WEEK-7

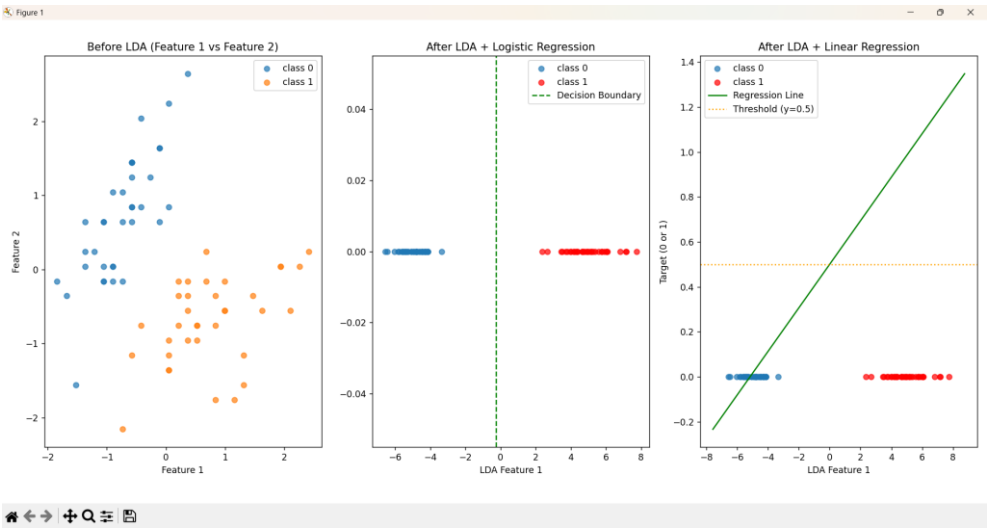
Aim: Write a program to perform Linear Discriminant Analysis for binary classification considering a real time dataset.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.model_selection import train_test_split
iris = load_iris()
X = iris.data
y = iris.target
mask = y < 2
X = X[mask]
y = y[mask]
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.3, random_state=42, stratify=)
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
X_train_2D = X_train_std[:, :2]
X_test_2D = X_test_std[:, :2]
lda = LinearDiscriminantAnalysis(n_components=1)
X_train_lda = lda.fit_transform(X_train_std, y_train)
X_test_lda = lda.transform(X_test_std)
log_reg = LogisticRegression()
log_reg.fit(X_train_lda, y_train)
log_bou = -log_reg.intercept_[0] / log_reg.coef_[0][0]
lin_reg = LinearRegression()
lin_reg.fit(X_train_lda, y_train)
X_min, X_max = X_train_lda.min() - 1, X_train_lda.max() + 1
XX = np.linspace(X_min, X_max, 100).reshape(-1, 1)
yy_lin = lin_reg.predict(XX)
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].scatter(X_train_2D[y_train == 0, 0], X_train_2D[y_train == 0, 1],
label='class 0', alpha=0.7)
axes[0].scatter(X_train_2D[y_train == 1, 0], X_train_2D[y_train == 1, 1],
```

