EASWARI ENGINEERING COLLEGE

(Autonomous)

Bharathi Salai, Ramapuram, Chennai-600 089

Department: Master of Computer Applications

Name of the Lab: 244MCC311L/MACHINE LEARNING LABORATORY

Name :

Reg No. :

Semester :

Year :

Branch :

EASWARI ENGINEERING COLLEGE

(Autonomous)

Bharathi Salai, Ramapuram, Chennai-600 089

Department: Master of Computer Applications

PRATICAL EXAMINATIONS NOVEMBER 2025

BONAFIDE CERTIFICATE

This is to Certify that this practical work titled

244MCC311L/MACHINE LEARNING LABORATORY is the Bonafide work of

Mr./Miss._____

With Registration Number_____

in _____Semester of _____Year in the Department of

MASTER OF COMPUTER APPLICATION during the academic year **2025-2026**

Faculty Incharge **Head of the Department**

Submitted for Practical Examination held on ____/____/_____at
Easwari Engineering College, Ramapuram, Chennai-89

Internal Examiner External Examiner

CONTENTS

Ex.No:1

Date:

Structuring Data for Machine Learning

Aim:

To write a python program to structure the data for machine learning.

Algorithm:

Step 1: Load the Iris dataset.

Step 2: Separate features (X) and target (y).

Step 3: Encode target labels numerically.

Step 4: Split the dataset into training and testing sets.

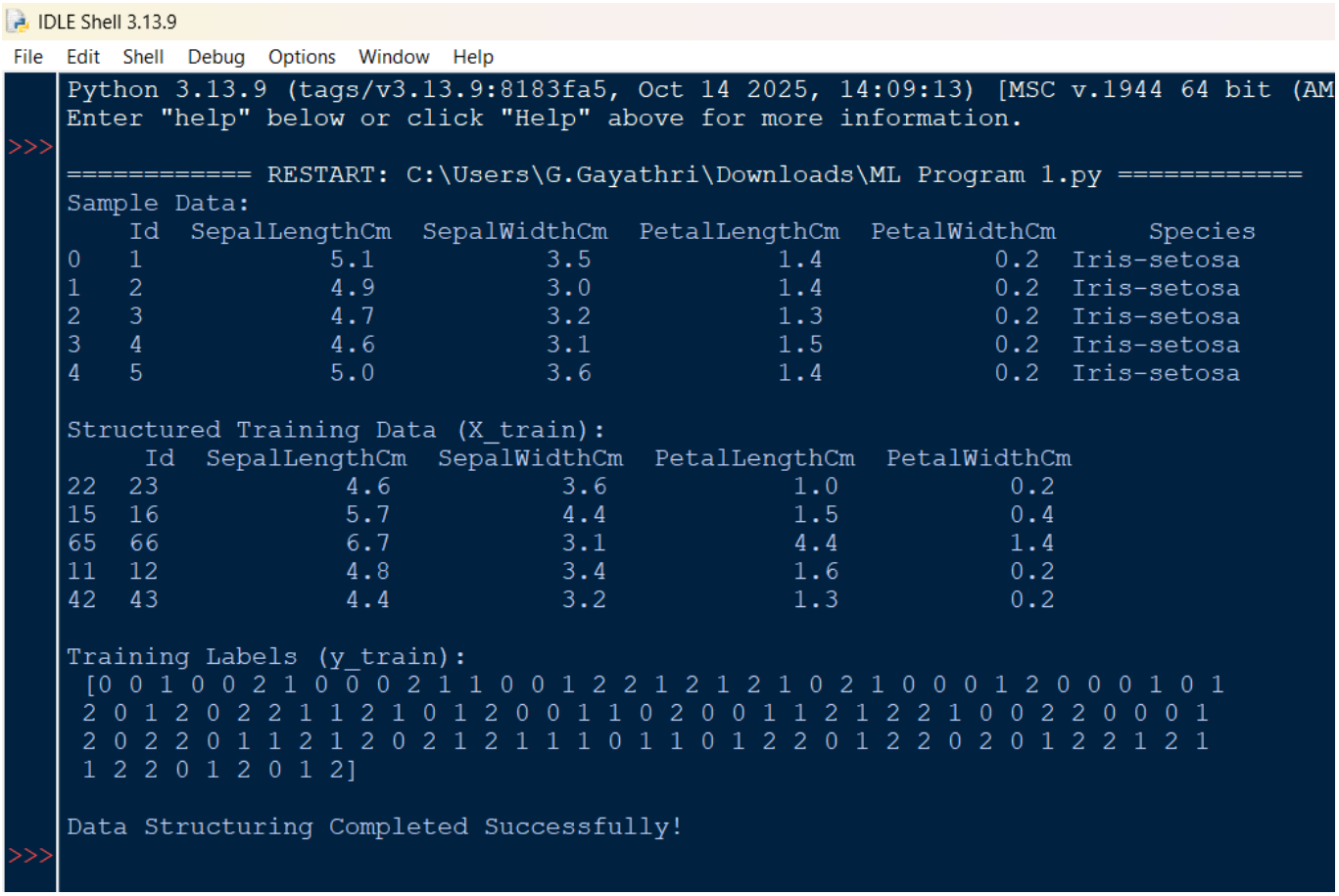
Step 5: Display structured data and labels.

#Program 1: Structuring Data for ML

Source Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load the dataset
url = "E:/Iris.csv"
data = pd.read_csv(url)

# Display the dataset
print("Sample Data:\n", data.head())

#Identify features (X) and target (y)
X = data.drop('Species', axis=1)
y = data['Species']

# Encode categorical target variable
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

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

# Display structured data
print("\nStructured Training Data (X_train):\n", X_train.head())
print("\nTraining Labels (y_train):\n", y_train[:5])

