Index

Practical No :1

Aim: Implementation of Data Preprocessing Techniques [Handling Missing Data, Encoding and Normalization,Feature Scaling]

Theory:
Data preprocessing is a crucial step in the data mining process that transforms raw data into an understandable and usable format.Here is a theoretical explanation of the techniques you mentioned.

1. Handling Missing Data
Missing data occurs when no value is stored for a feature (attribute) in an observation (row). Dealing with it is essential because many machine learning algorithms cannot handle missing values, or their performance will be severely degraded.

The common strategies for handling missing data are:
*** Imputation: Filling in the missing values with a substituted value.**

*** Mean/Median Imputation: Replacing missing values in a numerical column with the mean (good for normally distributed data) or median (more robust to outliers) of that column.**

*** Mode Imputation: Replacing missing values in a categorical column with the mode (most frequent value) of that column.**

*** Advanced Imputation: Using more sophisticated methods like K-Nearest Neighbors (KNN) imputation, which estimates a missing value based on the values of its nearest neighbors, or using a predictive model (like linear regression) to predict the missing values.**

*** Deletion: Removing observations (rows) or features (columns) that contain missing values.**

* Row Deletion (Listwise Deletion): Removing the entire observation if one or more values are missing. This is suitable when the amount of missing data is small, but it can
lead to a significant loss of information if many rows have missing data.

* Column Deletion: Removing a feature entirely if a large percentage of its values are missing. This is only done when the feature is deemed non-essential or too sparse to be
Useful.

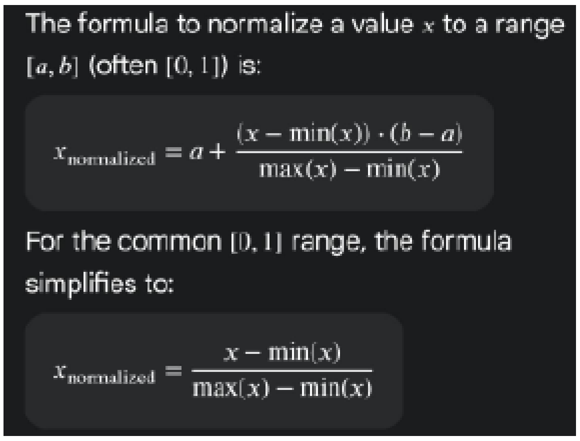
2. Encoding and Normalization
Encoding Categorical Data
Encoding is the process of converting categorical data (data that represents categories, like 'Color' or 'City') into a numerical format that machine learning algorithms can process.

* Ordinal Encoding: Used for categorical features where the order matters (e.g., "Low",'Medium', 'High'). The categories are mapped to integer values (e.g., 1, 2, 3).

* One-Hot Encoding: Used for nominal categorical data where no order exists (e.g., 'Red','Blue', 'Green'). It creates a new binary feature (column) for each unique category. An observation gets a '1' in the column corresponding to its category and '0' in all others. This prevents the model from assuming an arbitrary ordinal relationship between categories.

Normalization (Min-Max Scaling)

Normalization, often referred to as Min-Max Scaling, rescales the features to a fixed range, typically between 0 and 1. This is particularly useful for algorithms that depend on distance measures (like KNN or K-Means).



The formula to normalize a value x to a range $[a, b]$ (often $[0, 1]$) is:

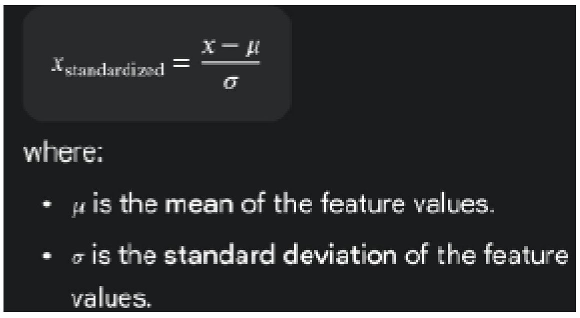
$$x_{\text{normalized}} = a + \frac{(x - \min(x)) \cdot (b - a)}{\max(x) - \min(x)}$$

For the common $[0, 1]$ range, the formula simplifies to:

$$x_{\text{normalized}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

3. Feature Scaling
Feature scaling is a method used to standardize or normalize the range of independent features (variables). If the range of values in features varies widely, the feature with the largest range might dominate the learning process, which is generally undesirable. Scaling ensures that all features contribute proportionally to the result.

Standardization (Z-Score Normalization)
Standardization (or Z-score normalization) transforms the data such that the resulting
distribution has a mean of 0 and a standard deviation of 1. This is particularly effective
when the data follows a Gaussian (normal) distribution.
The formula for standardization is



$$x_{\text{standardized}} = \frac{x - \mu}{\sigma}$$

where:

- μ is the mean of the feature values.
- σ is the standard deviation of the feature values.

Code & Output:

```
import pandas as pd

import numpy as np

data= { 'Name':[ 'Ammar','hafeez', 'hurara ', 'wasim'],

    'Age': [21, 30, 24, np.nan],

    'Marks': [ np.nan, 90, 90, 90],

    'City': [ 'Mumbai', 'Pune', 'Bangalore', np.nan] }

df= pd.DataFrame(data)

df.head()
```

```
Name        Age   Marks       City

0      Ammar       21.0  NaN  Mumbai

1      hafeez      30.0  90.0  Pune

2      hurara      24.0  90.0  Bangalore

3      wasim       NaN  90.0  NaN
```

```
df_drop=df.dropna()

df_drop.head()
```

```
Name        Age   Marks       City

1      hafeez      30.0  90.0  Pune

2      hurara      24.0  90.0  Bangalore
```

```
df_drop_col=df.dropna(axis=1)
```

```
df_drop_col.head()
```

```
Name
0       Ammar
1       hafeez
2       hurara
3       Wasim
```

```
df['Age']=df['Age'].fillna(df['Age'].mean())
df['Marks']=df['Marks'].fillna(df['Marks'].median())
df['City']=df['City'].fillna(df['City'].mode())
df.head()
```

```
Name        Age    Marks        City
0       Ammar       21.0   90.0   Mumbai
1       hafeez      30.0   90.0   Pune
2       hurara      24.0   90.0   Bangalore
3       wasim       25.0   90.0   NaN
```

```
df_ffill=df.fillna(method='ffill')
df_ffill.head()
```

```
/tmp/ipython-input-1175066473.py:1: FutureWarning: DataFrame.fillna
with 'method' is deprecated and will raise in a future version. Use
obj.ffill() or obj.bfill() instead.

  df_ffill=df.fillna(method='ffill')

Name        Age   Marks       City

0      Ammar      21.0  90.0  Mumbai

1      hafeez     30.0  90.0  Pune

2      hurara     24.0  90.0  Bangalore

3      wasim      25.0  90.0  Bangalore
```

4) Encoding Categorical Data **

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, StandardScaler,
RobustScaler, MaxAbsScaler


df = pd.read_csv('diabetes.csv')

df.head()
```

```
Pregnancies      Glucose     BloodPressure    SkinThickness    Insulin
       BMI   DiabetesPedigreeFunction    Age   Outcome

0      6      148    72      35      0      33.6  0.627 50      1
```

```
1    1    85    66    29    0     26.6  0.35131    0

2    8    183   64    0     0     23.3  0.67232    1

3    1    89    66    23    94    28.1  0.16721    0

4    0    137   40    35    168   43.1  2.28833    1
```

```
from google.colab import auth

auth.authenticate_user()


from google.colab import drive

drive.mount('/content/drive')


import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, StandardScaler,
RobustScaler, MaxAbsScaler


# Load data

df = pd.read_csv("diabetes.csv")


# Scalers

min_scaler = MinMaxScaler()

standard_scaler = StandardScaler()

rob_scaler = RobustScaler()
```

```
mabs_scaler = MaxAbsScaler()

# Fit Transform
min_df = min_scaler.fit_transform(df)
standard_df = standard_scaler.fit_transform(df)
rob_df = rob_scaler.fit_transform(df)
mabs_df = mabs_scaler.fit_transform(df)

# Print results
print("Min-Max Scaled Data:\n", min_df)
print("\nStandard Scaled Data:\n", standard_df)
print("\nRobust Scaled Data:\n", rob_df)
print("\nMaxAbs Scaled Data:\n", mabs_df)
print("\nPractical 1 Done by Devesh Vijay Patil")
```

```
Min-Max Scaled Data:
 [[0.35294118 0.74371859 0.59016393 ... 0.23441503 0.48333333 1.
]
 [0.05882353 0.42713568 0.54098361 ... 0.11656704 0.16666667 0.
]
 [0.47058824 0.91959799 0.52459016 ... 0.25362938 0.18333333 1.
]
 ...
 [0.29411765 0.6080402  0.59016393 ... 0.07130658 0.15       0.        ]
 [0.05882353 0.63316583 0.49180328 ... 0.11571307 0.43333333 1.
]
 [0.05882353 0.46733668 0.57377049 ... 0.10119556 0.03333333 0.
]]


Standard Scaled Data:
 [[ 0.63994726  0.84832379  0.14964075 ...  0.46849198  1.4259954
   1.36589591]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.36506078 -0.19067191
  -0.73212021]
 [ 1.23388019  1.94372388 -0.26394125 ...  0.60439732 -0.10558415
   1.36589591]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.68519336 -0.27575966
  -0.73212021]
 [-0.84488505  0.1597866  -0.47073225 ... -0.37110101  1.17073215
   1.36589591]
 [-0.84488505 -0.8730192   0.04624525 ... -0.47378505 -0.87137393
  -0.73212021]]


Robust Scaled Data:
 [[ 0.6         0.75151515  0.         ...  0.66535948  1.23529412
   1.        ]
 [-0.4        -0.77575758 -0.33333333 ... -0.05620915  0.11764706
```

```
  0.        ]
 [ 1.          1.6        -0.44444444 ...  0.78300654  0.17647059
   1.        ]
 ...
 [ 0.4         0.0969697   0.         ... -0.33333333  0.05882353
   0.        ]
 [-0.4         0.21818182 -0.66666667 ... -0.06143791  1.05882353
   1.        ]
 [-0.4        -0.58181818 -0.11111111 ... -0.1503268  -0.35294118
   0.        ]]
```

```
MaxAbs Scaled Data:
 [[0.35294118 0.74371859 0.59016393 ... 0.25909091 0.61728395 1.
]
 [0.05882353 0.42713568 0.54098361 ... 0.14504132 0.38271605 0.
]
 [0.47058824 0.91959799 0.52459016 ... 0.27768595 0.39506173 1.
]
 ...
 [0.29411765 0.6080402  0.59016393 ... 0.10123967 0.37037037 0.
]
 [0.05882353 0.63316583 0.49180328 ... 0.14421488 0.58024691 1.
]
 [0.05882353 0.46733668 0.57377049 ... 0.13016529 0.28395062 0.
]]
```

Practical 1 Done by Devesh Vijay Patil

3) Feature Scaling ****

```
import pandas as pd

from sklearn.preprocessing import LabelEncoder

data = {'color': ['Red', 'Blue', 'Pink', 'Green', 'Pink', 'Blue']}

df = pd.DataFrame(data)

encoder = LabelEncoder()

df['encoded_color'] = encoder.fit_transform(df['color'])


print(df)
```

```
color  encoded_color
0    Red              3
1   Blue             0
2   Pink             2
3  Green              1
4   Pink             2
5   Blue             0
```

```
df_onehot=pd.get_dummies(df,columns=['color'])

df_onehot.head()
```

```
encoded_color    color_Blue  color_Green        color_Pink  color_Red
```

```
0       3       False False False True
1       0       True  False False False
2       2       False False True  False
3       1       False True  False False
4       2       False False True  False
```

```
import pandas as pd
data = {'color': ['Red', 'Blue', 'Pink', 'Green', 'Pink', 'Blue']}
df = pd.DataFrame(data)

# Frequency encoding
freq_map = df['color'].value_counts().to_dict()
df['color_Encoded'] = df['color'].map(freq_map)
print(df)
print("Practical 1 Done by Devesh Vijay Patil")
```

```
color  color_Encoded
0    Red              1
1   Blue              2
2   Pink              2
3  Green              1
4   Pink              2
5   Blue              2
```

Practical 1 Done by Devesh Vijay Patil

Practical No: 2

Aim: Implementation of Linear Regression with Performance Evaluation Metrics

Theory:

Linear Regression is a type of Supervised Machine-learning algorithm that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets. It assumes that there is a linear relationship between the input and output, meaning the output changes at a constant rate as the input changes. This relationship is represented by a straight line.

For example : we want to predict a student's exam score based on how many hours they studied. We observe that as students study more hours, their scores go up. In the example of predicting exam scores based on hours studied. Here

- Independent Variable (input): Hours studied because it’s the factor we control or observe.
- Dependent variable (output): Exam score because it depends on how many hours were studied.

Evaluation Metrics for Linear Regression:

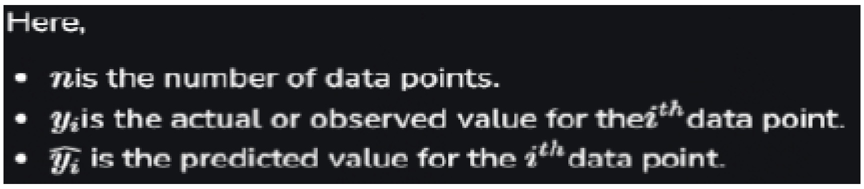
A variety of evaluation measures can be used to determine the strength of any linear regression model. These assessment metrics often give an indication of how well the model is producing the observed outputs.

The most common measurements are:

1. Mean Square Error (MSE)

Mean Squared Error (MSE) is an evaluation metric that calculates the average of the squared differences between the actual and predicted values for all the data points. The difference is squared to ensure that negative and positive differences don't cancel each other out.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \widehat{y_i}\right)^2$$

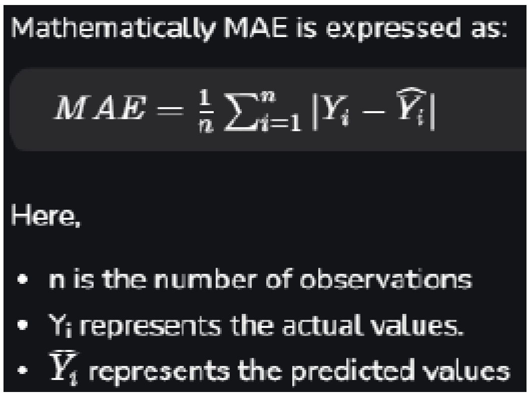


Here,

- $n$is the number of data points.
- $y_i$is the actual or observed value for thei^{th}data point.
- $\widehat{y_i}$ is the predicted value for the i^{th} data point.

2. Mean Absolute Error (MAE)

Mean Absolute Error is an evaluation metric used to calculate the accuracy of a regression model. MAE measures the average absolute difference between the predicted values and actual values.



Mathematically MAE is expressed as:

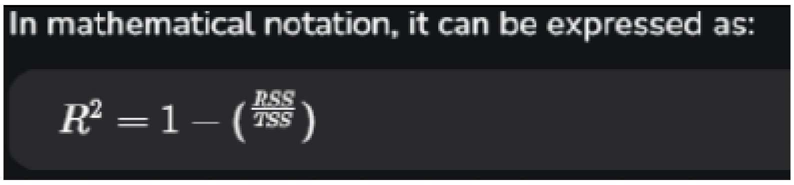
$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \widehat{Y_i}|$$

Here,

- n is the number of observations
- Y_i represents the actual values.
- $\overline{Y}_1$ represents the predicted values

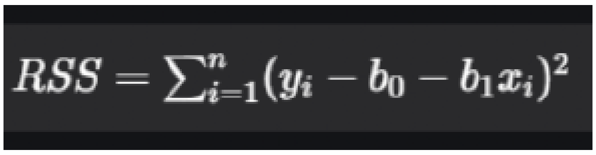
3.Coefficient of Determination (R-squared)

R-Squared is a statistic that indicates how much variation the developed model can explain or capture. It is always in the range of 0 to 1. In general, the better the model matches the data, the greater the R-squared number.

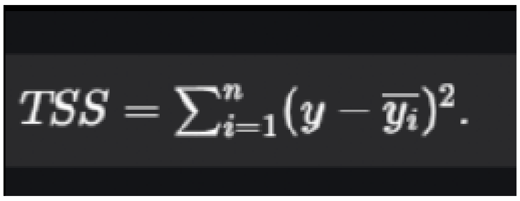


In mathematical notation, it can be expressed as:

$$R^2 = 1 - (\frac{RSS}{TSS})$$



$$RSS = \sum_{i=1}^{n}(y_i - b_0 - b_1 x_i)^2$$



$$TSS = \sum_{i=1}^{n}(y - \overline{y_i})^2.$$

Advantages.

- Linear regression is computationally efficient and can handle large datasets effectively. It can be trained quickly on large datasets, making it suitable for real-time applications.

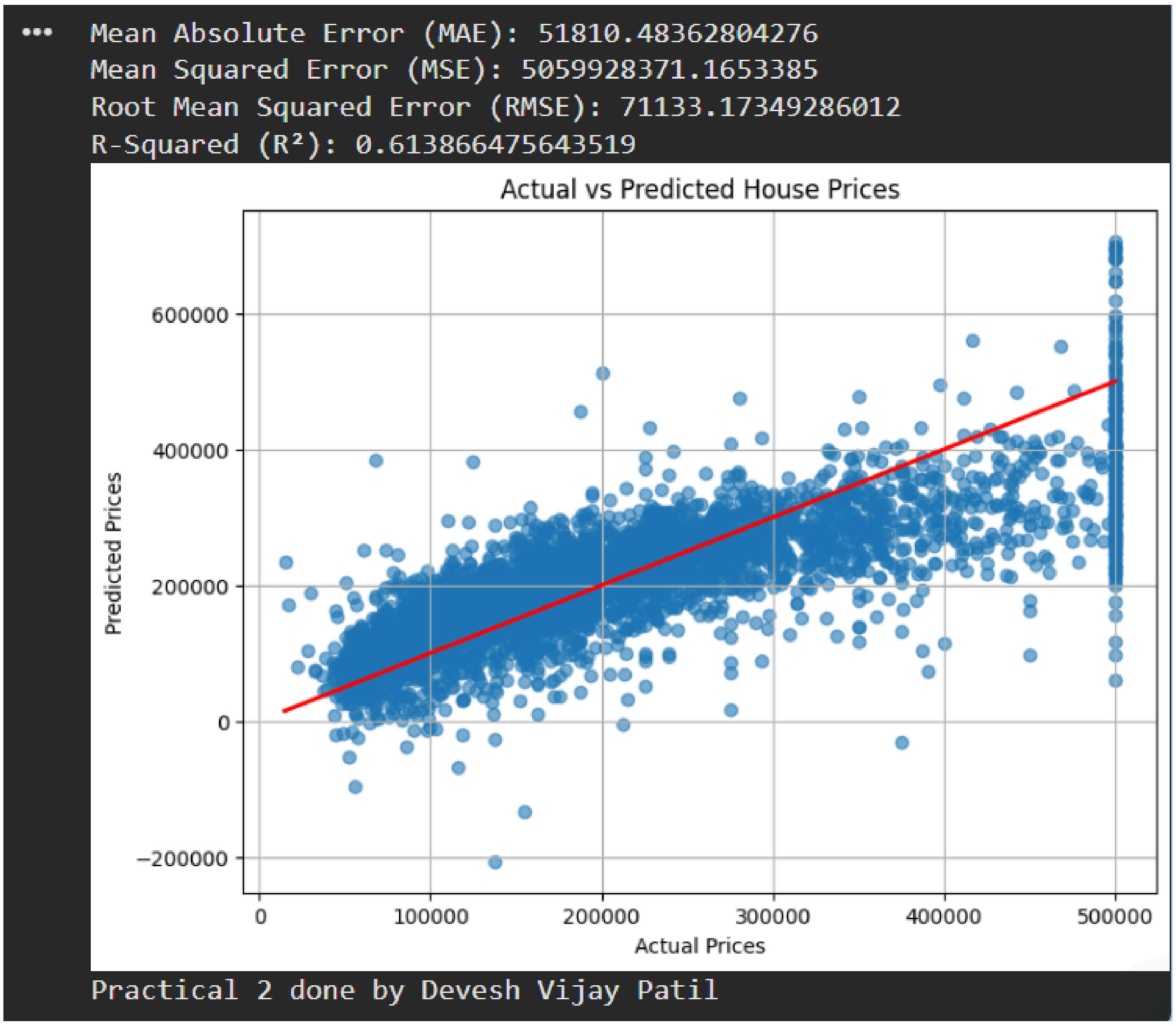
Disadvantages

- Linear regression assumes a linear relationship between the dependent and independent variables. If the relationship is not linear, the model may not perform well.

Code & Output:

```
# Importing required libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import matplotlib.pyplot as plt


# Load dataset (CSV version)

url = "https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv"

df = pd.read_csv(url)


# Remove categorical column

df = df.drop("ocean_proximity", axis=1)


# Handle missing values (IMPORTANT STEP)

df = df.fillna(df.median())
```

```
# Split features and target
X = df.drop("median_house_value", axis=1)
y = df["median_house_value"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Print results
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)

print("Root Mean Squared Error (RMSE):", rmse)

print("R-Squared (R²):", r2)


# Plot Actual vs Predicted values

plt.figure(figsize=(8, 6))

plt.scatter(y_test, y_pred, alpha=0.6)

plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linewidth=2)

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual vs Predicted House Prices")

plt.grid(True)

plt.show()


print("Practical 2 done by Devesh Vijay Patil")
```



```
Mean Absolute Error (MAE): 51810.48362804276
Mean Squared Error (MSE): 5059928371.1653385
Root Mean Squared Error (RMSE): 71133.17349286012
R-Squared (R²): 0.613866475643519
```

```
Practical 2 done by Devesh Vijay Patil
```

import numpy as np

y_true = [2, 3, 1, 4, 5.2]

y_pred = [2, 3, 6, 3, 5.0]

# Convert to numpy arrays

y_true = np.array(y_true)

y_pred = np.array(y_pred)

# Difference

d = y_true - y_pred

# Manual calculations

mae_manual = np.mean(np.abs(d))

mse_manual = np.mean(d**2)

rmse_manual = np.sqrt(mse_manual)

# R² manual

r2_manual = 1 - (np.sum(d**2) / np.sum((y_true - np.mean(y_true))**2))

# Print results

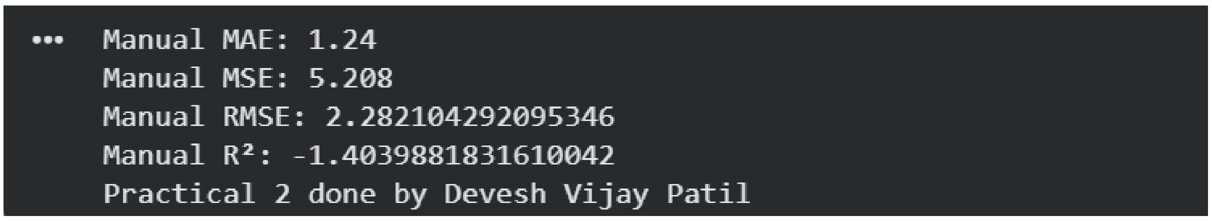
print("Manual MAE:", mae_manual)

print("Manual MSE:", mse_manual)

print("Manual RMSE:", rmse_manual)

print("Manual R²:", r2_manual)

print("Practical 2 done by Devesh Vijay Patil")



```
Manual MAE: 1.24
Manual MSE: 5.208
Manual RMSE: 2.282104292095346
Manual R²: -1.4039881831610042
Practical 2 done by Devesh Vijay Patil
```

Or

Manual MAE: 1.24

Manual MSE: 5.208

Manual RMSE: 2.282104292095346

Manual R²: -1.4039881831610042

Practical 2 done by Devesh Vijay Patil

Conclusion: Implementing Linear Regression on the California housing dataset demonstrates that the model is able to identify a general linear relationship between the actual price and the predicted price. The evaluation metrics suggest that MAE deviates from the actual house prices by 0.53 units while the MSE and RMSE values show the model predicts well but still some variance between predicted and actual values R2 shows that the model explains about 57.6% of the variation.

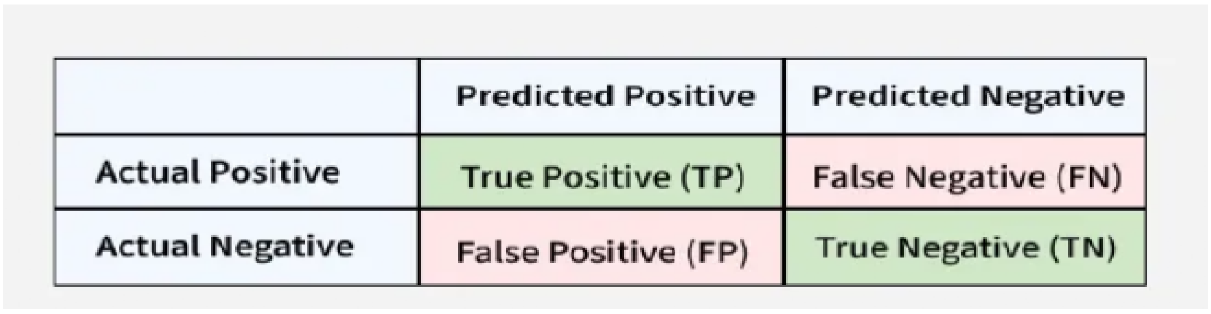
Practical No: 3

Aim: Implementation of Logistic Regression with Evaluation using Confusion Matrix

Theory:

Logistic Regression is a supervised machine learning algorithm used for classification problems. Unlike linear regression which predicts continuous values it predicts the probability that an input belongs to a specific class. It is used for binary classification where the output can be one of two possible categories such as Yes/No, True/False or 0/1. It uses sigmoid function to convert inputs into a probability value between O and 1. In this article, we will see the basics of logistic regression and its core concepts. Logistic regression can be classified into three main types based on the nature of the dependent variable:Binomial Logistic Regression: This type is used when the dependent variable has only two possible categories. Examples include Yes/No, Pass/Fail or 0/1. It is the most common form of logistic regression and is used for binary classification problems.Multinomial Logistic Regression: This is used when the dependent variable has three or more possible categories that are not ordered. For example, classifying animals into categories like "cat," "dog" or "sheep." It extends the binary logistic regression to handle multiple classes.Ordinal Logistic Regression: This type applies when the dependent variable has three or more categories with a natural order or ranking. Examples include ratings like "low," "medium" and "high." It takes the order of the categories into account when modeling.

Confusion matrix is a simple table used to measure how well a classification model is performing. It compares the predictions made by the model with the actual results and shows where the model was right or wrong. This helps you understand where the model is making mistakes so you can improve it. It breaks down the predictions into four categories: True Positive (TP): The model correctly predicted a positive outcome i.e the actual outcome was positive. True Negative (TN): The model correctly predicted a negative outcome i.e the actual outcome was negative.False Positive (FP): The model incorrectly predicted a positive outcome i.e the actual outcome was negative. It is also known as a Type I error. False Negative (FN): The model incorrectly predicted a negative outcome i.e the actual outcome was positive. It is also known as a Type II error.



	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Code & Output:

```
import pandas as pd
pima = pd.read_csv("diabetes.csv")
pima.head()
```

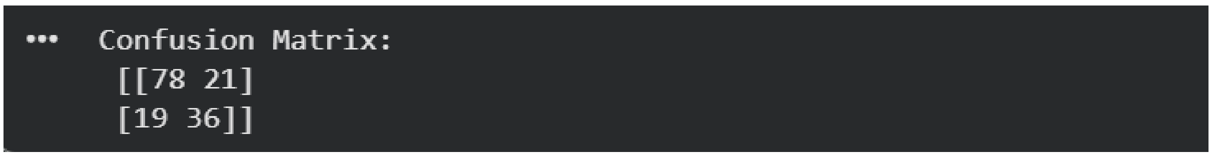
```
Pregnancies      Glucose     BloodPressure    SkinThickness    Insulin
        BMI   DiabetesPedigreeFunction    Age   Outcome
0       6       148   72      35      0       33.6  0.62750     1
1       1       85    66      29      0       26.6  0.35131     0
2       8       183   64      0       0       23.3  0.67232     1
3       1       89    66      23      94      28.1  0.16721     0
4       0       137   40      35      168     43.1  2.28833     1
```

```
# Split dataset into features and target variable
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age',
             'Glucose', 'BloodPressure', 'DiabetesPedigreeFunction']

X = pima[feature_cols]      # Features
y = pima['Outcome']        # Target variable

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

```
# Initialize model
logreg = LogisticRegression(max_iter=1000)

# Train model
logreg.fit(X_train, y_train)

# Predict on test set
y_pred = logreg.predict(X_test)

# Confusion Matrix
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cnf_matrix)
```

```
Confusion Matrix:
 [[78 21]
 [19 36]]
```

Or

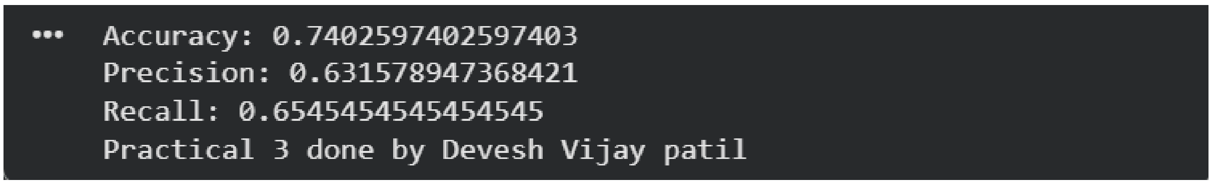


```
••• Confusion Matrix:
     [[78 21]
     [19 36]]
```

```
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(y_test, y_pred))
print("Recall:", metrics.recall_score(y_test, y_pred))

print("Practical 3 done by Devesh Vijay patil")
```

```
Accuracy: 0.7402597402597403
Precision: 0.631578947368421
Recall: 0.6545454545454545
Practical 3 done by Devesh Vijay patil
```

Or



```
••• Accuracy: 0.7402597402597403
    Precision: 0.631578947368421
    Recall: 0.6545454545454545
    Practical 3 done by Devesh Vijay patil
```

Conclusion: Implementation of the Logistic Regression and evaluating the model through a confusion matrix has shown that the model has correctly predicted 141 [True Negatives] ,16 [False Positive], 35[False Negatives],39 [True Positives]. The model has more correct predictions than wrong predictions.

Practical No: 4

Aim: Implementation of polynomial Regression

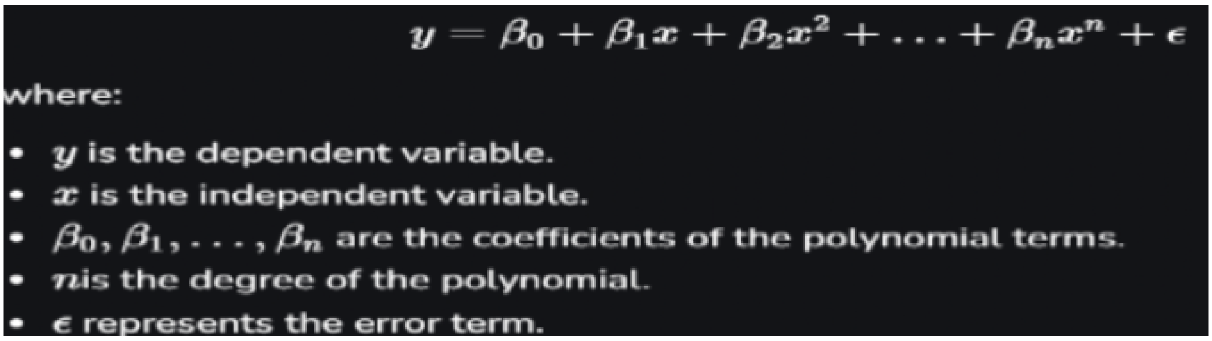
Theory:

Polynomial Regression is a form of linear regression where the relationship between the independent variable (x) and the dependent variable (y) is modelled as an nth degree polynomial. It is useful when the data exhibits a non-linear relationship allowing the model to fit a curve to the data.

Need for Polynomial Regression

- Non-linear Relationships: Polynomial regression is used when the relationship between the independent variable (input) and dependent variable (output) is non-linear. Unlike linear regression which fits a straight line, it fits a polynomial equation to capture the curve in the data.
- Better Fit for Curved Data: When a researcher hypothesizes a curvilinear relationship,polynomial terms are added to the model. A linear model often results in residuals with noticeable patterns which shows a poor fit. It can capture these non-linear patterns effectively.

Polynomial regression is an extension of linear regression where higher-degree terms are added to model non-linear relationships. The general form of the equation for a polynomial regression of degree n is:

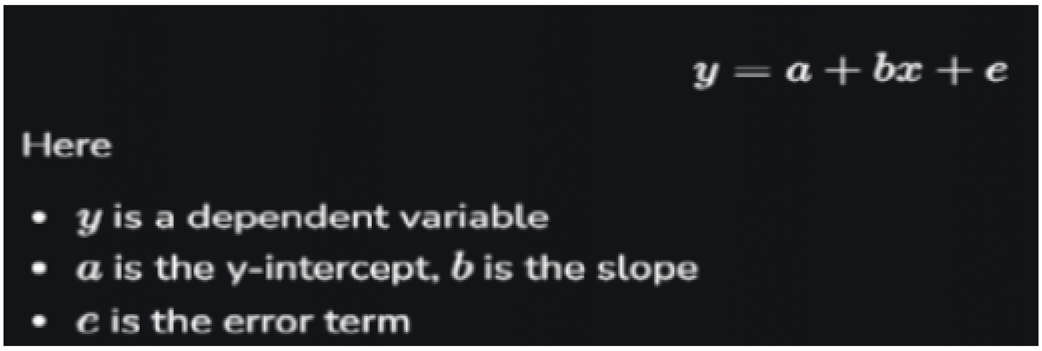


$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \epsilon$$

where:

- y is the dependent variable.
- x is the independent variable.
- $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients of the polynomial terms.
- $n$is the degree of the polynomial.
- ϵ represents the error term.

The goal of regression analysis is to model the expected value of a dependent variable y in terms of an independent variable x. In simple linear regression, this relationship is modeled as:



$$y = a + bx + e$$

Here

- y is a dependent variable
- a is the y-intercept, b is the slope
- e is the error term

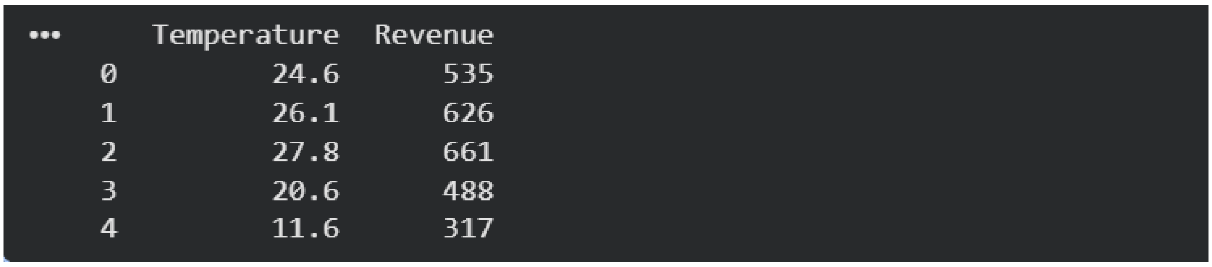
However in cases where the relationship between the variables is nonlinear such as modelling chemical synthesis based on temperature, a linear model may not be sufficient. Instead, we use polynomial regression which introduces higher-degree terms such as X2 to better capture the relationship.

Code & Output:

```
# Importing all packages at start
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Load CSV file
data = pd.read_csv('/content/Ice Cream.csv')

# Display first few rows
print(data.head())
```

```
Temperature  Revenue
0        24.6      535
1        26.1      626
2        27.8      661
3        20.6      488
4        11.6      317
```

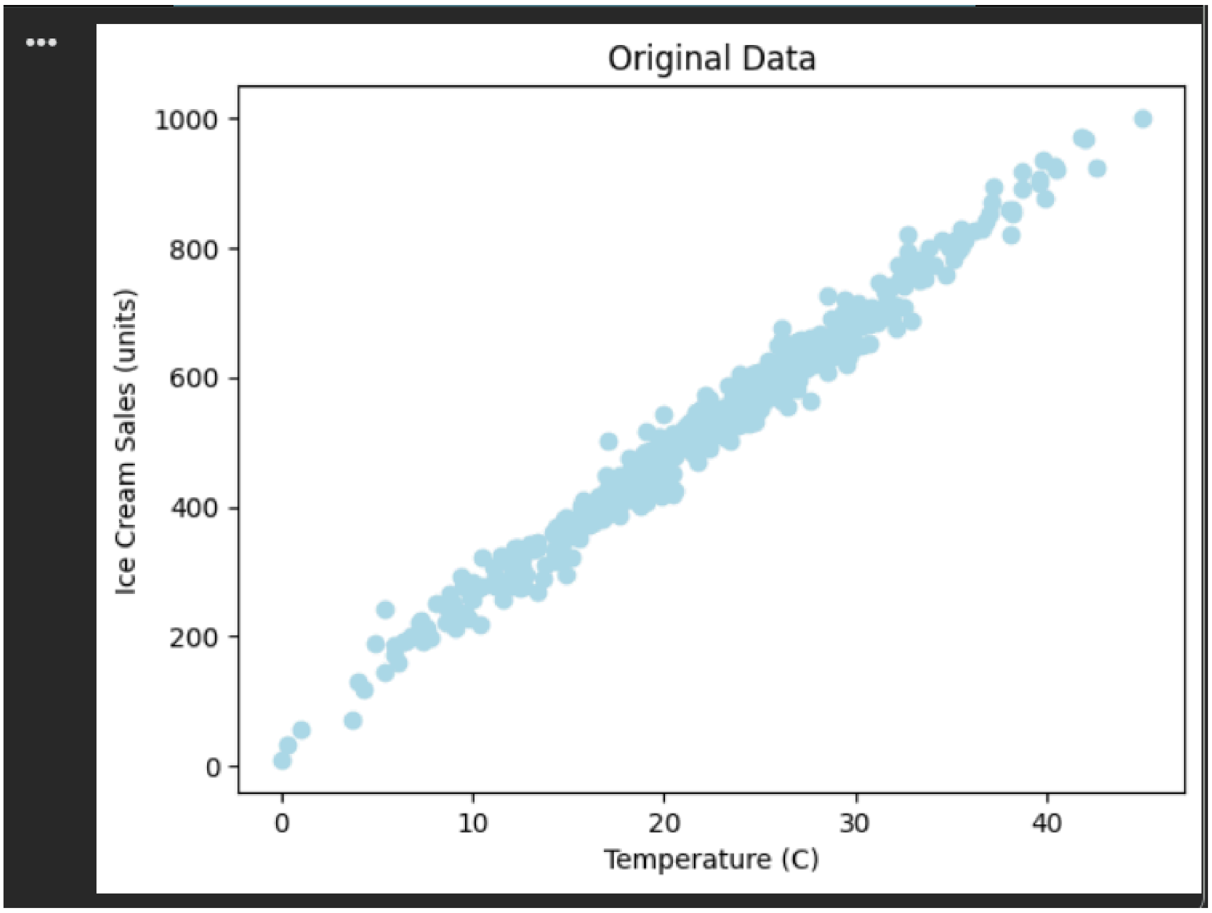
Or



```
•••     Temperature  Revenue
    0          24.6      535
    1          26.1      626
    2          27.8      661
    3          20.6      488
    4          11.6      317
```

