Sr. No.	Practical	Signature
1	To implement various descriptive statistics methods 1.1) central tendency, quartile and interquartile 1.2) univariate, bivariate and multivariate.	
2	To implement data cleaning 2.1) Removing leading or lagging spaces from a data entry 2.2) Removing nonprintable characters from a data entry 2.3) Data cleaning: handling missing values, type conversion, data transformations, removing duplicates. 2.4) To detect outliers in the given data.	
3	Regression Analysis 3.1) To perform regression analysis using single linear regression. 3.2) To perform regression analysis using multiple linear regression. 3.3) To perform logistic regression analysis	
4	Classification 4.1) To implement classification using decision tree induction 4.2) To implement classification using Naïve Bayes algorithm 4.3) To implement classification using decision tree induction with various attribute selection methods(Information Gain, Gini index and Gain ratio)	
5	Clustering Algorithm 5.1) To implement clustering using K-Means Algorithm 5.2) To perform hierarchical clustering	
6	To implement PCA (Principal Component Analysis).	
7	To explore the given data and identify the patterns in it.	
8	8.1) To evaluate binary classification model using confusion matrix along with precision and recall. 8.2) To evaluate multi-class classification model using confusion matrix along with precision and recall.	

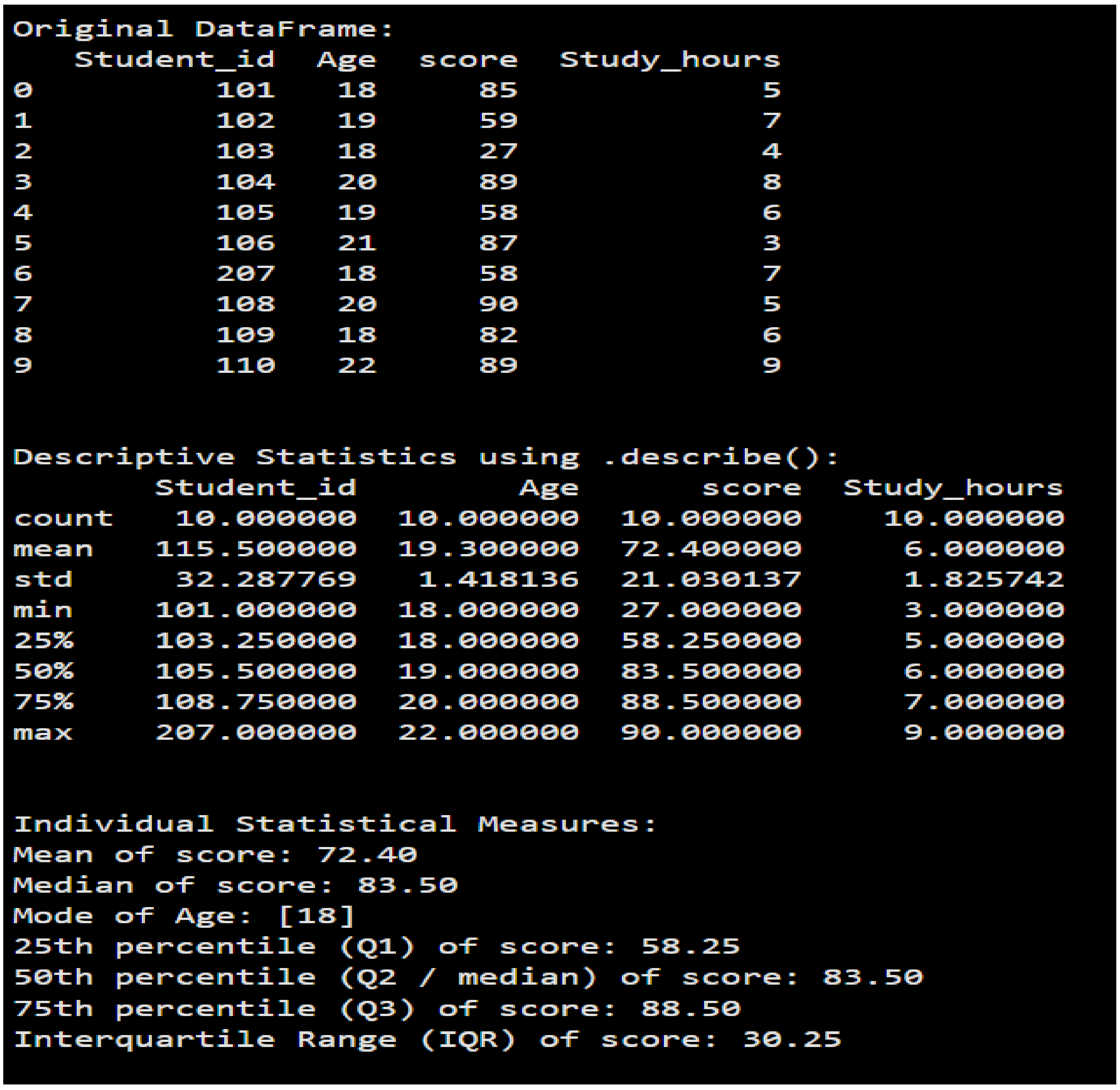
9.	Use an appropriate dataset and create a supervised learning model, Analyse the model with ROC-AUC.	
10.	Consider a case study problem and implement an appropriate model and evaluate it.	
11.	Write a program to implement 11.1 Bagging and boosting model. 11.2 Cross validation methods	

Aim: To implement various descriptive statistics methods

1.1)central tendency, quartile and interquartile

```
import pandas as pd

import numpy as np

# Ensure all columns have exactly 10 entries
Data = {
   'Student_id': [101, 102, 103, 104, 105, 106, 207, 108, 109, 110],
   'Age': [18, 19, 18, 20, 19, 21, 18, 20, 18, 22],
   'score': [85, 59, 27, 89, 58, 87, 58, 90, 82, 89],
   'Study_hours': [5, 7, 4, 8, 6, 3, 7, 5, 6, 9]
}
df = pd.DataFrame(Data)
print("Original DataFrame:")
print(df)
print("\n")
print("Descriptive Statistics using .describe():")
print(df.describe())
print("\n")
print("Individual Statistical Measures:")

# Central tendency
print(f"Mean of score: {df['score'].mean():.2f}")
print(f"Median of score: {df['score'].median():.2f}")
print(f"Mode of Age: {df['Age'].mode().tolist()}")

#  Quartiles
print(f"25th percentile (Q1) of score: {df['score'].quantile(0.25):.2f}")
print(f"50th percentile (Q2 / median) of score: {df['score'].quantile(0.50):.2f}")
print(f"75th percentile (Q3) of score: {df['score'].quantile(0.75):.2f}")

# Interquartile Range
iqr_score = df['score'].quantile(0.75) - df['score'].quantile(0.25)
print(f"Interquartile Range (IQR) of score: {iqr_score:.2f}")
```

OUTPUT:



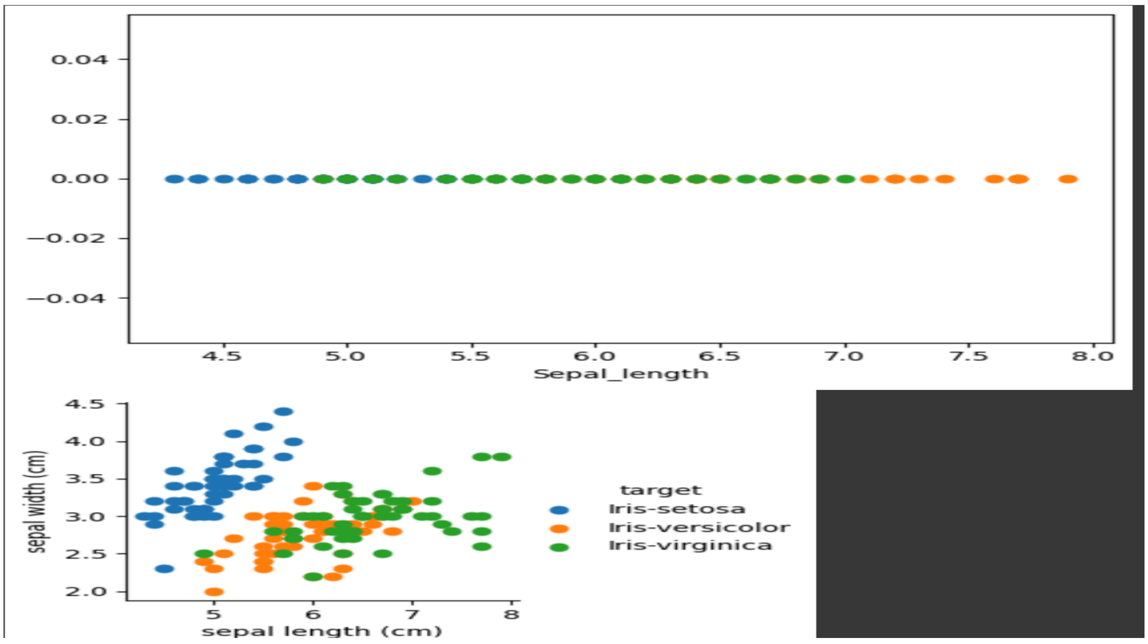
```
Original DataFrame:
   Student_id  Age  score  Study_hours
0         101   18     85            5
1         102   19     59            7
2         103   18     27            4
3         104   20     89            8
4         105   19     58            6
5         106   21     87            3
6         207   18     58            7
7         108   20     90            5
8         109   18     82            6
9         110   22     89            9


Descriptive Statistics using .describe():
       Student_id        Age      score  Study_hours
count   10.000000  10.000000  10.000000    10.000000
mean   115.500000  19.300000  72.400000     6.000000
std     32.287769   1.418136  21.030137     1.825742
min    101.000000  18.000000  27.000000     3.000000
25%    103.250000  18.000000  58.250000     5.000000
50%    105.500000  19.000000  83.500000     6.000000
75%    108.750000  20.000000  88.500000     7.000000
max    207.000000  22.000000  90.000000     9.000000


Individual Statistical Measures:
Mean of score: 72.40
Median of score: 83.50
Mode of Age: [18]
25th percentile (Q1) of score: 58.25
50th percentile (Q2 / median) of score: 83.50
75th percentile (Q3) of score: 88.50
Interquartile Range (IQR) of score: 30.25
```

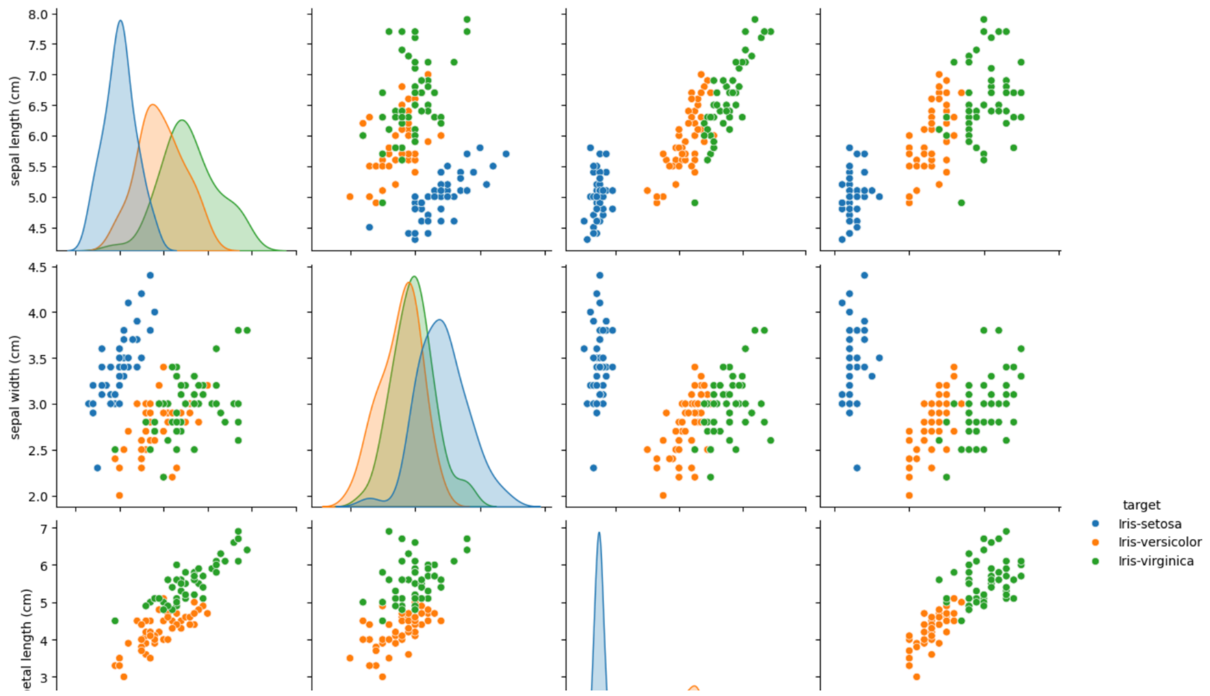
Aim: To implement various descriptive statistics methods

1.2) univariate, bivariate and multivariate.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("/content/Iris.csv")

#univariate
df_setosa=df.loc[df['Species']=='Iris-setosa']
df_virginica=df.loc[df['Species']=='Iris-virginica']
df_versicolor=df.loc[df['Species']=='Iris-versicolor']
plt.plot(df_setosa['Sepal_length'],np.zeros_like(df_setosa['Sepal_length']),'o')
plt.plot(df_virginica['Sepal_length'],np.zeros_like(df_virginica['Sepal_length']),'o')
plt.plot(df_versicolor['Sepal_length'],np.zeros_like(df_versicolor['Sepal_length']),'o')
plt.xlabel("Sepal_length")
plt.show()

#bivariate
sns.FacetGrid(df,hue='Species').map(plt.scatter,'Sepal_length','Sepal_width').add_legend()
plt.show()

#Multivariate
sns.pairplot(df,hue='Species',size=3)
plt.show()
```

OUTPUT:



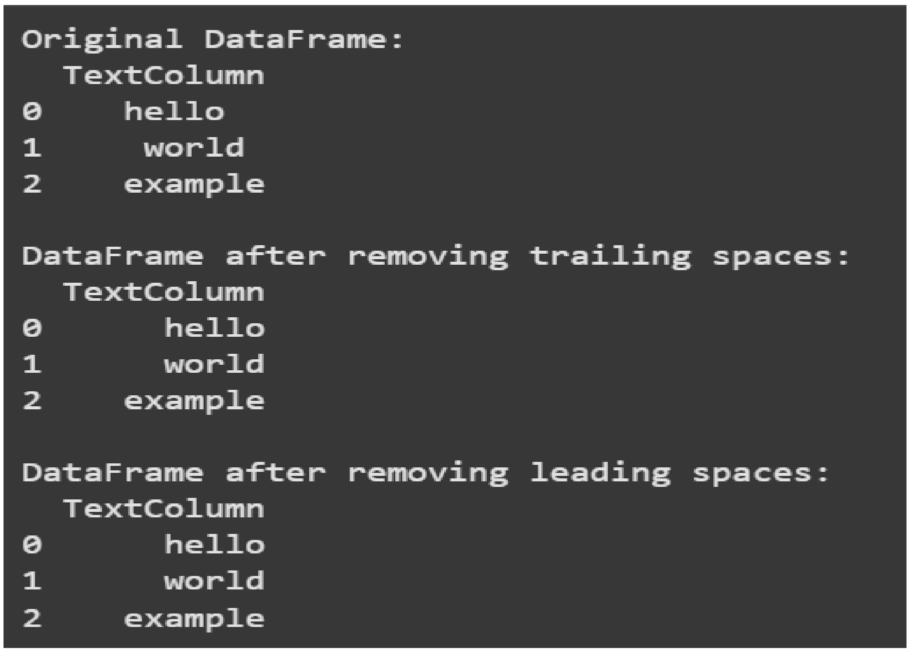


Ashish Ashtekar 411

Aim: To implement data cleaning
2.1) Removing leading or lagging spaces from a data entry

```
# Create a sample DataFrame with leading/trailing spaces
data = {'TextColumn': ['  hello  ', 'world ', '  example']}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)

# Remove trailing spaces from the 'TextColumn'
df['TextColumn'] = df['TextColumn'].str.rstrip()

print("\nDataFrame after removing trailing spaces:")
print(df)

# Remove leading spaces from the 'TextColumn'
df['TextColumn'] = df['TextColumn'].str.lstrip()

print("\nDataFrame after removing leading spaces:")
print(df)
```

OUTPUT:

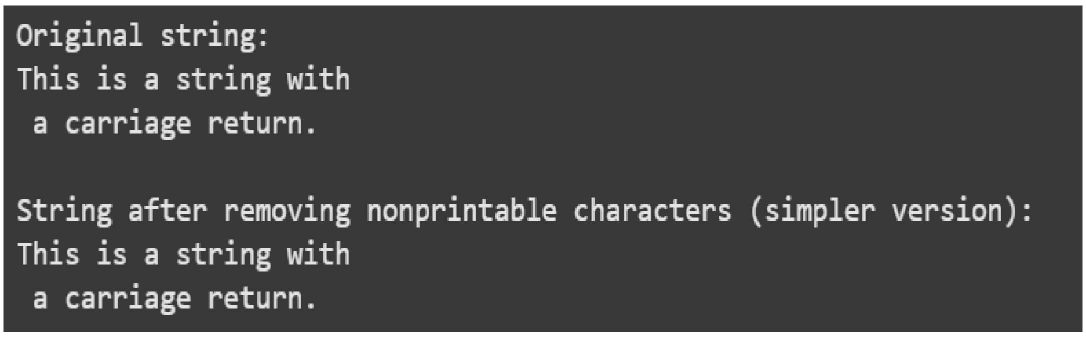


```
Original DataFrame:
  TextColumn
0    hello
1     world
2    example

DataFrame after removing trailing spaces:
  TextColumn
0      hello
1      world
2    example

DataFrame after removing leading spaces:
  TextColumn
0      hello
1      world
2    example
```

2.2) Removing nonprintable characters from a data entry

```
import string
# Create a set of printable characters
printable = set(string.printable)

# Sample data with nonprintable characters
data_with_nonprintable = "This is a string with \n a newline and \r a carriage return."

print("Original string:")
print(data_with_nonprintable)

# Remove nonprintable characters using a list comprehension
cleaned_data = ''.join([char for char in data_with_nonprintable if char in printable])

print("\nString after removing nonprintable characters (simpler version):")
print(cleaned_data)
```

OUTPUT:



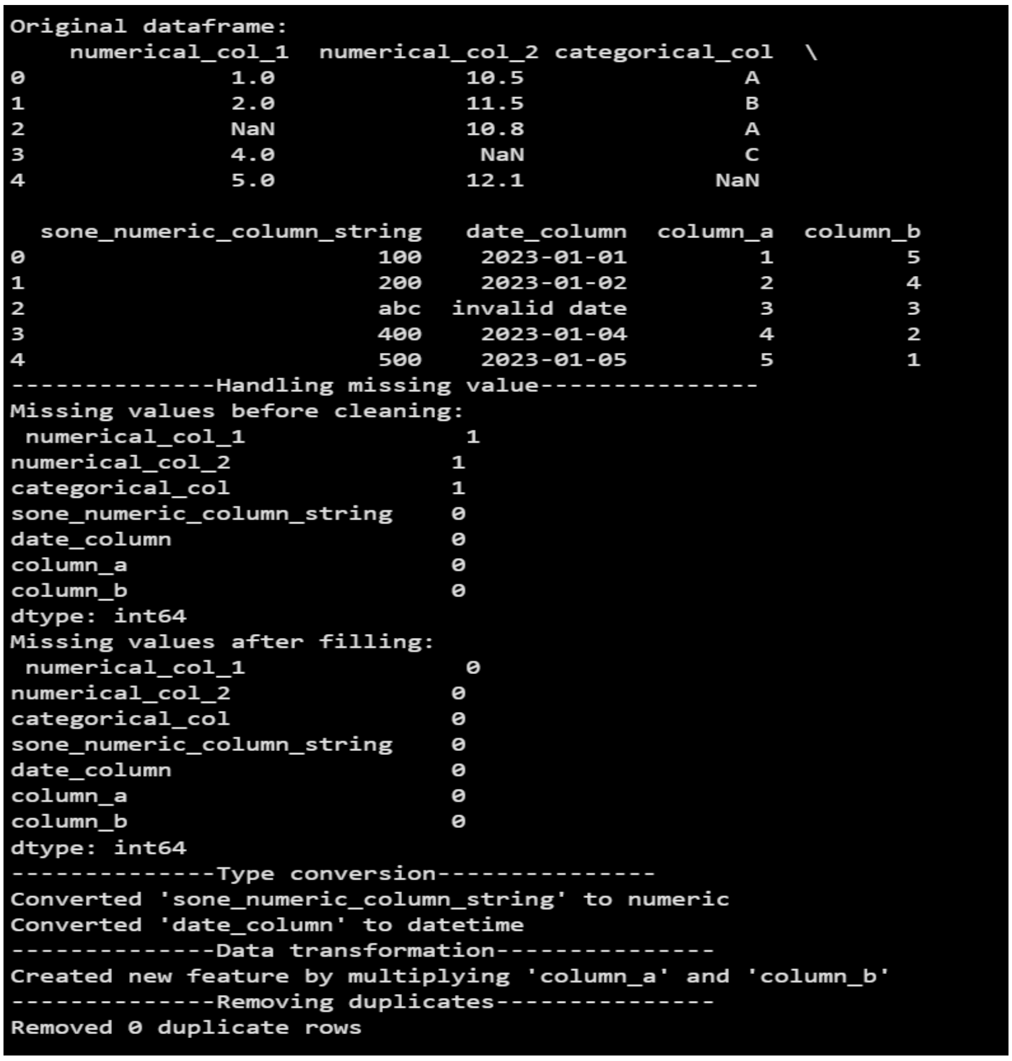
```
Original string:
This is a string with
 a carriage return.

String after removing nonprintable characters (simpler version):
This is a string with
 a carriage return.
```

2.3) Data cleaning: handling missing values, type conversion, data transformations, removing duplicates.

```
import pandas as pd
import numpy as np
def clean_dataset(df):
    print("--------------Handling missing value---------------")
    print("Missing values before cleaning:\n", df.isnull().sum())

    # 1. Fill numeric missing values
    for col in df.select_dtypes(include=np.number).columns:
        if df[col].isnull().any():
            df[col] = df[col].fillna(df[col].mean())

    # Fill categorical missing values
    for col in df.select_dtypes(include='object').columns:
        if df[col].isnull().any():
            df[col] = df[col].fillna(df[col].mode()[0])

    print("Missing values after filling:\n", df.isnull().sum())

    # 2. Type conversion
    print("--------------Type conversion---------------")
    if 'sone_numeric_column_string' in df.columns:
        df['sone_numeric_column_string'] = pd.to_numeric(
            df['sone_numeric_column_string'], errors='coerce')
        df['sone_numeric_column_string'] = df['sone_numeric_column_string'].fillna(
            df['sone_numeric_column_string'].mean())
        print("Converted 'sone_numeric_column_string' to numeric")

    if 'date_column' in df.columns:
        df['date_column'] = pd.to_datetime(df['date_column'], errors='coerce')
        print("Converted 'date_column' to datetime")

    # 3. Data transformation
    print("--------------Data transformation---------------")
    if 'column_a' in df.columns and 'column_b' in df.columns:
        df['new_feature'] = df['column_a'] * df['column_b']
        print("Created new feature by multiplying 'column_a' and 'column_b'")

    # 4. Removing duplicates
    print("--------------Removing duplicates---------------")
    initial_rows = len(df)
    df.drop_duplicates(inplace=True)
    print(f"Removed {initial_rows - len(df)} duplicate rows")

    return df
```

```
if __name__ == '__main__':
    data = {
        'numerical_col_1': [1, 2, np.nan, 4, 5],
        'numerical_col_2': [10.5, 11.5, 10.8, np.nan, 12.1],
        'categorical_col': ['A', 'B', 'A', 'C', np.nan],
        'sone_numeric_column_string': ['100', '200', 'abc', '400', '500'],
        'date_column': ['2023-01-01', '2023-01-02', 'invalid date', '2023-01-04', '2023-01-05'],
        'column_a': [1, 2, 3, 4, 5],
        'column_b': [5, 4, 3, 2, 1],
    }

    sample_df = pd.DataFrame(data)
    print("Original dataframe:\n", sample_df)

    cleaned_df = clean_dataset(sample_df.copy())
    print("\nCleaned Dataframe:\n", cleaned_df)
```

OUTPUT:

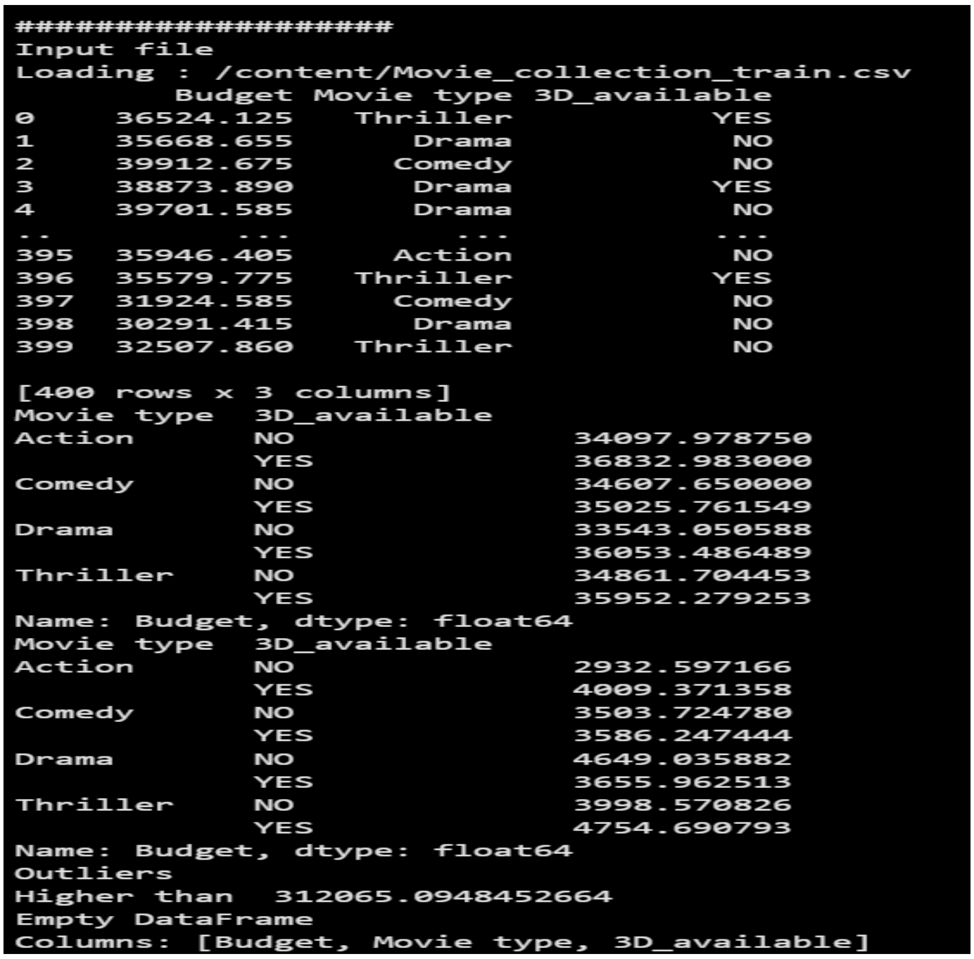


```
Original dataframe:
    numerical_col_1  numerical_col_2 categorical_col  \
0              1.0             10.5               A
1              2.0             11.5               B
2              NaN             10.8               A
3              4.0              NaN               C
4              5.0             12.1             NaN

  sone_numeric_column_string   date_column  column_a  column_b
0                        100    2023-01-01         1         5
1                        200    2023-01-02         2         4
2                        abc  invalid date         3         3
3                        400    2023-01-04         4         2
4                        500    2023-01-05         5         1
--------------Handling missing value---------------
Missing values before cleaning:
 numerical_col_1               1
numerical_col_2               1
categorical_col               1
sone_numeric_column_string    0
date_column                   0
column_a                      0
column_b                      0
dtype: int64
Missing values after filling:
 numerical_col_1               0
numerical_col_2               0
categorical_col               0
sone_numeric_column_string    0
date_column                   0
column_a                      0
column_b                      0
dtype: int64
--------------Type conversion---------------
Converted 'sone_numeric_column_string' to numeric
Converted 'date_column' to datetime
--------------Data transformation---------------
Created new feature by multiplying 'column_a' and 'column_b'
--------------Removing duplicates---------------
Removed 0 duplicate rows
```

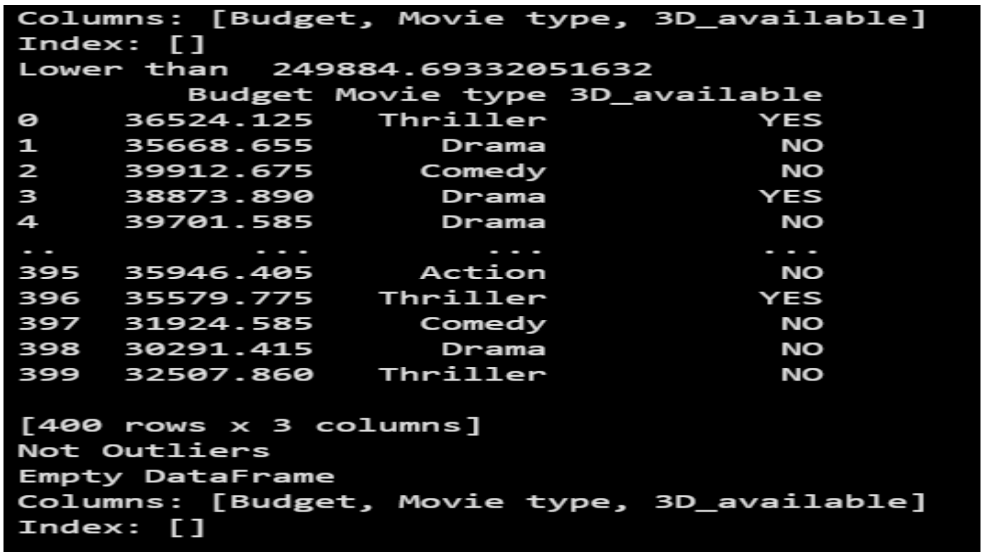
2.4) To detect outliers in the given data.

```
import pandas as pd

import numpy as np

InputFileName='Movie_collection_train.csv'

print('###################')

print("Input file")

sFileName='/content/Movie_collection_train.csv'

print('Loading :',sFileName)

Movie_DATA_ALL = pd.read_csv(sFileName, header=0, usecols=['Genre', '3D_available', 'Budget'],
encoding='latin-1')

Movie_DATA_ALL.rename(columns={'Genre':'Movie type'},inplace=True)

print(Movie_DATA_ALL)

MeanData=Movie_DATA_ALL.groupby(['Movie type','3D_available'])['Budget'].mean()

stdData=Movie_DATA_ALL.groupby(['Movie type','3D_available'])['Budget'].std()

print(MeanData);

print(stdData);

print('Outliers')

UpperBound = float(sum(MeanData) + sum(stdData))

print('Higher than ', UpperBound)

OutliersHigher = Movie_DATA_ALL[Movie_DATA_ALL.Budget > UpperBound]

print(OutliersHigher)

LowerBound = float(sum(MeanData) - sum(stdData))

print('Lower than ', LowerBound)

OutliersLower = Movie_DATA_ALL[Movie_DATA_ALL.Budget < LowerBound]

print(OutliersLower)

print('Not Outliers')

OutliersNot = Movie_DATA_ALL[(Movie_DATA_ALL.Budget > LowerBound) &
(Movie_DATA_ALL.Budget <= UpperBound)]

print(OutliersNot)
```

OUTPUT:



```
####################
Input file
Loading : /content/Movie_collection_train.csv
        Budget Movie type 3D_available
0    36524.125   Thriller          YES
1    35668.655      Drama           NO
2    39912.675     Comedy           NO
3    38873.890      Drama          YES
4    39701.585      Drama           NO
..         ...        ...          ...
395  35946.405     Action           NO
396  35579.775   Thriller          YES
397  31924.585     Comedy           NO
398  30291.415      Drama           NO
399  32507.860   Thriller           NO

[400 rows x 3 columns]
Movie type  3D_available
Action      NO              34097.978750
            YES             36832.983000
Comedy      NO              34607.650000
            YES             35025.761549
Drama       NO              33543.050588
            YES             36053.486489
Thriller    NO              34861.704453
            YES             35952.279253
Name: Budget, dtype: float64
Movie type  3D_available
Action      NO              2932.597166
            YES             4009.371358
Comedy      NO              3503.724780
            YES             3586.247444
Drama       NO              4649.035882
            YES             3655.962513
Thriller    NO              3998.570826
            YES             4754.690793
Name: Budget, dtype: float64
Outliers
Higher than  312065.0948452664
Empty DataFrame
Columns: [Budget, Movie type, 3D_available]
```

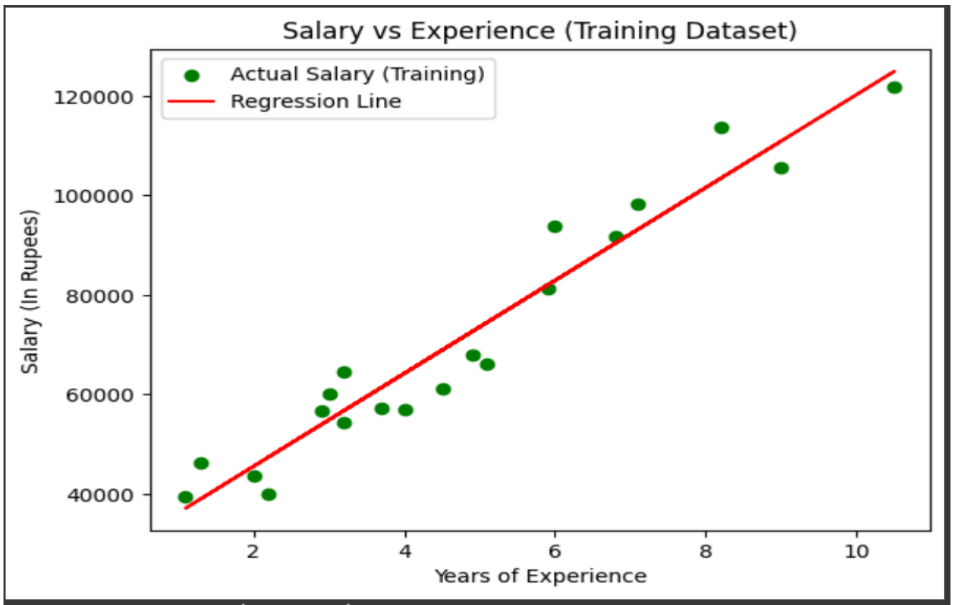


```
Columns: [Budget, Movie type, 3D_available]
Index: []
Lower than  249884.69332051632
        Budget Movie type 3D_available
0    36524.125   Thriller          YES
1    35668.655      Drama           NO
2    39912.675     Comedy           NO
3    38873.890      Drama          YES
4    39701.585      Drama           NO
..         ...        ...          ...
395  35946.405     Action           NO
396  35579.775   Thriller          YES
397  31924.585     Comedy           NO
398  30291.415      Drama           NO
399  32507.860   Thriller           NO

[400 rows x 3 columns]
Not Outliers
Empty DataFrame
Columns: [Budget, Movie type, 3D_available]
Index: []
```

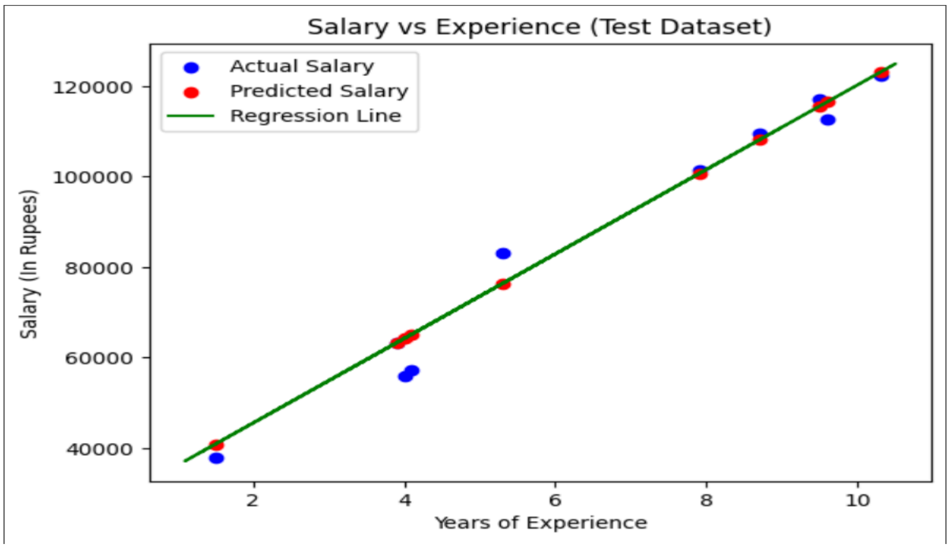
Aim: 3.1) To perform regression analysis using single linear regression.

```
y = data_set.iloc[:, 1].values
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn. linear_model import LinearRegression
from sklearn.metrics import r2_score
# Load dataset
data_set = pd.read_csv("/content/Salary_Data - Salary_Data.csv")
# Independent variable (experience) and dependent variable (salary)
x = data_set.iloc[:, :- 1].values
y = data_set.iloc[:, 1].values
# Split into training and test sets
x_train, x_test, y_train, y_test = train_test_split( x, y, test_size=1/3,
random_state=0)
# Train the model
regressor = LinearRegression()
regressor. fit(x_train, y_train)
y_pred_train = regressor.predict(x_train)
mtp.scatter(x_train, y_train, color="green", label="Actual Salary (Training)")
mtp.plot(x_train, y_pred_train, color="red", label="Regression Line")
mtp.title("Salary vs Experience (Training Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary (In Rupees)")
mtp.legend()
mtp.show()
# Predict test data
y_pred_test = regressor.predict(x_test)
# Find accuracy (R2 score)
accuracy = r2_score(y_test, y_pred_test)
print(f"Test Data Accuracy (R2 Score): {accuracy:.2f}")
# Plot actual vs predicted for test set
mtp.scatter(x_test, y_test, color="blue", label="Actual Salary")
mtp.scatter(x_test, y_pred_test, color="red", label="Predicted Salary")
mtp.plot( x_train, regressor.predict(x_train), color="green",
label="Regression Line")
mtp.title("Salary vs Experience (Test Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary (In Rupees)")
mtp.legend()
mtp.show()
```

OUTPUT:



```
Test Data Accuracy (R2 Score): 0.97
```

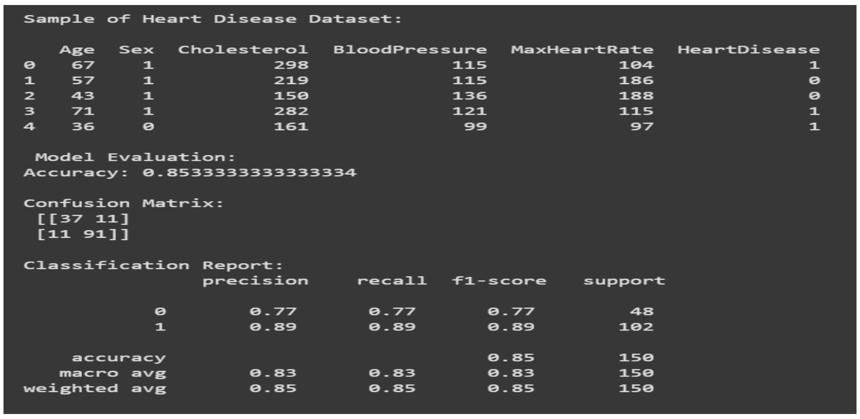


Aim:3.2) To perform regression analysis using multiple linear regression.

Aim:3.3) To perform logistic regression analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn. linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Step 1: Create a dataset (Heart Disease like) as DataFrame
np.random.seed(42)
n_samples = 500
data = pd.DataFrame({
"Age": np.random.randint( 29,77, n_samples),
"Sex": np. random.randint( 0,2, n_samples), # 0 = female, 1 = male
"Cholesterol": np.random.randint(150, 300, n_samples),
"BloodPressure": np.random.randint( 90, 180, n_samples),
"MaxHeartRate": np.random.randint( 90,200, n_samples)
})
# Target variable (rule-based: high Cholesterol, high BP, or low MaxHR + higher risk)
data["HeartDisease"] =((data["Cholesterol"] > 240) |
 (data["BloodPressure"] > 140) |
 (data["MaxHeartRate"] < 120)).astype(int)
print("Sample of Heart Disease Dataset: \n")
print(data.head())
# Step 2: Split features & target
X = data. drop("HeartDisease",axis=1)
y = data["HeartDisease"]
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=42)
# Step 3: Train Logistic Regression
model = LogisticRegression(max_iter=500)
model.fit(X_train, y_train)
# Step 4: Predictions & Evaluation
y_pred = model.predict(X_test)
print("\n Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,y_pred))
```

OUTPUT:

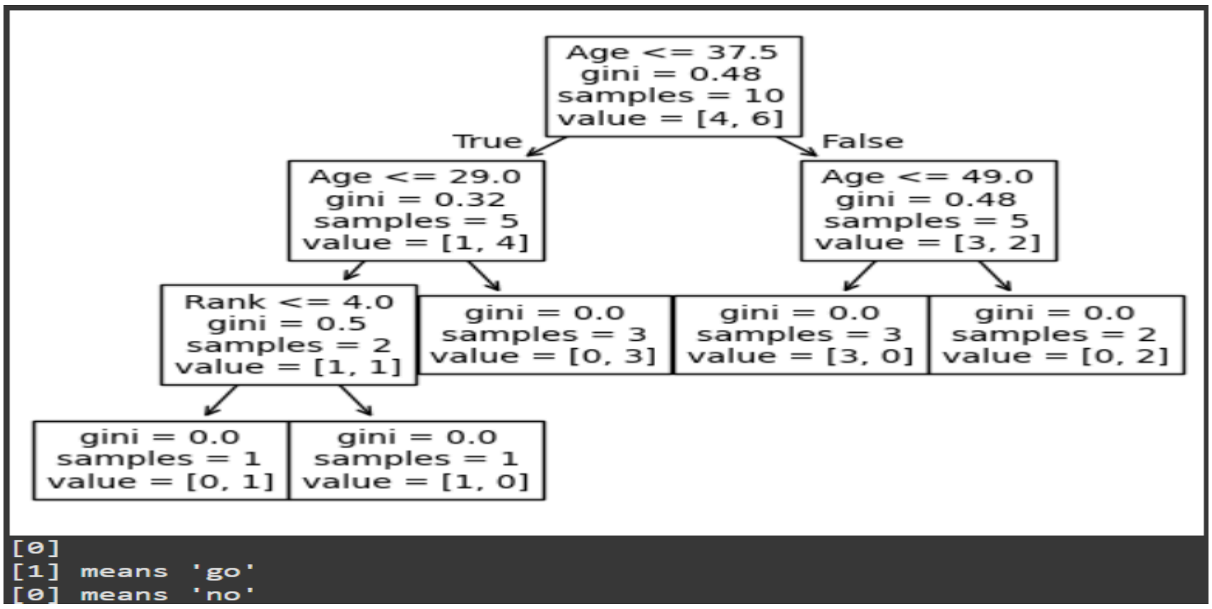


```
Sample of Heart Disease Dataset:

   Age  Sex  Cholesterol  BloodPressure  MaxHeartRate  HeartDisease
0   67    1          298            115           104             1
1   57    1          219            115           186             0
2   43    1          150            136           188             0
3   71    1          282            121           115             1
4   36    0          161             99            97             1

 Model Evaluation:
Accuracy: 0.8533333333333334

Confusion Matrix:
 [[37 11]
 [11 91]]

Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.77      0.77        48
           1       0.89      0.89      0.89       102

    accuracy                           0.85       150
   macro avg       0.83      0.83      0.83       150
weighted avg       0.85      0.85      0.85       150
```

4.1) To implement classification using decision tree induction

```
import pandas as pd
import sys
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import pandas as pd
data = {'Age': [30, 45, 25, 35, 40, 50, 28, 32, 48, 55],
    'Experience': [5, 15, 2, 10, 12, 20, 4, 7, 18, 25],
    'Rank': [7, 9, 3, 8, 7, 10, 5, 6, 9, 10],
    'Nationality': ['UK', 'USA', 'N', 'UK', 'USA', 'UK', 'N', 'USA', 'UK', 'N'],
    'Go': ['YES', 'NO', 'YES', 'YES', 'NO', 'YES', 'NO', 'YES', 'NO', 'YES']}

df = pd.DataFrame(data)
df.to_csv('/content/dataset.csv', index=False)
d={'Uk':0,'USA':1,'N':2}
df['Nationality']=df['Nationality'].map(d)
d={'YES':1,'NO':0}
df['Go']=df['Go'].map(d)
fea=['Age','Experience','Rank','Nationality']
x=df[fea]
y=df['Go']
dtree=DecisionTreeClassifier()
dtree=dtree.fit(x,y)
tree.plot_tree(dtree,feature_names=fea)
plt.savefig("result.png")
plt.show()
print(dtree.predict([[40,10,7,1]]))
print("[1] means 'go'")
print("[0] means 'no'")
```

OUTPUT:



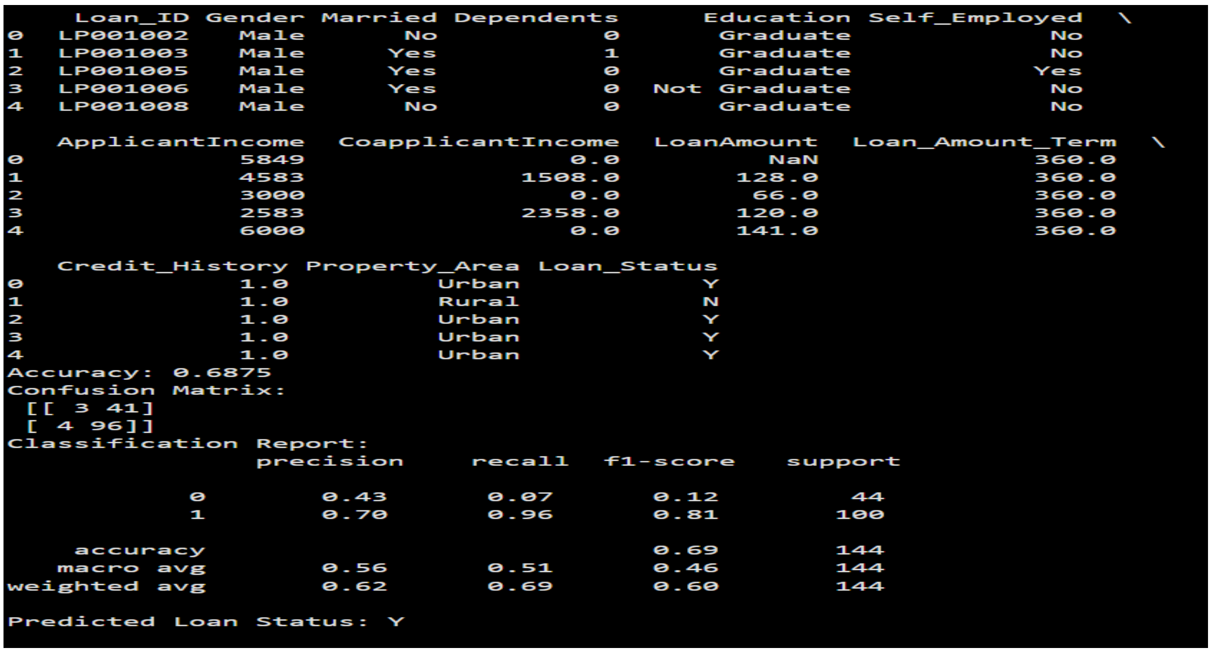
```
[0]
[1] means 'go'
[0] means 'no'
```

4.2) To implement classification using Naïve Bayes algorithm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Load the dataset
data = pd.read_csv('/content/loan.csv')
# View the first few rows
print(data.head())
# Drop rows with missing values (you can also choose to impute)
data.dropna(inplace=True)
# Encode categorical variables directly in original DataFrame to avoid SettingWithCopyWarning
le = LabelEncoder()
for col in ['Gender', 'Married', 'Education', 'Self_Employed']:
  data.loc[:, col] = le.fit_transform(data[col])
# Encode target variable
data['Loan_Status'] = le. fit_transform(data[ 'Loan_Status' ])
# Select features and target after
X = data[['Gender', 'Married', 'Education', 'Self_Employed', 'ApplicantIncome', 'LoanAmount' ]]
```

```
y = data['Loan_Status' ]
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=42)
# Initialize and train Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)
#Predict on test set
y_pred = model.predict(X_test)
#Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
# Create a sample DataFrame with the correct feature names and order
sample = pd.DataFrame ({
'Gender': [1], # Male encoded as 1
'Married': [1], # Yes encoded as 1
'Education': [1], # Graduate encoded as 1
'Self_Employed': [0], # No encoded as 0
'ApplicantIncome': [5000],
'LoanAmount': [128]
})
# Predict the class for the unknown sample or evidence
predicted_class = model.predict(sample)
# Map prediction back to label
loan_status_map = {0: 'N', 1: 'Y'}
print(f"Predicted Loan Status: {loan_status_map[predicted_class[0]]}")
```

OUTPUT:



```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
Accuracy: 0.6875
Confusion Matrix:
 [[ 3 41]
 [ 4 96]]
Classification Report:
               precision    recall  f1-score   support

           0       0.43      0.07      0.12        44
           1       0.70      0.96      0.81       100

    accuracy                           0.69       144
   macro avg       0.56      0.51      0.46       144
weighted avg       0.62      0.69      0.60       144

Predicted Loan Status: Y
```

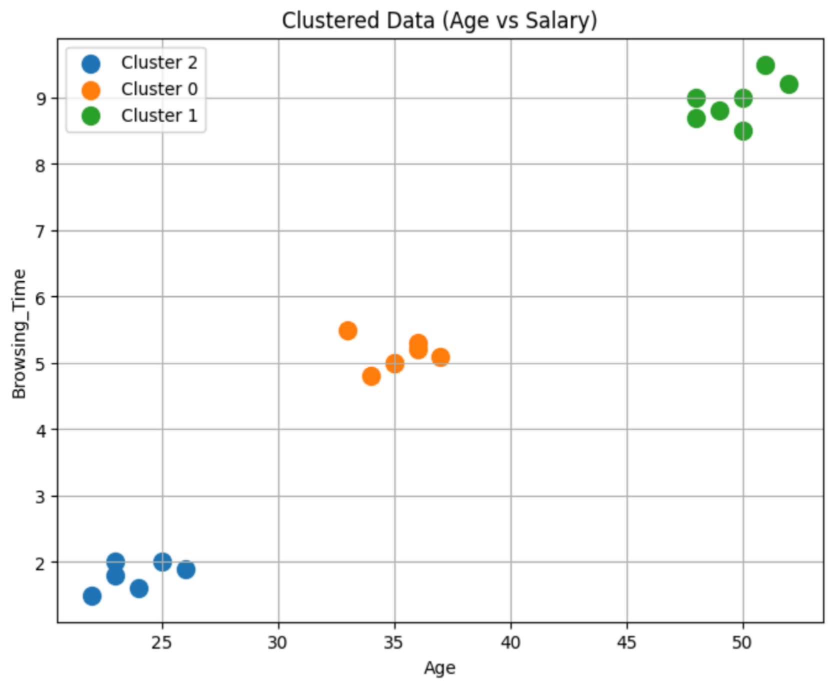
Aim:4.3) To implement classification using decision tree induction with various attribute selection methods(Information Gain, Gini index and Gain ratio)

Aim: 5.1To implement clustering using K-Means Algorithm

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Step 1: Synthetic cluster-friendly dataset
data = {
    'Age': [22, 23, 25, 24, 26,      # Group 1: Young, low salary, short browsing
            35, 36, 34, 33, 37,      # Group 2: Mid-age, mid salary, medium browsing
            48, 50, 52, 49, 51,      # Group 3: Older, high salary, long browsing
            23, 36, 50, 35, 48],     # Mix for variation
    'Salary': [25000, 27000, 26000, 28000, 24000,
               60000, 62000, 58000, 61000, 59000,
               100000, 98000, 105000, 97000, 102000,
               25500, 60500, 101000, 61500, 99000],
    'Browsing_Time': [1.5, 1.8, 2.0, 1.6, 1.9,
                      5.0, 5.2, 4.8, 5.5, 5.1,
                      9.0, 8.5, 9.2, 8.8, 9.5,
                      2.0, 5.3, 9.0, 5.0, 8.7]
}

df = pd.DataFrame(data)
```

```
# Step 2: Scale features
scaler = StandardScaler()
scaled = scaler.fit_transform(df)

# Step 3: KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled)

# Step 4: Plot clusters (Age vs Salary)
plt.figure(figsize=(8, 6))
for cluster in df['Cluster'].unique():
    cluster_data = df[df['Cluster'] == cluster]
    plt.scatter(cluster_data['Age'], cluster_data['Browsing_Time'], label=f'Cluster {cluster}', s=100)

plt.title('Clustered Data (Age vs Salary)')
plt.xlabel('Age')
plt.ylabel('Browsing_Time')
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:

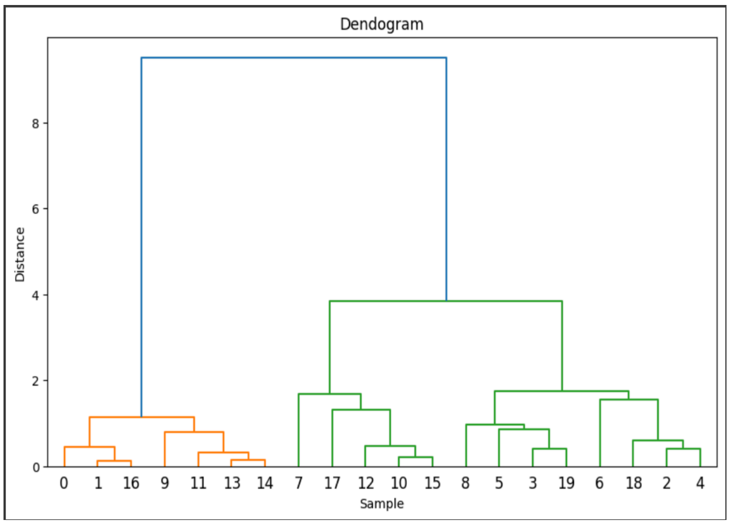


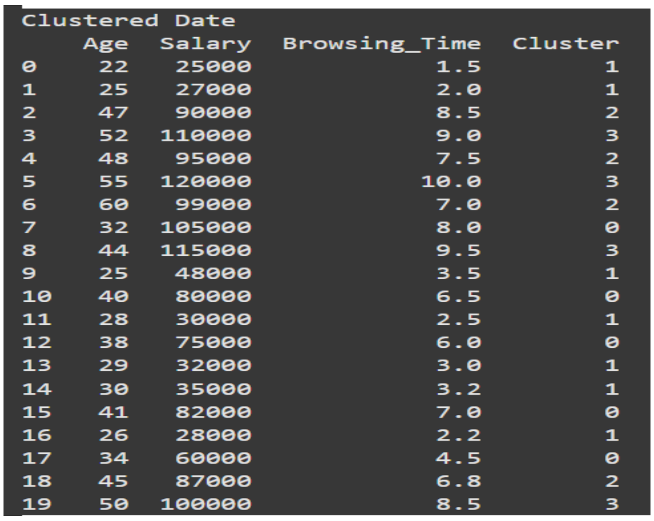
Aim: 5.2) To perform hierarchical clustering

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.cluster.hierarchy import linkage,dendrogram
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering

#Step1:
data={
   'Age':[22,25,47,52,48,55,60,32,44,25,
         40,28,38,29,30,41,26,34,45,50],
   'Salary':[25000,27000,90000,110000,95000,120000,99000,105000,115000,48000,
           80000,30000,75000,32000,35000,82000,28000,60000,87000,100000],
   'Browsing_Time':[1.5,2.0,8.5,9.0,7.5,10.0,7.0,8.0 ,9.5,3.5,
                 6.5,2.5,6.0,3.0,3.2,7.0,2.2,4.5,6.8,8.5]
}
df=pd.DataFrame(data)

#Step2
scaler=StandardScaler()
X_scaled=scaler.fit_transform(df)

#step3
plt.figure(figsize=(10,6))
```

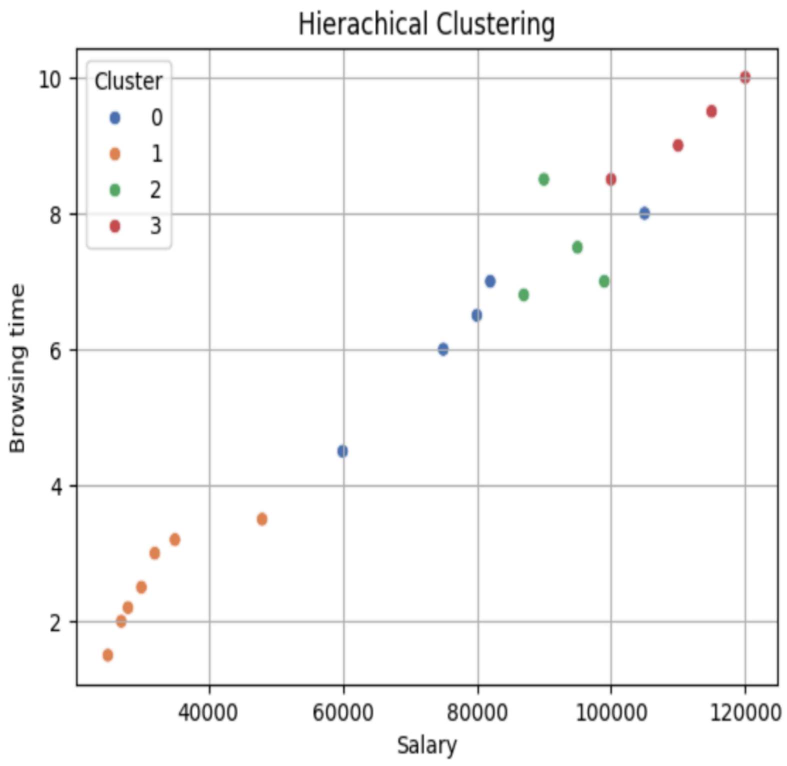
```
linked=linkage(X_scaled,method='ward')
dendrogram(linked,
        orientation='top',
        distance_sort='ascending',
        show_leaf_counts=True)
plt.title("Dendogram")
plt.xlabel("Sample")
plt.ylabel("Distance")
plt.show()

#step4
cluster=AgglomerativeClustering(n_clusters=4,linkage='ward')
df['Cluster']=cluster.fit_predict(X_scaled)

#step5
print("Clustered Date")
print(df)
sns.scatterplot(data=df,x='Salary',y='Browsing_Time',hue='Cluster',palette='deep')
plt.title("Hierachical Clustering")
plt.xlabel("Salary")
plt.ylabel("Browsing time")
plt.grid(True)
plt.show()
```

OUTPUT:





```
Clustered Date
    Age  Salary  Browsing_Time  Cluster
0    22   25000            1.5        1
1    25   27000            2.0        1
2    47   90000            8.5        2
3    52  110000            9.0        3
4    48   95000            7.5        2
5    55  120000           10.0        3
6    60   99000            7.0        2
7    32  105000            8.0        0
8    44  115000            9.5        3
9    25   48000            3.5        1
10   40   80000            6.5        0
11   28   30000            2.5        1
12   38   75000            6.0        0
13   29   32000            3.0        1
14   30   35000            3.2        1
15   41   82000            7.0        0
16   26   28000            2.2        1
17   34   60000            4.5        0
18   45   87000            6.8        2
19   50  100000            8.5        3
```

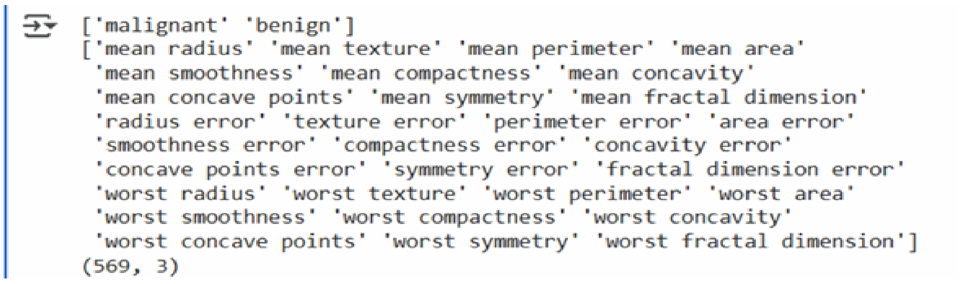


Practical 6: To implement PCA (Principal Component Analysis).

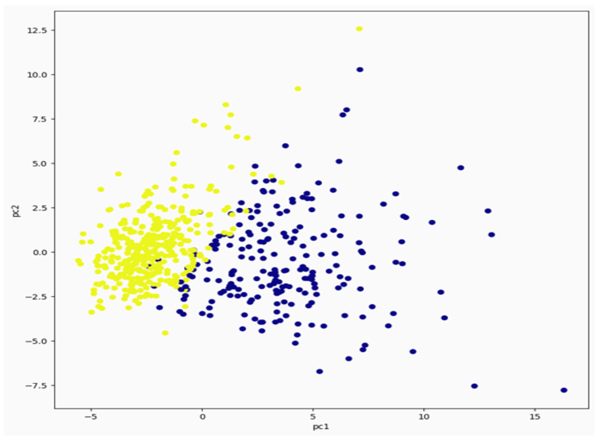
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer
data=load_breast_cancer()
data.keys()
```

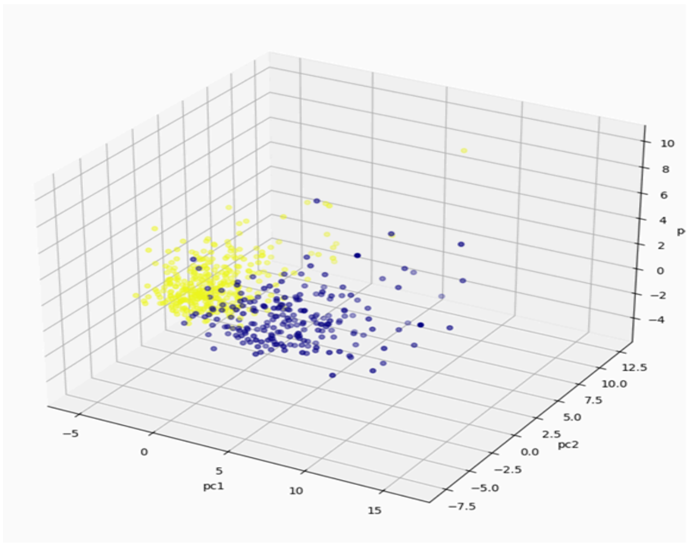
```
print(data['target_names']) #check the output class

print(data['feature_names']) #check the input features

df1=pd.DataFrame(data['data'],columns=data['feature_names'])

scaling=StandardScaler()

scaling.fit(df1)

scaled_data=scaling.transform(df1)

principal=PCA(n_components=3)  #set n_componets=3

principal.fit(scaled_data)

x=principal.transform(scaled_data)

print(x.shape)

plt.figure(figsize=(10,10))

plt.scatter(x[:,0],x[:,1],c=data['target'],cmap='plasma')

plt.xlabel('pc1')

plt.ylabel('pc2')

plt.show()

from mpl_toolkits.mplot3d import Axes3D

fig=plt.figure(figsize=(10,10))

axis=fig.add_subplot(111,projection='3d')

axis.scatter(x[:,0],x[:,1],x[:,2],c=data['target'],cmap='plasma')




axis.set_xlabel('pc1',fontsize=10)

axis.set_ylabel('pc2',fontsize=10)

axis.set_zlabel('pc3',fontsize=10)

plt.show()
```

OUTPUT:



```
['malignant' 'benign']
['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']
(569, 3)
```





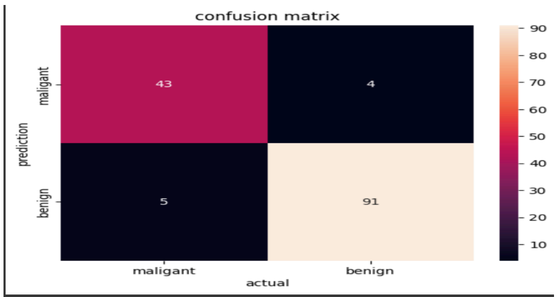
Practical 7 :To explore the given data and identify the patterns in it.

Aim: 8.1) To evaluate binary classification model using confusion matrix along with precision and recall.

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
#load dataset
X,y=load_breast_cancer(return_X_y=True)
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25)
#Train model
tree=DecisionTreeClassifier(random_state=23)
tree.fit(X_train,y_train)
y_pred=tree.predict(X_test)
#compute the matrix
cm=confusion_matrix(y_test,y_pred)
#plot the matrix
sns.heatmap(cm,
        annot=True,
        fmt='g',
        xticklabels=['maligant','benign'],
        yticklabels=['maligant','benign'])
plt.ylabel("prediction")
plt.xlabel("actual")
plt.title("confusion matrix")
plt.show()
#finding presion and recall
accuracy=accuracy_score(y_test,y_pred)
print("accuracy:",accuracy)
precision=precision_score(y_test,y_pred)
print("presion:",precision)
recall=recall_score(y_test,y_pred)
print("recall:",recall)
f=f1_score(y_test,y_test)
print("f1:",f)
```

OUTPUT:



```
accuracy: 0.9370629370629371
presion: 0.9578947368421052
recall: 0.9479166666666666
f1: 1.0
```

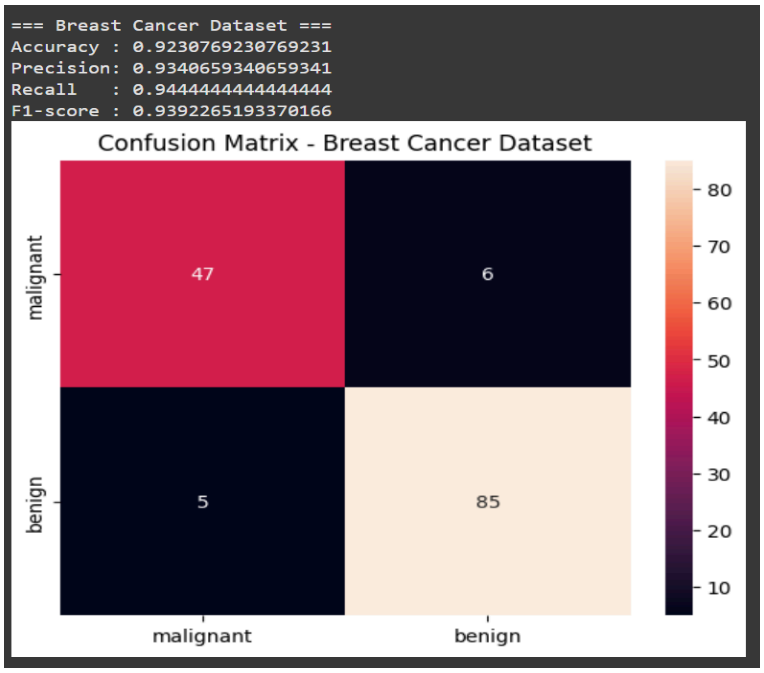
Aim:8.2) To evaluate multi-class classification model using confusion matrix along with precision and recall.

Aim:9(Use an appropriate dataset and create a supervised learning model, Analyse the model with ROC-AUC.

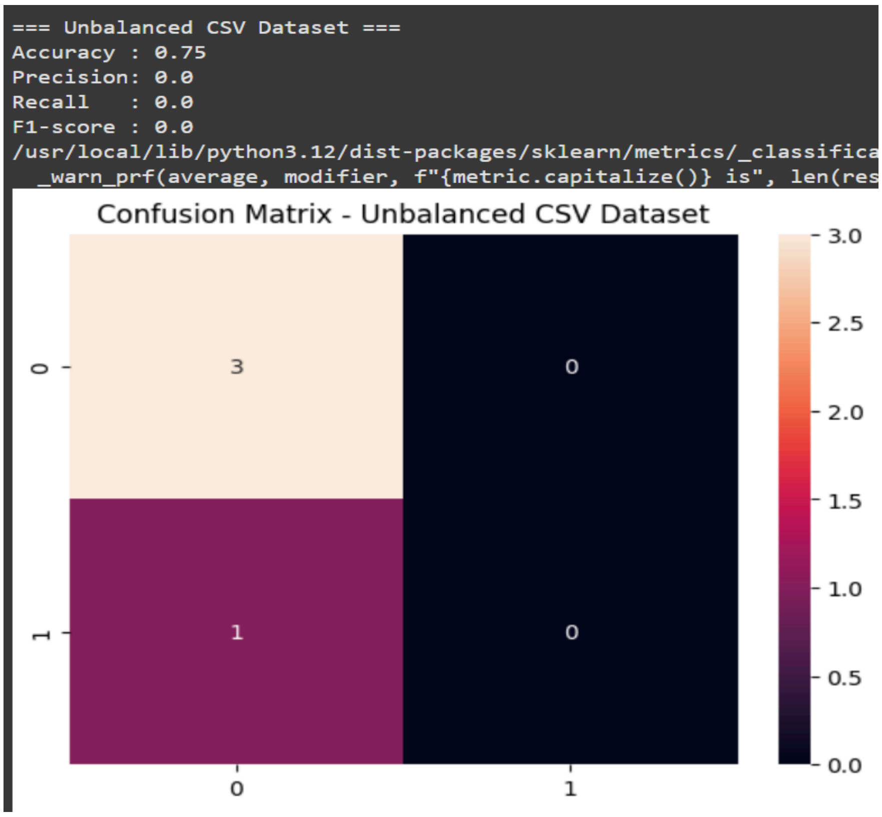
```
#Use an appropriate dataset and create a supervised learning model, Analyse the model with ROC-AUC.

import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score, roc_curve, auc)
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=23, stratify=y
)
tree1 = DecisionTreeClassifier(random_state=23)
tree1.fit(X_train, y_train)
y_pred1 = tree1.predict(X_test)
y_proba1 = tree1.predict_proba(X_test)[:, 1]
print("=== Breast Cancer Dataset ===")
print("Accuracy :", accuracy_score(y_test, y_pred1))
print("Precision:", precision_score(y_test, y_pred1))
print("Recall   :", recall_score(y_test, y_pred1))
print("F1-score :", f1_score(y_test, y_pred1))

cm1 = confusion_matrix(y_test, y_pred1)
sns.heatmap(cm1, annot=True, fmt="g",
            xticklabels=["malignant", "benign"],
            yticklabels=["malignant", "benign"])
plt.title("Confusion Matrix - Breast Cancer Dataset")
plt.show()
fpr1, tpr1, _ = roc_curve(y_test, y_proba1)
roc_auc1 = auc(fpr1, tpr1)

data = pd.DataFrame({
    "education":
["bach","mast","diploma","mast","diploma","bach","mast","mast","diploma","mast","bach","mast",
"mast","mast","bach"],

    "job": [1,0,0,0,0,0,0,0,0,1,0,0,0,0,0]

})


# Encode categorical features

X2 = pd.get_dummies(data.drop("job", axis=1))

y2 = data["job"]


X2_train, X2_test, y2_train, y2_test = train_test_split(

    X2, y2, test_size=0.25, random_state=23, stratify=y2

)


tree2 = DecisionTreeClassifier(random_state=23)

tree2.fit(X2_train, y2_train)


y2_pred = tree2.predict(X2_test)

y2_proba = tree2.predict_proba(X2_test)[:, 1]


print("\n=== Unbalanced CSV Dataset ===")

print("Accuracy :", accuracy_score(y2_test, y2_pred))

print("Precision:", precision_score(y2_test, y2_pred))

print("Recall   :", recall_score(y2_test, y2_pred))

print("F1-score :", f1_score(y2_test, y2_pred))


cm2 = confusion_matrix(y2_test, y2_pred)

sns.heatmap(cm2, annot=True, fmt="g")

plt.title("Confusion Matrix - Unbalanced CSV Dataset")

plt.show()


fpr2, tpr2, _ = roc_curve(y2_test, y2_proba)
```

```
roc_auc2 = auc(fpr2, tpr2)


plt.figure(figsize=(7, 5))
plt.plot(fpr1, tpr1, label=f"Breast Cancer (AUC={roc_auc1:.2f})")
plt.plot(fpr2, tpr2, label=f"Unbalanced CSV (AUC={roc_auc2:.2f})")

plt.plot([0, 1], [0, 1], "r--", label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves Comparison")
plt.legend()
plt.grid(True)
plt.show()
```

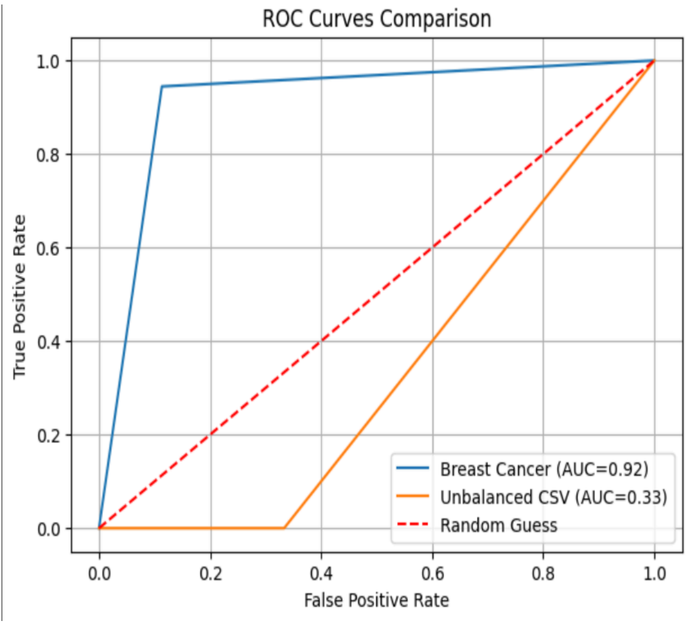
OUTPUT:



```
=== Breast Cancer Dataset ===
Accuracy : 0.9230769230769231
Precision: 0.9340659340659341
Recall   : 0.9444444444444444
F1-score : 0.9392265193370166
```



```
=== Unbalanced CSV Dataset ===
Accuracy : 0.75
Precision: 0.0
Recall   : 0.0
F1-score : 0.0
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classifica
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(res
```



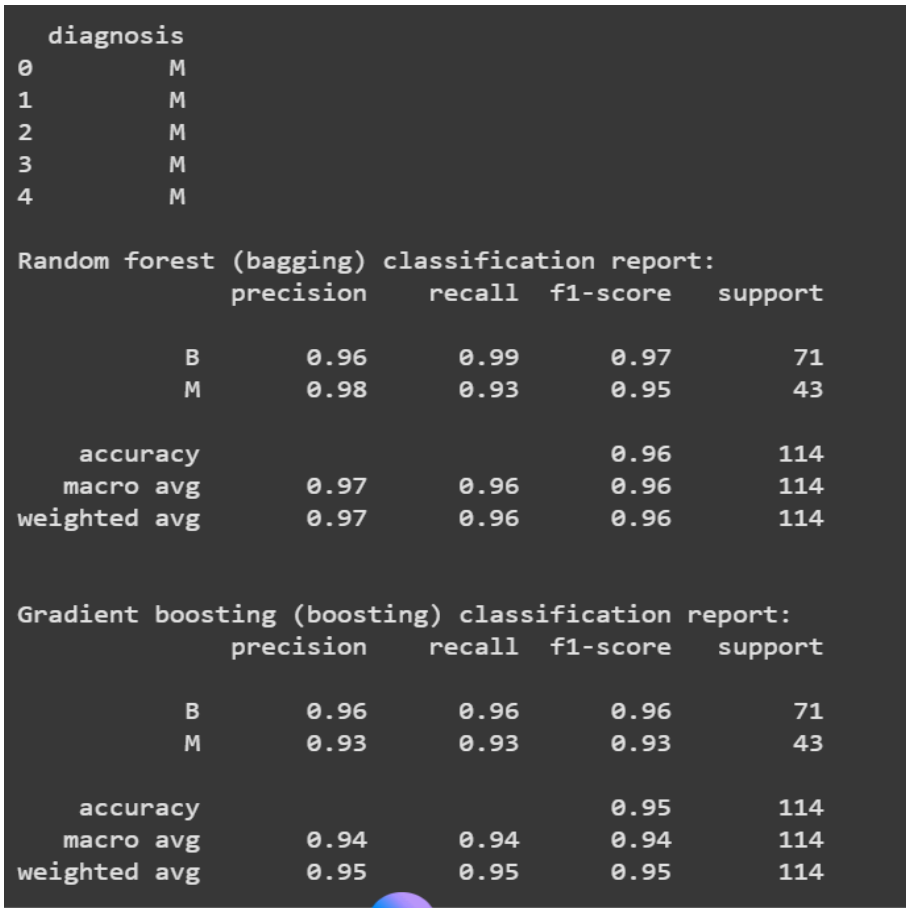
Ashish Ashtekar 411

Practical 10.:Consider a case study problem and implement an appropriate model and evaluate it.

Aim: 11.1)Bagging and boosting model.

```
#bagging and boosting
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.metrics import classification_report
import pandas as pd
data = pd.read_csv("/content/breast-cancer.csv")
x=data.drop("diagnosis", axis=1)
y=data.diagnosis
df=pd.DataFrame(y)
print(df.head())
#splitting data
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
#initialize model
rf=RandomForestClassifier(n_estimators=100,random_state=42)
gb=GradientBoostingClassifier(n_estimators=100,learning_rate=0.3,random_state=42)
#train model
rf.fit(x_train,y_train)
gb.fit(x_train,y_train)
#predict model
y_pred_rf=rf.predict(x_test)
y_pred_gb=gb.predict(x_test)
#Evaluate and print result
print("\nRandom forest (bagging) classification report: ")
print(classification_report(y_test,y_pred_rf))
print("\nGradient boosting (boosting) classification report: ")
print(classification_report(y_test,y_pred_gb))
```

OUTPUT:

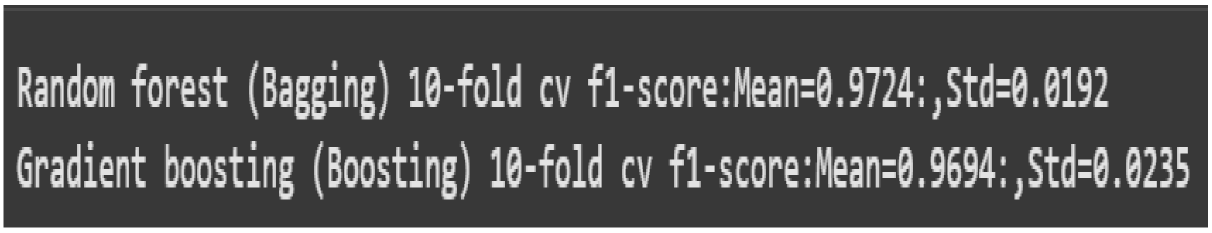


```
  diagnosis
0         M
1         M
2         M
3         M
4         M

Random forest (bagging) classification report:
              precision    recall  f1-score   support

           B       0.96      0.99      0.97        71
           M       0.98      0.93      0.95        43

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114


Gradient boosting (boosting) classification report:
              precision    recall  f1-score   support

           B       0.96      0.96      0.96        71
           M       0.93      0.93      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

Aim:11.2 Cross validation methods

```
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
import numpy as np
data = load_breast_cancer()
X = data.data
y = data.target
kf = StratifiedKFold(n_splits=10)
rf = RandomForestClassifier(n_estimators=100, n_jobs=42)
gb = GradientBoostingClassifier(n_estimators=100,learning_rate=0.1,random_state=42)
rf_scores = cross_val_score(rf,X,y,cv=kf,scoring='f1')
gb_scores = cross_val_score(gb,X,y,cv=kf,scoring='f1')
print(f"Random forest (Bagging) 10-fold cv f1-score:"
f"Mean={rf_scores.mean():.4f}:,Std={rf_scores.std():.4f}")
print(f"Gradient boosting (Boosting) 10-fold cv f1-score:"
f"Mean={gb_scores.mean():.4f}:,Std={gb_scores.std():.4f}")
```

OUTPUT:



```
Random forest (Bagging) 10-fold cv f1-score:Mean=0.9724:,Std=0.0192
Gradient boosting (Boosting) 10-fold cv f1-score:Mean=0.9694:,Std=0.0235
```