print("\nData Structuring Completed Successfully!")
```

OUTPUT:


IDLE Shell 3.13.9
File Edit Shell Debug Options Window Help
Python 3.13.9 (tags/v3.13.9:8183fa5, Oct 14 2025, 14:09:13) [MSC v.1944 64 bit (AM
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\G.Gayathri\Downloads\ML Program 1.py ==========
Sample Data:
Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species
0 1 5.1 3.5 1.4 0.2 Iris-setosa
1 2 4.9 3.0 1.4 0.2 Iris-setosa
2 3 4.7 3.2 1.3 0.2 Iris-setosa
3 4 4.6 3.1 1.5 0.2 Iris-setosa
4 5 5.0 3.6 1.4 0.2 Iris-setosa
Structured Training Data (X_train):
Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm
22 23 4.6 3.6 1.0 0.2
15 16 5.7 4.4 1.5 0.4
65 66 6.7 3.1 4.4 1.4
11 12 4.8 3.4 1.6 0.2
42 43 4.4 3.2 1.3 0.2
Training Labels (y_train):
[0 0 1 0 0 2 1 0 0 0 2 1 1 0 0 1 2 2 1 2 1 2 1 0 2 1 0 0 0 1 2 0 0 0 1 0 1
2 0 1 2 0 2 2 1 1 2 1 0 1 2 0 0 1 1 0 2 0 0 1 1 2 1 2 2 1 0 0 2 2 0 0 0 1
2 0 2 2 0 1 1 2 1 2 0 2 1 2 1 1 1 0 1 1 0 1 2 2 0 1 2 2 0 2 0 1 2 2 1 2 1
1 2 2 0 1 2 0 1 2]
Data Structuring Completed Successfully!
>>>

Ex.No:2

Date:

Data Preprocessing on Real-world Dataset

Aim:

To write a python program to preprocess the data in real-world dataset

Algorithm:

Step 1: Load the Titanic dataset.

Step 2: Handle missing values using mean/mode.

Step 3: Drop irrelevant or highly missing columns.

Step 4: Detect and remove outliers using IQR.

Step 5: Encode categorical data numerically.

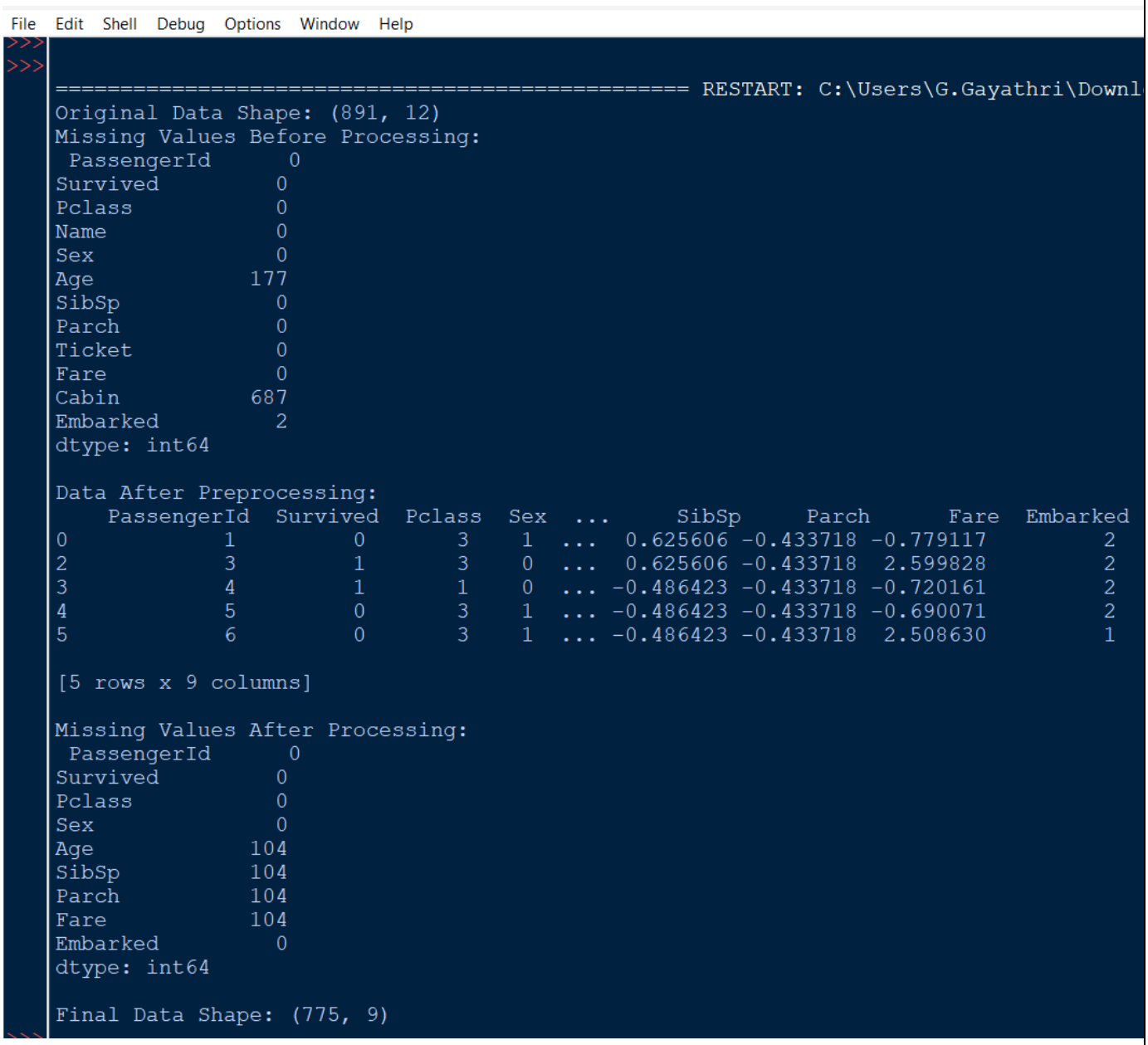
Step 6: Apply standard scaling to numeric columns.

Step 7: Display preprocessed dataset.

#Program 2: Data Preprocessing on Real-World Dataset

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Load dataset
url = "E:/ML Program Dataset/Titanic-Dataset.csv"
data = pd.read_csv(url)

print("Original Data Shape:", data.shape)
print("Missing Values Before Processing:\n", data.isnull().sum())

# Handle missing values
data['Age'] = data['Age'].fillna(data['Age'].mean())
data['Embarked'] = data['Embarked'].fillna(data['Embarked'].mode()[0])

# Drop irrelevant or highly missing columns
data = data.drop(columns=['Cabin', 'Name', 'Ticket'])

# Handle outliers using IQR method for 'Fare'
Q1 = data['Fare'].quantile(0.25)
Q3 = data['Fare'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR
data = data[(data['Fare'] >= lower) & (data['Fare'] <= upper)]

#Encode categorical variables
encoder = LabelEncoder()
data['Sex'] = encoder.fit_transform(data['Sex'])
data['Embarked'] = encoder.fit_transform(data['Embarked'])

# Feature scaling
scaler = StandardScaler()
scaled_features = scaler.fit_transform(data[['Age', 'Fare', 'SibSp', 'Parch']])
scaled_df = pd.DataFrame(scaled_features, columns=['Age', 'Fare', 'SibSp', 'Parch'])

# Replace original numeric columns with scaled values
data[['Age', 'Fare', 'SibSp', 'Parch']] = scaled_df

# Display cleaned and preprocessed dataset
print("\nData After Preprocessing:\n", data.head())
print("\nMissing Values After Processing:\n", data.isnull().sum())
print("\nFinal Data Shape:", data.shape)
```

OUTPUT:



```
File  Edit  Shell  Debug  Options  Window  Help
>>>
>>>
========================================== RESTART: C:\Users\G.Gayathri\Downl
Original Data Shape: (891, 12)
Missing Values Before Processing:
 PassengerId      0
Survived          0
Pclass            0
Name              0
Sex               0
Age             177
SibSp             0
Parch             0
Ticket            0
Fare              0
Cabin           687
Embarked          2
dtype: int64

Data After Preprocessing:
    PassengerId  Survived  Pclass  Sex  ...     SibSp     Parch      Fare  Embarked
0             1         0       3    1  ...  0.625606 -0.433718 -0.779117         2
2             3         1       3    0  ...  0.625606 -0.433718  2.599828         2
3             4         1       1    0  ... -0.486423 -0.433718 -0.720161         2
4             5         0       3    1  ... -0.486423 -0.433718 -0.690071         2
5             6         0       3    1  ... -0.486423 -0.433718  2.508630         1

[5 rows x 9 columns]

Missing Values After Processing:
 PassengerId      0
Survived          0
Pclass            0
Sex               0
Age             104
SibSp           104
Parch           104
Fare            104
Embarked          0
dtype: int64

Final Data Shape: (775, 9)
>>>
```

Ex.No:3

Date:

Feature Subset Selection Techniques

Aim:

To write a python program to implement feature subset selection technique on a given dataset

Algorithm:

Step 1: Load the Titanic dataset.

Step 2: Clean and encode data.

Step 3: Separate features and target.

Step 4: Use correlation matrix to identify related features.

Step 5: Apply Recursive Feature Elimination (RFE).

Step 6: Use Random Forest for feature importance ranking.

Step 7: Display top selected features.

Program 3: Feature Subset Selection Techniques

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import warnings

# Suppress all harmless warnings
warnings.filterwarnings("ignore")

# Load the dataset
url = "E:/ML Program Dataset/Titanic-Dataset.csv"
data = pd.read_csv(url).copy()   # ensure fresh copy to avoid chained assignment issues

# Preprocess dataset (safe assignments)
data['Age'] = data['Age'].fillna(data['Age'].mean())
data['Embarked'] = data['Embarked'].fillna(data['Embarked'].mode()[0])
data = data.drop(columns=['Cabin', 'Name', 'Ticket'])

# Encode categorical variables
encoder = LabelEncoder()
data['Sex'] = encoder.fit_transform(data['Sex'])
data['Embarked'] = encoder.fit_transform(data['Embarked'])

# Split features and target
X = data[['Pclass', 'Sex', 'Age', 'Fare', 'SibSp', 'Parch', 'Embarked']]
y = data['Survived']

# Filter Method (Correlation-based)
correlation = X.corr(numeric_only=True)  # explicit for sklearn >=1.5 compatibility
print("Correlation Matrix:\n", correlation)
print("\nHighly Correlated Features (>|0.8|):")
print(correlation[(correlation > 0.8) | (correlation < -0.8)])

# Wrapper Method (Recursive Feature Elimination - RFE)
model = LogisticRegression(max_iter=2000, solver='lbfgs')
rfe = RFE(model, n_features_to_select=4)
fit = rfe.fit(X, y)

print("\nSelected Features using RFE:")
for i, col in enumerate(X.columns):
    if fit.support_[i]:
```

```
    print(f"- {col}")

# Embedded Method (Feature Importance using Random Forest)
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)
importances = pd.Series(rf.feature_importances_,
index=X.columns).sort_values(ascending=False)

print("\nFeature Importance (Random Forest):\n", importances)

# Display top features
top_features = importances.head(4).index.tolist()
print("\nTop 4 Important Features (Final Selected):", top_features)
```

OUTPUT:

```
File  Edit  Shell  Debug  Options  Window  Help
>>> ================================================ RESTART: C:\Users\G.Gayathri\Do
Correlation Matrix:
              Pclass       Sex       Age      Fare     SibSp     Parch  Embarked
Pclass      1.000000  0.131900 -0.331339 -0.549500  0.083081  0.018443  0.162098
Sex         0.131900  1.000000  0.084153 -0.182333 -0.114631 -0.245489  0.108262
Age        -0.331339  0.084153  1.000000  0.091566 -0.232625 -0.179191 -0.026749
Fare       -0.549500 -0.182333  0.091566  1.000000  0.159651  0.216225 -0.224719
SibSp       0.083081 -0.114631 -0.232625  0.159651  1.000000  0.414838  0.068230
Parch       0.018443 -0.245489 -0.179191  0.216225  0.414838  1.000000  0.039798
Embarked    0.162098  0.108262 -0.026749 -0.224719  0.068230  0.039798  1.000000

Highly Correlated Features (>|0.8|):
          Pclass  Sex  Age  Fare  SibSp  Parch  Embarked
Pclass       1.0  NaN  NaN   NaN    NaN    NaN       NaN
Sex          NaN  1.0  NaN   NaN    NaN    NaN       NaN
Age          NaN  NaN  1.0   NaN    NaN    NaN       NaN
Fare         NaN  NaN  NaN   1.0    NaN    NaN       NaN
SibSp        NaN  NaN  NaN   NaN    1.0    NaN       NaN
Parch        NaN  NaN  NaN   NaN    NaN    1.0       NaN
Embarked     NaN  NaN  NaN   NaN    NaN    NaN       1.0

