30) write a python program to implement decision tree regressor on Advertising.csv and print mean absolute error

```
from sklearn.preprocessing import StandardScaler


# Load the dataset

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


# Split the data into features and target variable

X = df.drop('Sales', axis=1)

y = df['Sales']


# Split data 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

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


# Initialize and fit the regressor

regressor = DecisionTreeRegressor(random_state=42)

regressor.fit(X_train, y_train)


# Predict the test set results

y_pred = regressor.predict(X_test)


# Calculate the mean absolute error

mae = mean_absolute_error(y_test, y_pred)

print("Mean Absolute Error:", mae)
```

31) write a python program to implement decision tree regressor on Advertising.csv and print mean square error

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv("Advertising.csv")

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

# Split data 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
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Initialize and fit the regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)
```

```
# Predict the test set results

y_pred = regressor.predict(X_test)


# Calculate the mean squared error (MSE)

mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error:", mse)
```

32) write a python program to implement decision tree regressor on Advertising.csv and print root mean absolute error

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean_squared_error

from sklearn.preprocessing import StandardScaler

import numpy as np


# Load the dataset

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


# Split the data into features and target variable

X = df.drop('Sales', axis=1)

y = df['Sales']


# Split data 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

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


# Initialize and fit the regressor

regressor = DecisionTreeRegressor(random_state=42)

regressor.fit(X_train, y_train)


# Predict the test set results

y_pred = regressor.predict(X_test)


# Calculate the mean squared error (MSE)

mse = mean_squared_error(y_test, y_pred)


# Calculate the root mean squared error (RMSE)

rmse = np.sqrt(mse)

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

33) write a python program to implement knn classifier on Iris.csv dataset and print accuracy score (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score

from sklearn.preprocessing import LabelEncoder
```

```
# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

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

# Split data 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)

# Initialize and fit the KNN classifier
k = 3  # Number of neighbors
classifier = KNeighborsClassifier(n_neighbors=k)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
```

print("Accuracy Score:", accuracy)

34) write a python program to implement knn classifier on Iris.csv dataset and print classification report (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report

from sklearn.preprocessing import LabelEncoder


# Load the dataset

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


# Convert categorical column 'Species' to numerical using LabelEncoder

label_encoder = LabelEncoder()

df['Species'] = label_encoder.fit_transform(df['Species'])


# Split the data into features and target variable

X = df.drop('Species', axis=1)

y = df['Species']


# Split data 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)


# Initialize and fit the KNN classifier

k = 3  # Number of neighbors
```

```
classifier = KNeighborsClassifier(n_neighbors=k)

classifier.fit(X_train, y_train)



# Predict the test set results

y_pred = classifier.predict(X_test)



# Print the classification report

print("Classification Report:")

print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

35) write a python program to implement knn classifier on Iris.csv dataset and print confusion matrix (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

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

# Split data 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)

# Initialize and fit the KNN classifier
k = 3  # Number of neighbors
classifier = KNeighborsClassifier(n_neighbors=k)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)
```

```
# Print the confusion matrix
print("Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

36) write a python program to implement support vector classifier on Iris.csv dataset and print accuracy score (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

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

# Split data 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)

# Initialize and fit the SVC classifier
classifier = SVC(kernel='linear', random_state=42)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

37) write a python program to implement support vector classifier on Iris.csv dataset and print classification report (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")
```

```
# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

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

# Split data 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)

# Initialize and fit the SVC classifier
classifier = SVC(kernel='linear', random_state=42)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

38) write a python program to implement support vector classifier on Iris.csv dataset and print confusion matrix (hint:use labelencoder from sklearn to convert the column Species)

```
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

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

# Split data 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)

# Initialize and fit the SVC classifier
classifier = SVC(kernel='linear', random_state=42)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)
```

```
# Print the confusion matrix
print("Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

39) write a python program to implement Decision Tree classifier on Iris.csv dataset and print accuracy score (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

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

# Split data 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)

# Initialize and fit the Decision Tree classifier
classifier = DecisionTreeClassifier(random_state=42)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

40) write a python program to implement Decision Tree classifier on Iris.csv dataset and print classification report (hint:use labelencoder from sklearn to convert the column Species)

```
from sklearn.metrics import classification_report
```

```
from sklearn.preprocessing import LabelEncoder


# Load the dataset

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


# Convert categorical column 'Species' to numerical using LabelEncoder

label_encoder = LabelEncoder()

df['Species'] = label_encoder.fit_transform(df['Species'])


# Split the data into features and target variable

X = df.drop('Species', axis=1)

y = df['Species']


# Split data 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)


# Initialize and fit the Decision Tree classifier

classifier = DecisionTreeClassifier(random_state=42)

classifier.fit(X_train, y_train)


# Predict the test set results

y_pred = classifier.predict(X_test)


# Print the classification report

print("Classification Report:")
```

```
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

41) write a python program to implement Decision Tree classifier on Iris.csv dataset and print confusion matrix (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion_matrix

from sklearn.preprocessing import LabelEncoder


# Load the dataset

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


# Convert categorical column 'Species' to numerical using LabelEncoder

label_encoder = LabelEncoder()

df['Species'] = label_encoder.fit_transform(df['Species'])


# Split the data into features and target variable

X = df.drop('Species', axis=1)

y = df['Species']


# Split data 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)


# Initialize and fit the Decision Tree classifier

classifier = DecisionTreeClassifier(random_state=42)

classifier.fit(X_train, y_train)
```

```
# Predict the test set results

y_pred = classifier.predict(X_test)


# Print the confusion matrix

print("Confusion Matrix:")

conf_matrix = confusion_matrix(y_test, y_pred)

print(conf_matrix)
```

42) write a python program to implement linear regression on salary.csv dataset (varied test size, eg. test_szie = 0.2, 0.3, 0.4, 0.5)and print a plot bar chart between varied test size and mean squared error

```
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 LinearRegression

from sklearn.metrics import mean_squared_error


# Load the dataset

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


# Separate features and target variable

X = df[['YearsExperience']]

y = df['Salary']
```

```
# Initialize an empty dictionary to store MSE for each test size
mse_dict = {}

# Iterate over different test sizes
test_sizes = [0.2, 0.3, 0.4, 0.5]
for test_size in test_sizes:
    # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
random_state=42)

    # Initialize and fit the linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

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

    # Calculate mean squared error
    mse = mean_squared_error(y_test, y_pred)

    # Store the MSE for the current test size
    mse_dict[test_size] = mse

# Plotting the bar chart
plt.bar(mse_dict.keys(), mse_dict.values(), color='skyblue')
```

```
plt.xlabel('Test Size')

plt.ylabel('Mean Squared Error')

plt.title('Mean Squared Error vs. Test Size')

plt.xticks(np.arange(0.2, 0.6, 0.1))

plt.show()
```

43) Write a Python program that reads iris.csv with multiple features. Perform Principal Component Analysis (PCA) on the dataset to reduce its dimensionality (n_components=3).

```
from sklearn.preprocessing import StandardScaler


# Load the dataset

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


# Separate features and target variable

X = df.drop('Species', axis=1)

y = df['Species']


# Standardize the features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Perform PCA with 3 components

pca = PCA(n_components=3)

X_pca = pca.fit_transform(X_scaled)


# Create a DataFrame for the PCA components
```

```
df_pca = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2', 'PC3'])


# Concatenate the PCA components with the target variable

df_final = pd.concat([df_pca, y], axis=1)


# Print the final DataFrame with PCA components

print("DataFrame after PCA:")

print(df_final.head())
```

44) Implement LDA in Python using scikit-learn or another machine learning library. Visualize the reduced data in a scatter plot with different classes represented by distinct colors. (read HearDisease1.csv)

```
df = pd.read_csv("HeartDisease1.csv")


# Encode the target variable 'target' into numerical labels

label_encoder = LabelEncoder()

df['target'] = label_encoder.fit_transform(df['target'])


# Separate features and target variable

X = df.drop('target', axis=1)

y = df['target']


# Initialize and fit the LDA model

lda = LinearDiscriminantAnalysis(n_components=2)

X_lda = lda.fit_transform(X, y)


# Create a DataFrame for the LDA components
```

```
df_lda = pd.DataFrame(data=X_lda, columns=['LD1', 'LD2'])