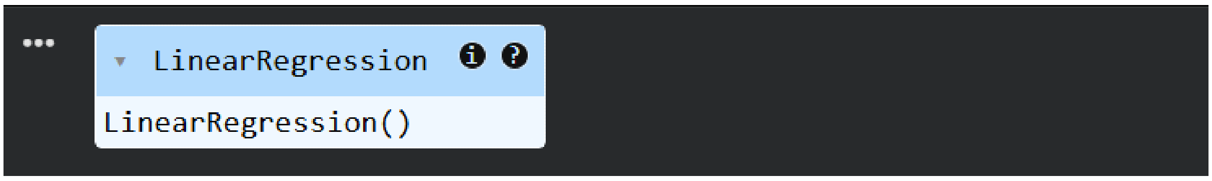
```
# Select features and target
X = data.iloc[:, 0].values.reshape(-1, 1)   # Temperature column
```

```
y = data.iloc[:, 1].values              # Ice Cream Sales column

# Visualize the original data points
plt.scatter(X, y, color="lightblue")  # You can change color

plt.title("Original Data")
plt.xlabel("Temperature (C)")
plt.ylabel("Ice Cream Sales (units)")
plt.show()
```



```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Create polynomial features (degree 2)
poly_features = PolynomialFeatures(degree=2)

# Transform X into polynomial form
X_poly = poly_features.fit_transform(X)
```

```
# Create linear regression model
pr_model = LinearRegression()

# Fit model on polynomial features
pr_model.fit(X_poly, y)
```

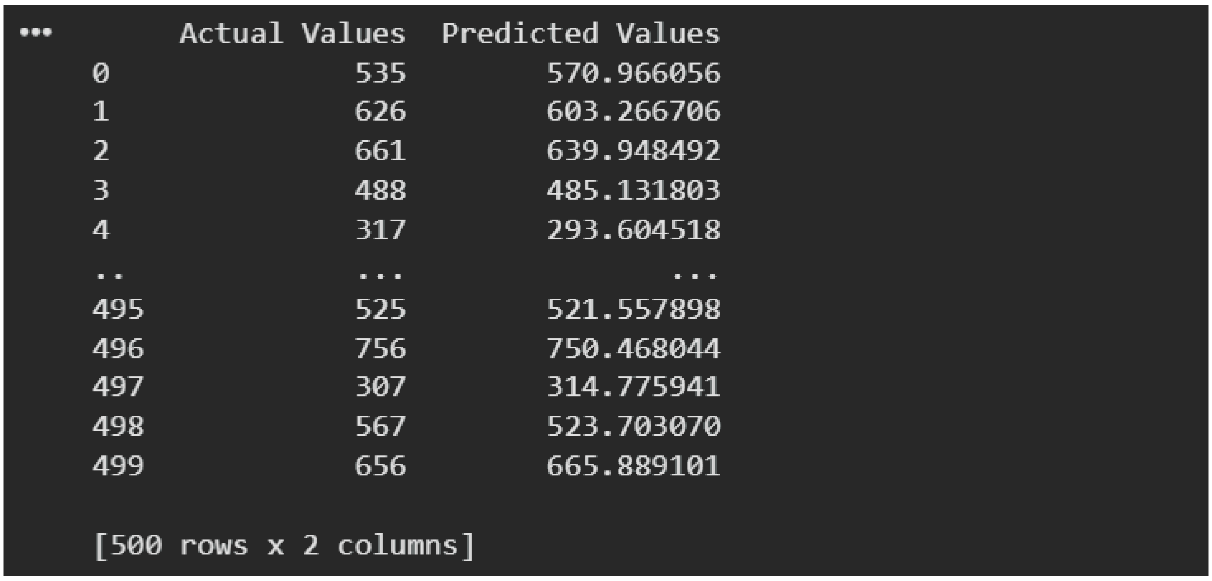


```
LinearRegression
LinearRegression()
```

```
# Predict values using polynomial regression model
y_pred = pr_model.predict(X_poly)

# Create a dataframe of actual vs predicted values
actual_predicted = pd.DataFrame({
    'Actual Values': y,
    'Predicted Values': y_pred
})

print(actual_predicted)
```

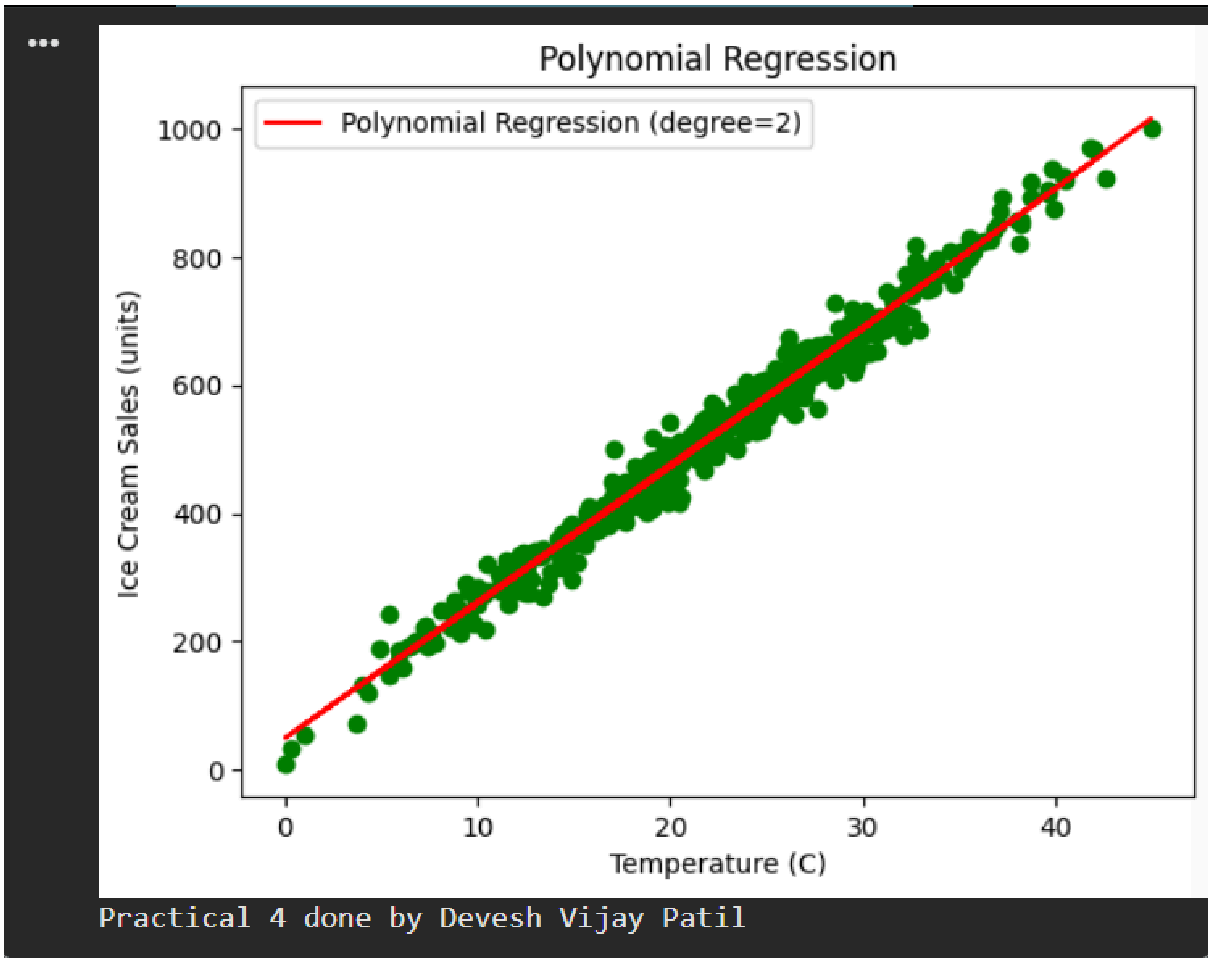
```
Actual Values  Predicted Values
0            535       570.966056
1            626       603.266706
2            661       639.948492
3            488       485.131803
4            317       293.604518
..           ...              ...
495            525       521.557898
496            756       750.468044
497            307       314.775941
498            567       523.703070
499            656       665.889101

[500 rows x 2 columns]
```

Or



```
•••       Actual Values  Predicted Values
    0               535        570.966056
    1               626        603.266706
    2               661        639.948492
    3               488        485.131803
    4               317        293.604518
    ..              ...               ...
    495             525        521.557898
    496             756        750.468044
    497             307        314.775941
    498             567        523.703070
    499             656        665.889101

    [500 rows x 2 columns]
```

```
# Visualize the polynomial regression results
plt.scatter(X, y, color="green")

# Plot polynomial regression curve
plt.plot(X, y_pred, color='red', label='Polynomial Regression (degree=2)')

plt.xlabel("Temperature (C)")
plt.ylabel("Ice Cream Sales (units)")
plt.legend()
plt.title('Polynomial Regression')
plt.show()

print("Practical 4 done by Devesh Vijay Patil")
```



```
Practical 4 done by Devesh Vijay Patil
```

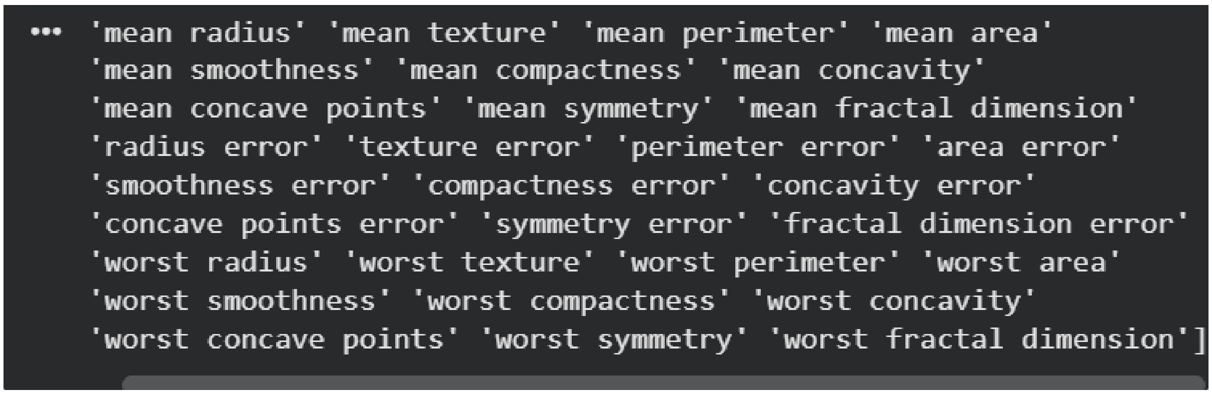
```
# Predict real value or unseen data
X_new = np.array([[1.9929]])  # Example temperature value to predict

# Transform new input into polynomial form
X_new_poly = poly_features.transform(X_new)

# Predict using the polynomial regression model
y_new_pred = pr_model.predict(X_new_poly)

print("Predicted Ice Cream Sales:", y_new_pred)
```

```
Predicted Ice Cream Sales: [91.60190947]
```

```
# Step 1: Import Libraries
```

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
RobustScaler
from sklearn.datasets import load_breast_cancer

# Load the breast cancer dataset
cancer = load_breast_cancer()

# Print feature names
print(cancer.feature_names)
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```



```
••• 'mean radius' 'mean texture' 'mean perimeter' 'mean area'
    'mean smoothness' 'mean compactness' 'mean concavity'
    'mean concave points' 'mean symmetry' 'mean fractal dimension'
    'radius error' 'texture error' 'perimeter error' 'area error'
    'smoothness error' 'compactness error' 'concavity error'
    'concave points error' 'symmetry error' 'fractal dimension error'
    'worst radius' 'worst texture' 'worst perimeter' 'worst area'
    'worst smoothness' 'worst compactness' 'worst concavity'
    'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

```
from sklearn.datasets import load_breast_cancer

# Load the dataset
data = load_breast_cancer()

# Features and target
X = data.data
y = data.target
```

# Print the label types (malignant, benign)
print("Labels:", data.target_names)

Labels: ['malignant' 'benign']

```
•••   Labels: ['malignant' 'benign']
```

print("Data (feature) shape:", cancer.data.shape)

Data (feature) shape: (569, 30)
Or

```
    Data (feature) shape: (569, 30)
```

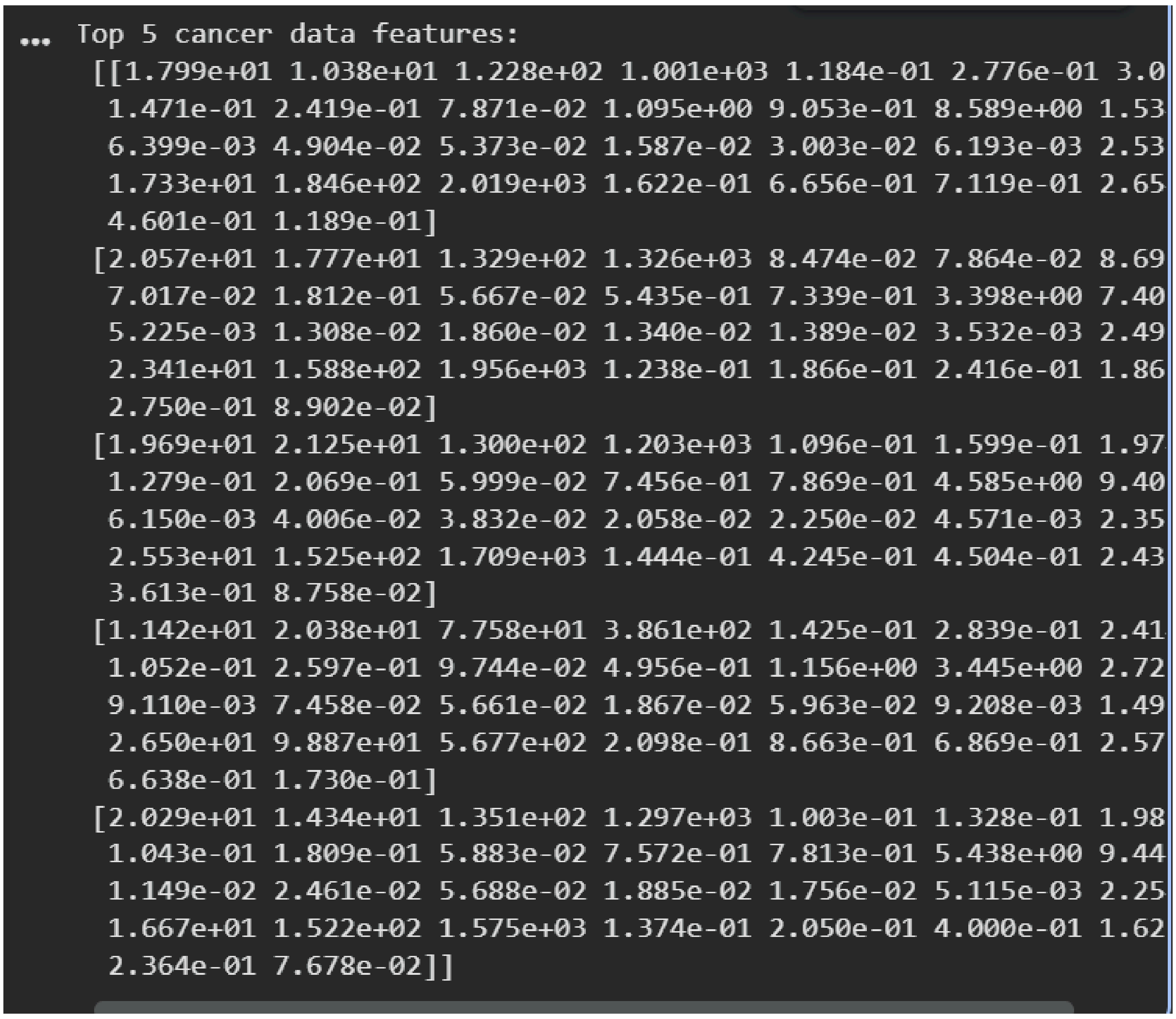
Cancer.data.shape

(569, 30)

```
•••   (569, 30)
```

print("Top 5 cancer data features:\n", cancer.data[:5])
Top 5 cancer data features:
 [[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01

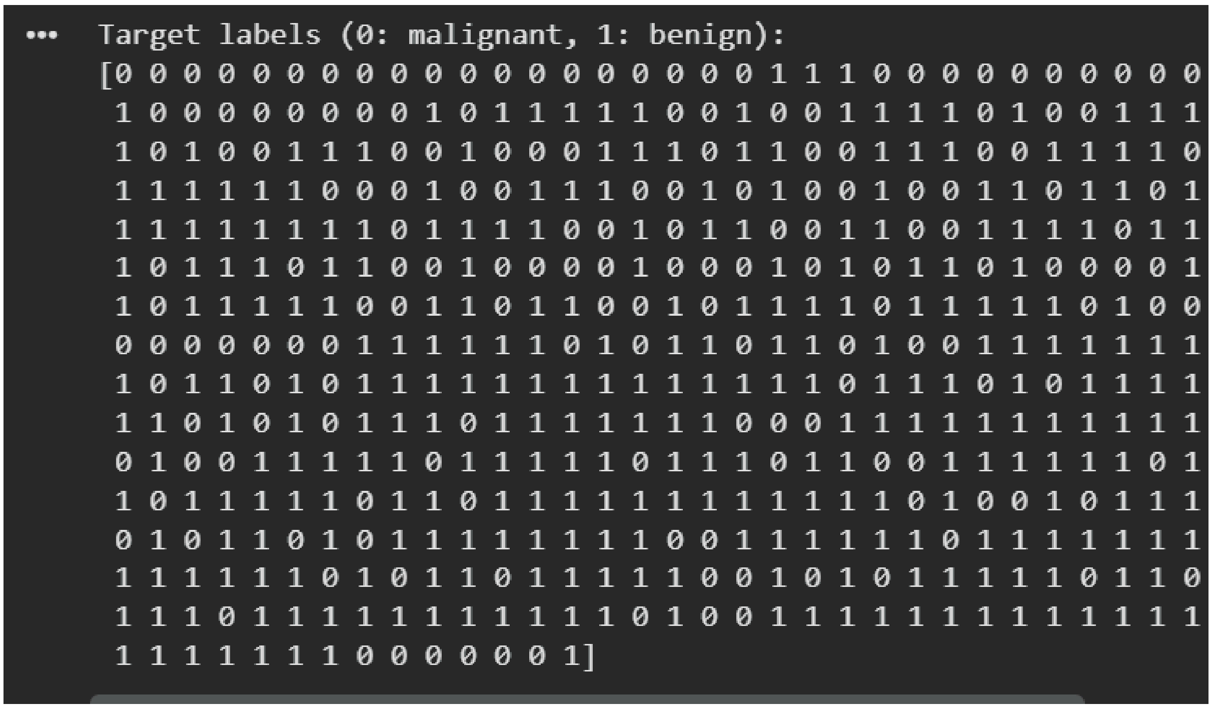
```
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01
1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01
1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00
9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03
2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01
2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01
2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00
2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03
1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01
2.575e-01
  6.638e-01 1.730e-01]
 [2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01
1.980e-01
  1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00
9.444e+01
  1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03
2.254e+01
  1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01
1.625e-01
  2.364e-01 7.678e-02]]
```
Or



```
... Top 5 cancer data features:
    [[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.0
      1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.53
      6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.53
      1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.65
      4.601e-01 1.189e-01]
     [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.69
      7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.40
      5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.49
      2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.86
      2.750e-01 8.902e-02]
     [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.97
      1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.40
      6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.35
      2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.43
      3.613e-01 8.758e-02]
     [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.41
      1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.72
      9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.49
      2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.57
      6.638e-01 1.730e-01]
     [2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.98
      1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.44
      1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.25
      1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.62
      2.364e-01 7.678e-02]]
```

```
print("Target labels (0: malignant, 1: benign):")
print(cancer.target)
Target labels (0: malignant, 1: benign):
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

Or



```
Target labels (0: malignant, 1: benign):
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn import svm
from sklearn import metrics

# Load dataset
cancer = load_breast_cancer()

# Split the dataset into training set (70%) and test set (30%)
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data,        # features
    cancer.target,      # labels (0: malignant, 1: benign)
    test_size=0.3,      # 30% for testing
    random_state=109    # ensures reproducibility
)

# Create an SVM classifier with linear kernel
```

```
clf = svm.SVC(kernel='linear')

# Train the model on training data
clf.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = clf.predict(X_test)

# Print model accuracy
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9649122807017544
```

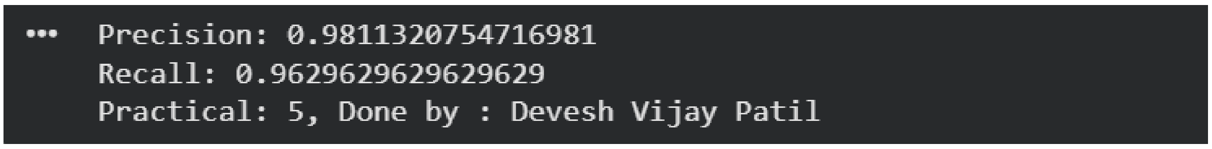
or

```
••• Accuracy: 0.9649122807017544
```

```
from sklearn import metrics

# Print precision and recall
print("Precision:", metrics.precision_score(y_test, y_pred))
print("Recall:", metrics.recall_score(y_test, y_pred))

print("Practical: 5, Done by : Devesh Vijay Patil")
```

```
Precision: 0.9811320754716981
Recall: 0.9629629629629629
Practical: 5, Done by : Devesh Vijay Patil
```

Or



```
••• Precision: 0.9811320754716981
    Recall: 0.9629629629629629
    Practical: 5, Done by : Devesh Vijay Patil
```

Conclusion: Implementing polynomial regression on the ice cream selling dataset we have
transformed the features into polynomial features of degree 2 i.e(yCBu + le + Bzxz) then this
polynomial feature is used to train the linear regression model; the r2 score shows that the model

is best fit to learn from such kind of polynomial features.

Practical No: 5

Aim:Implementation and Evaluation of Support Vector Machine using Accuracy, Precision and Recall

Theory:

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It tries to find the best boundary known as a hyperplane that separates different classes in the data.The main goal of SVM is to maximize the margin between the two classes. The larger the margin the better the model performs on new and unseen data. Key Concepts of Support Vector Machine

- Hyperplane: A decision boundary separating different classes in feature space and is represented by the equation wx + b = 0 in linear classification.
- Support Vectors: The closest data points to the hyperplane, crucial for determining the hyperplane and margin in SVM. o Margin: The distance between the hyperplane and the support vectors. SVM aims to maximize this margin for better classification performance.
- Kemel: A function that maps data to a higher-dimensional space enabling SVM to handle non-linearly separable data.
- Hard Margin: A maximum-margin hyperplane that perfectly separates the data without misclassifications.
- Soft Margin: Allows some misclassifications by introducing slack variables, balancing margin maximization and misclassification penalties when data is not perfectly separable.

Classification Metrics
Classification problems aim to predict discrete categories. To evaluate the performance of
classification models, we use the following metrics:

1. Accuracy
Accuracy is a fundamental metric used for evaluating the performance of a classification model. It tells us the proportion of correct predictions made by the model out of all predictions.While accuracy provides a quick snapshot, it can be misleading in cases of imbalanced datasets.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

2. Precision
It measures how many of the positive predictions made by the model are actually correct. It's useful when the cost of false positives is high such as in medical diagnoses where predicting a disease when it’s not present can have serious

consequences.Precision helps ensure that when the model predicts a positive outcome, it’s likely to be correct.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Where:
TP = True Positives
FP = False Positives

3. Recall
Recall or Sensitivity measures how many of the actual positive cases were correctly identified by the model. It is important when missing a positive case (false negative) is more costly than false positives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Where:
TP=True Positives
FN = False Negatives

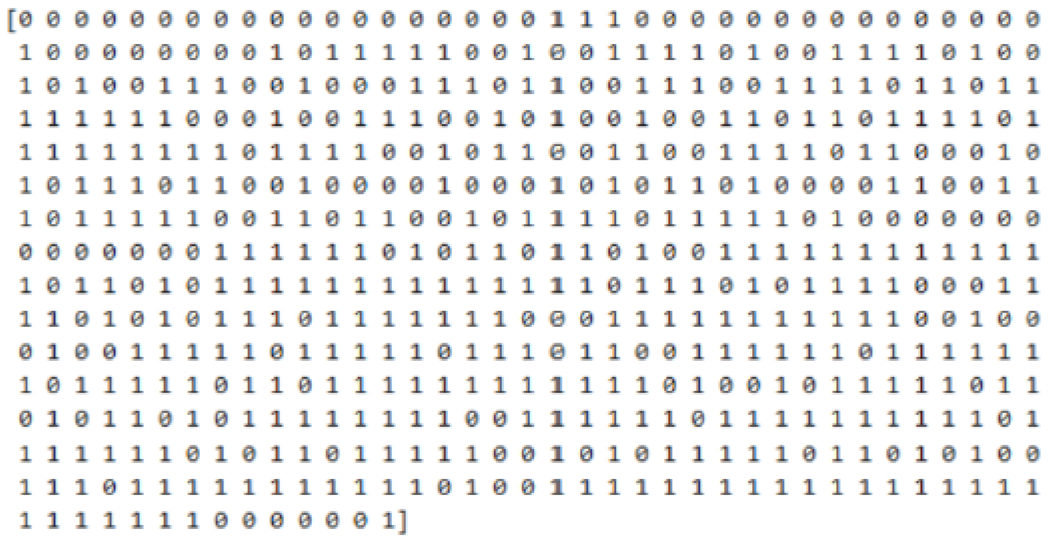
Code & Output:

```
from sklearn import datasets
cancer=datasets.load breast_cancer ()
from sklearn.model selection import train_test split
from sklearn import svm
from sklearn import metrics
print("Features:",cancer.featureinames
Features: ['mean radius' 'mean texture' 'mean perimeter’ 'mean area’
"mean smoothness' 'mean compactness’ ‘mean concavity'
"mean concave points' ‘mean symmetry' ‘mean fractal dimension’
‘radius error' 'texture error' 'perimeter error' ‘area error’
‘smoothness error' 'compactness error' ‘concavity error’
*concave points error’ ‘symmetry error’ ‘fractal dimension error’
‘worst radius’' 'worst texture' ‘worst perimeter’ 'worst area’
"worst smoothness' ‘worst compactness' ‘worst concavity'
‘worst concave points' ‘worst symmetry' ‘worst fractal dimension']
print("Labels:",cancer.targetinames)
```

```
                    [ Lobels: ['malignant' 'benign']
Cancer.data.shape
                                        (569, 30)
print (cancer.data[0:5])

[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01
3.001e-01
1.471e-01 2.419¢-01 7.871e-02 1.095¢+00 9.053-01 8.589e+00
1.534e+02
6.3992-03 4.904e-02 5.373¢-02 1.5872-02 3.003e-02 6.193¢-03
2.538e+01
1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01
2.654e-01
4.601e-01 1.189e-01]
[2.057e401 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02
8.690e-02
7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398+00
7.408e+01
5.225¢-03 1.308¢-02 1.860c-02 1.340e-02 1.389¢-02 3.532e-03
2.499e+01
2.341e+01 1.588e+02 1.956¢+03 1.238e-01 1.866e-01 2.416e-01
1.860e-01
2.750e-01 8.902e-02]
[1.969e401 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01
1.974e-01
1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00
9.403e+01
6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03
2.357e+01
2.553e+01 1.525¢+02 1.709¢+03 1.4442-01 4.245¢-01 4.504c-01
2.430e-01
3.6132-01 8.758¢-02]
[1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.83%e-01
2.414e-01
1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00
2.723e+01
```

9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03
1.491e+01
2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01
2.575e-01
6.638e-01 1.730e-01]
[2.029¢+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01
1.980e-01
1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00
9.444e+01
1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03
2.254e+01
1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01
1.625¢-01
2.364e-01 7.678e-02]]

print (cancer.target)



```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 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 0 0 0 0 0 0 1]
```

X train,X test,y train,y test=ttresta siplnit (cancer.data,cancer.target, testisize:() .3, random_state=109)

clf=svm.SVC (kernel="linear")
clf.fit(X_train,y_train)

```
y_pred=clf.predict (X_test)
print ("accuarcy",metrics.accuracy score(y_test,y pred))
                    accuarcy 0.9649122807017544

print ("Precision:",metrics.precision_score(y_test, y_pred))
print ("Recall:",metrics. recall score(y_test, y pred))
                    print (“Done by Devesh Patil”)
                    accuarcy 0.9649122807017544
                    Precision: ©.9811320754716981
                    Recall: ©.9629629629629629
```

Conclusion: Implementation of Support Vector Machine (SVM) on the iris dataset for classification has shown that the model can accurately predict about 80% of the name of the species and with the precision of 0.805 with recall of 0.8.

Practical No: 6

Aim:Implementation of K-Nearest Neighbours Algorithm with Performance Evaluation

Theory:

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm generally used for classification but can also be used for regression tasks. It works by finding the "k" closest data points (neighbors) to a given input and makes predictions based on the majority class (for classification) or the average value (for regression). Since KNN makes no assumptions about the underlying data distribution it makes it a non-parametric and instance-based learning method.K-Nearest Neighbors is also called as a lazy learner algorithm because it does not learn from the training set immediately instead it stores the entire dataset and performs computations only at the time of classification.In the k-Nearest Neighbours algorithm k is just a number that tells the algorithm how many nearby points or neighbors to look at when it makes a decision. Example: Imagine you're deciding which fruit it is based on its shape and size. You compare it to fruits you already know.

- Ifk=3, the algorithm looks at the 3 closest fruits to the new one.
- If 2 of those 3 fruits are apples and 1 is a banana, the algorithm says the new fruit is an apple because most of its neighbors are apples.

Distance Metrics Used in KNN Algorithm

KNN uses distance metrics to identify nearest neighbors; these neighbors are used for classification and regression tasks. To identify nearest neighbor we use below distance metrics:

1. Minkowski Distance

Minkowski distance is like a family of distances, which includes both Euclidean and Manhattan distances as special cases.

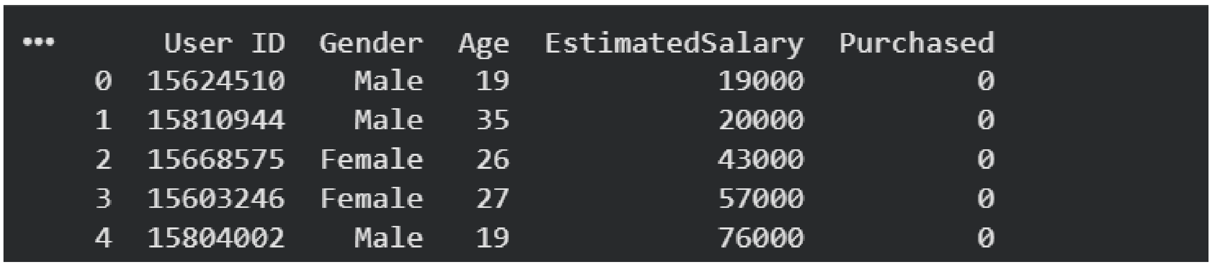
$$d\left(x,y\right)=\left(\sum_{i=1}^{n}\left(x_i-y_i\right)^p\right)^{\frac{1}{p}}$$

From the formula above, when p=2, it becomes the same as the Euclidean distance formula and when p=1, it turns into the Manhattan distance formula. Minkowski distance is essentially a flexible formula that can represent either Euclidean or Manhattan distance depending on the value of p.

Code & Output:

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

# Load dataset
dataset = pd.read_csv('Social_Network_Ads.csv')

# Show first 5 rows
print(dataset.head())

    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0
```

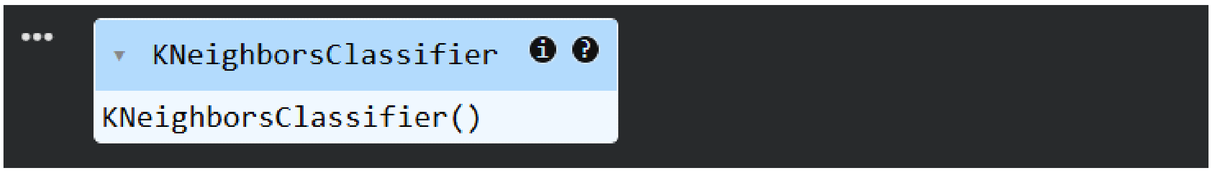
Or



```
•••      User ID  Gender  Age  EstimatedSalary  Purchased
    0  15624510    Male   19            19000          0
    1  15810944    Male   35            20000          0
    2  15668575  Female   26            43000          0
    3  15603246  Female   27            57000          0
    4  15804002    Male   19            76000          0
```

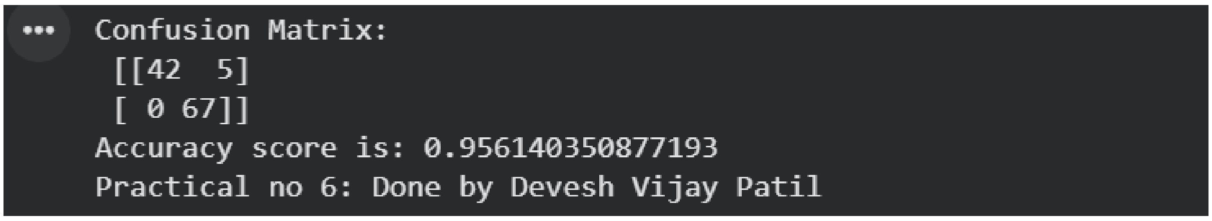
```
# Splitting the dataset into the Training set and
Test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, 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)

# Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5,
metric='minkowski', p=2)
classifier.fit(X_train, y_train)
```



```
KNeighborsClassifier
KNeighborsClassifier()
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix,
accuracy_score

# Confusion Matrix & Accuracy
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)

print("Confusion Matrix:\n", cm)
print("Accuracy score is:", ac)

print("Practical no 6: Done by Devesh Vijay Patil")

Confusion Matrix:
 [[42  5]
 [ 0 67]]
Accuracy score is: 0.956140350877193
Practical no 6: Done by Devesh Vijay Patil

Or
```



```
Confusion Matrix:
 [[42  5]
 [ 0 67]]
Accuracy score is: 0.956140350877193
Practical no 6: Done by Devesh Vijay Patil
```

Conclusion: Implementing k-neareast neighbors on the social network ads and using standardscaler to standardizes features and wuse this features on the model with kNeighborsClassifier using minkowski distance along with k values to 5 and p value set to 2 the confusion matrix stated that the no of time the model has predicted the values true positive and false positive and model has an accuracy score of around 90% which means the model can predict accurately.

Practical No: 7

Aim:Implementation and Visualization of K-Means Clustering

Theory:
K-Means Clustering is an unsupervised machine learning algorithm that helps group data points into clusters based on their inherent similarity. Unlike supervised learning, where we train models using labeled data, K-Means is used when we have data that is not labeled and the goal is to uncover hidden patterns or structures. For example, an online store can use K-Means to segment customers into groups like "Budget Shoppers," "Frequent Buyers," and "Big Spenders" based on their purchase history. ‘Working of K-Means Clustering Suppose we are given a data set of items with certain features and values for these features like a vector. The task is to categorize those items into groups. To achieve this we will use the K-means algorithm."k" represents the number of groups or clusters we want to classify our items into. The algorithm will categorize the items into "k" groups or clusters of similarity. To calculate that similarity we will use the Euclidean distance as a measurement. The algorithm works as follows: Initialization: We begin by randomly selecting k cluster centroids. Assignment Step: Each data point is assigned to the nearest centroid, forming clusters.

- Update Step: After the assignment, we recalculate the centroid of each cluster by averaging the points within it.
- Repeat: This process repeats until the centroids no longer change or the maximum number of iterations is reached.

The goal is to partition the dataset into k clusters such that data points within each cluster are more similar to each other than to those in other clusters.

Performance Evaluation

Confusion matrix is a simple table used to measure how well a classification model is performing. It compares the predictions made by the model with the actual results and shows where the model was right or wrong. This helps you understand where the model is making mistakes so you can improve it. It breaks down the predictions into four categories:

Accuracy
Accuracy is a fundamental metric used for evaluating the performance of a classification model.
It tells us the proportion of correct predictions made by the model out of all predictions.While accuracy provides a quick snapshot, it can be misleading in cases of imbalanced datasets.

Code & Output:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

# Load IRIS dataset
dataset = datasets.load_iris()

print(dataset)

{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
[5.1, 3.3, 1.7, 0.5],
[4.8, 3.4, 1.9, 0.2],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
[7.2, 3.2, 6. , 1.8],
[6.2, 2.8, 4.8, 1.8],
[6.1, 3. , 4.9, 1.8],
[6.4, 2.8, 5.6, 2.1],
[7.2, 3. , 5.8, 1.6],
[7.4, 2.8, 6.1, 1.9],
[7.9, 3.8, 6.4, 2. ],
[6.4, 2.8, 5.6, 2.2],
[6.3, 2.8, 5.1, 1.5],
[6.1, 2.6, 5.6, 1.4],
[7.7, 3. , 6.1, 2.3],
[6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]), 'target': array([0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 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, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2]), 'frame': None, 'target_names':
array(['setosa', 'versicolor', 'virginica'],
dtype='<U10'), 'DESCR': '.. _iris_dataset:\n\nIris
plants dataset\n--------------------\n\n**Data Set
Characteristics:**\n\n:Number of Instances: 150 (50
in each of three classes)\n:Number of Attributes: 4
numeric, predictive attributes and the
class\n:Attribute Information:\n    - sepal length
in cm\n    - sepal width in cm\n    - petal length
in cm\n    - petal width in cm\n    - class:\n
- Iris-Setosa\n            - Iris-Versicolour\n
- Iris-Virginica\n\n:Summary
```

```
Statistics:\n\n============== ==== ==== =======
===== ====================\n                Min  Max
Mean    SD   Class Correlation\n============== ====
==== ======= ===== ====================\nsepal
length:   4.3  7.9   5.84   0.83    0.7826\nsepal
width:    2.0  4.4   3.05   0.43   -0.4194\npetal
length:   1.0  6.9   3.76   1.76    0.9490
(high!)\npetal width:    0.1  2.5   1.20   0.76
0.9565  (high!)\n============== ==== ==== =======
===== ====================\n\n:Missing Attribute
Values: None\n:Class Distribution: 33.3% for each of
3 classes.\n:Creator: R.A. Fisher\n:Donor: Michael
Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n:Date:
July, 1988\n\nThe famous Iris database, first used
by Sir R.A. Fisher. The dataset is taken\nfrom
Fisher\'s paper. Note that it\'s the same as in R,
but not as in the UCI\nMachine Learning Repository,
which has two wrong data points.\n\nThis is perhaps
the best known database to be found in the\npattern
recognition literature.  Fisher\'s paper is a
classic in the field and\nis referenced frequently
to this day.  (See Duda & Hart, for example.)
The\ndata set contains 3 classes of 50 instances
each, where each class refers to a\ntype of iris
plant.  One class is linearly separable from the
other 2; the\nlatter are NOT linearly separable from
each other.\n\n.. dropdown:: References\n\n  -
Fisher, R.A. "The use of multiple measurements in
taxonomic problems"\n    Annual Eugenics, 7, Part
II, 179-188 (1936); also in "Contributions to\n
Mathematical Statistics" (John Wiley, NY, 1950).\n
- Duda, R.O., & Hart, P.E. (1973) Pattern
Classification and Scene Analysis.\n    (Q327.D83)
John Wiley & Sons.  ISBN 0-471-22361-1.  See page
218.\n  - Dasarathy, B.V. (1980) "Nosing Around the
Neighborhood: A New System\n    Structure and
Classification Rule for Recognition in Partially
Exposed\n    Environments".  IEEE Transactions on
```