Selected Features using RFE:
- Pclass
- Sex
- SibSp
- Embarked

Feature Importance (Random Forest):
 Fare        0.268581
Age          0.265754
Sex          0.260338
Pclass       0.085593
SibSp        0.046811
Parch        0.040439
Embarked     0.032483
dtype: float64

Top 4 Important Features (Final Selected): ['Fare', 'Age', 'Sex', 'Pclass']
>>>
>>>
```

Ex.No:4

Date:

Measuring Model Performance on Real-World Dataset

Aim:

To write a python program to measure the model performance on real world dataset.

Algorithm:

Step 1: Load the Breast Cancer dataset.

Step 2: Split data into training and testing sets.

Step 3: Normalize features using StandardScaler.

Step 4: Train Logistic Regression classifier.

Step 5: Predict test labels.

Step 6: Evaluate model using accuracy, precision, recall, F1, and confusion matrix.

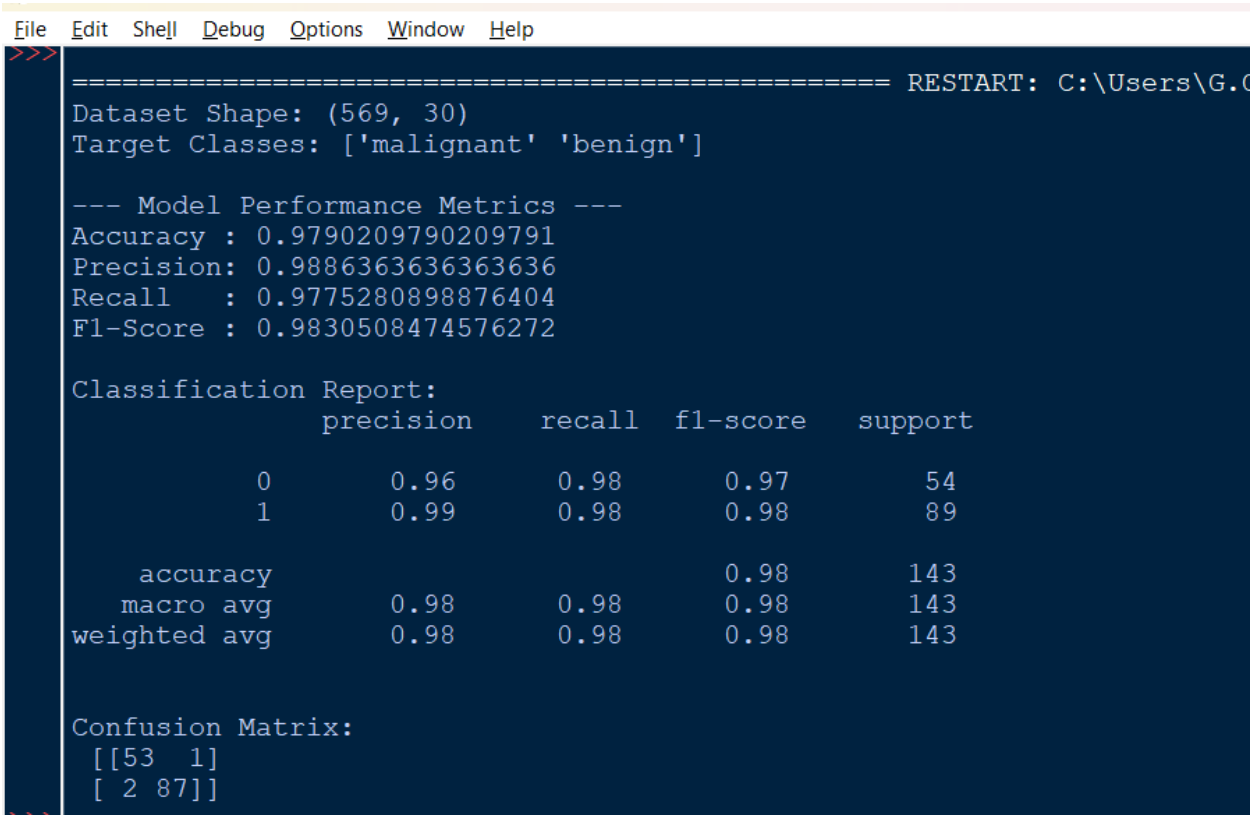
Program 4: Measuring Model Performance on Real-World Dataset

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, confusion_matrix,
classification_report
)

# Load real-world dataset
cancer = load_breast_cancer()
X = pd.DataFrame(cancer.data, columns=cancer.feature_names)
y = pd.Series(cancer.target)

print("Dataset Shape:", X.shape)
print("Target Classes:", cancer.target_names)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train a machine learning model
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Measure model performance
print("\n--- Model Performance Metrics ---")
print("Accuracy :", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall   :", recall_score(y_test, y_pred))
print("F1-Score :", f1_score(y_test, y_pred))

# Detailed classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

 # Confusion matrix

```
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", cm)
```

OUTPUT:



Ex.No:5

Date:

Naïve Bayesian Classifier

Aim:

To write a python program to implement naïve Bayesian Classifier.

Algorithm:

Step 1: Load Cancer dataset.

Step 2: Remove unnecessary columns and handle missing data.

Step 3: Split data into features (X) and target (y).

Step 4: Train/test split the dataset.

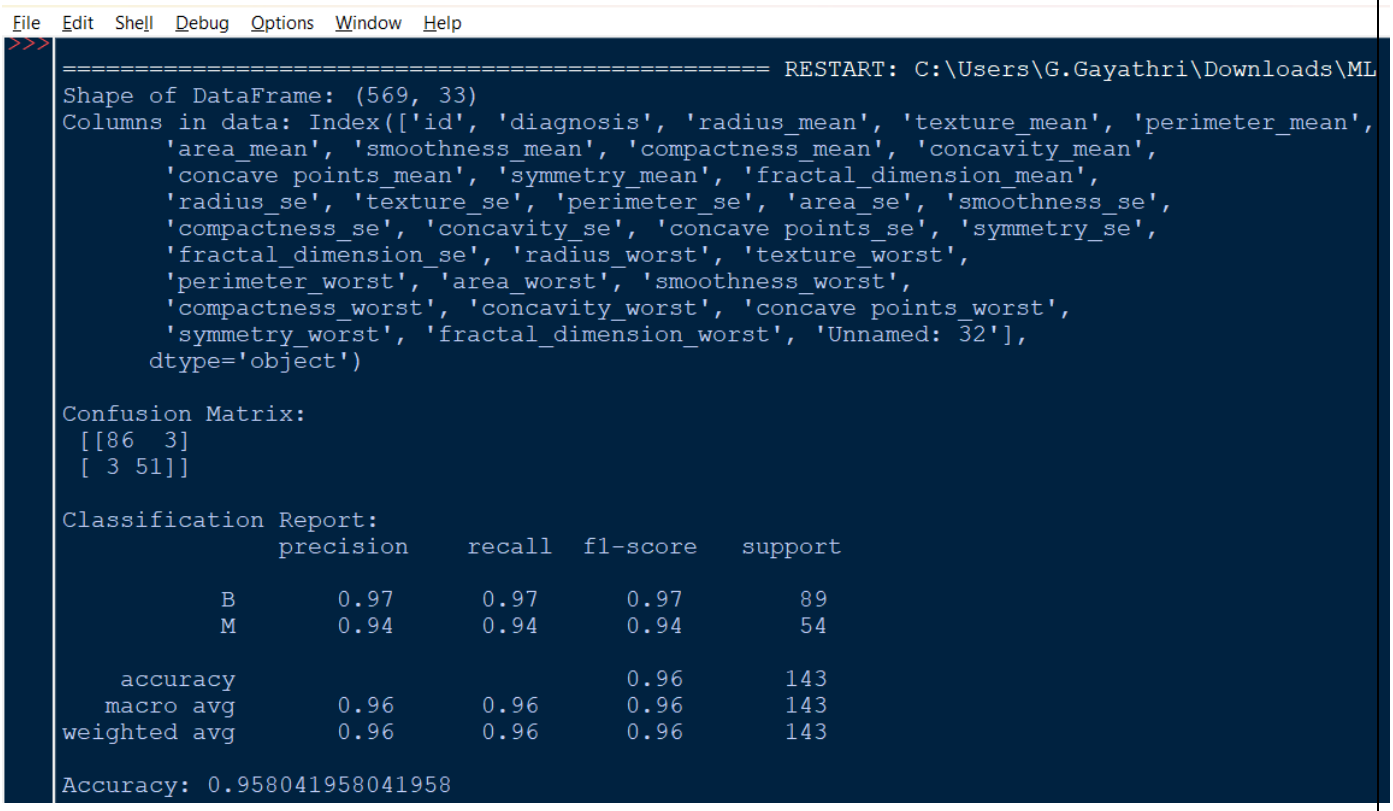
Step 5: Train Gaussian Naïve Bayes model.

Step 6: Predict and evaluate model accuracy and report.

#Program 5: naïve Bayesian Classifier

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load dataset
data = pd.read_csv("E:/ML Program Dataset/Cancer_Data.csv")

#: Inspect the dataset
print("Shape of DataFrame:", data.shape)
print("Columns in data:", data.columns)

data = data.drop(['id', 'Unnamed: 32'], axis=1)
data = data.dropna()

# Split features and target
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)

# Train the model
model = GaussianNB()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

OUTPUT:


File Edit Shell Debug Options Window Help
>>>
========== RESTART: C:\Users\G.Gayathri\Downloads\ML
Shape of DataFrame: (569, 33)
Columns in data: Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
'fractal_dimension_se', 'radius_worst', 'texture_worst',
'perimeter_worst', 'area_worst', 'smoothness_worst',
'compactness_worst', 'concavity_worst', 'concave points_worst',
'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
dtype='object')
Confusion Matrix:
[[86 3]
[3 51]]
Classification Report:
precision recall f1-score support
B 0.97 0.97 0.97 89
M 0.94 0.94 0.94 54
accuracy 0.96 143
macro avg 0.96 0.96 0.96 143
weighted avg 0.96 0.96 0.96 143
Accuracy: 0.958041958041958

Ex.No:6

Date:

Bayesian Network for Heart Disease Diagnosis

Aim:

To write a python program to construct a Bayesian network for heart disease diagnosis

Algorithm:

Step 1: Define discrete and conditional probability tables.

Step 2: Create network states for each variable.

Step 3: Build Bayesian Network structure.

Step 4: Add edges and make the model.

Step 5: Perform prediction with given evidence.

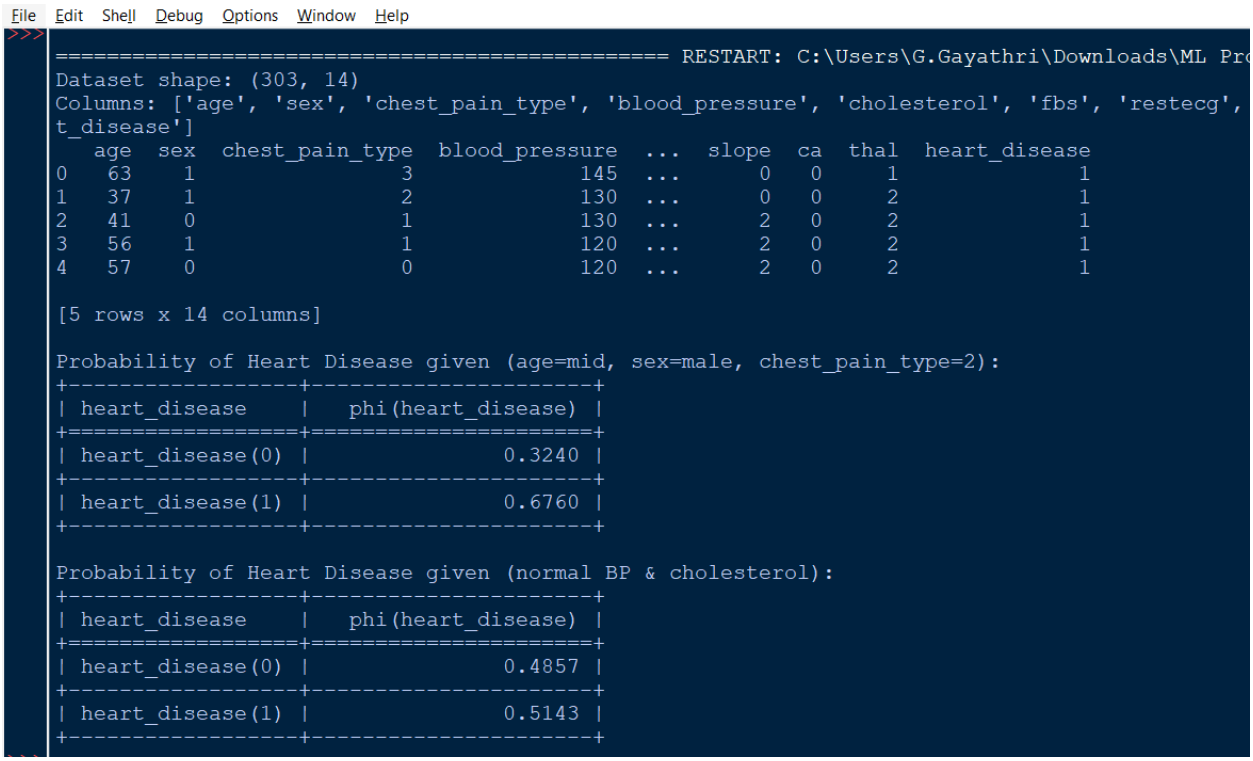
Step 6: Display probabilistic outcomes.

Program 6: Bayesian Network for Heart Disease Diagnosis

```
import logging
logging.getLogger('pgmpy').setLevel(logging.ERROR)

import pandas as pd
from pgmpy.models import DiscreteBayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

# Load the Heart Disease dataset
data = pd.read_csv(r"E:\ML Program Dataset\heart.csv")

# Rename columns for clarity
data.rename(columns={
    'cp': 'chest_pain_type',
    'trestbps': 'blood_pressure',
    'chol': 'cholesterol',
    'target': 'heart_disease'
}, inplace=True)

print("Dataset shape:", data.shape)
print("Columns:", data.columns.tolist())
print(data.head())

# Discretize continuous variables into categories
# Bayesian Networks require discrete data
for col in ['age', 'blood_pressure', 'cholesterol', 'thalach']:
    data[col] = pd.cut(data[col], bins=3, labels=[0, 1, 2])

# Convert all columns to categorical (discrete)
for col in data.columns:
    data[col] = data[col].astype('category')

# Define the Bayesian Network structure (based on domain knowledge)
model = DiscreteBayesianNetwork([
    ('age', 'blood_pressure'),
    ('sex', 'cholesterol'),
    ('blood_pressure', 'heart_disease'),
    ('cholesterol', 'heart_disease'),
    ('thalach', 'heart_disease'),
    ('chest_pain_type', 'heart_disease'),
    ('restecg', 'heart_disease'),
    ('thal', 'heart_disease')
])
```

```
# Fit the model using Maximum Likelihood Estimation
model.fit(data, estimator=MaximumLikelihoodEstimator)

# Perform inference
infer = VariableElimination(model)

# Query 1: Probability of heart disease given age, sex, and chest pain type
query_result = infer.query(
    variables=['heart_disease'],
    evidence={'age': 1, 'sex': 1, 'chest_pain_type': 2}
)
print("\nProbability of Heart Disease given (age=mid, sex=male, chest_pain_type=2):")
print(query_result)

# Query 2: Probability of heart disease given normal BP and cholesterol
query_result2 = infer.query(
    variables=['heart_disease'],
    evidence={'blood_pressure': 1, 'cholesterol': 1}
)
print("\nProbability of Heart Disease given (normal BP & cholesterol):")
print(query_result2)
```

OUTPUT:



```
File  Edit  Shell  Debug  Options  Window  Help
>>>
========== RESTART: C:\Users\G.Gayathri\Downloads\ML Pr
Dataset shape: (303, 14)
Columns: ['age', 'sex', 'chest_pain_type', 'blood_pressure', 'cholesterol', 'fbs', 'restecg',
t_disease']
   age  sex  chest_pain_type  blood_pressure  ...  slope  ca  thal  heart_disease
0   63    1                3             145  ...      0   0     1              1
1   37    1                2             130  ...      0   0     2              1
2   41    0                1             130  ...      2   0     2              1
3   56    1                1             120  ...      2   0     2              1
4   57    0                0             120  ...      2   0     2              1

[5 rows x 14 columns]

Probability of Heart Disease given (age=mid, sex=male, chest_pain_type=2):
+------------------+----------------------+
| heart_disease    |   phi(heart_disease) |
+==================+======================+
| heart_disease(0) |               0.3240 |
+------------------+----------------------+
| heart_disease(1) |               0.6760 |
+------------------+----------------------+

Probability of Heart Disease given (normal BP & cholesterol):
+------------------+----------------------+
| heart_disease    |   phi(heart_disease) |
+==================+======================+
| heart_disease(0) |               0.4857 |
+------------------+----------------------+
| heart_disease(1) |               0.5143 |
+------------------+----------------------+
>>>
```

Ex.No:7

Date:

Apply EM algorithm to cluster a set of data

Aim:

To write a python program to apply Expectation Maximization Algorithm to cluster a set of data

Algorithm:

Step 1: Load Wine Quality dataset.

Step 2: Separate features and scale them.

Step 3: Apply Gaussian Mixture Model for clustering.

Step 4: Predict cluster labels for data points.

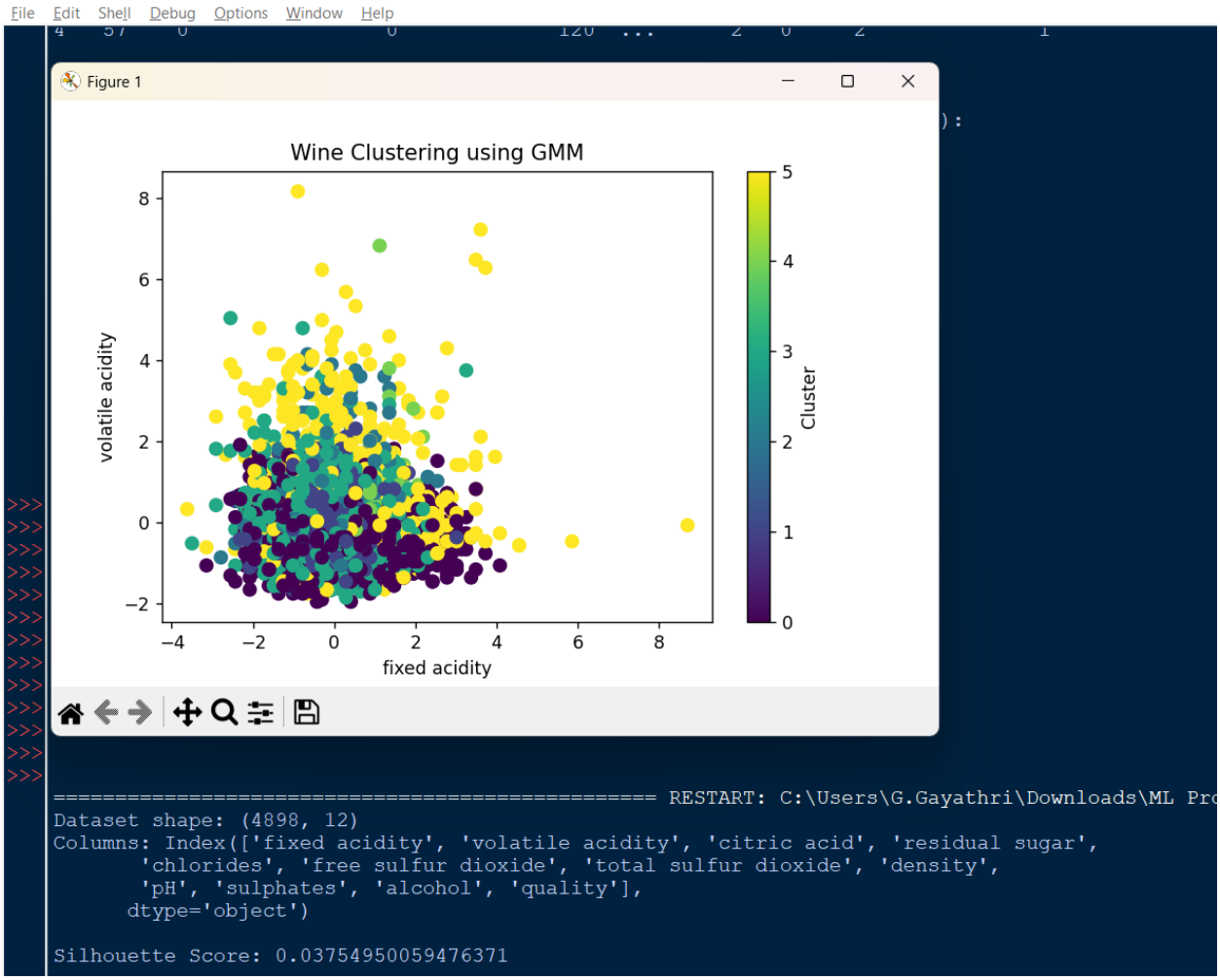
Step 5: Evaluate clustering using silhouette score.

Step 6: Visualize clusters with scatter plot.

#Program 7: Apply EM algorithm to cluster a set of data

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Load the Wine Quality Dataset
url = "E:/ML Program Dataset/winequality-white.csv"
data = pd.read_csv(url, sep=";")

# Inspect the dataset
print("Dataset shape:", data.shape)
print("Columns:", data.columns)

# Preprocess the data
X = data.drop('quality', axis=1)  # Features
y = data['quality']  # Target (not used in unsupervised learning)

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply Gaussian Mixture Model (GMM)
n_clusters = 6
gmm = GaussianMixture(n_components=n_clusters, covariance_type='full', random_state=42)
gmm.fit(X_scaled)

# Predict cluster labels
labels = gmm.predict(X_scaled)
data['Cluster'] = labels

# Evaluate clustering quality
sil_score = silhouette_score(X_scaled, labels)
print("\nSilhouette Score:", sil_score)

# Visualize clusters (first two features)
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis', s=50)
plt.title('Wine Clustering using GMM')
plt.xlabel(data.columns[0])
plt.ylabel(data.columns[1])
plt.colorbar(label='Cluster')
plt.show()
```

OUTPUT:


File Edit Shell Debug Options Window Help
Figure 1
Wine Clustering using GMM
volatile acidity
fixed acidity
Cluster
>>>
========== RESTART: C:\Users\G.Gayathri\Downloads\ML Pro
Dataset shape: (4898, 12)
Columns: Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
'pH', 'sulphates', 'alcohol', 'quality'],
dtype='object')
Silhouette Score: 0.03754950059476371

Ex.No:8

Date:

k-NN Classification with automatic k selection on SMS Spam dataset

Aim:

To write a python program to apply k-NN Classification with automatic k selection on SMS Spam dataset.

Algorithm:

Step 1: Load the Spam dataset.

Step 2: Assign proper feature names.

Step 3: Separate features and target.

Step 4: Split into training and testing sets.

Step 5: Scale data using StandardScaler.

Step 6: Train k-NN classifier with k=5.

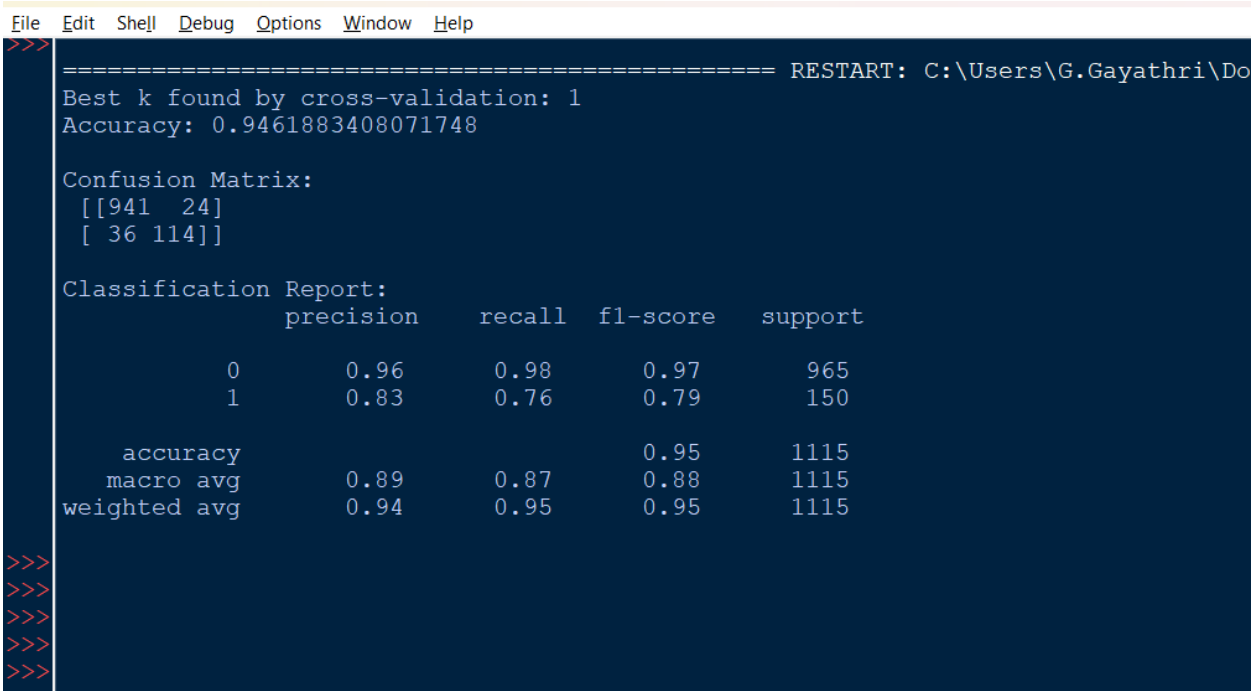
Step 7: Predict and evaluate using accuracy and confusion matrix.

#Program 8: k-NN Classification with automatic k selection on SMS Spam dataset

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load dataset
data = pd.read_csv("E:/ML Program Dataset/spam.csv", encoding='latin-1')
data = data[['v1', 'v2']]  # 'v1' = label, 'v2' = message
data.rename(columns={'v1': 'label', 'v2': 'message'}, inplace=True)

# Encode labels: 'ham' = 0, 'spam' = 1
data['label'] = data['label'].map({'ham': 0, 'spam': 1})

# Prepare features and target
X_text = data['message']
y = data['label']

# Convert text to numeric features using TF-IDF
vectorizer = TfidfVectorizer(stop_words='english', max_features=3000)
X = vectorizer.fit_transform(X_text).toarray()

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Use GridSearchCV to find the best k
param_grid = {'n_neighbors': list(range(1, 21))}  # try k from 1 to 20
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# Best k value
best_k = grid_search.best_params_['n_neighbors']
print("Best k found by cross-validation:", best_k)
```

```
# Train k-NN with best k
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train_scaled, y_train)

# Make predictions
y_pred = knn_best.predict(X_test_scaled)

# Evaluate model
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:


File Edit Shell Debug Options Window Help
>>>
========== RESTART: C:\Users\G.Gayathri\Do
Best k found by cross-validation: 1
Accuracy: 0.9461883408071748
Confusion Matrix:
[[941 24]
[36 114]]
Classification Report:
precision recall f1-score support
0 0.96 0.98 0.97 965
1 0.83 0.76 0.79 150
accuracy 0.95 1115
macro avg 0.89 0.87 0.88 1115
weighted avg 0.94 0.95 0.95 1115
>>>
>>>
>>>
>>>
>>>

Ex.No:9

Date:

Decision Tree Pruning on Noisy Monk2 Data

Aim:

To write a python program to build a decision tree and perform pruning on noisy monk2 data

Algorithm:

Step 1: Load and encode Monk2 dataset.

Step 2: Split data into train and test sets.

Step 3: Train an unpruned decision tree.

Step 4: Compute cost complexity pruning path.

Step 5: Train multiple pruned trees for different alpha values.

Step 6: Select best alpha and prune tree.

Step 7: Compare unpruned and pruned trees (accuracy, size).

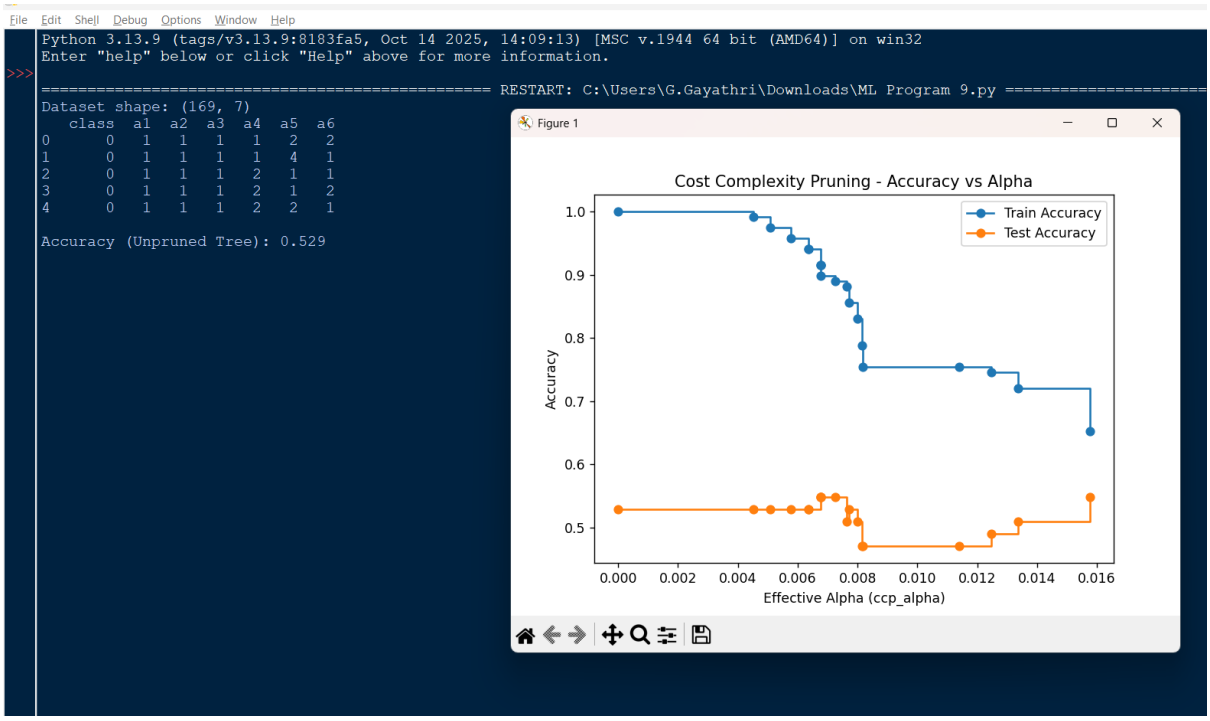
#Program 9:Decision Tree Pruning on Noisy Monk2 Data

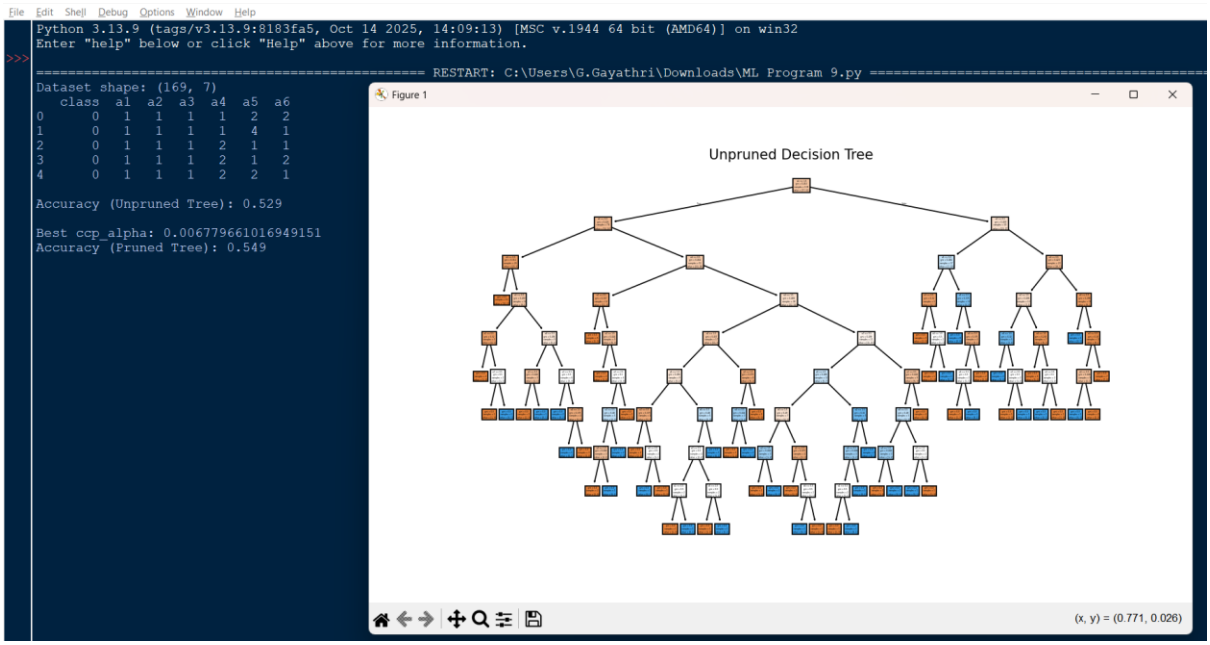
```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

data = pd.read_csv("E:/ML Program Dataset/monk2.csv")

data = data.drop('id', axis=1)
print("Dataset shape:", data.shape)
print(data.head())

# Convert categorical attributes to numeric (Label Encoding)
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for col in data.columns:
    data[col] = le.fit_transform(data[col])

# Split into features and target
X = data.drop('class', axis=1)
y = data['class']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

#Train Unpruned Decision Tree
clf_unpruned = DecisionTreeClassifier(random_state=42)
clf_unpruned.fit(X_train, y_train)

# Evaluate
y_pred_unpruned = clf_unpruned.predict(X_test)
acc_unpruned = accuracy_score(y_test, y_pred_unpruned)
print("\nAccuracy (Unpruned Tree):", round(acc_unpruned, 3))

#Apply Cost Complexity Pruning

path = clf_unpruned.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas  # array of effective alphas

# Train trees with different pruning strengths
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=42, ccp_alpha=ccp_alpha)
```

```
    clf.fit(X_train, y_train)
    clfs.append(clf)

# Evaluate accuracy for each alpha
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

# Plot accuracy vs alpha
plt.figure(figsize=(7, 5))
plt.plot(ccp_alphas, train_scores, marker='o', label='Train Accuracy', drawstyle="steps-post")
plt.plot(ccp_alphas, test_scores, marker='o', label='Test Accuracy', drawstyle="steps-post")
plt.xlabel("Effective Alpha (ccp_alpha)")
plt.ylabel("Accuracy")
plt.title("Cost Complexity Pruning - Accuracy vs Alpha")
plt.legend()
plt.show()

#Select Best Alpha and Prune Tree

best_alpha = ccp_alphas[np.argmax(test_scores)]
clf_pruned = DecisionTreeClassifier(random_state=42, ccp_alpha=best_alpha)
clf_pruned.fit(X_train, y_train)

# Evaluate pruned tree
y_pred_pruned = clf_pruned.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_pruned)

print("\nBest ccp_alpha:", best_alpha)
print("Accuracy (Pruned Tree):", round(acc_pruned, 3))


# Compare Tree Structures
plt.figure(figsize=(10, 6))
plot_tree(clf_unpruned, filled=True, feature_names=X.columns)
plt.title("Unpruned Decision Tree")
plt.show()

plt.figure(figsize=(10, 6))
plot_tree(clf_pruned, filled=True, feature_names=X.columns)
plt.title("Pruned Decision Tree")
plt.show()

print("\nNumber of nodes in Unpruned Tree:", clf_unpruned.tree_.node_count)
print("Number of nodes in Pruned Tree:", clf_pruned.tree_.node_count)

print("\nAnalysis:")
```

```
print(f"Unpruned Tree Accuracy = {acc_unpruned:.3f}")
print(f"Pruned Tree Accuracy   = {acc_pruned:.3f}")
print("The pruned tree is smaller and often generalizes better on noisy Monk2 data.")
```

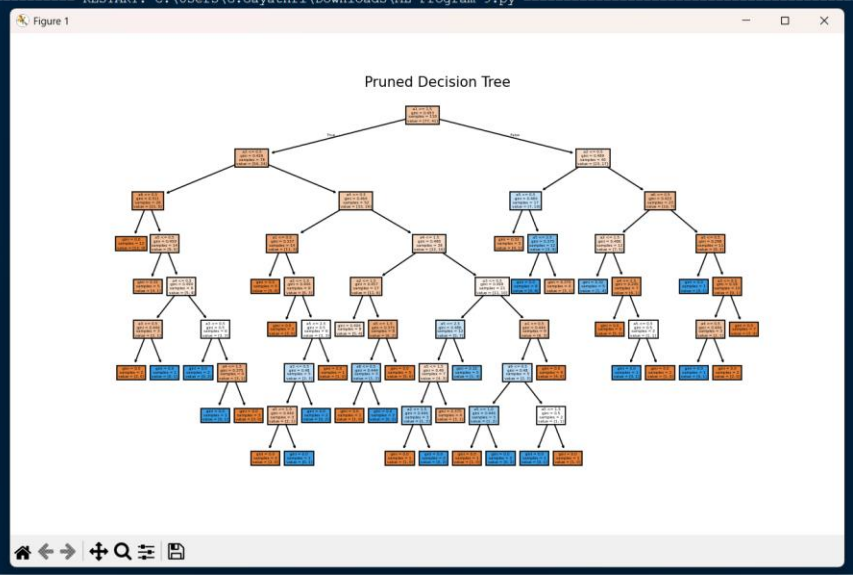
OUTPUT:


File Edit Shell Debug Options Window Help
Python 3.13.9 (tags/v3.13.9:8183fa5, Oct 14 2025, 14:09:13) [MSC v.1944 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\G.Gayathri\Downloads\ML Program 9.py ==========
Dataset shape: (169, 7)
class a1 a2 a3 a4 a5 a6
0 0 1 1 1 1 2 2
1 0 1 1 1 1 4 1
2 0 1 1 1 2 1 1
3 0 1 1 1 2 1 2
4 0 1 1 1 2 2 1
Accuracy (Unpruned Tree): 0.529
Figure 1
Cost Complexity Pruning - Accuracy vs Alpha
Train Accuracy
Test Accuracy
Accuracy
Effective Alpha (ccp_alpha)


File Edit Shell Debug Options Window Help
Python 3.13.9 (tags/v3.13.9:8183fa5, Oct 14 2025, 14:09:13) [MSC v.1944 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\G.Gayathri\Downloads\ML Program 9.py ==========
Dataset shape: (169, 7)
class a1 a2 a3 a4 a5 a6
0 0 1 1 1 1 2 2
1 0 1 1 1 1 4 1
2 0 1 1 1 2 1 1
3 0 1 1 1 2 1 2
4 0 1 1 1 2 2 1
Accuracy (Unpruned Tree): 0.529
Best ccp_alpha: 0.006779661016949151
Accuracy (Pruned Tree): 0.549
Figure 1
Unpruned Decision Tree
(x, y) = (0.771, 0.026)

File Edit Shell Debug Options Window Help

```
Python 3.13.9 (tags/v3.13.9:8183fa5, Oct 14 2025, 14:09:13) [MSC v.1944 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\G.Gayathri\Downloads\ML Program 9.py ==========
Dataset shape: (169, 7)
   class  a1  a2  a3  a4  a5  a6
0      0   1   1   1   1   2   2
1      0   1   1   1   1   4   1
2      0   1   1   1   2   1   1
3      0   1   1   1   2   1   2
4      0   1   1   1   2   2   1

Accuracy (Unpruned Tree): 0.529

Best ccp_alpha: 0.006779661016949151
Accuracy (Pruned Tree): 0.549
```



Ex.No:10

Date:

Artificial Neural Network -Backpropagation algorithm

Aim:

To write a python program to build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data set.

Algorithm:

Step 1: Load and scale the Diabetes dataset.

Step 2: Initialize neural network weights and biases.

Step 3: Perform forward propagation using sigmoid activation.

Step 4: Compute loss using Mean Squared Error.

Step 5: Apply backpropagation to update weights.

Step 6: Repeat for multiple epochs.

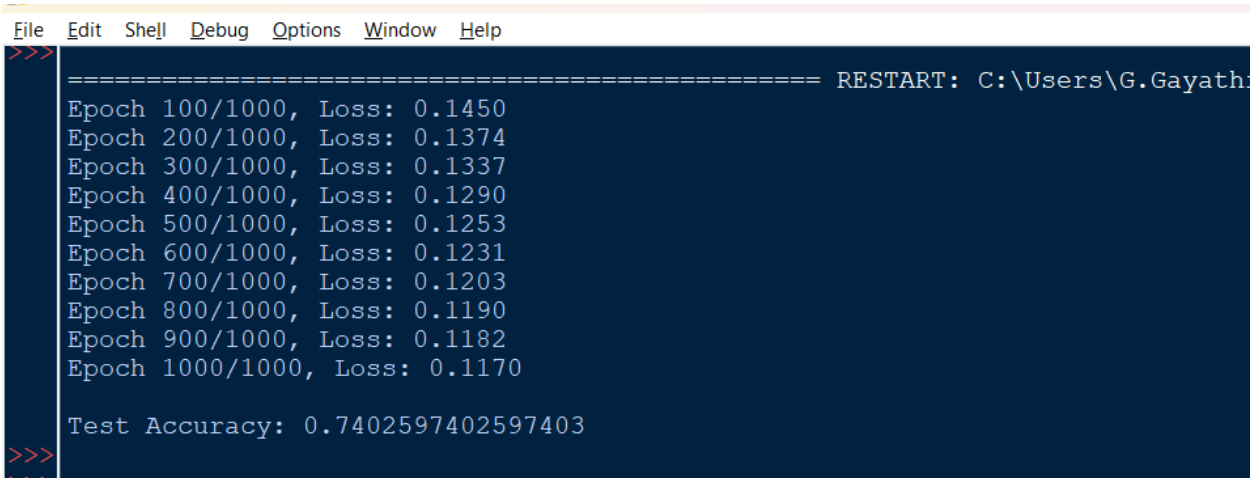
Step 7: Test network and compute accuracy.

#Program 10: Artificial Neural Network -Backpropagation algorithm

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load dataset
url = "E:/ML Program Dataset/diabetes.csv"
data = pd.read_csv(url)

X = data.drop('Outcome', axis=1).values
y = data['Outcome'].values.reshape(-1, 1)  # Column vector

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define helper functions

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Initialize network parameters
input_dim = X_train.shape[1]
hidden_dim = 5
output_dim = 1
learning_rate = 0.1
epochs = 1000

# Initialize weights randomly
np.random.seed(42)
W1 = np.random.randn(input_dim, hidden_dim)
b1 = np.zeros((1, hidden_dim))
W2 = np.random.randn(hidden_dim, output_dim)
b2 = np.zeros((1, output_dim))
```

```
# Train network using backpropagation
for epoch in range(epochs):

    Z1 = np.dot(X_train, W1) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(A1, W2) + b2
    A2 = sigmoid(Z2)  # Output layer

    # Compute loss (Mean Squared Error)
    loss = np.mean((y_train - A2) ** 2)

    # Backward pass
    dA2 = -(y_train - A2) * sigmoid_derivative(A2)
    dW2 = np.dot(A1.T, dA2)
    db2 = np.sum(dA2, axis=0, keepdims=True)

    dA1 = np.dot(dA2, W2.T) * sigmoid_derivative(A1)
    dW1 = np.dot(X_train.T, dA1)
    db1 = np.sum(dA1, axis=0, keepdims=True)

    # Update weights
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2
    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1

    # Print loss every 100 epochs
    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch+1}/{epochs}, Loss: {loss:.4f}")

# Test network
Z1_test = np.dot(X_test, W1) + b1
A1_test = sigmoid(Z1_test)
Z2_test = np.dot(A1_test, W2) + b2
A2_test = sigmoid(Z2_test)

y_pred = (A2_test > 0.5).astype(int)

# Accuracy
accuracy = np.mean(y_pred == y_test)
print("\nTest Accuracy:", accuracy)
```

OUTPUT:


File Edit Shell Debug Options Window Help
>>>
========== RESTART: C:\Users\G.Gayathi
Epoch 100/1000, Loss: 0.1450
Epoch 200/1000, Loss: 0.1374
Epoch 300/1000, Loss: 0.1337
Epoch 400/1000, Loss: 0.1290
Epoch 500/1000, Loss: 0.1253
Epoch 600/1000, Loss: 0.1231
Epoch 700/1000, Loss: 0.1203
Epoch 800/1000, Loss: 0.1190
Epoch 900/1000, Loss: 0.1182
Epoch 1000/1000, Loss: 0.1170
Test Accuracy: 0.7402597402597403
>>>

Ex.No:11

Date:

Support Vector Classification (SVC) with Linear Kernel

Aim:

To write a python program to implement Support Vector Classification (SVC) with linear kernel

Algorithm:

Step 1: Load the Diabetes dataset.

Step 2: Split data into features and target.

Step 3: Normalize features using StandardScaler.

Step 4: Train Support Vector Classifier with linear kernel.

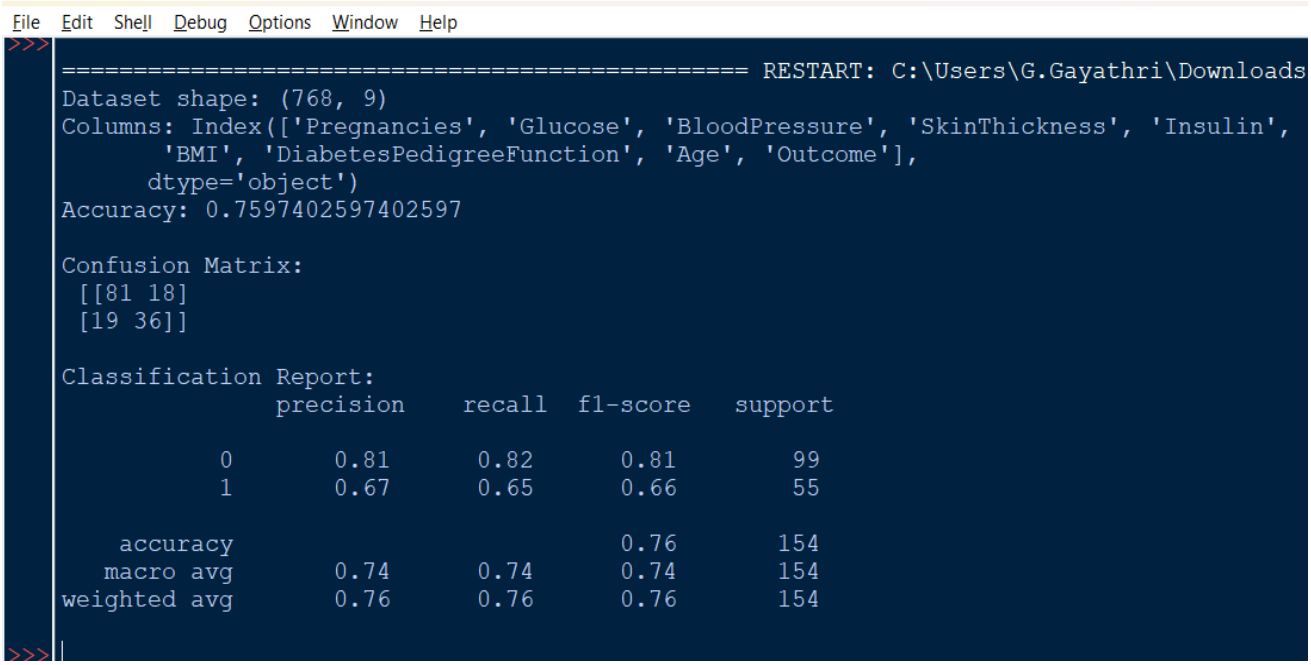
Step 5: Predict outcomes on test data.

Step 6: Evaluate performance using accuracy, confusion matrix, and report.

Program 11:Support Vector Classification (SVC) with Linear Kernel

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
url = "E:/ML Program Dataset/diabetes.csv"  # Replace with your path
data = pd.read_csv(url)

# Inspect dataset
print("Dataset shape:", data.shape)
print("Columns:", data.columns)

# Prepare features and target
X = data.drop('Outcome', axis=1)  # Features
y = data['Outcome']               # Target (1 = diabetes, 0 = no diabetes)

# Split 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
)

# Feature scaling (important for SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train SVC with linear kernel
svc = SVC(kernel='linear', random_state=42)
svc.fit(X_train_scaled, y_train)

# Make predictions
y_pred = svc.predict(X_test_scaled)

# Evaluate the model
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:


File Edit Shell Debug Options Window Help
>>>
========== RESTART: C:\Users\G.Gayathri\Downloads
Dataset shape: (768, 9)
Columns: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
dtype='object')
Accuracy: 0.7597402597402597
Confusion Matrix:
[[81 18]
[19 36]]
Classification Report:
precision recall f1-score support
0 0.81 0.82 0.81 99
1 0.67 0.65 0.66 55
accuracy 0.76 154
macro avg 0.74 0.74 0.74 154
weighted avg 0.76 0.76 0.76 154
>>>

Ex.No:12

Date:

Logistic Regression for Diabetes Prediction

Aim:

To write a python program to implement logistic regression for binary classification.

Algorithm:

Step 1: Load the Diabetes dataset.

Step 2: Separate features and target variable.

Step 3: Split dataset into training and testing sets.

Step 4: Standardize features using StandardScaler.

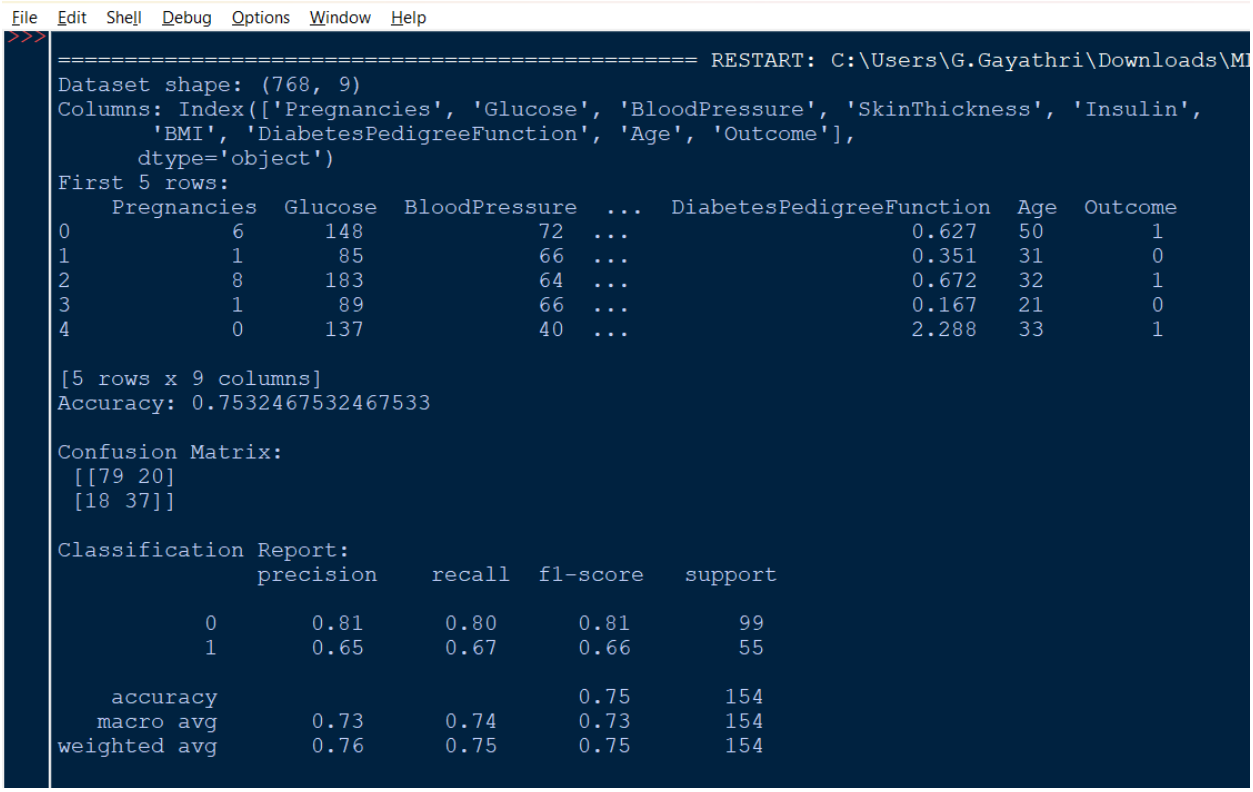
Step 5: Train Logistic Regression model.

Step 6: Predict on test set and evaluate using performance metrics.

Program 12:Logistic Regression for Diabetes Prediction

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
# Replace the path with your local CSV file
# Example: "diabetes.csv" from Kaggle or UCI
url = "E:/ML Program Dataset/diabetes.csv"
data = pd.read_csv(url)

# Inspect the dataset
print("Dataset shape:", data.shape)
print("Columns:", data.columns)
print("First 5 rows:\n", data.head())

# Prepare features and target
# Assume the dataset has a column 'Outcome' as target (1 = diabetes, 0 = no diabetes)
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Split 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
)
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Logistic Regression model
logreg = LogisticRegression(random_state=42)
logreg.fit(X_train_scaled, y_train)

# Make predictions
y_pred = logreg.predict(X_test_scaled)

# Evaluate the model
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:


File Edit Shell Debug Options Window Help
>>>
========== RESTART: C:\Users\G.Gayathri\Downloads\M
Dataset shape: (768, 9)
Columns: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
dtype='object')
First 5 rows:
Pregnancies Glucose BloodPressure ... DiabetesPedigreeFunction Age Outcome
0 6 148 72 ... 0.627 50 1
1 1 85 66 ... 0.351 31 0
2 8 183 64 ... 0.672 32 1
3 1 89 66 ... 0.167 21 0
4 0 137 40 ... 2.288 33 1
[5 rows x 9 columns]
Accuracy: 0.7532467532467533
Confusion Matrix:
[[79 20]
[18 37]]
Classification Report:
precision recall f1-score support
0 0.81 0.80 0.81 99
1 0.65 0.67 0.66 55
accuracy 0.75 154
macro avg 0.73 0.74 0.73 154
weighted avg 0.76 0.75 0.75 154