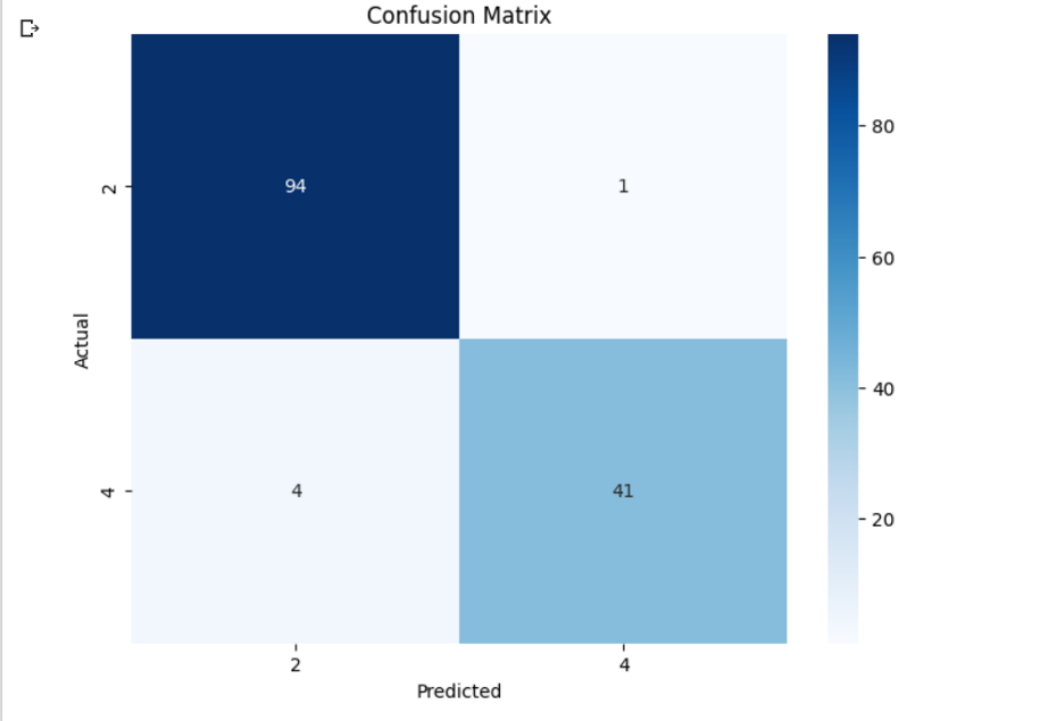
```
from sklearn.metrics import accuracy_score
accuracy_score(te.iloc[:,-1], svc.predict(te.iloc[:,1:-1]))
import matplotlib.pyplot as plt
for col in data.iloc[:,1:-1].columns:
  # data[[col, 'Class']].plot(kind = 'scatter')
  plt.figure()
  plt.scatter(data['Class'], data[col])
  plt.show()
plt.plot(te.iloc[:,-1], 'o')
plt.plot(svc.predict(te.iloc[:,1:-1]), '+')
```



```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

cm = confusion_matrix(te.iloc[:,-1], svc.predict(te.iloc[:,1:-1]))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

[ ]

Accuracy: 0.96



Decision Tree

```
# importing the required libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from matplotlib.colors import ListedColormap
```

```
# Create synthetic data for classification
X, y = make_classification(n_samples=200, n_features=2, n_classes=2, n_clusters_per_class=1, n_redundant=0, random_state=42)
```

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