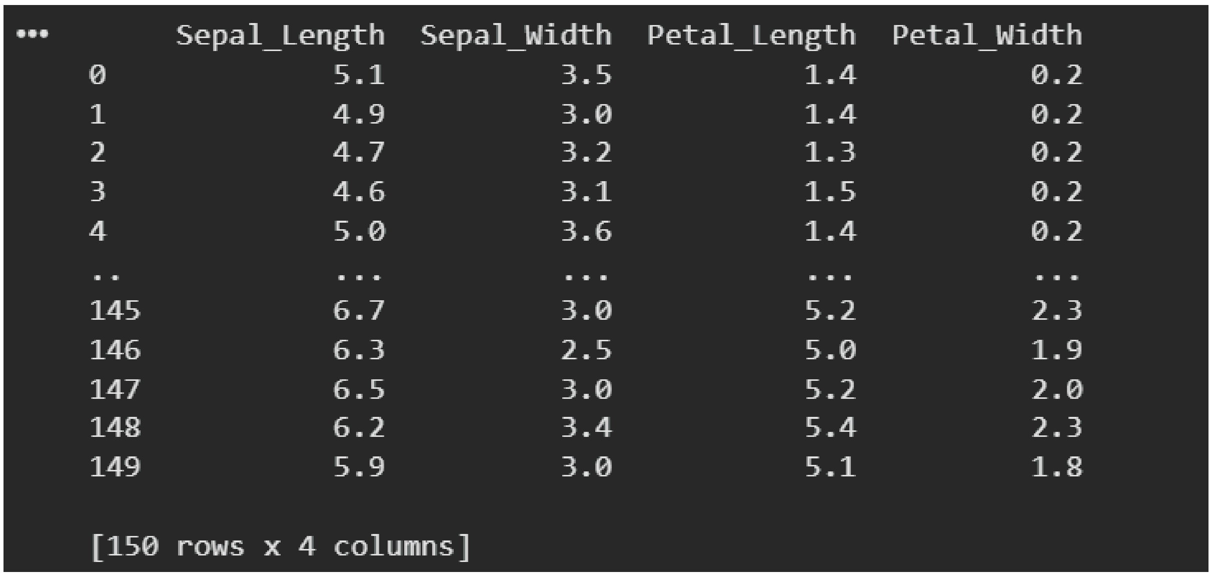
```
Pattern Analysis and Machine\n    Intelligence, Vol.
PAMI-2, No. 1, 67-71.\n  - Gates, G.W. (1972) "The
Reduced Nearest Neighbor Rule".  IEEE Transactions\n
on Information Theory, May 1972, 431-433.\n  - See
also: 1988 MLC Proceedings, 54-64.  Cheeseman et
al"s AUTOCLASS II\n    conceptual clustering system
finds 3 classes in the data.\n  - Many, many more
...\n', 'feature_names': ['sepal length (cm)',
'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)'], 'filename': 'iris.csv', 'data_module':
'sklearn.datasets.data'}
```

```
X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Lengt
h','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
print(X)
```

```
     Sepal_Length  Sepal_Width  Petal_Length
Petal_Width
0             5.1          3.5           1.4
0.2
1             4.9          3.0           1.4
0.2
2             4.7          3.2           1.3
0.2
3             4.6          3.1           1.5
0.2
4             5.0          3.6           1.4
0.2
..            ...          ...           ...
...
145           6.7          3.0           5.2
2.3
146           6.3          2.5           5.0
1.9
```

```
147             6.5           3.0            5.2
2.0
148             6.2           3.4            5.4
2.3
149             5.9           3.0            5.1
1.8

[150 rows x 4 columns]
```

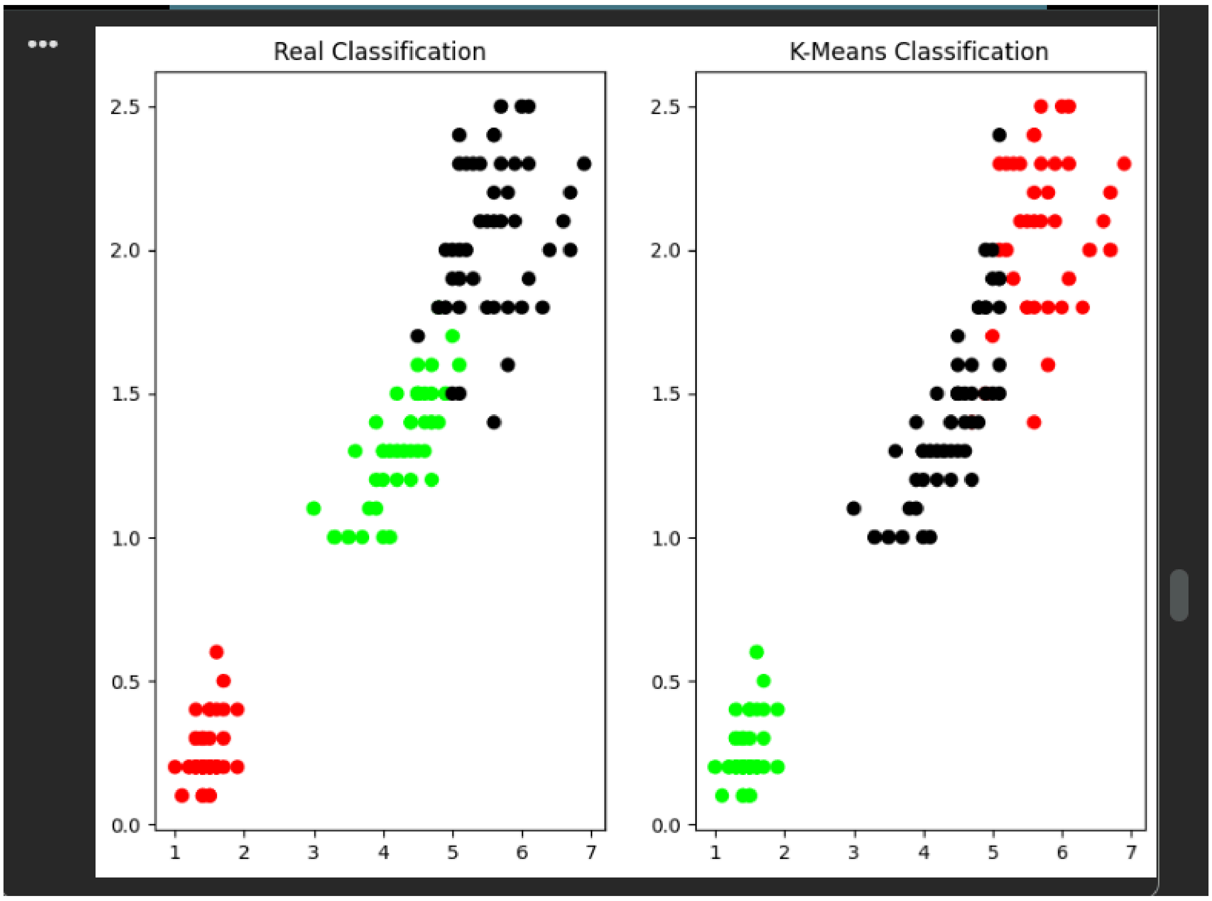
Or



```
       Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
   0            5.1          3.5           1.4          0.2
   1            4.9          3.0           1.4          0.2
   2            4.7          3.2           1.3          0.2
   3            4.6          3.1           1.5          0.2
   4            5.0          3.6           1.4          0.2
   ..           ...          ...           ...          ...
   145          6.7          3.0           5.2          2.3
   146          6.3          2.5           5.0          1.9
   147          6.5          3.0           5.2          2.0
   148          6.2          3.4           5.4          2.3
   149          5.9          3.0           5.1          1.8

   [150 rows x 4 columns]
```

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

# Load dataset
iris = datasets.load_iris()

# Convert into DataFrame
X = pd.DataFrame(iris.data,
columns=iris.feature_names)
y = pd.DataFrame(iris.target, columns=['target'])

# Colors
```

```
colormap = np.array(['red', 'lime', 'black'])

# Create figure
plt.figure(figsize=(14, 7))

# ---------------- REAL CLASSIFICATION
----------------
plt.subplot(1, 3, 1)
plt.scatter(X['petal length (cm)'], X['petal width
(cm)'],
            c=colormap[y['target']], s=40)
plt.title('Real Classification')

# ---------------- K-MEANS CLASSIFICATION
----------------
plt.subplot(1, 3, 2)
model = KMeans(n_clusters=3, random_state=42)
model.fit(X)

predY = np.choose(model.labels_, [0, 1,
2]).astype(int)

plt.scatter(X['petal length (cm)'], X['petal width
(cm)'],
            c=colormap[predY], s=40)
plt.title('K-Means Classification')

plt.show()
```



Conclusion: Implementing the K-Means Clustering on the iris dataset the number of clusters that is specified in the model initialization that are 3 as the visualization we can get the model have separated the dataset into three clusters of red , black and green and model has an accuracy of 93%.

Practical No: 8

Aim: Implementation and Visualization of Hierarchical Clustering using Dendrogram.

Theory:

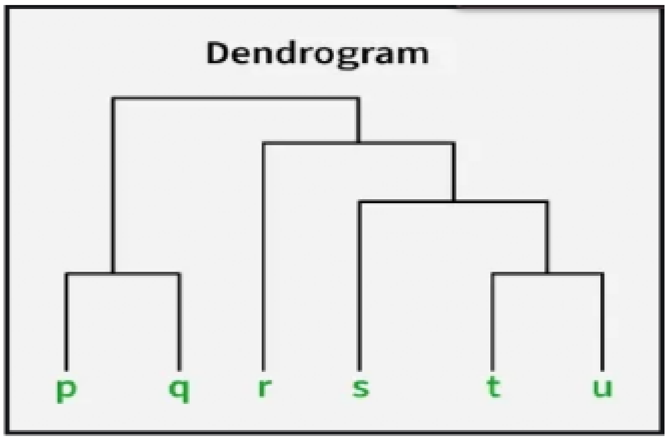
Hierarchical clustering is an unsupervised learning technique used to group similar data points into clusters by building a hierarchy (tree-like structure). Unlike flat clustering like k-means hierarchical clustering does not require specifying the number of clusters in advance.
The algorithm builds clusters step by step either by progressively merging smaller clusters or by splitting a large cluster into smaller ones. The process is often visualized using a dendrogram, which helps to understand data similarity.
Imagine we have four fruits with different weights: an apple (100g), a banana (120g), a cherry (50g) and a grape (30g). Hierarchical clustering starts by treating each fruit as its own group.

- Start with each fruit as its own cluster.
- Merge the closest items: grape (30g) and cherry (50g) are grouped first.
- Next, apple (100g) and banana (120g) are grouped.
- Finally, these two clusters merge into one.

Finally all the fruits are merged into one large group, showing how hierarchical clustering progressively combines the most similar data points.

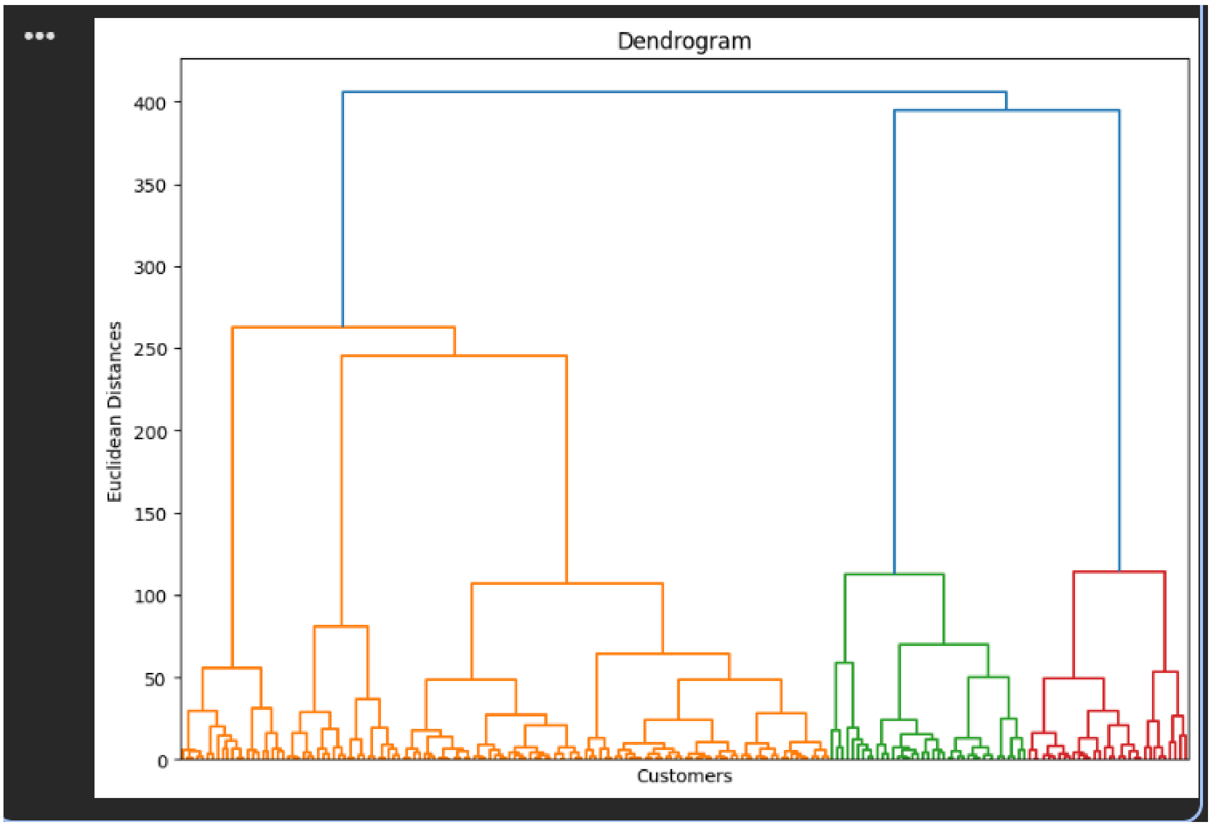
Dendrogram

A dendrogram is like a family tree for clusters. It shows how individual data points or groups of data merge together. The bottom shows each data point as its own group and as we move up, similar groups are combined. The lower the merge point, the more similar the groups are. It helps us see how things are grouped step by step.



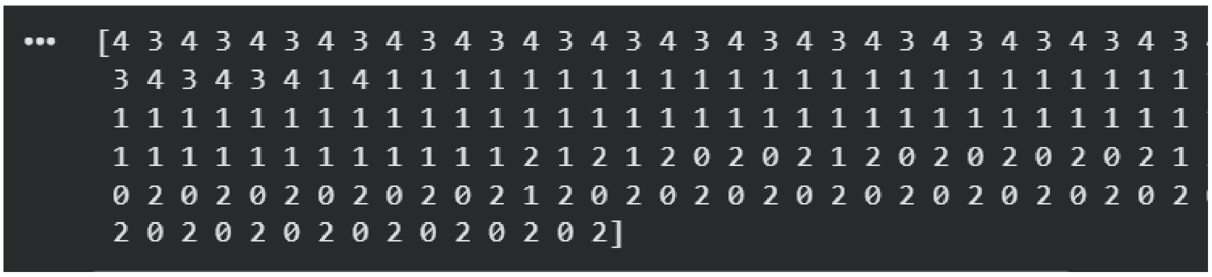
Code & Output :

```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.cluster.hierarchy as sch

# Load dataset
dataset = pd.read_csv('Mall_Customers.csv')

# Select only Annual Income and Spending Score
columns
X = dataset.iloc[:, [3, 4]].values

# Create Dendrogram
plt.figure(figsize=(10, 7))
dendro = sch.dendrogram(sch.linkage(X,
method='ward'), no_labels=True)

plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean Distances')
plt.show()
```



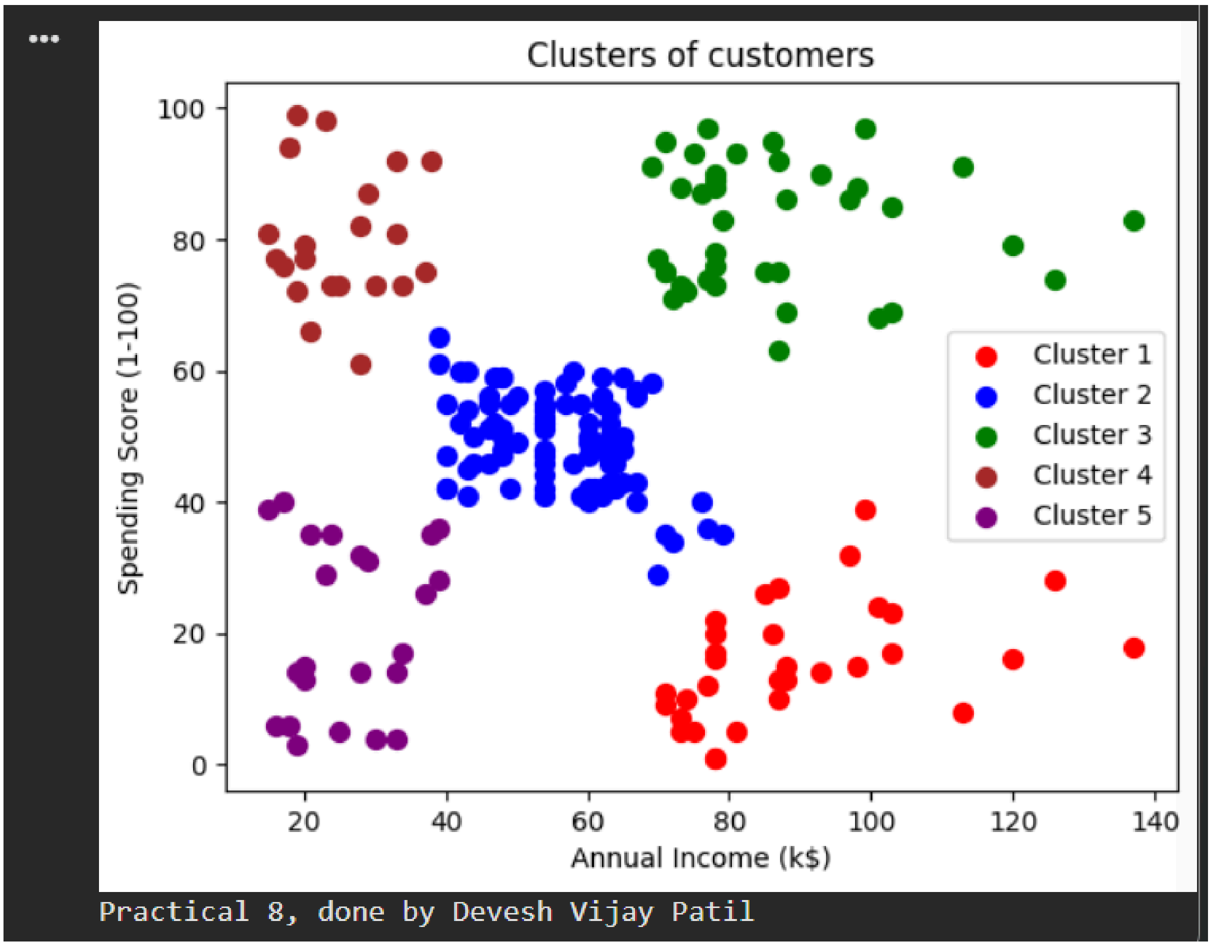
```
from sklearn.cluster import AgglomerativeClustering

# Recommended version (new sklearn uses 'metric')
hc = AgglomerativeClustering(n_clusters=5,
metric='euclidean', linkage='ward')

# Fit & Predict clusters
y_hc = hc.fit_predict(X)

print(y_hc)

[4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3
4 3 4 3 4 3 4 3 4 3 4
 3 4 3 4 3 4 1 4 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 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 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 0 2 0 2 1 2 0 2 0
2 0 2 0 2 1 2 0 2 1 2
```

```
 0 2 0 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 0 2 0 2
0 2 0 2 0 2 0 2 0 2 0
 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
Or
```



```
•••  [4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3
      3 4 3 4 3 4 1 4 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 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 2 1 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 1
      0 2 0 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2
      2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
```

```
# Visualise the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s=50,
c='red', label='Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s=50,
c='blue', label='Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s=50,
c='green', label='Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s=50,
c='brown', label='Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s=50,
c='purple', label='Cluster 5')

plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()

print("Practical 8, done by Devesh Vijay Patil")4
```



```
Practical 8, done by Devesh Vijay Patil
```

Conclusion: The Agglomerative Hierarchical Clustering grouped customers into five distinct clusters based on their features. The dendrogram and scatter plot confirm clear separation and similarity within clusters. This analysis provides insights for effective customer segmentation and targeted strategies.

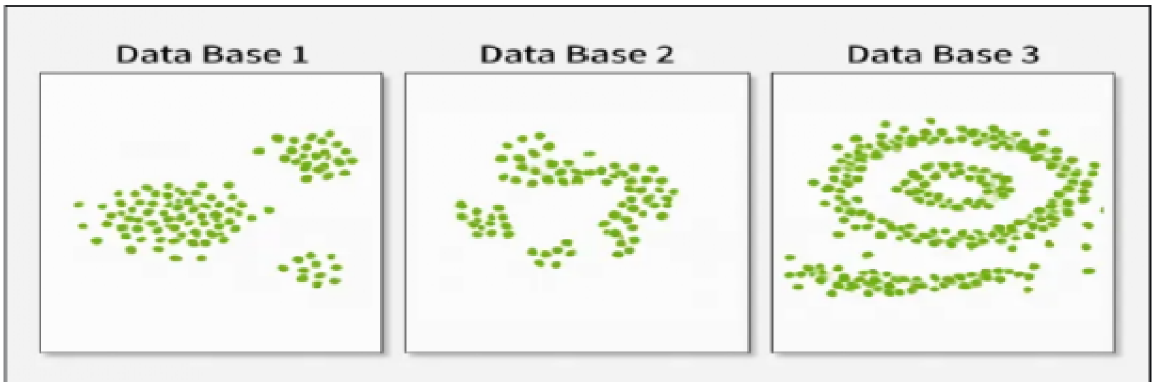
Practical No:9

Aim: Implementation of Density-Based Spatial Clustering of Application of with Noise

Theory:

DBSCAN is a density-based clustering algorithm that groups data points that are closely packed together and marks outliers as noise based on their density in the feature space. It identifies clusters as dense regions in the data space separated by areas of lower density. Unlike K-Means or hierarchical clustering which assumes clusters are compact and spherical, DBSCAN perform well in handling real-world data irregularities such as:

- Arbitrary-Shaped Clusters: Clusters can take any shape not just circular or convex.
- Noise and Outliers: It effectively identifies and handles noise points without assigning them to any cluster.



The figure above shows a data set with clustering algorithms: K-Means and Hierarchical handling compact, spherical clusters with varying noise tolerance while DBSCAN manages arbitrary-shaped clusters and noise handling.

Key Parameters in DBSCAN

1. eps: This defines the radius of the neighborhood around a data point. If the distance between two points is less than or equal to eps they are considered neighbors. A common method to determine eps is by analyzing the k-distance graph.

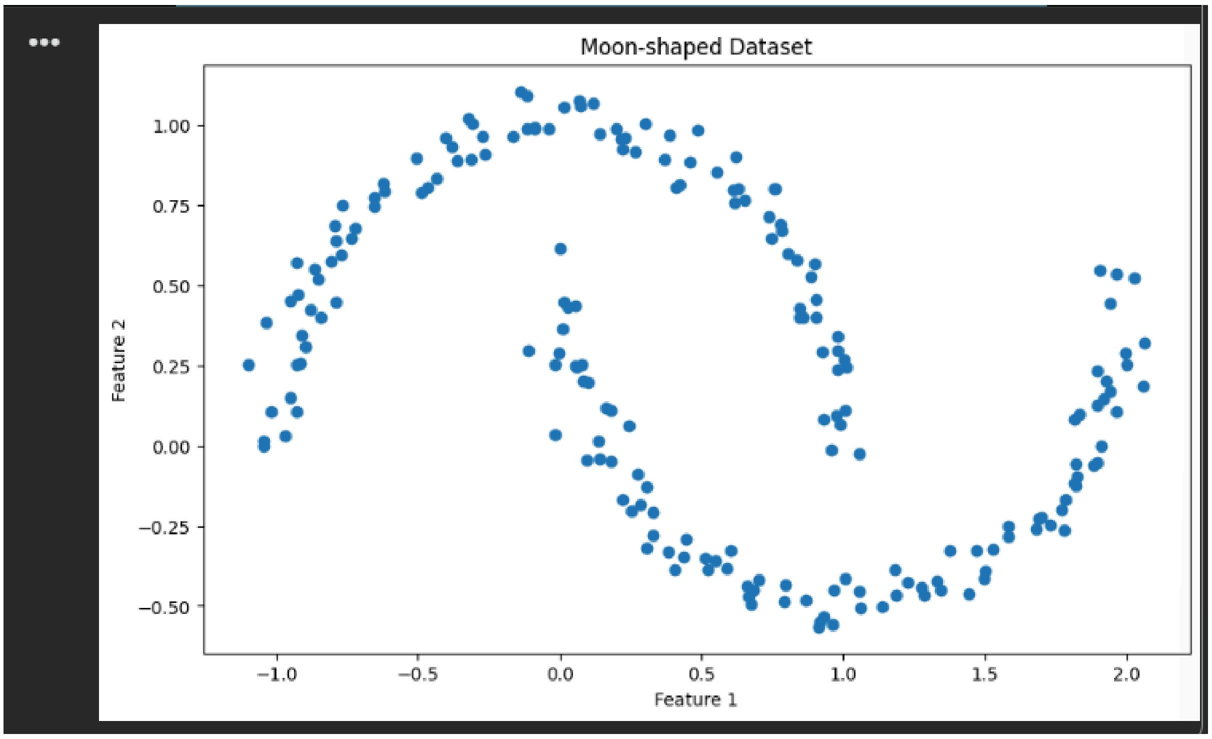
Choosing the right eps is important:

If eps is too small most points will be classified as noise.

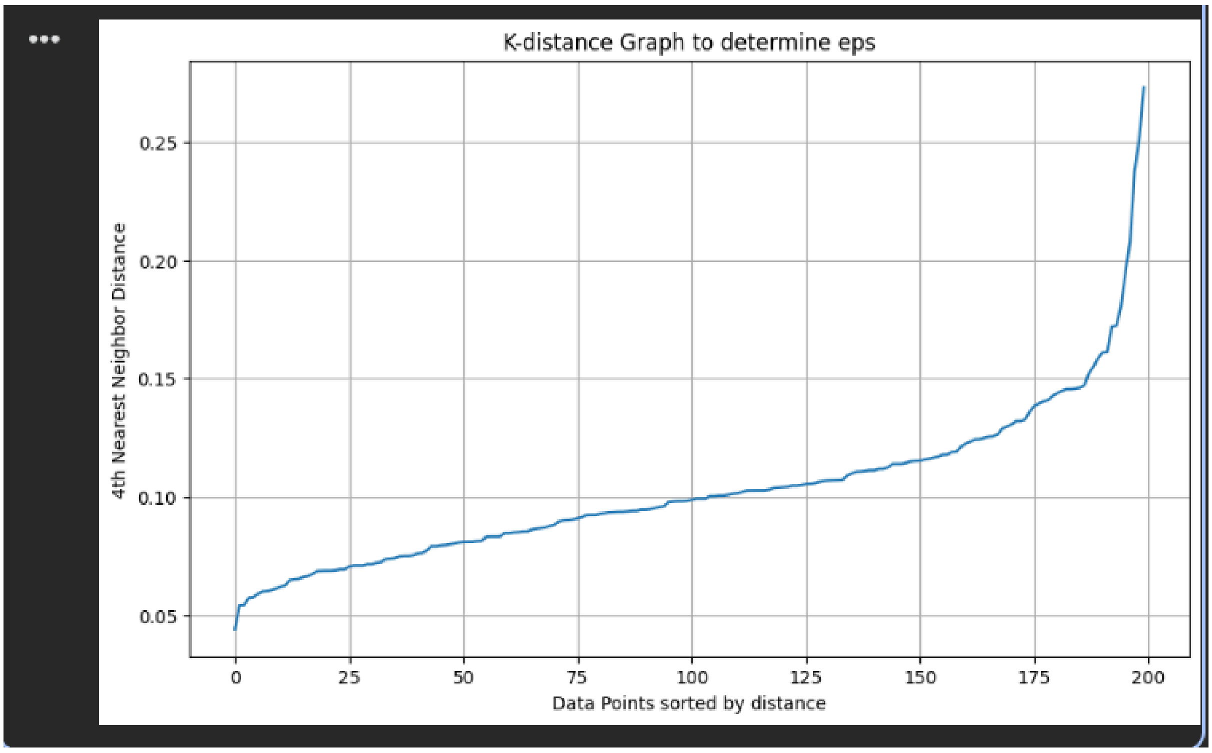
If eps is too large, clusters may merge and the algorithm may fail to distinguish between them.
2. MinPts: This is the minimum number of points required within the eps radius to form a dense region. A general rule of thumb is to set MinPts >= D+1 where D is the number of dimensions in the dataset.

Code & Output :

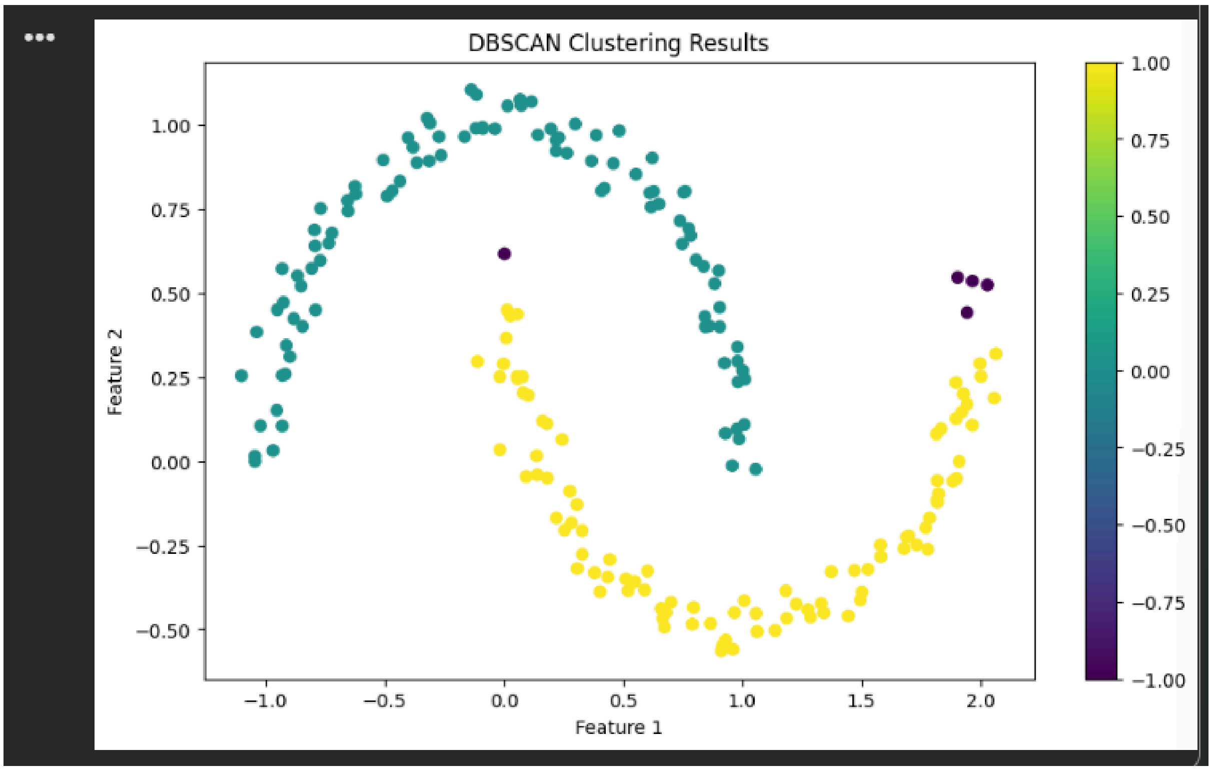
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors

# Generate dataset
X, _ = make_moons(n_samples=200, noise=0.05,
random_state=42)

# Visualize the dataset
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1])
plt.title('Moon-shaped Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



```
# Step 2: Use Nearest Neighbors to find optimal eps
neigh = NearestNeighbors(n_neighbors=5)
nbrs = neigh.fit(X)
distances, indices = nbrs.kneighbors(X)

# Sort distances
distances = np.sort(distances[:, 4])

# Plot k-distance graph
plt.figure(figsize=(10, 6))
plt.plot(distances)
plt.title('K-distance Graph to determine eps')
plt.xlabel('Data Points sorted by distance')
plt.ylabel('4th Nearest Neighbor Distance')
plt.grid(True)
plt.show()
```

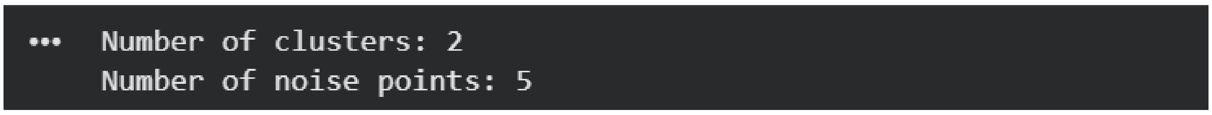


```
# Perform DBSCAN clustering
epsilon = 0.15   # chosen from K-distance graph
min_samples = 5  # minimum points to form cluster

dbscan = DBSCAN(eps=epsilon,
min_samples=min_samples)
clusters = dbscan.fit_predict(X)

# Visualize the results
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X[:, 0], X[:, 1], c=clusters,
cmap='viridis')
plt.colorbar(scatter)

plt.title('DBSCAN Clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



```
# Print number of clusters and noise points
n_clusters = len(set(clusters)) - (1 if -1 in
clusters else 0)
n_noise = list(clusters).count(-1)

print(f'Number of clusters: {n_clusters}')
print(f'Number of noise points: {n_noise}')

Number of clusters: 2
Number of noise points: 5
```

Or

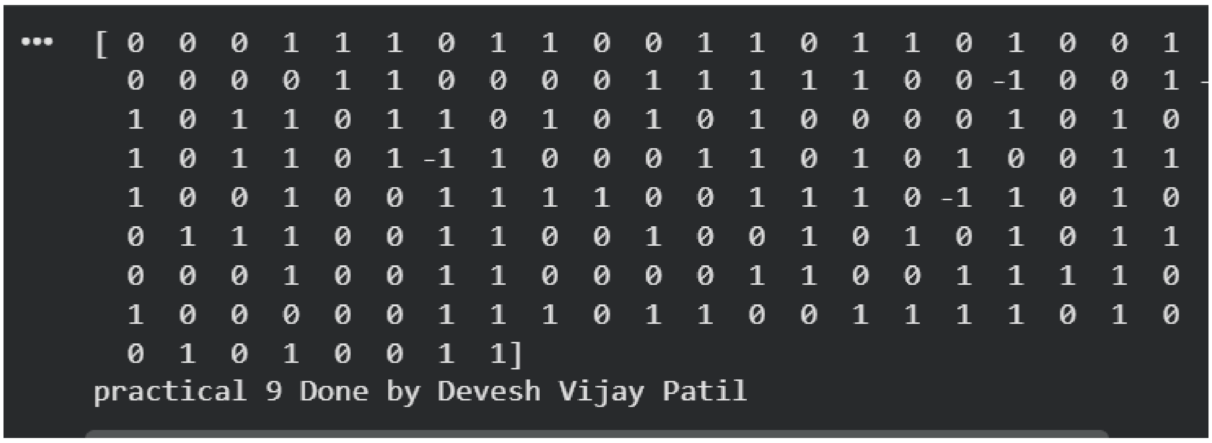


```
•••   Number of clusters: 2
      Number of noise points: 5
```

```
print(clusters)
print("practical 9 Done by Devesh Vijay Patil")
```

```
[ 0  0  0  1  1  1  0  1  1  0  0  1  1  0  1  1  0
1  0  0  1  0  1  0
  0  0  0  0  1  1  0  0  0  0  1  1  1  1  1  0  0
-1  0  0  1 -1  0  0
  1  0  1  1  0  1  1  0  1  0  1  0  1  0  0  0  0
1  0  1  0  1  0  1
  1  0  1  1  0  1 -1  1  0  0  0  1  1  0  1  0  1
0  0  1  1  0  0  1
  1  0  0  1  0  0  1  1  1  1  0  0  1  1  1  0 -1
1  0  1  0  0  1  0
  0  1  1  1  0  0  1  1  0  0  1  0  0  1  0  1  0
1  0  1  1  1  1  0
  0  0  0  1  0  0  1  1  0  0  0  0  1  1  0  0  1
1  1  1  0  0  1  0
  1  0  0  0  0  0  1  1  1  0  1  1  0  0  1  1  1
1  0  1  0  0  1 -1
  0  1  0  1  0  0  1  1]

practical 9 Done by Devesh Vijay Patil
```

Or



```
[ 0  0  0  1  1  1  0  1  1  0  0  1  1  0  1  1  0  1  0  0  1
  0  0  0  0  1  1  0  0  0  0  1  1  1  1  1  0  0 -1  0  0  1 -
  1  0  1  1  0  1  1  0  1  0  1  0  1  0  0  0  0  1  0  1  0
  1  0  1  1  0  1 -1  1  0  0  0  1  1  0  1  0  1  0  0  1  1
  1  0  0  1  0  0  1  1  1  1  0  0  1  1  1  0 -1  1  0  1  0
  0  1  1  1  0  0  1  1  0  0  1  0  0  1  0  1  0  1  0  1  1
  0  0  0  1  0  0  1  1  0  0  0  0  1  1  0  0  1  1  1  1  0
  1  0  0  0  0  0  1  1  1  0  1  1  0  0  1  1  1  1  0  1  0
  0  1  0  1  0  0  1  1]
practical 9 Done by Devesh Vijay Patil
```

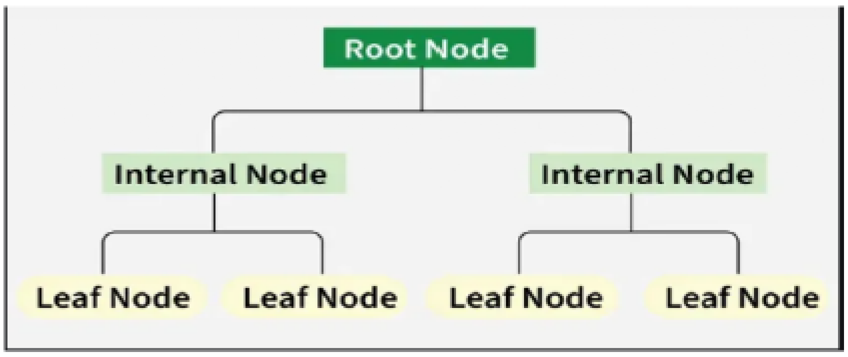
Conclusion :The implementation of the Density -Based Spatial Clustering Algorithm has stated that the algorithm has grouped the points that are closer to each other and marked them as one cluster and those away from each other mark them as noise.

Practical No: 10

Aim:Comparative Analysis of Decision Tree Classifier and Random Forest Classifier

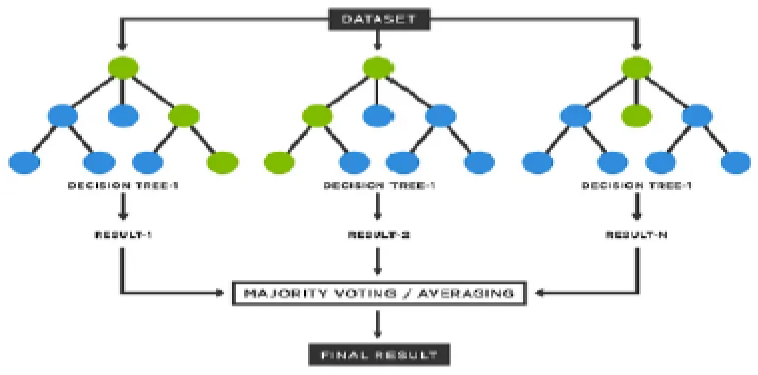
Theory:
A Decision Tree helps us to make decisions by mapping out different choices and their possible outcomes. It’s used in machine learning for tasks like classification and prediction.A Decision Tree helps us make decisions by showing different options and how they are related. It has a tree-like structure that starts with one main question called the root node which represents the entire dataset. From there, the tree branches out into different possibilities based on features in the data.

- Root Node: Starting point representing the whole dataset.
- Branches: Lines connecting nodes showing the flow from one decision to another.
- Internal Nodes: Points where decisions are made based on data features.
- Leaf Nodes: End points of the tree where the final decision or prediction is made.



A Decision Tree also helps with decision-making by showing possible outcomes clearly. By looking at the "branches" we can quickly compare options and figure out the best choice.
There are mainly two types of Decision Trees based on the target variable:

- Classification Trees: Used for predicting categorical outcomes like spam or not spam.These trees split the data based on features to classify data into predefined categories.
- Regression Trees: Used for predicting continuous outcomes like predicting house prices.Instead of assigning categories, it provides numerical predictions based on the input features



Random Forest

Random Forest is a machine learning algorithm that uses many decision trees to make better predictions. Each tree looks at different random parts of the data and their results are combined by voting for classification or averaging for regression which makes it an ensemble learning technique. This helps in improving accuracy and reducing errors.
‘Working of Random Forest Algorithm

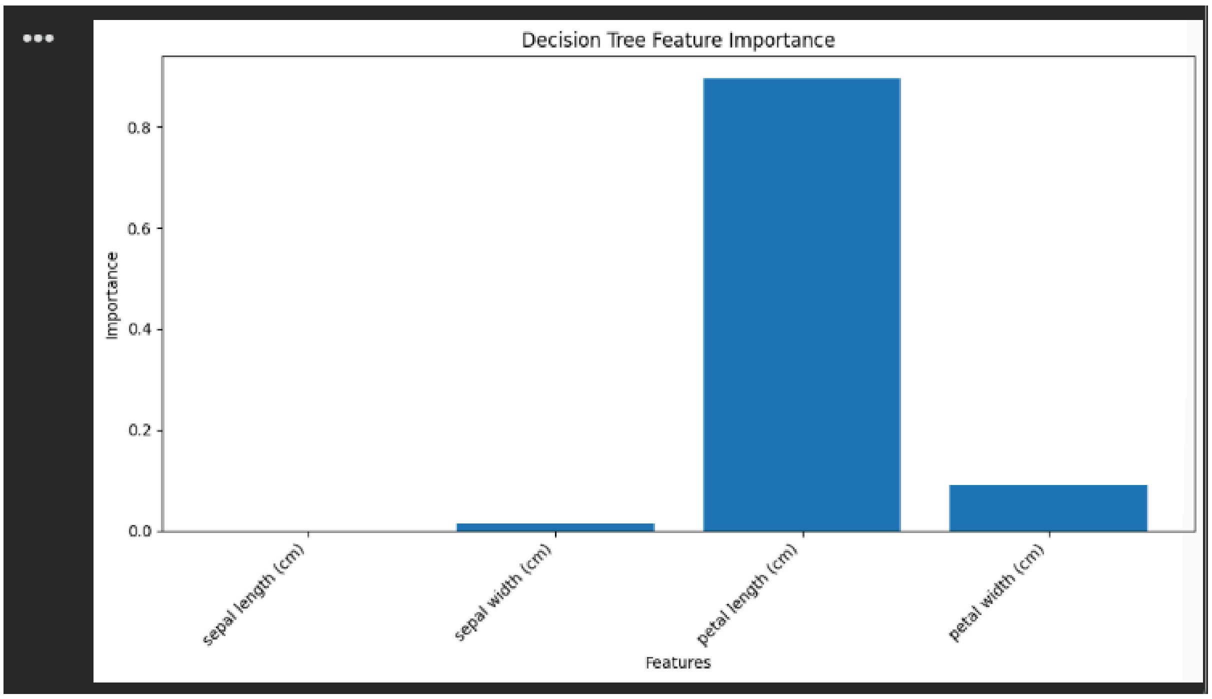
- Create Many Decision Trees: The algorithm makes many decision trees each using a random part of the data. So every tree is a bit different.
- Pick Random Features: When building each tree it doesn’t look at all the features
- (columns) at once. It picks a few at random to decide how to split the data. This helps the trees stay different from each other.
- Each Tree Makes a Prediction: Every tree gives its own answer or prediction based on what it learned from its part of the data.
- Combine the Predictions: For classification we choose a category as the final answer is the one that most trees agree on i.e majority voting and for regression we predict a
- number as the final answer is the average of all the trees predictions.
- ‘Why It Works Well: Using random data and features for each tree helps avoid overfitting and makes the overall prediction more accurate and trustworthy.

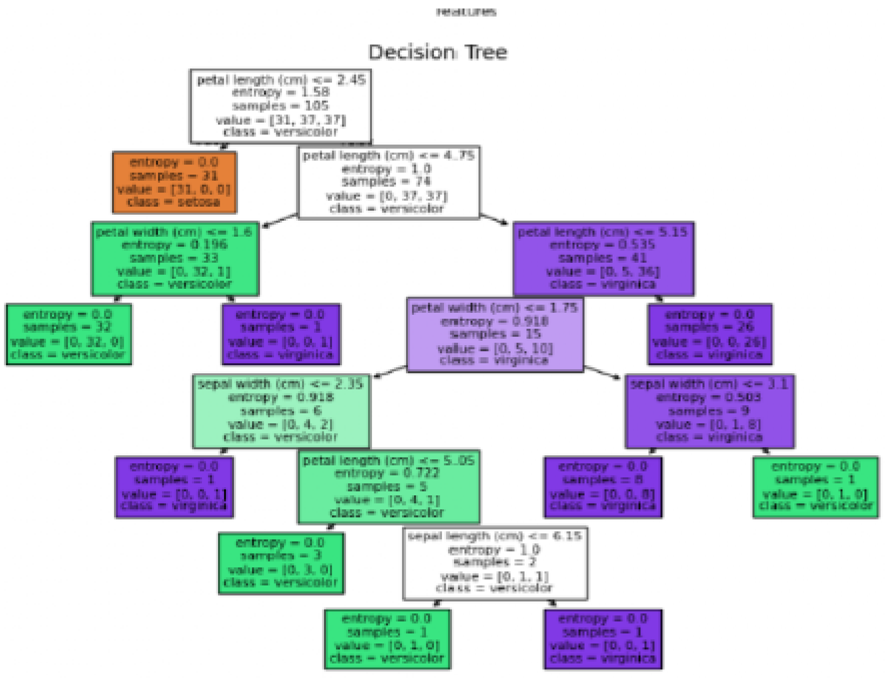
Code & Outputs :

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier,
plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Load the Iris Dataset
iris = load_iris()
X = iris.data
y = iris.target

# 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
)

print('Training data shape:', X_train.shape)
print('Testing data shape:', X_test.shape)

Training data shape: (120, 4)
Testing data shape: (30, 4)
Or
```

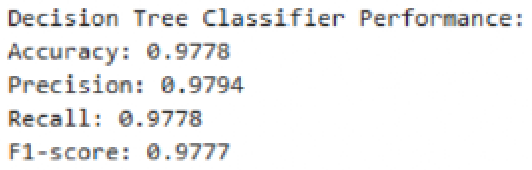
```
Training data shape: (120, 4)
Testing data shape: (30, 4)
```

```
# Initiate a Decision Tree classifier and fit it to
the training data
dt_clf = DecisionTreeClassifier(criterion="entropy",
random_state=42)
dt_clf.fit(X_train, y_train)

# Access the feature importances
feature_importances = dt_clf.feature_importances_

# Create a bar plot to visualize the feature
importance
plt.figure(figsize=(10, 6))
plt.bar(iris.feature_names, feature_importances)

plt.title("Decision Tree Feature Importance")
plt.xlabel("Features")
plt.ylabel("Importance")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```





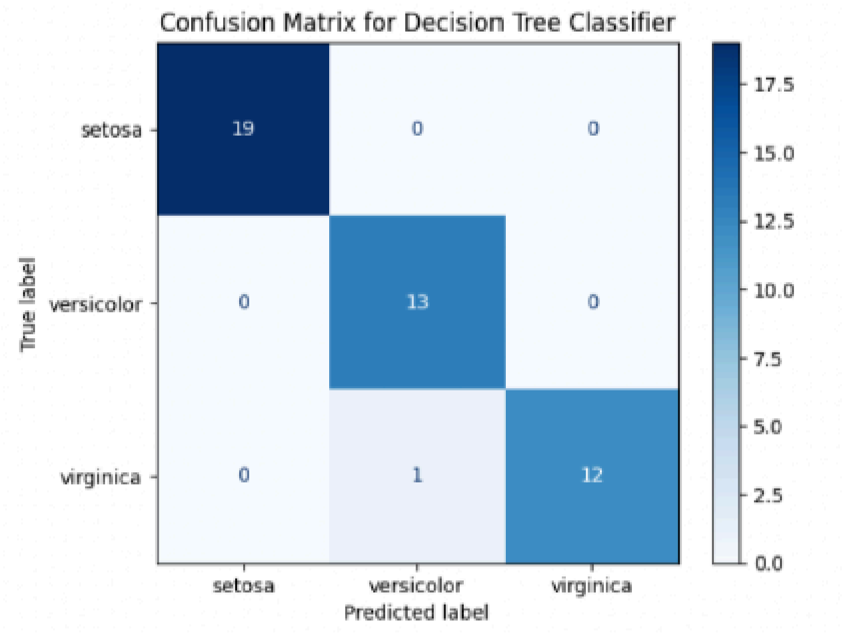
```
from sklearn.metrics import accuracy score,
precision_score,
recall score, fl score
# Predict on the test set
y_preddt = dt_clf.predict(X_test
# Evaluate the model
accuracy_dt = accuracy_score(Y_test, y pred dt)
precision_dt = precision_score(Y_test, y preddt
average='weighted"')
recall _dt = recall score(Y_test, y pred dt,
average='weighted')
fl dt = fl_score(Y_test, y pred dt,
average='weighted')
print (f"Decision Tree Classifier Performance:")
print (f"Accuracy: {accuracy_dt:.4f}1")
print (f"Precision: {precision_dt:.4f}")
print (f"Recall: {recall dt:.4f})")
print (f"Fl-score: {f1 dt:.4f}")
print (“Done By Devesh Patil”)
```

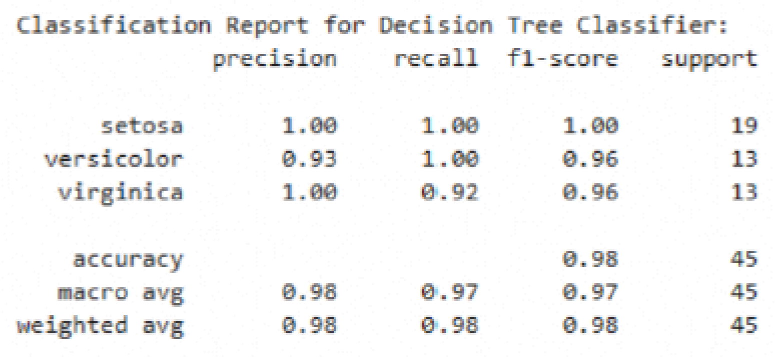


```
Decision Tree Classifier Performance:
Accuracy: 0.9778
Precision: 0.9794
Recall: 0.9778
F1-score: 0.9777
```

```
from sklearn.metrics import confusion matrix,
classification_report, ConfusionMatrixDisplay
cm = confusion matrix(Y test, y pred dt)
disp = ConfusionMat(rcoinfxusDioin msatprlixa=cym,
display labels=iris.tanragmeest)
disp.plot (cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Decision Tree
Classifier")
plt.show()
```

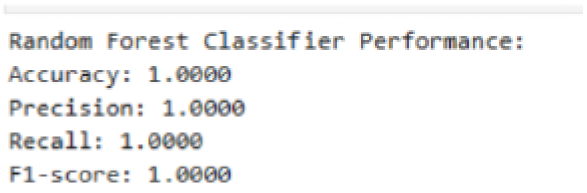
```
print ("Classification Report for Decision Tree
Classifie
print(classification_report (Y_test, y_pred dt,
target names=iris.target names))
```





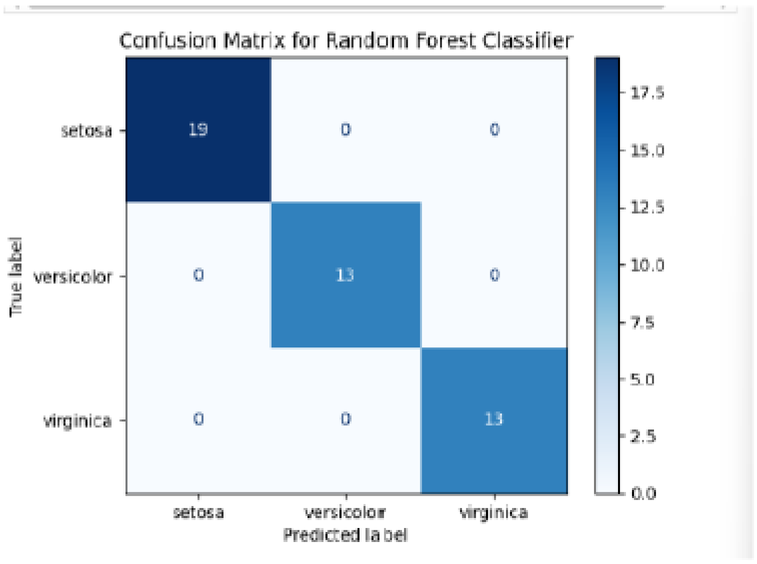
```
Classification Report for Decision Tree Classifier:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       0.93      1.00      0.96        13
   virginica       1.00      0.92      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
```

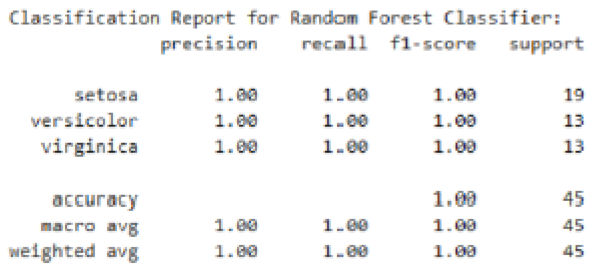
```
rf clf = RandomForestClassifier(randomfistate:42)
#default DT=100
rf clf.fit(X_train, Y train)
# Access the feature importances
rf_feature importances = rf clf.feature importances_
from sklearn.metrics import accuracy score,
precision_score,
recall score, fl score
y_prerdf = rf clf.predict(X_test
accuracy_rf = accuracy_score(Y_test, y pred rf)
precisiornf = precision_score(Y_test, y prerdf
average='weighted"')
recalrlf = recall score(Y_test, y pred rf,
average='weighted')
fl_rf = fl_score(Y_test, y pred rf,
average='weighted')
print (f"Random Forest Classifier Performance:")
print (f"Accuracy: {accuracy_rf:.4f}1")
print (f"Precision: {precision_rf:.4f}")
print (f"Recall: {recall rf:.4f}")
print (f"Fl-score: {f1 rf:.4£}")
```



```
Random Forest Classifier Performance:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-score: 1.0000
```

```
from sklearn.metrics import confusion matrix,
classification_report, ConfusionMatrixDisplay
cm = confusion matrix(Y test, y pred rf)
disp = ConfusionMatrixDisplay (confusion matrix=cm,
display labels=iris.target names)
disp.plot (cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Random Forest
Classifier")
plt.show()
print ("Classification Report for Random Forest
Classifier:")
print (classification_report(Y_test, y pred rf,
target names=iris.target names))
```





```
Classification Report for Random Forest Classifier:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Conclusion: The Random Forest Classifier outperforms the Decision Tree Classifier across all metrics. It provides a more robust and generalized model by combining multiple decision trees, which minimizes errors due to variance or overfitting.

Practical No: 11

Aim: Implementation of Feedforward Neural Network for Handwritten Digit Recognition using MNIST Dataset

Theory:

TensorFlow is an open-source framework for machine learning (ML) and artificial intelligence (AI) that was developed by Google Brain. It was designed to facilitate the development of machine learning models, particularly deep learning models by providing tools to build, train and deploy them across different platforms. It supports a wide range of applications from natural language processing (NLP) and computer vision (CV) to time series forecasting and reinforcement learning.

TensorFlow operates on a computational graph architecture where nodes represent mathematical operations and edges represent tensors flowing between them. This graph-based execution model allows TensorFlow to perform efficient parallel computation across multiple CPUs, GPUs, and TPUs, making it highly scalable for large-scale machine learning tasks.

The framework supports both low-level and high-level APIs. The low-level API provides fine-grained control over operations, while the high-level API, primarily implemented through Keras, simplifies the process of building and training neural networks. TensorFlow 2.x introduced eager execution, which allows operations to execute immediately as they are called, improving usability and debugging compared to the static graph execution of earlier versions.

TensorFlow offers extensive modules for handling data, model optimization, and deployment. It includes tf.data for data input pipelines, tfkeras for model construction, and tf.distribute for distributed training. TensorFlow also supports model deployment through the TensorFlow Serving and TensorFlow Lite frameworks, enabling seamless integration across mobile, web, and cloud environments.

The MNIST (Modified National Institute of Standards and Technology) dataset is one of the most widely used benchmark datasets in the field of machine learning and deep learning. It is primarily used for training and testing image processing systems, particularly in the domain of handwritten digit recognition. The dataset serves as a foundational standard for evaluating the performance of various classification algorithms and neural network architectures.

The MNIST dataset consists of 70,000 grayscale images of handwritten digits from 0 to 9, each image having a fixed size of 28 x 28 pixels, resulting in a total of 784 input features per image. Out of these, 60,000 images are designated for training, and 10,000 images are reserved for testing purposes. Each image in the dataset is accompanied by a corresponding label that indicates the correct digit class, enabling supervised learning.

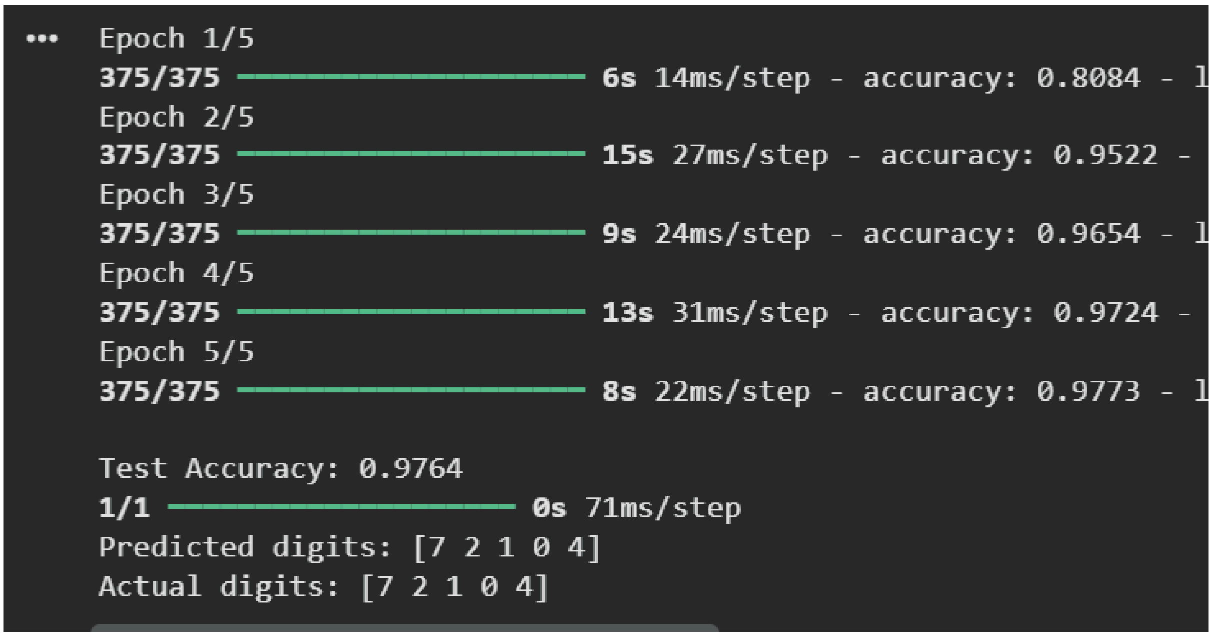
Code & Output:

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,
BatchNormalization, Dropout, Flatten, Activation
import matplotlib.pyplot as plt
import numpy as np

# 1. MNIST Dataset load karein [cite: 1561, 1562]
(x_train, y_train), (x_test, y_test) =
mnist.load_data()

# 2. Preprocessing: Pixel values ko [0,1] ke beech
normalize karein [cite: 1565]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# 3. Model Architecture taiyar karein [cite: 1570,
1571]
model = Sequential([
    Flatten(input_shape=(28, 28)), # 2D image ko 1D
vector mein badalna
    Dense(256, activation='relu'), # 1st Hidden
Layer [cite: 1573, 1575]
    Dropout(0.3),                  # Overfitting
rokne ke liye [cite: 1576]
    Dense(128, activation='relu'), # 2nd Hidden
Layer [cite: 1577, 1578]
    Dense(10, activation='softmax') # Output Layer
(10 digits ke liye)
])

# 4. Model Compile karein [cite: 1588, 1589]
```

```
model.compile(optimizer='adam',

loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 5. Training [cite: 1590]
history = model.fit(x_train, y_train, epochs=5,
batch_size=128, validation_split=0.2)

# 6. Evaluation aur Predictions [cite: 1681, 1691]
test_loss, test_acc = model.evaluate(x_test, y_test,
verbose=0)
print(f"\nTest Accuracy: {test_acc:.4f}")

# Example Prediction dikhayein [cite: 1698]
predictions = model.predict(x_test[:5])
print("Predicted digits:", np.argmax(predictions,
axis=1))
print("Actual digits:", y_test[:5])

Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s
14ms/step - accuracy: 0.8084 - loss: 0.6450 -
val_accuracy: 0.9582 - val_loss: 0.1429
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15s
27ms/step - accuracy: 0.9522 - loss: 0.1598 -
val_accuracy: 0.9683 - val_loss: 0.1060
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9s
24ms/step - accuracy: 0.9654 - loss: 0.1113 -
val_accuracy: 0.9722 - val_loss: 0.0989
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13s
31ms/step - accuracy: 0.9724 - loss: 0.0909 -
val_accuracy: 0.9761 - val_loss: 0.0811
Epoch 5/5
```

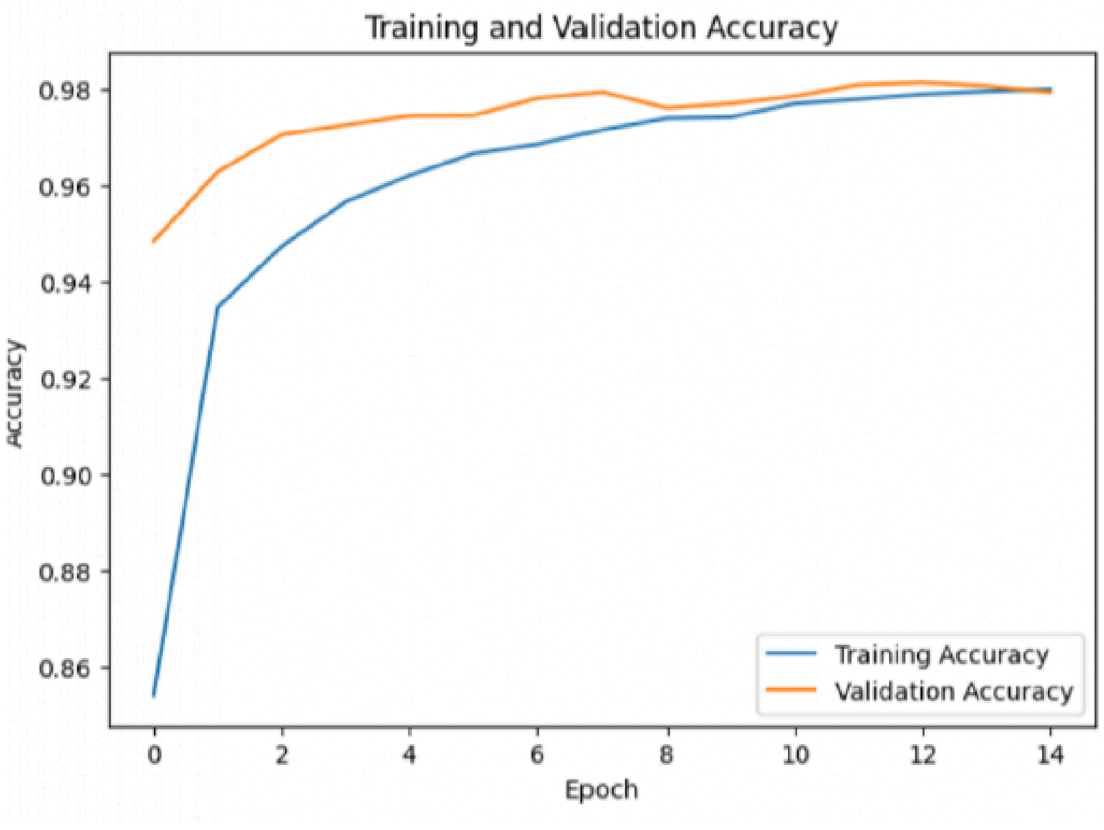
```
375/375 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8s
22ms/step - accuracy: 0.9773 - loss: 0.0735 -
val_accuracy: 0.9772 - val_loss: 0.0818

Test Accuracy: 0.9764
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
Predicted digits: [7 2 1 0 4]
Actual digits: [7 2 1 0 4]
```

Or

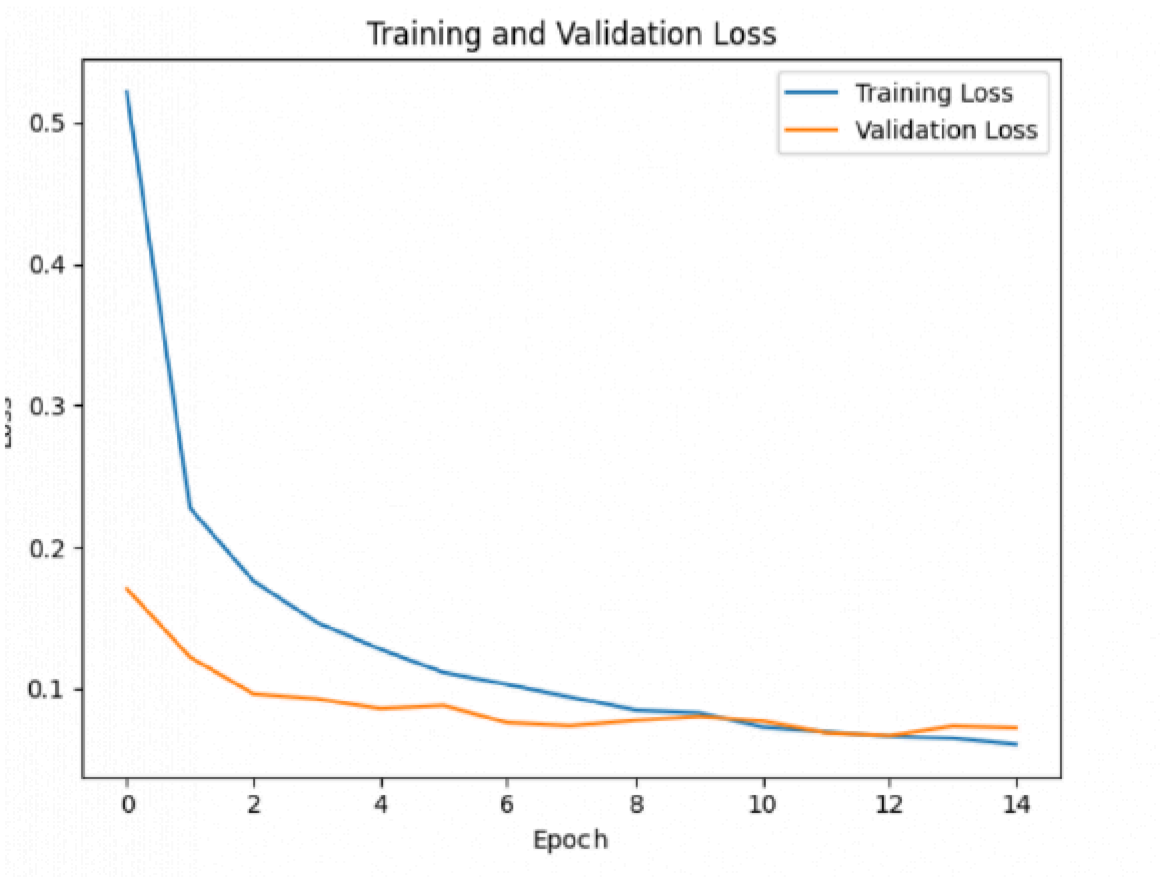


```
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - accuracy: 0.8084 - l
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 15s 27ms/step - accuracy: 0.9522 -
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.9654 - l
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 13s 31ms/step - accuracy: 0.9724 -
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 8s 22ms/step - accuracy: 0.9773 - l

Test Accuracy: 0.9764
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
Predicted digits: [7 2 1 0 4]
Actual digits: [7 2 1 0 4]
```

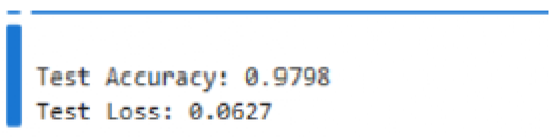
```
plt.figure (figsize=(16, 5))
plt.subplot(1, 2, 1)
plt.plot (history.history['accuracy'],
label='Training Accuracy')
plt.plot (history.history['val accuracy'],
label='Validation
Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel ("Epoch')
plt.ylabel ('Accuracy')
plt.legend()
```



```
plt.figure (figsize=(12, 5))
plt.subplot(l, 2, 2)
plt.plot (history.history['loss'], label='Training
Loss')
plt.plot (history.history['val loss'],
label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel ("Epoch')
plt.ylabel('Loss'")
plt.legend()
plt.tight_layout ()
plt.show()
```



```
test_loss,
verbose=0)
test_accuracy
print (f"\nTest Accuracy:
print (f"Test Loss: {test_
= model.evaluate(x_test, y_t est,
{test_accuracy:.4f}")
loss:.4f}1")
```

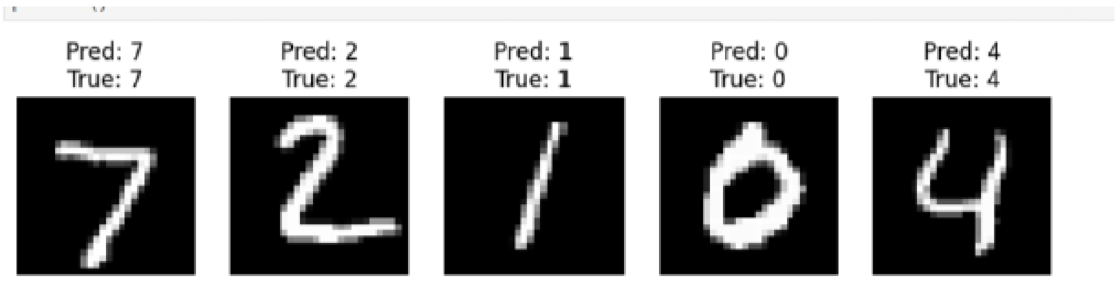


```
Test Accuracy: 0.9798
Test Loss: 0.0627
```

```
y_pred =
y_pred classes =
model.predict (x_
np.argmax(y_pred,
0.0627
test)
axis=1)
y_test _classes = np.argmax(y test, axis=1)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
```

```
plt.figure (figsize=(10, 2))
for 1 in range(5):
plt.subplot(l, 5, i+1)
plt.imshow(x_test[i].reshape (28, 28), cmap='gray')
plt.title(f'Pred: {y_pred classes[i]}\nTrue:
{y_test_classes[i]}")
plt.axis ('off")
plt.show ()
```



Conclusion: The implementation of a Feedforward Neural Network on the MNIST dataset demonstrates the model’s ability to accurately recognize handwritten digits. The study highlights the effectiveness of neural networks in pattern recognition tasks. This approach provides a foundation for more advanced deep learning techniques in image classification applications.

Practical No: 12

Aim:Implementation of Bayesian Optimization

Theory :

Bayesian Optimization is a strategy for optimizing expensive-to-evaluate functions. It operates by building a probabilistic model of the objective function and using this model to select the most promising points to evaluate next. This approach is particularly useful in scenarios where the objective function is unknown, noisy, or costly to evaluate, as it aims to minimize the number of evaluations required to find the optimal solution.The optimization process involves two main components:

- Surrogate Model: A probabilistic model (often a Gaussian Process) that approximates the objective function.
- Acquisition Function: A utility function that guides the selection of the next point to evaluate based on the surrogate model.

Key Concepts in Bayesian Optimization

- Gaussian Process (GP): A Gaussian Process is a non-parametric model that defines a distribution over functions. In Bayesian Optimization, GPs are often used as the surrogate model because they provide not only an estimate of the objective function but also a measure of uncertainty.
- Acquisition Functions:
- Expected Improvement (EI): A popular acquisition function that selects points where the expected improvement over the current best solution is maximized.

- Probability of Improvement (PI): Chooses points with the highest probability of improving the current best solution.

- Upper Confidence Bound (UCB): Balances exploration and exploitation by selecting points based on a confidence interval around the GP prediction.

- Exploration vs. Exploitation: Exploration involves searching in areas of the search space with high uncertainty, while exploitation focuses on areas where the surrogate model predicts good outcomes.

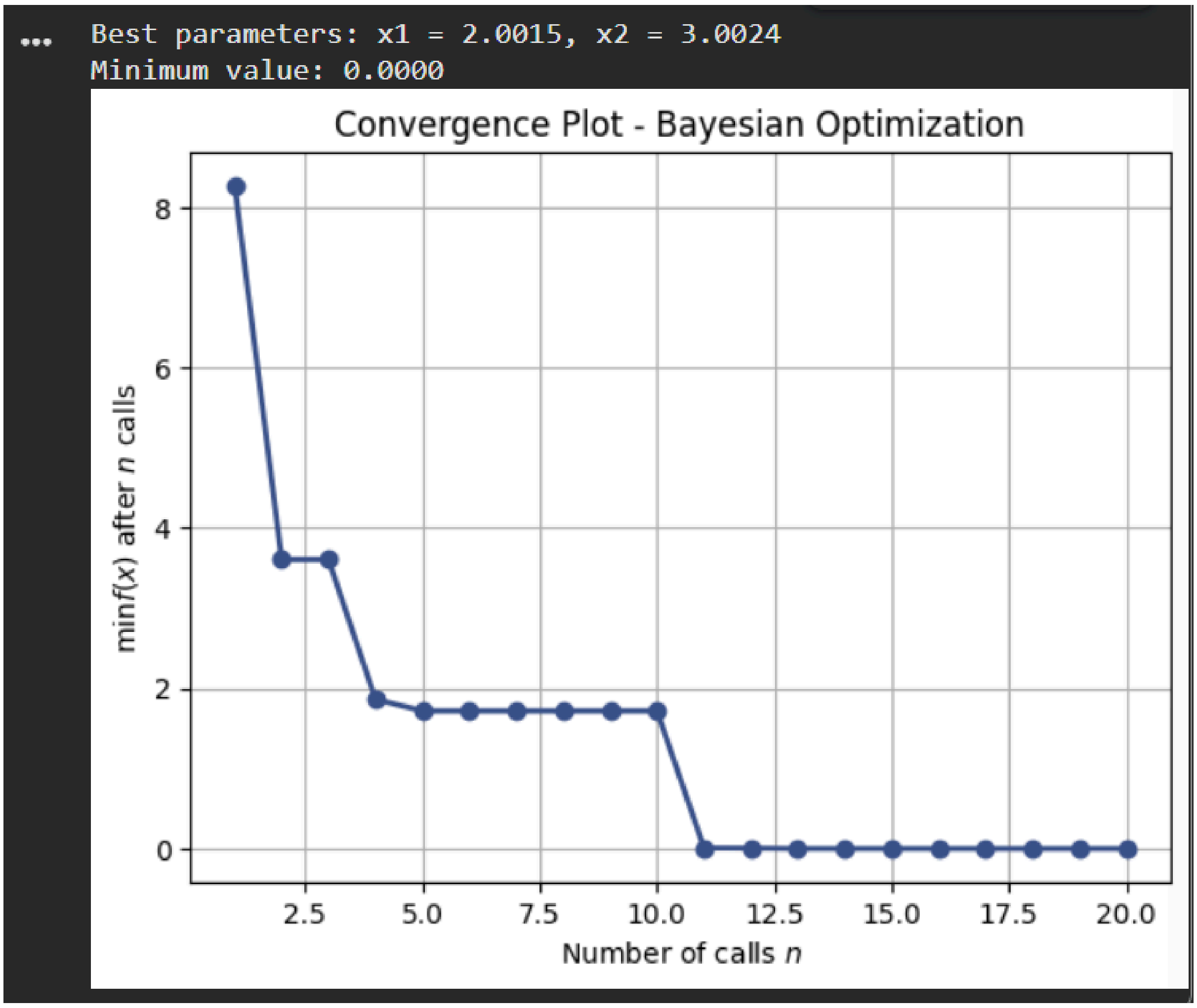
Advantages of Bayesian Optimization

- Efficiency: Bayesian Optimization is highly efficient in finding the optimum with a minimal number of evaluations, making it ideal for expensive or time-consuming objective functions.

Code & Output :

```
import numpy as np
import matplotlib.pyplot as plt

from skopt import gp_minimize
from skopt.space import Real
from skopt.plots import plot_convergence

# 1. Objective function (to minimize)
def objective_function(x):
    # (x1 - 2)^2 + (x2 - 3)^2
    return (x[0] - 2)**2 + (x[1] - 3)**2

# 2. Search space
space = [
    Real(0.0, 5.0, name='x1'),
    Real(0.0, 5.0, name='x2')
]

# 3. Bayesian Optimization
result = gp_minimize(
    objective_function,
    space,
    n_calls=20,
    random_state=42
)

# 4. Best parameters and minimum value
print(f"Best parameters: x1 = {result.x[0]:.4f}, x2
= {result.x[1]:.4f}")
print(f"Minimum value: {result.fun:.4f}")

# 5. Convergence plot
plot_convergence(result)
```

```
plt.title("Convergence Plot - Bayesian
Optimization")
plt.show()
```



```
Best parameters: x1 = 2.0015, x2 = 3.0024
Minimum value: 0.0000
```