```
label='class 1', alpha=0.7)
axes[0].set_title("Before LDA (Feature 1 vs Feature 2)")
axes[0].set_xlabel("Feature 1")
axes[0].set_ylabel("Feature 2")
axes[0].legend()
axes[1].scatter(X_train_lda[y_train == 0], np.zeros_like(X_train_lda[y_train
== 0]), label='class 0', alpha=0.7)
axes[1].scatter(X_train_lda[y_train == 1], np.zeros_like(X_train_lda[y_train
== 1]), label='class 1', alpha=0.7, color='red')
axes[1].axvline(x=log_bou, linestyle='--', color='g', label="Decision Boundary")
axes[1].set_title("After LDA + Logistic Regression")
axes[1].set_xlabel("LDA Feature 1")
axes[1].set_ylabel("")
axes[1].legend()
axes[2].scatter(X_train_lda[y_train == 0], np.zeros_like(X_train_lda[y_train
== 0]), label='class 0', alpha=0.7)
axes[2].scatter(X_train_lda[y_train == 1], np.zeros_like(X_train_lda[y_train
== 1]), label='class 1', alpha=0.7, color='red')
axes[2].plot(XX, yy_lin, color="green", label="Regression Line")
axes[2].axhline(y=0.5, color='orange', linestyle=':', label='Threshold (y=0.5)')
axes[2].set_title("After LDA + Linear Regression")
axes[2].set_xlabel("LDA Feature 1")
axes[2].set_ylabel("Target (0 or 1)")
axes[2].legend()
plt.tight_layout()
plt.show()
```

Output:



WEEK-8

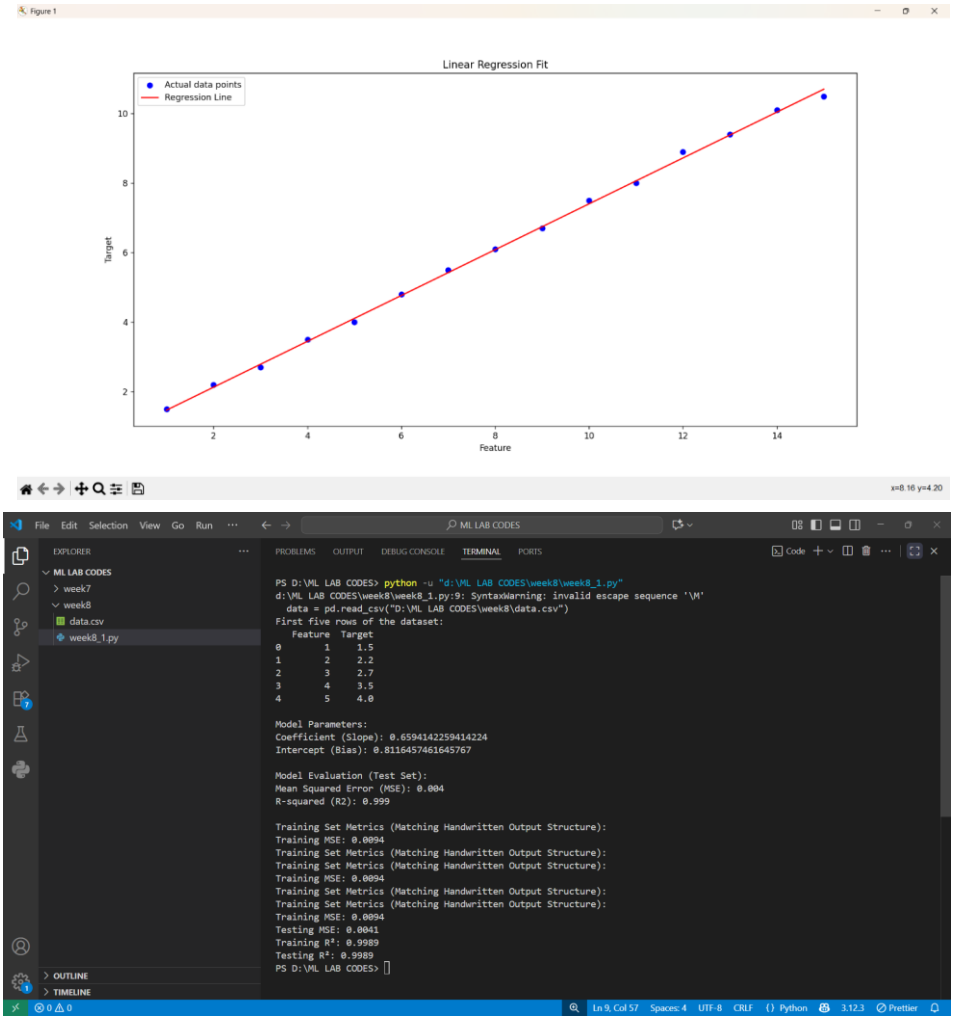
Aim: Write a program to implement the linear Regression for a sample training data set stored as a .CSV file

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
try:
   data = pd.read_csv("D:\ML LAB CODES\week8\data.csv")
except FileNotFoundError:
   print("Error: 'weeks.csv' not found at the specified path.")
   print("Using dummy data for execution.")
   dummy_data = {
      'Feature': np.arange(10).reshape(-1, 1).flatten(),
      'Target': (np.arange(10) * 0.6) + 1 + (np.random.rand(10) * 0.5)
   }
   data = pd.DataFrame(dummy_data)
print("First five rows of the dataset:")
print(data.head(5))
X = data['Feature'].values.reshape(-1, 1)
y = data['Target'].values
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=0)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("\nModel Parameters:")
print(f"Coefficient (Slope): {model.coef_[0]}")
print(f"Intercept (Bias): {model.intercept_}")
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("\nModel Evaluation (Test Set):")
print(f"Mean Squared Error (MSE): {round(mse, 3)}")
print(f"R-squared (R2): {round(r2, 3)}")
X_plot = np.sort(X, axis=0)
y_plot = model.predict(X_plot)
```

```
plt.scatter(X, y, color="blue", label="Actual data points")
plt.plot(X_plot, y_plot, color="red", label="Regression Line")
plt.xlabel("Feature")
plt.ylabel("Target")
plt.title("Linear Regression Fit")
plt.legend()
plt.show()
y_train_pred = model.predict(X_train)
train_mse = mean_squared_error(y_train, y_train_pred)
train_r2 = r2_score(y_train, y_train_pred)
print("\nTraining Set Metrics (Matching Handwritten Output Structure):")
print(f"Training MSE: {round(train_mse, 4)}")
print(f"Testing MSE: {round(mse, 4)}")
print(f"Training R²: {round(train_r2, 4)}")
print(f"Testing R²: {round(r2, 4)}")
```