```
# Create and train the Decision Tree classifier
tree_classifier = DecisionTreeClassifier()
tree_classifier.fit(X_train, y_train)
```

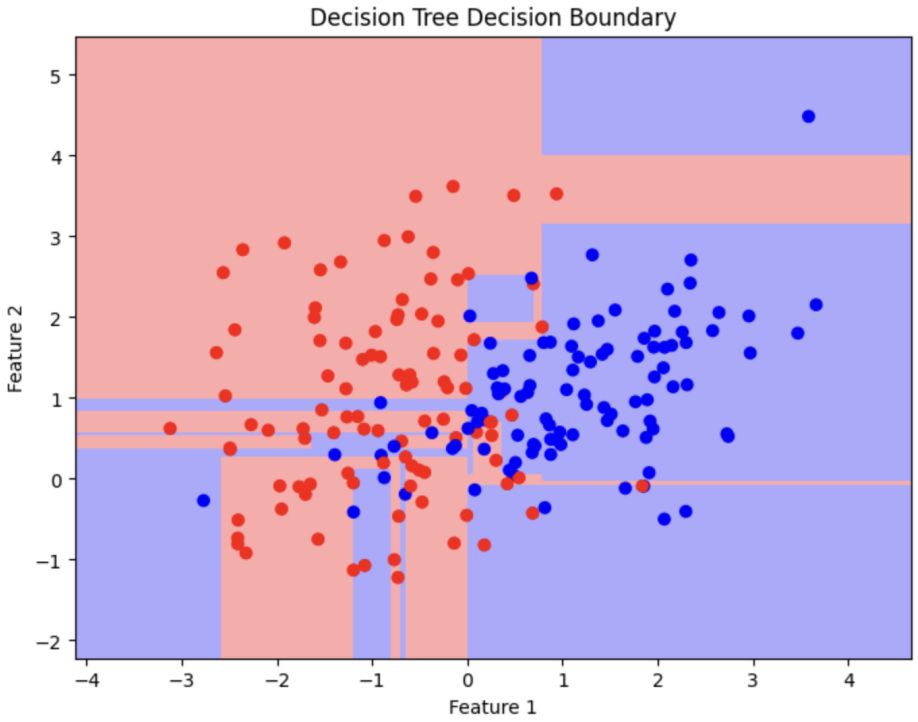
```
# Make predictions on the test data
y_pred = tree_classifier.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.80
```

```
# Create a mesh grid for visualization
h = 0.02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

```
# Predict the class for each point in the mesh grid
Z = tree_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

```
# Plot the decision boundary and data points
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(['#FF0000', '#0000FF']))
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Tree Decision Boundary')
plt.show()
```




```
# Print the confusion matrix
print("Confusion Matrix:")

cm = confusion_matrix(y_test, y_pred)

import pandas as pd
cm_df = pd.DataFrame(cm,
                     index = ['Actual 0', 'Actual 1'],
                     columns = ['Predicted 0', 'Predicted 1'])

print(cm_df)
```

```
Confusion Matrix:
          Predicted 0  Predicted 1
Actual 0           18            5
Actual 1            3           14
```

DATA SCIENCE

Laboratory Number - 06

EXPERIMENT:- Implement Random Forest classification using any dataset.

PROGRAM -

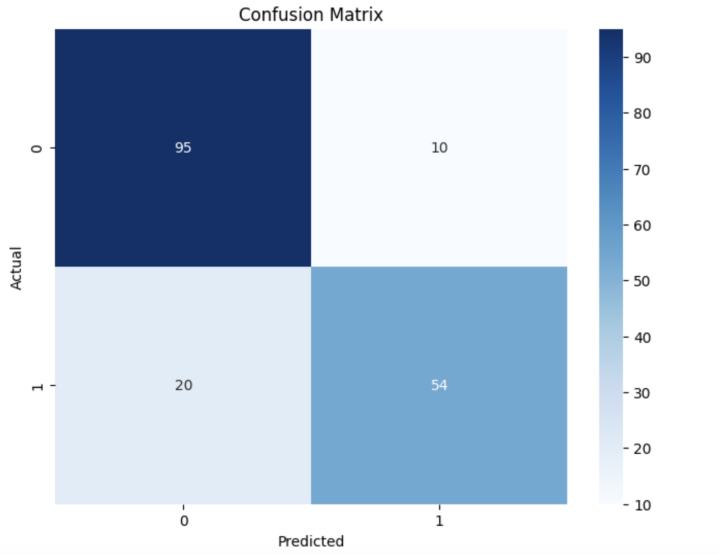
```
#importing libraries

from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv('train.csv')
data['Age'] = data['Age'].fillna(data['Age'].mean())
data['Embarked'] = data['Embarked'].fillna(data['Embarked'].mode())

label_encoder = preprocessing.LabelEncoder()
data['Sex'] = label_encoder.fit_transform(data['Sex'])
data['Embarked'] = label_encoder.fit_transform(data['Embarked'])

X = data.iloc[:, [2, 4, 5, 6, 7, 11]]
Y = data.iloc[:, 1]

# Performing feature scaling
sc = StandardScaler()
sc.fit(X)
X = sc.transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42)
```

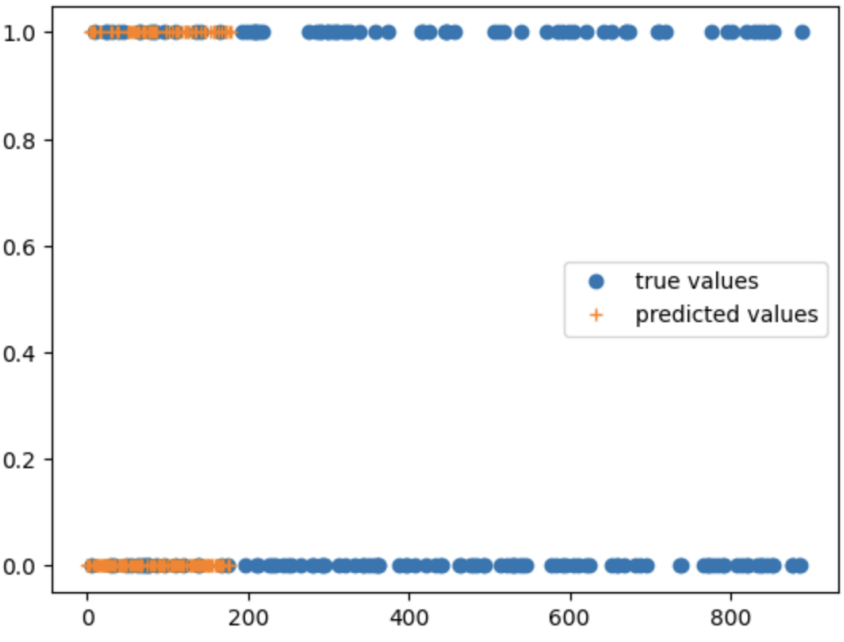
```
# Random Forest classifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100)
```

```
Accuracy: 83.24022346368714
```

```
# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```




```
# Plotting the results
plt.plot(y_test, 'o', label="true values")
plt.plot(y_pred, '+', label="predicted values")
plt.legend()
plt.show()
```



DATA SCIENCE

Laboratory Number - 07

EXPERIMENT:- Implement Naïve Bayes theorem to classify the English text.

PROGRAM -

```
import numpy as np
import pandas as pd
```

```
data = {
    "corpus": [
        "I love this movie, it's so entertaining!",
        "The weather is terrible today.",
        "This book is amazing, highly recommended.",
        "I don't like the taste of this dish.",
        "The performance was outstanding, I was truly impressed.",
        "I hate waiting in long lines.",
        "The food at the restaurant was delicious.",
        "The service was very slow and disappointing.",
        "The concert was fantastic, I had a great time.",
        "I'm not a fan of the new design.",
        "The team played poorly, it was a disappointing match.",
        "I adore the artwork in this museum.",
        "The customer support was helpful and friendly.",
        "I can't stand the noise in this neighborhood.",
        "The movie was boring, I fell asleep halfway through.",
        "The software is user-friendly and efficient.",
        "The traffic was unbearable, I was stuck for hours.",
        "This product is a waste of money.",
        "The atmosphere in the cafe was cozy and inviting.",
        "I'm impressed with the quality of this product.",
        "The hotel room was dirty and smelled bad.",
        "I enjoy spending time with my friends.",
        "The play was thought-provoking and well-executed.",
        "I couldn't stop laughing, the comedy show was hilarious.",
        "The experience was underwhelming, I expected more.",
    ],
    "label": [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0]
}
df = pd.DataFrame(data)
```

```
positive = df[df['label'] == 1]
negative = df[df['label'] == 0]

print("No. of Positive: ", len(positive))
print("No. of Negative: ", len(negative))
```

```
No. of Positive:  12
No. of Negative:  13
```

```
total = len(df)
prior_positive = len(positive) / total
prior_negative = len(negative) / total

print("Prior probability of Positive samples: ", prior_positive)
print("Prior probability of Negative samples: ", prior_negative)
```

```
Prior probability of Positive samples:  0.48
Prior probability of Negative samples:  0.52
```

```
positive_text = ' '.join(positive['corpus']).split()
negative_text = ' '.join(negative['corpus']).split()

vocabulary = list(set(negative_text + positive_text))
print('Vocabulary: \n', vocabulary)
```

```
Vocabulary:
 ['book', 'impressed', 'in', 'time', 'outstanding,', 'was', 'artwork', 'food', 'highly', 'impressed.', 'comedy', 'friendly.',
```

```
positive_word_freq = { word: positive_text.count(word) for word in vocabulary }
negative_word_freq = { word: negative_text.count(word) for word in vocabulary }

word_data = {
    'Positive Word Frequency': positive_word_freq,
    'Negative Word Frequency': negative_word_freq,
}
word_df = pd.DataFrame(word_data)
word_df.head(10)
```

	Positive Word Frequency	Negative Word Frequency
book	1	0
impressed	1	0
in	2	2
time	1	0
outstanding,	1	0
was	7	8
artwork	1	0
food	1	0
highly	1	0
impressed.	1	0

```
test_data = {
    "corpus": [
        "This restaurant serves delicious food and has excellent service.",
        "I'm really disappointed with the customer service I received.",
        "The weather is perfect for outdoor activities today.",
        "I can't believe how fast and efficient the delivery service was.",
        "The movie was a complete waste of time and money, I regret watching it."
    ],
    "label": [1, 0, 1, 1, 0]
}
test_df = pd.DataFrame(test_data)
```

✓ 0.0s

```
def predict(string, positive_word_freq, negative_word_freq, prior_positive, prior_negative, smoothing_factor = 1): # Using Laplace Smoothing for Likelihood Calculation

    new_text_words = string.split()
    likelihood_positive = 1
    for word in new_text_words:
        likelihood_positive *= (positive_word_freq.get(word, 0) + smoothing_factor) / (len(positive_text) + smoothing_factor * len(vocabulary))

    likelihood_negative = 1
    for word in new_text_words:
        likelihood_negative *= (negative_word_freq.get(word, 0) + smoothing_factor) / (len(negative_text) + smoothing_factor * len(vocabulary))

    # Apply Naive Bayes formula
    posterior_positive = prior_positive * likelihood_positive
    posterior_negative = prior_negative * likelihood_negative

    # Classify based on the higher posterior probability
    predicted_class = 1 if posterior_positive > posterior_negative else 0
    return predicted_class
```

```
test_df['Predicted Label'] = [
    predict(
        string, positive_word_freq, negative_word_freq, prior_positive, prior_negative
    ) for string in test_df['corpus']
]
```

✓ 0.0s

```
test_df.head()
```

✓ 0.0s

	corpus	label	Predicted Label
0	This restaurant serves delicious food and has ...	1	1
1	I'm really disappointed with the customer serv...	0	1
2	The weather is perfect for outdoor activities ...	1	0
3	I can't believe how fast and efficient the del...	1	0
4	The movie was a complete waste of time and mon...	0	0

DATA SCIENCE

Laboratory Number - 08

EXPERIMENT:- Implement clustering techniques (KMean, KMedoids).

PROGRAM -

K-Means

```
# importing the necessary packages
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

```
# Generate synthetic data
X, y = make_blobs(n_samples=300, centers=4, random_state=42)
```

```
# Create a K-means model with the desired number of clusters (k)
k = 4
kmeans = KMeans(n_clusters=k)
```

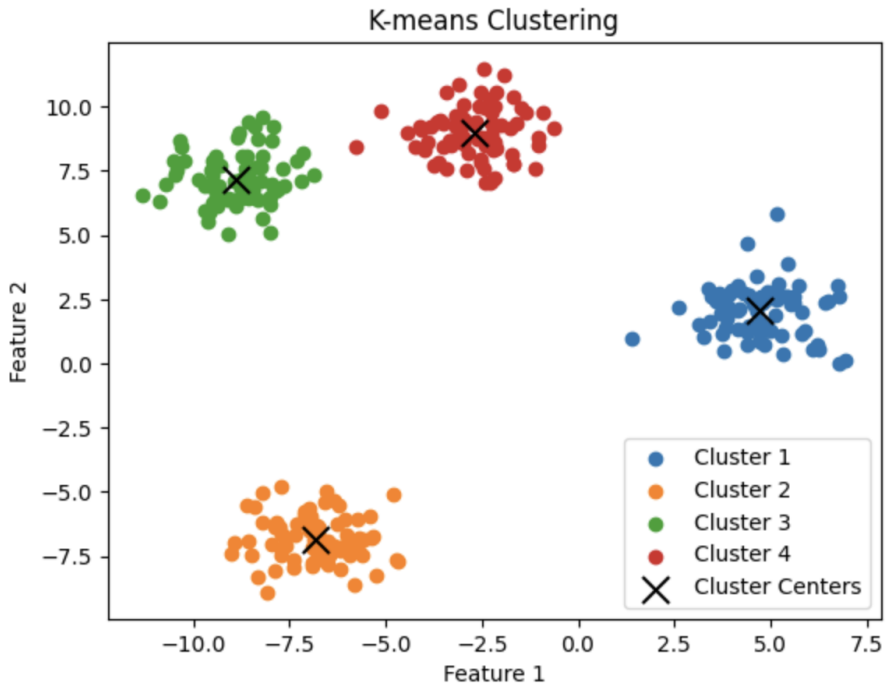
```
# Fit the model to the data and predict cluster labels
kmeans.fit(X)
cluster_labels = kmeans.labels_
```

```
# Get the coordinates of cluster centers
cluster_centers = kmeans.cluster_centers_

# Visualize the data and cluster centers
plt.figure(figsize=(8, 6))
```

```
# Plot data points with different colors for each cluster
for i in range(k):
    plt.scatter(X[cluster_labels == i, 0], X[cluster_labels == i, 1], label=f'Cluster {i+1}')

# Plot cluster centers
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1,], color='black', marker='x', s=150, label='Cluster Centers')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-means Clustering')
plt.legend()
plt.show()
```

```python
# Plot data points with different colors for each cluster
for i in range(k):
    plt.scatter(X[cluster_labels == i, 0], X[cluster_labels == i, 1], label=f'Cluster {i+1}')

# Plot cluster centers
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1,], color='black', marker='x', s=150, label='Cluster Centers')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-means Clustering')
plt.legend()
plt.show()
```

✓ 0.1s



K-Medoids

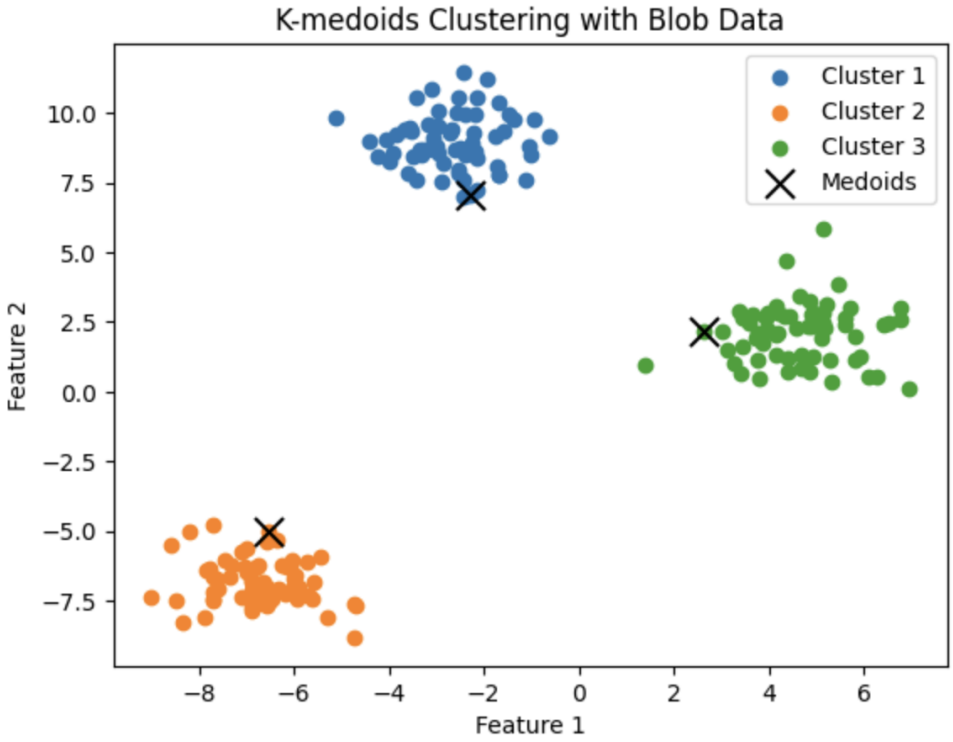
```python
# importing libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

```python
# Generate synthetic data with blobs
X, y = make_blobs(n_samples=200, centers=3, random_state=42)
```

```python
# Define the number of clusters (k)
k = 3

# Use K-means to initialize medoids
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X)
medoids = kmeans.cluster_centers_

# Maximum iterations
max_iterations = 100
```

```python
# K-medoids clustering algorithm
for _ in range(max_iterations):
    # Assign each point to the nearest medoid
    distances = np.linalg.norm(X[:, np.newaxis] - medoids, axis=2)
    labels = np.argmin(distances, axis=1)

    # Update medoids by selecting the data point that minimizes the total dissimilarity within the cluster
    new_medoids = np.array([X[labels == i][np.argmin(np.sum(distances[labels == i], axis=1))] for i in range(k)])

    # Check for convergence
    if np.all(medoids == new_medoids):
        break

    medoids = new_medoids
```

```python
# Visualize the clusters and medoids
for i in range(k):
    cluster_points = X[labels == i]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {i + 1}')

plt.scatter(medoids[:, 0], medoids[:, 1], color='black', marker='x', s=150, label='Medoids')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-medoids Clustering with Blob Data')
plt.legend()
plt.show()
```


K-medoids Clustering with Blob Data
Cluster 1
Cluster 2
Cluster 3
Medoids
Feature 1
Feature 2
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−8
−6
−4
−2
0
2
4
6

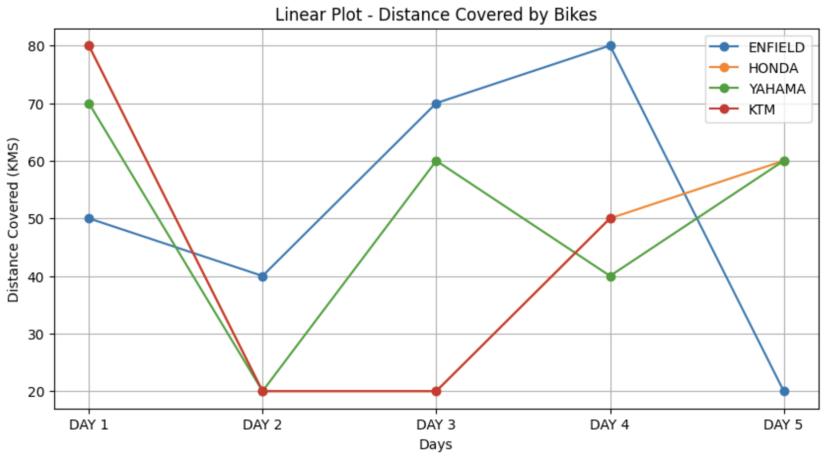
DATA SCIENCE

Laboratory Number - 09

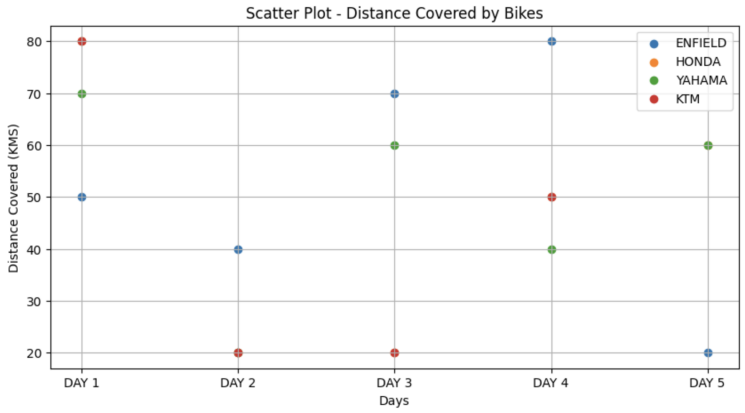
EXPERIMENT:- Visualize data using any plotting framework.

PROGRAM -

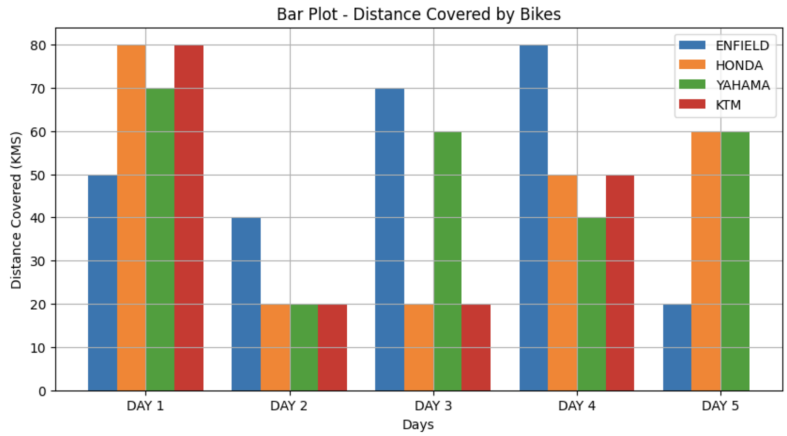
```
import pandas as pd
import matplotlib.pyplot as plt

# Create a DataFrame with the given data
data = {
        "DAYS": ["DAY 1", "DAY 2", "DAY 3", "DAY 4", "DAY 5"],
        "ENFIELD": [50, 40, 70, 80, 20],
        "HONDA": [80, 20, 20, 50, 60],
        "YAHAMA": [70, 20, 60, 40, 60],
        "KTM": [80, 20, 20, 50, None], # Note: Missing value for DAY 5
}

df = pd.DataFrame(data)

# Linear Plot
plt.figure(figsize=(10, 5))
plt.plot(df['DAYS'], df['ENFIELD'], label='ENFIELD', marker='o')
plt.plot(df['DAYS'], df['HONDA'], label='HONDA', marker='o')
plt.plot(df['DAYS'], df['YAHAMA'], label='YAHAMA', marker='o')
plt.plot(df['DAYS'], df['KTM'], label='KTM', marker='o')
plt.xlabel('Days')
plt.ylabel('Distance Covered (KMS)')
plt.title('Linear Plot - Distance Covered by Bikes')
plt.legend()
plt.grid(True)
plt.show()
```



```
# Scatter Plot
plt.figure(figsize=(10, 5))
plt.scatter(df['DAYS'], df['ENFIELD'], label='ENFIELD', marker='o')
plt.scatter(df['DAYS'], df['HONDA'], label='HONDA', marker='o')
plt.scatter(df['DAYS'], df['YAHAMA'], label='YAHAMA', marker='o')
plt.scatter(df['DAYS'], df['KTM'], label='KTM', marker='o')
plt.xlabel('Days')
plt.ylabel('Distance Covered (KMS)')
plt.title('Scatter Plot - Distance Covered by Bikes')
plt.legend()
plt.grid(True)
plt.show()
```




```
plt.figure(figsize=(10, 5))
width,x = 0.2, range(len(df['DAYS']))
plt.bar(x, df['ENFIELD'], width=width, label='ENFIELD')
plt.bar([i + width for i in x], df['HONDA'], width=width, label='HONDA')
plt.bar([i + 2 * width for i in x], df['YAHAMA'], width=width, label='YAHAMA')
plt.bar([i + 3 * width for i in x], df['KTM'], width=width, label='KTM')
plt.ylabel('Distance Covered (KMS)')
plt.title('Bar Plot - Distance Covered by Bikes')
plt.xticks([i + 1.5 * width for i in x], df['DAYS'])
plt.legend()
plt.grid(True)
plt.show()
```



DATA SCIENCE

Laboratory Number - 10

EXPERIMENT:- Solve a numerical problem on Normal Distribution using python.

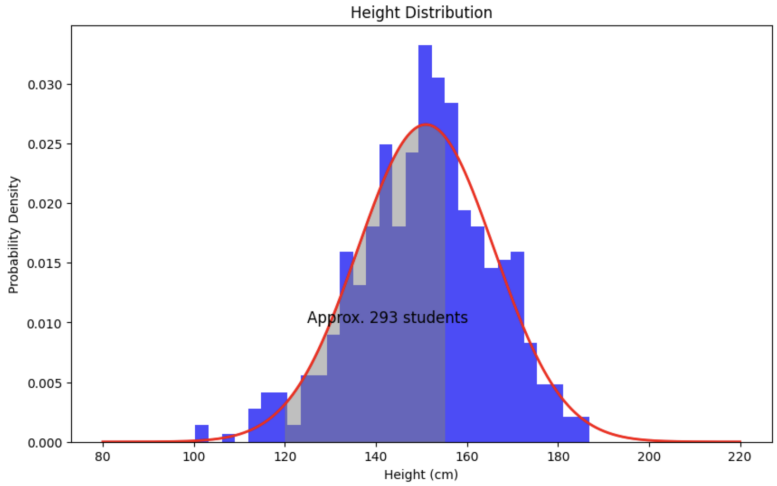
PROGRAM -

Q1. The mean height of 500 students is 151 cm and SD is 15 cm. Assuming that the heights are normally distributed, find how many student heights lie between 120 cm and 155 cm ?

```
import numpy as np
from scipy.stats import norm

mean = 151
sd = 15
n = 500

# P(120 < X < 155) = P(X < 155) - P(X < 120)

z1 = (120 - mean) / sd
z2 = (155 - mean) / sd

print("P(120 < X < 155) = P(X < 155) - P(X < 120)")
print("P(120 < X < 155) = P(Z < {:.2f}) - P(Z < {:.2f})".format(z2, z1))

p1 = norm.cdf(z1)
p2 = norm.cdf(z2)

print("P(120 < X < 155) = {:.4f} - {:.4f}".format(p2, p1))
print("P(120 < X < 155) = {:.4f}".format(p2 - p1))

print("Number of students height lie between 120 and 155cm is {:.0f}".format((p2 - p1) * n))
```

```
P(120 < X < 155) = P(X < 155) - P(X < 120)
P(120 < X < 155) = P(Z < 0.27) - P(Z < -2.07)
P(120 < X < 155) = 0.6051 - 0.0194
P(120 < X < 155) = 0.5858
Number of students height lie between 120 and 155cm is 293
```

```
x = np.linspace(80, 220, 1000)
pdf = norm.pdf(x, mean, sd)

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
plt.hist(np.random.normal(mean, sd, n), bins=30, density=True, alpha=0.7, color='blue')

plt.plot(x, pdf, color='red', lw=2)
plt.fill_between(x, 0, pdf, where=(x >= 120) & (x <= 155), color='gray', alpha=0.5)

plt.xlabel('Height (cm)')
plt.ylabel('Probability Density')
plt.title('Height Distribution')

plt.text(125, 0.01, f'Approx. {round((p2-p1) * n)} students', fontsize=12, color='black')
plt.show()
```



Q2. X is a normal variate with mean 30 and SD 5 find the probability that

a)26<=X<=40

b) X>=45

```
mean = 30
sd = 5

# a) P(26 <= X <= 40) = P(X <= 40) - P(X <= 26)
z1 = (26 - mean) / sd
z2 = (40 - mean) / sd
print("P(26 <= X <= 40) = P(X <= 40) - P(X <= 26)")
print("P(26 <= X <= 40) = P(Z <= {:.2f}) - P(Z <= {:.2f})".format(z2, z1))

p1 = norm.cdf(z1)
p2 = norm.cdf(z2)
print("P(26 <= X <= 40) = {:.4f} - {:.4f}".format(p2, p1))
print("P(26 <= X <= 40) = {:.4f}".format(p2 - p1))

# b) P(X >= 45) = 1 - P(X <= 45)
z = (45 - mean) / sd
print("P(X >= 45) = 1 - P(X <= 45)")
print("P(X >= 45) = 1 - P(Z <= {:.2f})".format(z))

p = norm.cdf(z)
print("P(X >= 45) = 1 - {:.4f}".format(p))
print("P(X >= 45) = {:.4f}".format(1 - p))
```

```
P(26 <= X <= 40) = P(X <= 40) - P(X <= 26)
P(26 <= X <= 40) = P(Z <= 2.00) - P(Z <= -0.80)
P(26 <= X <= 40) = 0.9772 - 0.2119
P(26 <= X <= 40) = 0.7654
P(X >= 45) = 1 - P(X <= 45)
P(X >= 45) = 1 - P(Z <= 3.00)
P(X >= 45) = 1 - 0.9987
P(X >= 45) = 0.0013
```

```python
x =  np.linspace(0, 60, 1000)
pdf = norm.pdf(x, mean, sd)
plt.figure(figsize=(10, 6))
plt.plot(x, pdf, color='red', lw=2)
plt.fill_between(x, 0, pdf, where=(x >= 26) & (x <= 40), color='gray', alpha=0.5)
plt.fill_between(x, 0, pdf, where=(x >= 45), color='green', alpha=0.5)
plt.xlabel('X')
plt.ylabel('Probability Density')
plt.title('Normal Distribution')
plt.text(30, 0.01, f'{round(p2-p1, 4)}', fontsize=12, color='black')
plt.text(45, 0.01, f'{round(1-p, 4)}', fontsize=12, color='black')
plt.show()
```