# Concatenate the LDA components with the target variable

df_final = pd.concat([df_lda, y], axis=1)


# Visualize the reduced data in a scatter plot with different classes represented by distinct colors

plt.figure(figsize=(10, 6))

sns.scatterplot(x='LD1', y='LD2', hue='target', data=df_final, palette='viridis', legend='full')

plt.title('Scatter Plot of LDA Components')

plt.xlabel('LD1')

plt.ylabel('LD2')

plt.show()
```

45) Write a Python program that reads iris.csv with multiple features. Perform Principal Component Analysis (PCA) and print confusion matrix

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.preprocessing import LabelEncoder


# Load the dataset

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


# Convert categorical column 'Species' to numerical using LabelEncoder
```

```
label_encoder = LabelEncoder()

df['Species'] = label_encoder.fit_transform(df['Species'])


# Separate features and target variable

X = df.drop('Species', axis=1)

y = df['Species']


# Split data 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)


# Perform PCA with 3 components

pca = PCA(n_components=3)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)


# Initialize and fit a classifier (Random Forest in this case)

classifier = RandomForestClassifier(random_state=42)

classifier.fit(X_train_pca, y_train)


# Predict the test set results

y_pred = classifier.predict(X_test_pca)


# Print the confusion matrix

print("Confusion Matrix:")

conf_matrix = confusion_matrix(y_test, y_pred)
```

```
print(conf_matrix)
```

46) Implement LDA in Python using scikit-learn or another machine learning library and print F1-score, precision and accuracy (read HearDisease1.csv)

```
import pandas as pd

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.model_selection import train_test_split

from sklearn.metrics import f1_score, precision_score, accuracy_score

from sklearn.preprocessing import LabelEncoder


# Load the dataset

df = pd.read_csv("HeartDisease1.csv")


# Encode the target variable 'target' into numerical labels

label_encoder = LabelEncoder()

df['target'] = label_encoder.fit_transform(df['target'])


# Separate features and target variable

X = df.drop('target', axis=1)

y = df['target']


# Split data 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)


# Initialize and fit the LDA model

lda = LinearDiscriminantAnalysis()
```

```
lda.fit(X_train, y_train)


# Predict the test set results

y_pred = lda.predict(X_test)


# Calculate the evaluation metrics

f1 = f1_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

accuracy = accuracy_score(y_test, y_pred)


# Print the evaluation metrics

print("F1-score:", f1)

print("Precision:", precision)

print("Accuracy:", accuracy)
```

47) Create a Python program that reads 'Advertising.csv' evaluates the performance of a multiple linear regression model. Calculate and display any one error metric for model evaluation.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error


# Load the dataset

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


# Separate features and target variable
```

```
X = df[['TV', 'Radio', 'Newspaper']]

y = df['Sales']


# Split data 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)


# Initialize and fit the multiple linear regression model

model = LinearRegression()

model.fit(X_train, y_train)


# Predict the test set results

y_pred = model.predict(X_test)


# Calculate the Mean Squared Error (MSE)

mse = mean_squared_error(y_test, y_pred)


# Print the Mean Squared Error

print("Mean Squared Error:", mse)
```

48) Create a Python program that reads 'Advertising.csv'; using multiple linear regression fill the missing values in the target variable after row 180

```
import pandas as pd

from sklearn.linear_model import LinearRegression


# Load the dataset

df = pd.read_csv("Advertising.csv")
```

```
# Separate the dataset into two parts: before and after row 180
df_before = df.loc[:179]
df_after = df.loc[180:]

# Prepare the data for model training and prediction
X_train = df_before[['TV', 'Radio', 'Newspaper']]
y_train = df_before['Sales']
X_test = df_after[['TV', 'Radio', 'Newspaper']]

# Initialize and fit the multiple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict the missing values
y_pred = model.predict(X_test)

# Fill the missing values in the target variable 'Sales' after row 180
df_after['Sales'] = y_pred

# Concatenate the dataframes back together
df_filled = pd.concat([df_before, df_after])

# Save the filled dataset to a new CSV file
df_filled.to_csv("Advertising_filled.csv", index=False)
```

print("Missing values in 'Sales' after row 180 have been filled using multiple linear regression.")

49) Create a Python program to read 'diabetes.csv' and evaluate the performance of a multivariate logistic regression model for multiclass classification. Calculate and display the confusion matrix.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix


# Load the dataset

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


# Separate features and target variable

X = df.drop('class', axis=1)

y = df['class']


# Split data 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)


# Initialize and fit the logistic regression model

model = LogisticRegression(multi_class='multinomial', solver='lbfgs')

model.fit(X_train, y_train)


# Predict the test set results

y_pred = model.predict(X_test)
```

```
# Calculate the confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)


# Print the confusion matrix

print("Confusion Matrix:")

print(conf_matrix)
```

50) Create a Python program to read 'diabetes.csv' and evaluate the performance of a multivariate logistic regression model for multiclass classification. Calculate and display accuracy

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


# Load the dataset

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


# Separate features and target variable

X = df.drop('class', axis=1)

y = df['class']


# Split data 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)
```

```
# Initialize and fit the logistic regression model

model = LogisticRegression(multi_class='multinomial', solver='lbfgs')

model.fit(X_train, y_train)


# Predict the test set results

y_pred = model.predict(X_test)


# Calculate and display the accuracy

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

10) write a python program to implement support vector classifier on diabetes.csv dataset and print accuracy score

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score


# Load the dataset

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


# Separate features and target variable

X = df.drop('class', axis=1)

y = df['class']


# Split data 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)


# Initialize and fit the SVC classifier

classifier = SVC(kernel='linear')

classifier.fit(X_train, y_train)


# Predict the test set results

y_pred = classifier.predict(X_test)


# Calculate and print the accuracy score

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy Score:", accuracy)
```

9) write a python program to implement support vector classifier on diabetes.csv dataset and print precision score

```
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 precision_score

from sklearn.metrics import classification_report


# Load the dataset

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


# Separate features and target variable
```

```
X = data.drop('Outcome', axis=1)

y = data['Outcome']


# 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)


# Standardize features by removing the mean and scaling to unit variance

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Create a Support Vector Classifier

svc = SVC(kernel='rbf', gamma='auto')


# Train the classifier

svc.fit(X_train_scaled, y_train)


# Predict the labels for test set

y_pred = svc.predict(X_test_scaled)


# Calculate precision score

precision = precision_score(y_test, y_pred)


print("Precision Score:", precision)

print("\nClassification Report:")
```

print(classification_report(y_test, y_pred))

8) write a python program to implement decision tree classifier on diabetes.csv dataset and print confusion matrix

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion_matrix


# Load the dataset

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


# Separate features and target variable

X = data.drop('Outcome', axis=1)

y = data['Outcome']


# 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)


# Standardize features by removing the mean and scaling to unit variance

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Create a Decision Tree Classifier

dt_classifier = DecisionTreeClassifier(random_state=42)
```

```
# Train the classifier

dt_classifier.fit(X_train_scaled, y_train)


# Predict the labels for test set

y_pred = dt_classifier.predict(X_test_scaled)


# Calculate confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)


print("Confusion Matrix:")

print(conf_matrix)
```

7) write a python program to implement decision tree classifier on diabetes.csv dataset and print classification report

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification_report


# Load the dataset

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


# Separate features and target variable

X = data.drop('Outcome', axis=1)

y = data['Outcome']
```

```
# 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)


# Standardize features by removing the mean and scaling to unit variance

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Create a Decision Tree Classifier

dt_classifier = DecisionTreeClassifier(random_state=42)


# Train the classifier

dt_classifier.fit(X_train_scaled, y_train)


# Predict the labels for test set

y_pred = dt_classifier.predict(X_test_scaled)


# Generate classification report

report = classification_report(y_test, y_pred)


print("Classification Report:")

print(report)
```

6) write a python program to implement a knn classifier on diabetes.csv and print confusion matrix

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

# Load the dataset
data = pd.read_csv("diabetes.csv")

# Separate features and target variable
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# 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)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a KNN Classifier with k=5
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier
```

```
knn_classifier.fit(X_train_scaled, y_train)


# Predict the labels for test set

y_pred = knn_classifier.predict(X_test_scaled)


# Calculate confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)


print("Confusion Matrix:")

print(conf_matrix)
```