Output:



WEEK-9

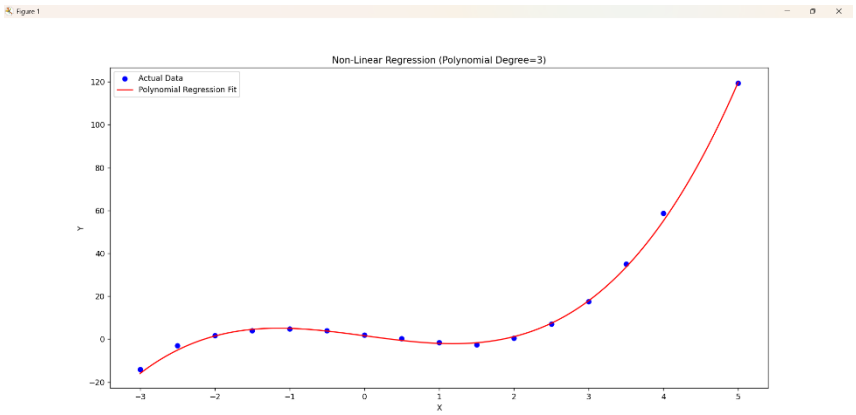
Aim: Write a program to implement the Non-linear Regression for a sample training data set stored as a .CSV file. Compute Mean Square Error by considering few test data sets.

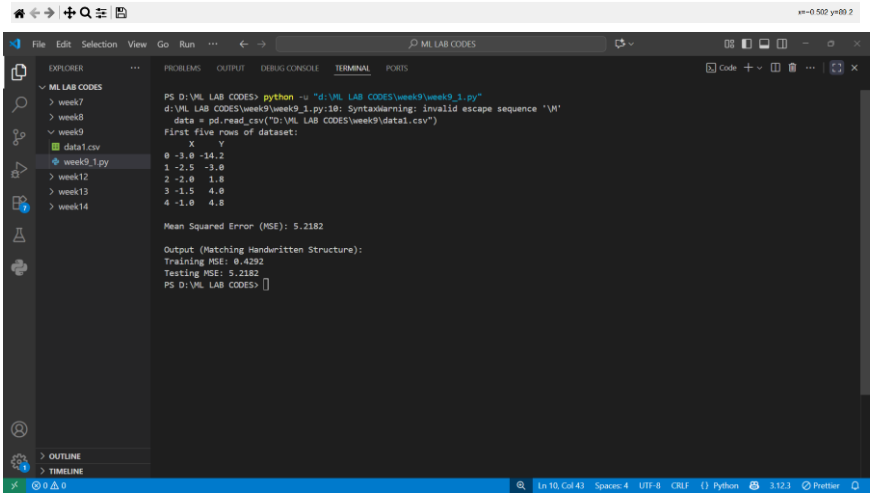
Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
try:
   data = pd.read_csv("D:\ML LAB CODES\week9\data1.csv ")
except FileNotFoundError:
   print("Error: 'data1.csv' not found at the specified path.")
   print("Using dummy data for execution.")
   X_dummy = np.arange(-5, 5, 0.5)
   y_dummy = 0.5 * X_dummy**3 - 10 * X_dummy + 5 + np.random.normal(0,
10, len(X_dummy))
   data = pd.DataFrame({'X': X_dummy, 'Y': y_dummy})
print("First five rows of dataset:")
print(data.head())
X = data['X'].values.reshape(-1, 1)
y = data['Y'].values
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=42)
degree = 3
poly = PolynomialFeatures(degree=degree)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)
model = LinearRegression()
model.fit(X_poly_train, y_train)
y_pred = model.predict(X_poly_test)
mse = mean_squared_error(y_test, y_pred)
print(f"\nMean Squared Error (MSE): {mse:.4f}")
y_train_pred = model.predict(X_poly_train)
train_mse = mean_squared_error(y_train, y_train_pred)
```

```
print("\nOutput (Matching Handwritten Structure):")
print(f"Training MSE: {train_mse:.4f}")
print(f"Testing MSE: {mse:.4f}")
plt.scatter(X, y, color='blue', label='Actual Data')
X_min, X_max = X.min(), X.max()
X_curve = np.linspace(X_min, X_max, 200).reshape(-1, 1)
X_curve_poly = poly.transform(X_curve)
y_curve = model.predict(X_curve_poly)
plt.plot(X_curve, y_curve, color='red', label='Polynomial Regression Fit')
plt.xlabel('X')
plt.ylabel('Y')
plt.title(f'Non-Linear Regression (Polynomial Degree={degree})')
plt.legend()
plt.show()
```

Output:





WEEK-10

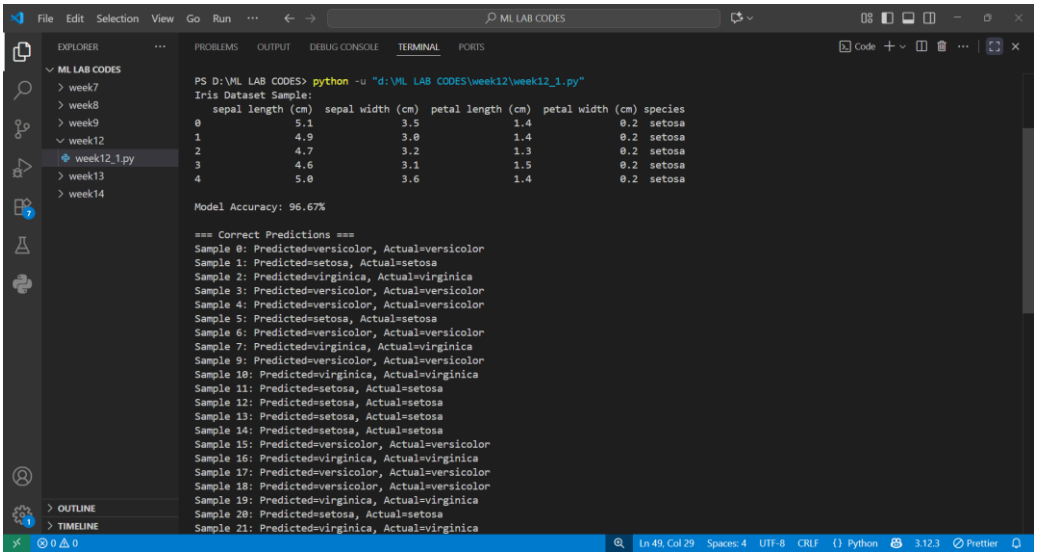
Aim: Write a program to implement the Support Vector Machine algorithm to classify the iris data set. Print both correct and wrong predictions.

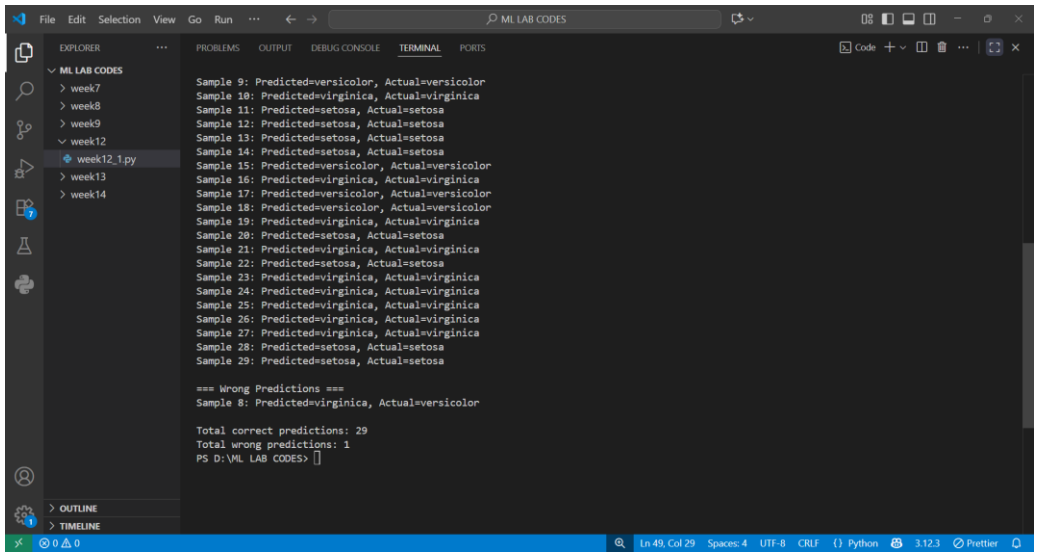
Program:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
df = pd.DataFrame(X, columns=feature_names)
df['species'] = [target_names[i] for i in y]
print("Iris Dataset Sample:")
print(df.head())
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm_model = SVC(kernel='linear', C=1.0, random_state=42)
svm_model.fit(X_train_scaled, y_train)
y_pred = svm_model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Accuracy: {accuracy * 100:.2f}%")
correct_predictions_count = 0
wrong_predictions_count = 0
print("\n=== Correct Predictions ===")
for i in range(len(y_test)):
   if y_test[i] == y_pred[i]:
      correct_predictions_count += 1
      print(f"Sample {i}: Predicted={target_names[y_pred[i]]},
Actual={target_names[y_test[i]]}")
```

```
print("\n=== Wrong Predictions ===")
for i in range(len(y_test)):
    if y_test[i] != y_pred[i]:
        wrong_predictions_count += 1
        print(f"Sample {i}: Predicted={target_names[y_pred[i]]},
Actual={target_names[y_test[i]]}")
print(f"\nTotal correct predictions: {correct_predictions_count}")
print(f"Total wrong predictions: {wrong_predictions_count}")
```

Output:





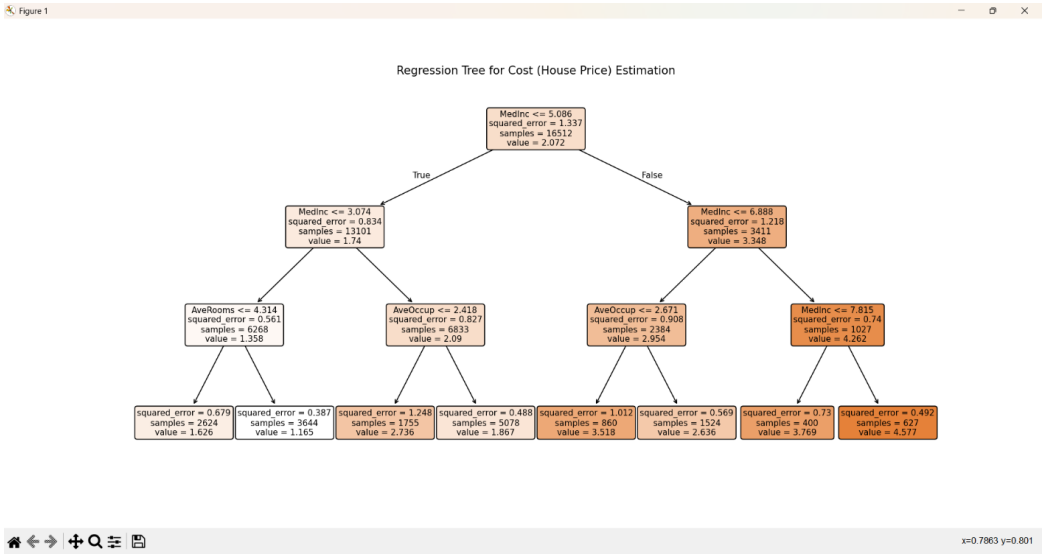
WEEK-11

Aim: Write a program to construct a Regression tree for cost estimation by assuming any numerical dataset.

Program:

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
import matplotlib.pyplot as plt
data = fetch_california_housing()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = DecisionTreeRegressor(max_depth=3, random_state=0)
model.fit(X_train, y_train)
plt.figure(figsize=(15,8))
plot_tree(model, feature_names=data.feature_names, filled=True,
rounded=True)
plt.title("Regression Tree for Cost (House Price) Estimation")
plt.show()
sample = [X_test[0]]
predicted_value = model.predict(sample)
print("Predicted House Value:", predicted_value[0])
```

Output:



WEEK-12

Aim: Write a program to perform Model diagnosis and tuning on any real time dataset using ensemblers
a) RandomForestClassifier
b) BaggingClassifier
c) GradientBoostClassifier

Program:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,
GradientBoostingClassifier
from sklearn.metrics import accuracy_score
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
rf = RandomForestClassifier(random_state=0)
rf_params = {'n_estimators': [50, 100, 150], 'max_depth': [2, 4, 6]}
rf_grid = GridSearchCV(rf, rf_params, cv=3)
rf_grid.fit(X_train, y_train)
rf_best = rf_grid.best_estimator_
rf_pred = rf_best.predict(X_test)
print("RandomForestClassifier Results:")
print("Best Parameters:", rf_grid.best_params_)
print("Accuracy:", accuracy_score(y_test, rf_pred))
print()
bag = BaggingClassifier(random_state=0)
bag_params = {'n_estimators': [10, 20, 30]}
bag_grid = GridSearchCV(bag, bag_params, cv=3)
bag_grid.fit(X_train, y_train)
bag_best = bag_grid.best_estimator_
bag_pred = bag_best.predict(X_test)
print("BaggingClassifier Results:")
print("Best Parameters:", bag_grid.best_params_)
print("Accuracy:", accuracy_score(y_test, bag_pred))
print()
gb = GradientBoostingClassifier(random_state=0)
gb_params = {'n_estimators': [50, 100], 'learning_rate': [0.05, 0.1],
```

```
'max_depth': [2, 3]}
gb_grid = GridSearchCV(gb, gb_params, cv=3)
gb_grid.fit(X_train, y_train)
gb_best = gb_grid.best_estimator_
gb_pred = gb_best.predict(X_test)
print("GradientBoostingClassifier Results:")
print("Best Parameters:", gb_grid.best_params_)
print("Accuracy:", accuracy_score(y_test, gb_pred))
print()
print("Model Performance Summary:")
print(f"Random Forest Accuracy: {accuracy_score(y_test, rf_pred):.2f}")
print(f"Bagging Classifier Accuracy: {accuracy_score(y_test, bag_pred):.2f}")
print(f"Gradient Boosting Accuracy: {accuracy_score(y_test, gb_pred):.2f}")
```

Output: