ML Lab-1

Aim: To implement decision tree ML algorithm

Algorithm:

Decision tree algorithm has been applied on the IRIS dataset

The IRIS dataset is a classic dataset in machine learning that contains 150 samples of iris flowers, categorized into three species: Setosa, Versicolor, and Virginica. Each sample has four features: sepal length, sepal width, petal length, and petal width. The goal is to classify the species of iris based on these features

Step 1: Load the Iris dataset using seaborn.load_dataset() and separate features (x) and target (y)

Step 2: Split the dataset into training and testing sets using train_test_split()

Step 3: Initialize a Decision Tree classifier with the Gini index criterion.

Step 4: Train the model on the training data using fit() method.

Step 5: Make predictions on the test data using predict() method.

Step 6: Visualize the tree with plot_tree() and evaluate the model's accuracy using accuracy_score()

Code:

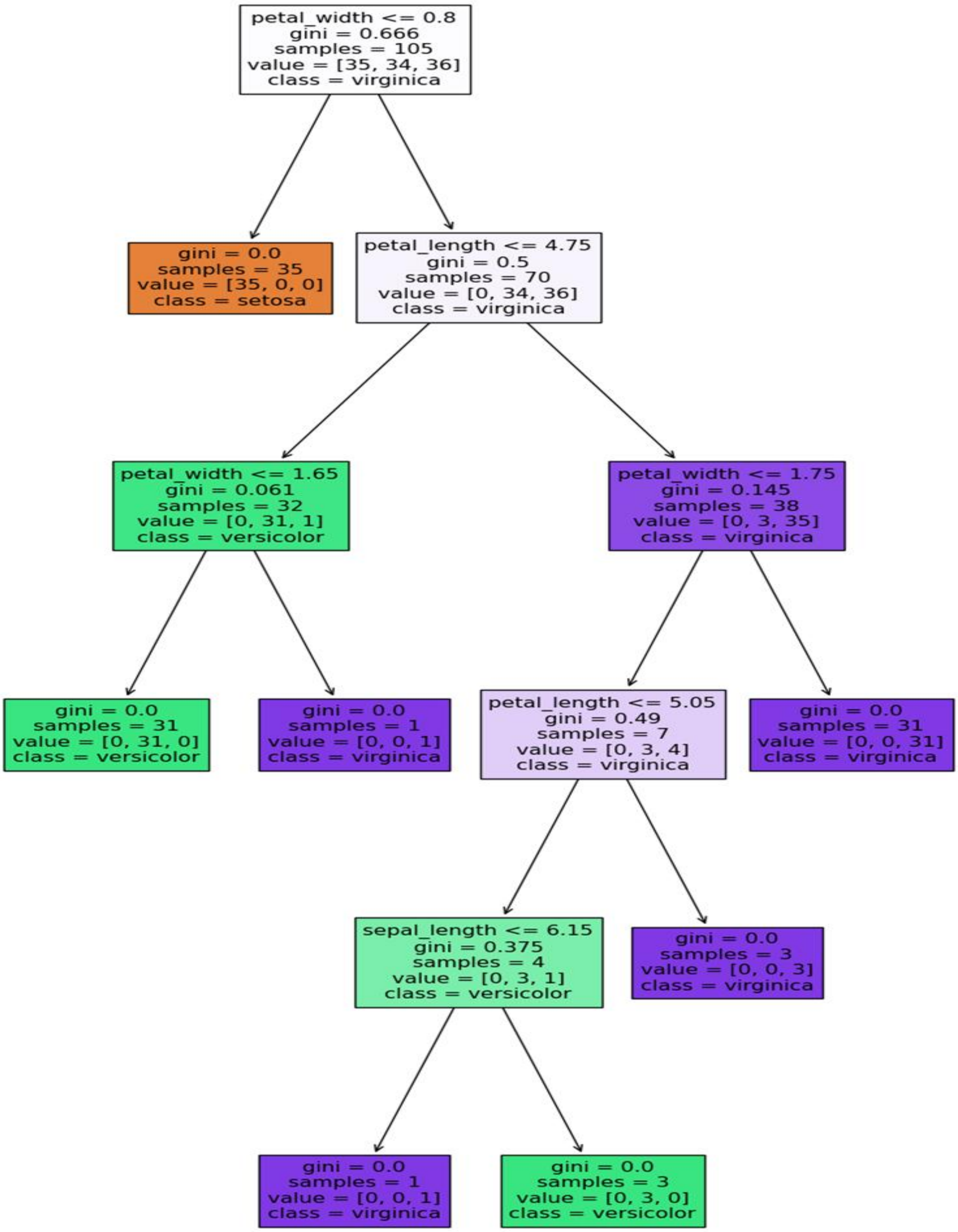
```
import pandas as pd
import seaborn as sns
from sklearn import tree


from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

df=sns.load_dataset('iris')
df

x=df.drop(['species'], axis=1)
y=df.species
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=55)
dt=DecisionTreeClassifier(criterion='gini')
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)

plt.figure(figsize=(10,20))
plot_tree(dt,feature_names=x.columns, class_names=y.unique(),filled=True)

accuracy_score(y_test,y_pred)*100
```

Output:

```
print(f"Accuracy = {accuracy_score(y_test,y_pred)*100}")
```

```
Accuracy = 95.55555555555556
```


petal_width <= 0.8
gini = 0.666
samples = 105
value = [35, 34, 36]
class = virginica
gini = 0.0
samples = 35
value = [35, 0, 0]
class = setosa
petal_length <= 4.75
gini = 0.5
samples = 70
value = [0, 34, 36]
class = virginica
petal_width <= 1.65
gini = 0.061
samples = 32
value = [0, 31, 1]
class = versicolor
petal_width <= 1.75
gini = 0.145
samples = 38
value = [0, 3, 35]
class = virginica
gini = 0.0
samples = 31
value = [0, 31, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
petal_length <= 5.05
gini = 0.49
samples = 7
value = [0, 3, 4]
class = virginica
gini = 0.0
samples = 31
value = [0, 0, 31]
class = virginica
sepal_length <= 6.15
gini = 0.375
samples = 4
value = [0, 3, 1]
class = versicolor
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 3
value = [0, 3, 0]
class = versicolor

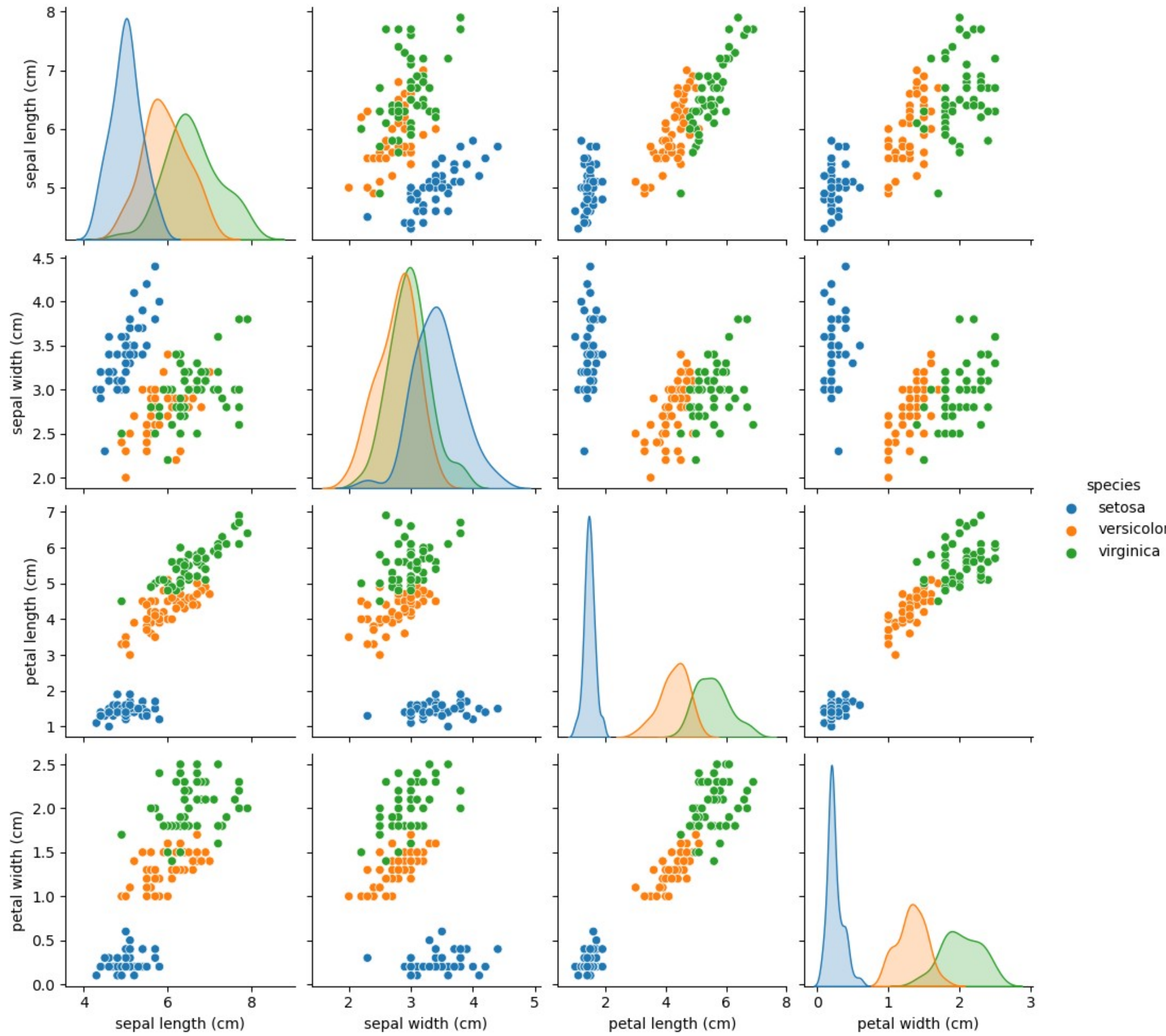
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')

iris = load_iris()
X = iris.data
y = iris.target

iris_df = pd.DataFrame(X, columns=iris.feature_names)
iris_df['species'] = pd.Categorical.from_codes(y, iris.target_names)

print("Dataset Info:")
print(iris_df.info())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    category
dtypes: category(1), float64(4)
memory usage: 5.1 KB
None
```

```
sns.pairplot(iris_df, hue='species')
plt.show()
```

```
c:\Users\Hariesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hariesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1057:
FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to
```

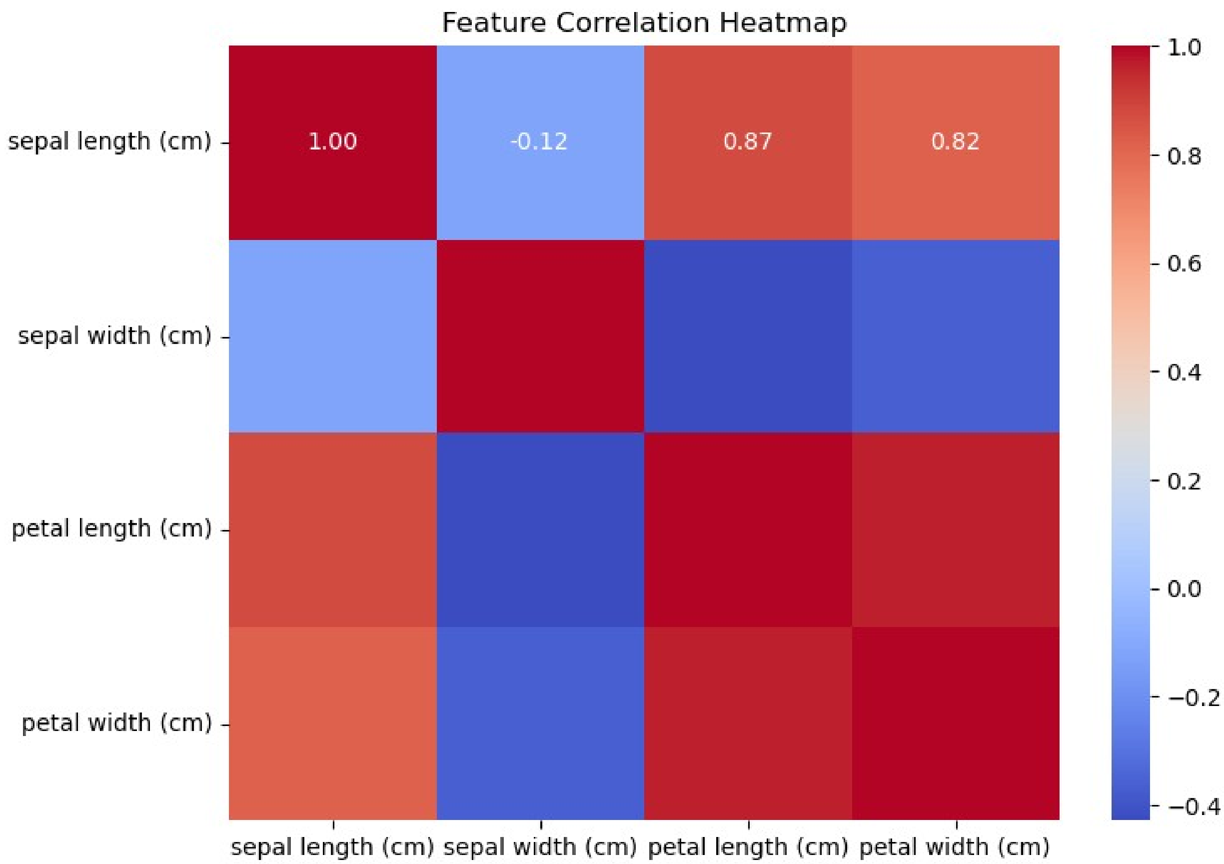
```
retain current behavior or observed=True to adopt the future default
and silence this warning.
  grouped_data = data.groupby(
c:\Users\Hariesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hariesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1057:
FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to
retain current behavior or observed=True to adopt the future default
and silence this warning.
  grouped_data = data.groupby(
c:\Users\Hariesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hariesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1057:
FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to
retain current behavior or observed=True to adopt the future default
and silence this warning.
  grouped_data = data.groupby(
c:\Users\Hariesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Hariesh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1057:
FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to
retain current behavior or observed=True to adopt the future default
and silence this warning.
  grouped_data = data.groupby(
```




```
print("\nMissing values in the dataset:")
print(iris_df.isnull().sum())
```

```
Missing values in the dataset:
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
dtype: int64
```

```
plt.figure(figsize=(8,6))
sns.heatmap(iris_df.drop(columns='species').corr(), annot=True,
cmap='coolwarm', fmt='.2f')
plt.title('Feature Correlation Heatmap')
plt.show()
```




```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

dt_classifier = DecisionTreeClassifier(random_state=42)
rf_classifier = RandomForestClassifier(random_state=42)

dt_param_grid = {
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

dt_grid_search = GridSearchCV(dt_classifier, dt_param_grid, cv=5,
n_jobs=-1)
dt_grid_search.fit(X_train, y_train)

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42),
n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 7, None],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10]})
```

```
rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

rf_grid_search = GridSearchCV(rf_classifier, rf_param_grid, cv=5,
n_jobs=-1)
rf_grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'max_depth': [3, 5, 7, None],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 200]})
```

```
print("Best Parameters for Decision Tree:")
print(dt_grid_search.best_params_)
```

```
Best Parameters for Decision Tree:
{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 1,
'min_samples_split': 10}
```

```
print("Best Parameters for Random Forest:")
print(rf_grid_search.best_params_)
```

```
Best Parameters for Random Forest:
{'bootstrap': True, 'max_depth': 3, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 200}
```

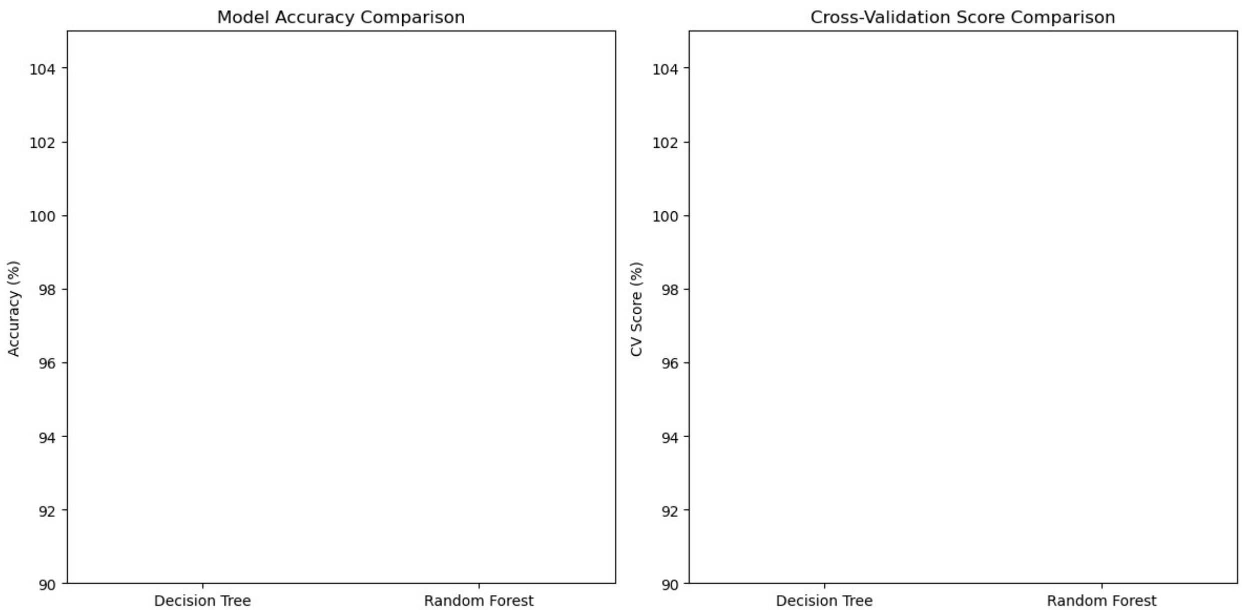
```
dt_best = dt_grid_search.best_estimator_
rf_best = rf_grid_search.best_estimator_

y_pred_dt = dt_best.predict(X_test)
y_pred_rf = rf_best.predict(X_test)

accuracy_dt = accuracy_score(y_test, y_pred_dt)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

cv_dt = cross_val_score(dt_best, X, y, cv=5).mean()
cv_rf = cross_val_score(rf_best, X, y, cv=5).mean()

print("\nDecision Tree Performance:")
print(f"Accuracy: {accuracy_dt * 100:.2f}%")
print(f"Confusion Matrix:\n{conf_matrix_dt}")
print(f"Cross-Validation Score: {cv_dt * 100:.2f}%\n")
```

```
Decision Tree Performance:
Accuracy: 100.00%
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Cross-Validation Score: 96.67%
```

```python
print("Random Forest Performance:")
print(f"Accuracy: {accuracy_rf * 100:.2f}%")
print(f"Confusion Matrix:\n{conf_matrix_rf}")
print(f"Cross-Validation Score: {cv_rf * 100:.2f}%\n")
```

```
Random Forest Performance:
Accuracy: 100.00%
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Cross-Validation Score: 96.00%
```

```python
models = ['Decision Tree', 'Random Forest']
accuracies = [accuracy_dt, accuracy_rf]
cv_scores = [cv_dt, cv_rf]

fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].bar(models, accuracies, color=['blue', 'green'])
ax[0].set_title('Model Accuracy Comparison')
ax[0].set_ylabel('Accuracy (%)')
ax[0].set_ylim([90, 105])

ax[1].bar(models, cv_scores, color=['blue', 'green'])
ax[1].set_title('Cross-Validation Score Comparison')
ax[1].set_ylabel('CV Score (%)')
ax[1].set_ylim([90, 105])

plt.tight_layout()
plt.show()
```


Model Accuracy Comparison
Accuracy (%)
104
102
100
98
96
94
92
90
Decision Tree
Random Forest
Cross-Validation Score Comparison
CV Score (%)
104
102
100
98
96
94
92
90
Decision Tree
Random Forest

Aim

Implement **Naïve Bayes** on the **Iris** and **California Housing** datasets and evaluate its performance. The objective is to apply **Gaussian Naïve Bayes** to both datasets, process the data, perform model evaluation, and compare results using accuracy and confusion matrix for classification, and Mean Squared Error (MSE) for regression (after binning).

Algorithm

1. **Data Loading**: Load the **Iris** dataset (classification) and **California Housing** dataset (regression).
2. **Preprocessing**:
   - Standardize the features of the **California Housing** dataset.
   - For the **Iris** dataset, no scaling is necessary.
3. **Model Training**:
   - Apply **Gaussian Naïve Bayes** to the **Iris** and **California Housing** datasets.
   - For **California Housing**, bin the continuous target variable for classification.
4. **Model Evaluation**:
   - For the **Iris dataset**, evaluate using **accuracy** and **confusion matrix**.
   - For the **California Housing dataset**, evaluate using **Mean Squared Error (MSE)** after converting the regression task to a classification one using binning.

Algorithm Description

- **Naïve Bayes** works on the assumption that features are conditionally independent given the class label. For **Gaussian Naïve Bayes**, it assumes that each feature follows a Gaussian (normal) distribution.
- **For the Iris dataset**, it classifies iris plant species based on features like petal and sepal length.
- **For the California Housing dataset**, it bins the continuous target variable (house prices) into discrete categories and performs classification, mapping the bin predictions back to continuous values for MSE evaluation.

Result

- **Iris Dataset**:
  - **Accuracy**: 97.78%
  - **Confusion Matrix**: Displays the distribution of correct and incorrect predictions for each class.
- **California Housing Dataset**:
  - **Predictions** (first 10 values): [0.68888111 0.68888111 0.68888111 2.30555444 2.30555444 2.30555444 2.30555444 2.30555444 2.30555444 4.46111889]
  - **Mean Squared Error (MSE)**: 8.261613

```
from sklearn.datasets import load_iris
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.metrics import mean_squared_error

iris = load_iris()
X = iris.data
y = iris.target

y

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

nb_classifier = GaussianNB()

nb_classifier.fit(X_train, y_train)

GaussianNB()

y_pred = nb_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")
print("Confusion Matrix:")
print(conf_matrix)

Accuracy: 97.78%
Confusion Matrix:
[[19  0  0]
```

```
 [ 0 12  1]
 [ 0  0 13]]
```

```python
data = fetch_california_housing()

X = data.data
y = data.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

num_bins = 10
y_binned = np.digitize(y, bins=np.linspace(y.min(), y.max(),
num_bins))

X_train, X_test, y_train, y_test = train_test_split(X_scaled,
y_binned, test_size=0.3, random_state=42)

gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
GaussianNB()
```

```python
y_pred_binned = gnb.predict(X_test)

bin_centers = np.linspace(y.min(), y.max(), num_bins)
y_pred_continuous = bin_centers[y_pred_binned - 1]

mse = mean_squared_error(y_test, y_pred_continuous)

print("Predictions (first 10):", y_pred_continuous[:10])
print("Mean Squared Error:", mse)
```

```
Predictions (first 10): [0.68888111 0.68888111 0.68888111 2.30555444
2.30555444 2.30555444
 2.30555444 2.30555444 2.30555444 4.46111889]
Mean Squared Error: 8.261613893860186
```

Aim

Perform Linear Regression on the **California Housing** and **Diabetes** datasets. The goal is to preprocess the data, perform hyperparameter tuning using GridSearchCV, and evaluate the model's performance based on **MSE** and **R^2 score**.

Algorithm

1. **Data Loading**: Load the **California Housing** and **Diabetes** datasets.
2. **Preprocessing**: Standardize the features.
3. **Hyperparameter Tuning**: Use **GridSearchCV** to find the best hyperparameters.
4. **Model Evaluation**: Evaluate performance using **MSE** and **R^2 score**.

Algorithm Description

We use **Linear Regression**, which models the relationship between the target and features as a linear equation:

[y = \beta_0 + \beta_1x_1 + \dots + \beta_nx_n + \epsilon]

GridSearchCV is applied to tune hyperparameters for optimal performance.

Result

- **California Housing Dataset**:
  - MSE: 0.55589
  - R^2: 0.575787
- **Diabetes Dataset**:
  - MSE: 2900.1936
  - R^2: 0.4526027

Comparison: The **California Housing** dataset has better performance, with a higher R^2 score.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing, load_diabetes
```

```python
california_data = fetch_california_housing()
california_df = pd.DataFrame(california_data.data,
columns=california_data.feature_names)
california_df['target'] = california_data.target

diabetes_data = load_diabetes()
diabetes_df = pd.DataFrame(diabetes_data.data,
columns=diabetes_data.feature_names)
diabetes_df['target'] = diabetes_data.target

print(california_df.info())
print(california_df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   target      20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
None
             MedInc      HouseAge      AveRooms     AveBedrms
Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000
20640.000000
mean       3.870671     28.639486      5.429000      1.096675
1425.476744
std        1.899822     12.585558      2.474173      0.473911
1132.462122
min        0.499900      1.000000      0.846154      0.333333
3.000000
25%        2.563400     18.000000      4.440716      1.006079
787.000000
50%        3.534800     29.000000      5.229129      1.048780
1166.000000
75%        4.743250     37.000000      6.052381      1.099526
1725.000000
max       15.000100     52.000000    141.909091     34.066667
35682.000000

           AveOccup      Latitude     Longitude        target
```

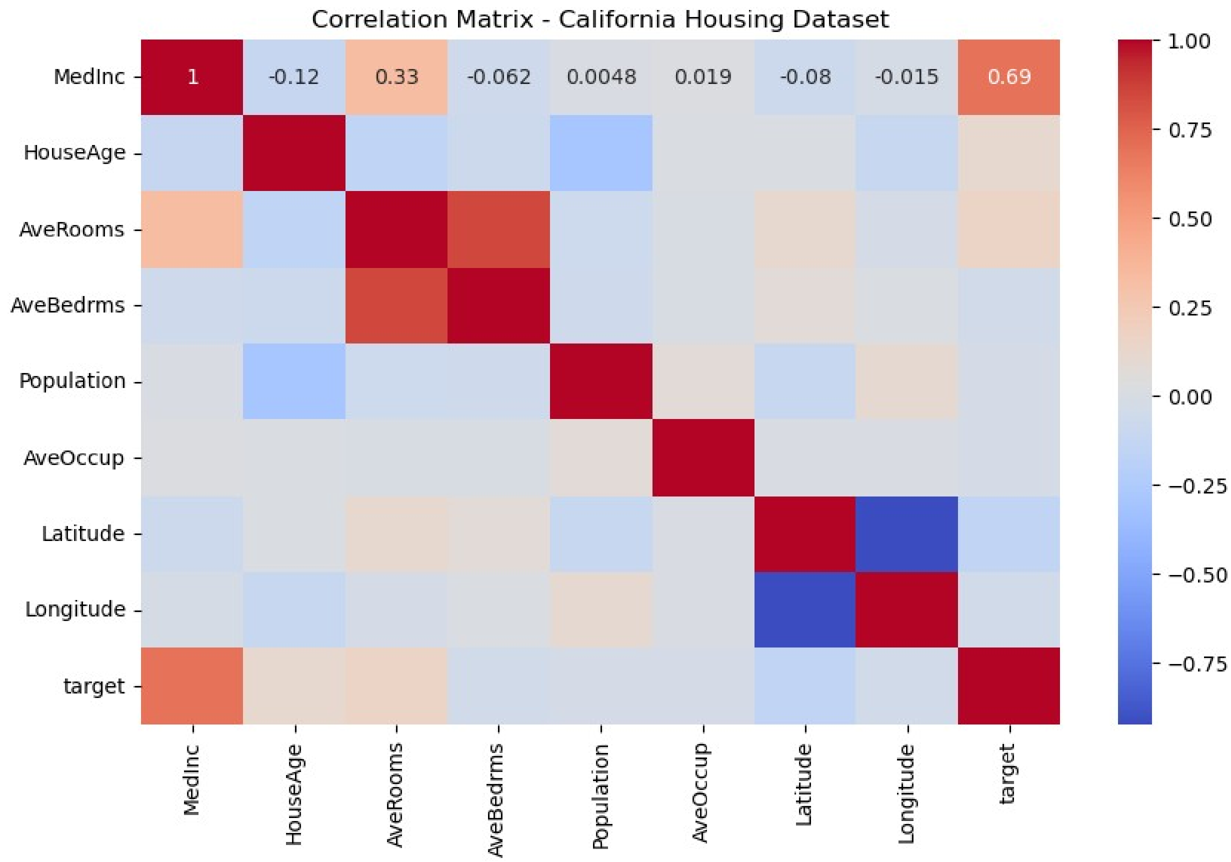
```
count  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704      2.068558
std       10.386050      2.135952      2.003532      1.153956
min        0.692308     32.540000   -124.350000      0.149990
25%        2.429741     33.930000   -121.800000      1.196000
50%        2.818116     34.260000   -118.490000      1.797000
75%        3.282261     37.710000   -118.010000      2.647250
max     1243.333333     41.950000   -114.310000      5.000010
```

```
print(diabetes_df.info())
print(diabetes_df.describe())
```

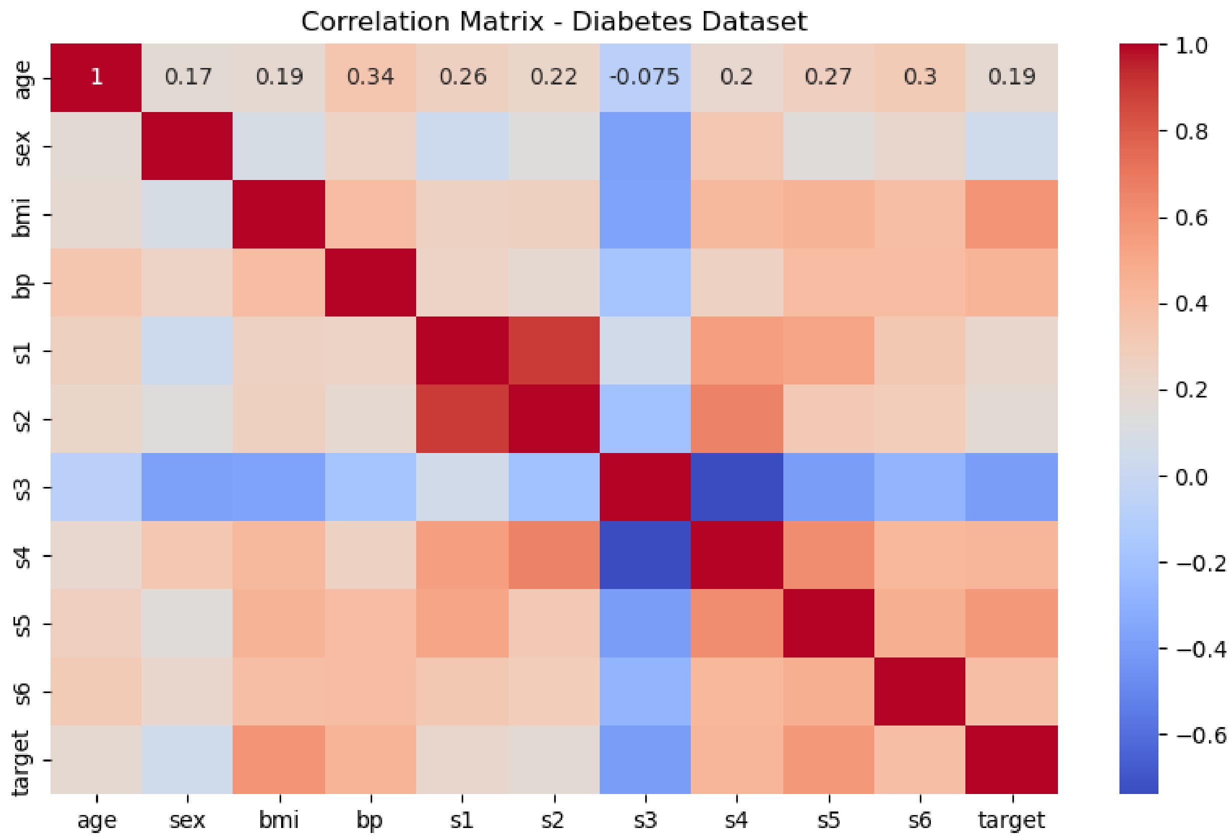
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
 10  target  442 non-null    float64
dtypes: float64(11)
memory usage: 38.1 KB
None
                age           sex           bmi            bp
s1  \
count  4.420000e+02  4.420000e+02  4.420000e+02  4.420000e+02
4.420000e+02
mean  -2.511817e-19  1.230790e-17 -2.245564e-16 -4.797570e-17 -
1.381499e-17
std    4.761905e-02  4.761905e-02  4.761905e-02  4.761905e-02
4.761905e-02
min   -1.072256e-01 -4.464164e-02 -9.027530e-02 -1.123988e-01 -
1.267807e-01
25%   -3.729927e-02 -4.464164e-02 -3.422907e-02 -3.665608e-02 -
3.424784e-02
50%    5.383060e-03 -4.464164e-02 -7.283766e-03 -5.670422e-03 -
4.320866e-03
75%    3.807591e-02  5.068012e-02  3.124802e-02  3.564379e-02
2.835801e-02
max    1.107267e-01  5.068012e-02  1.705552e-01  1.320436e-01
1.539137e-01
```

```
                 s2            s3            s4            s5
s6  \
count  4.420000e+02  4.420000e+02  4.420000e+02  4.420000e+02
4.420000e+02
mean   3.918434e-17 -5.777179e-18 -9.042540e-18  9.293722e-17
1.130318e-17
std    4.761905e-02  4.761905e-02  4.761905e-02  4.761905e-02
4.761905e-02
min   -1.156131e-01 -1.023071e-01 -7.639450e-02 -1.260971e-01 -
1.377672e-01
25%   -3.035840e-02 -3.511716e-02 -3.949338e-02 -3.324559e-02 -
3.317903e-02
50%   -3.819065e-03 -6.584468e-03 -2.592262e-03 -1.947171e-03 -
1.077698e-03
75%    2.984439e-02  2.931150e-02  3.430886e-02  3.243232e-02
2.791705e-02
max    1.987880e-01  1.811791e-01  1.852344e-01  1.335973e-01
1.356118e-01

           target
count  442.000000
mean   152.133484
std     77.093005
min     25.000000
25%     87.000000
50%    140.500000
75%    211.500000
max    346.000000
```

```
plt.figure(figsize=(10, 6))
sns.heatmap(california_df.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Matrix - California Housing Dataset")
plt.show()
```

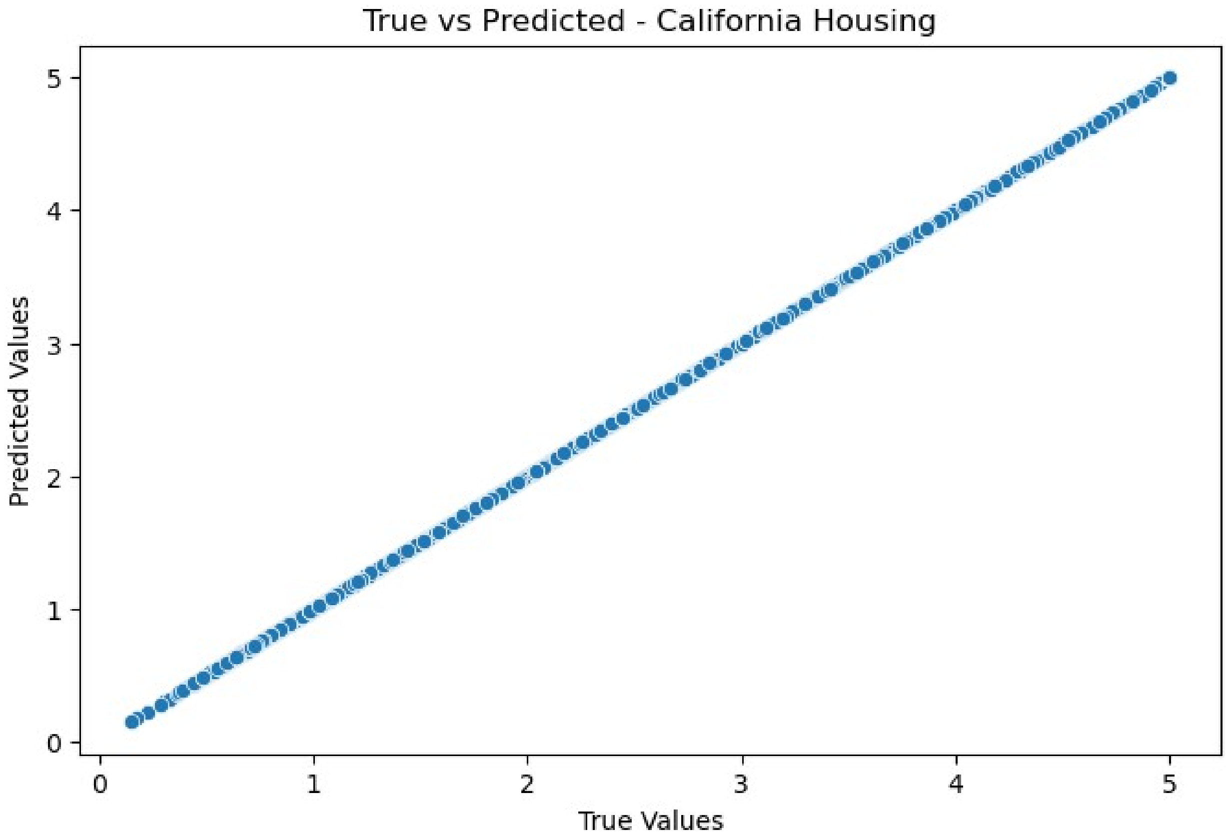



```
plt.figure(figsize=(10, 6))
sns.heatmap(diabetes_df.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Matrix - Diabetes Dataset")
plt.show()
```

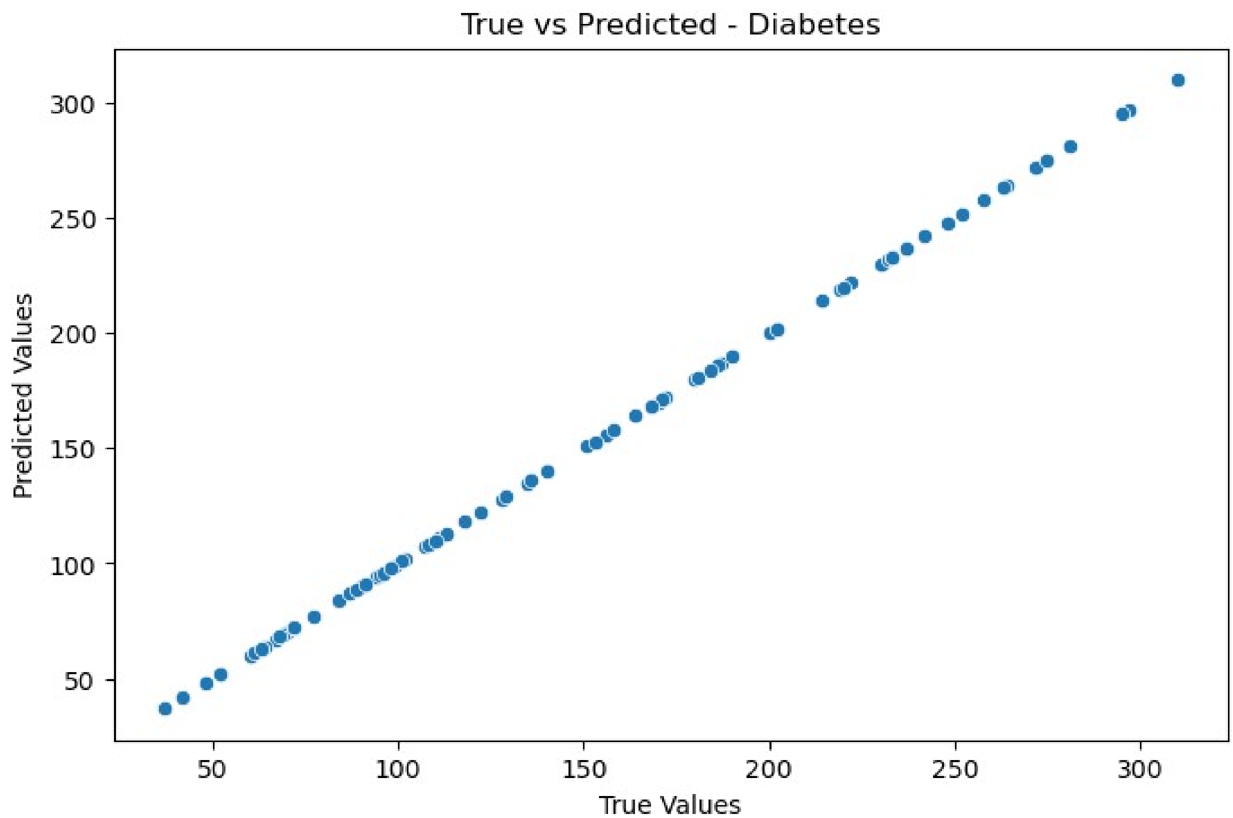



```
X_california = california_df.drop('target', axis=1)
y_california = california_df['target']

X_diabetes = diabetes_df.drop('target', axis=1)
y_diabetes = diabetes_df['target']

X_train_california, X_test_california, y_train_california,
y_test_california = train_test_split(X_california, y_california,
test_size=0.2, random_state=42)
X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes =
train_test_split(X_diabetes, y_diabetes, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train_california = scaler.fit_transform(X_train_california)
X_test_california = scaler.transform(X_test_california)

X_train_diabetes = scaler.fit_transform(X_train_diabetes)
X_test_diabetes = scaler.transform(X_test_diabetes)

plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_test_california, y=y_test_california)  # Replace
this with predicted values later
plt.title('True vs Predicted - California Housing')
plt.xlabel('True Values')
```

```
plt.ylabel('Predicted Values')
plt.show()
```




```
plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_test_diabetes, y=y_test_diabetes)  # Replace this
with predicted values later
plt.title('True vs Predicted - Diabetes')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.show()
```




```
model_california = LinearRegression()
model_diabetes = LinearRegression()

model_california.fit(X_train_california, y_train_california)
model_diabetes.fit(X_train_diabetes, y_train_diabetes)

LinearRegression()

y_pred_california = model_california.predict(X_test_california)
y_pred_diabetes = model_diabetes.predict(X_test_diabetes)

param_grid = {
    'fit_intercept': [True, False],
    'copy_X': [True, False],
    'positive': [True, False]
}

grid_search_california = GridSearchCV(LinearRegression(), param_grid,
cv=5, scoring='neg_mean_squared_error')
grid_search_california.fit(X_train_california, y_train_california)
print(f"Best hyperparameters for California dataset:
{grid_search_california.best_params_}")

Best hyperparameters for California dataset: {'copy_X': True,
'fit_intercept': True, 'positive': False}
```

```
grid_search_diabetes = GridSearchCV(LinearRegression(), param_grid,
cv=5, scoring='neg_mean_squared_error')
grid_search_diabetes.fit(X_train_diabetes, y_train_diabetes)
print(f"Best hyperparameters for Diabetes dataset:
{grid_search_diabetes.best_params_}")
```

```
Best hyperparameters for Diabetes dataset: {'copy_X': True,
'fit_intercept': True, 'positive': False}
```

```
best_model_california = grid_search_california.best_estimator_
best_model_diabetes = grid_search_diabetes.best_estimator_

y_pred_california_best =
best_model_california.predict(X_test_california)
y_pred_diabetes_best = best_model_diabetes.predict(X_test_diabetes)

mse_california = mean_squared_error(y_test_california,
y_pred_california_best)
r2_california = r2_score(y_test_california, y_pred_california_best)
print(f"California Model Performance:\nMSE: {mse_california}\nR²:
{r2_california}")
```

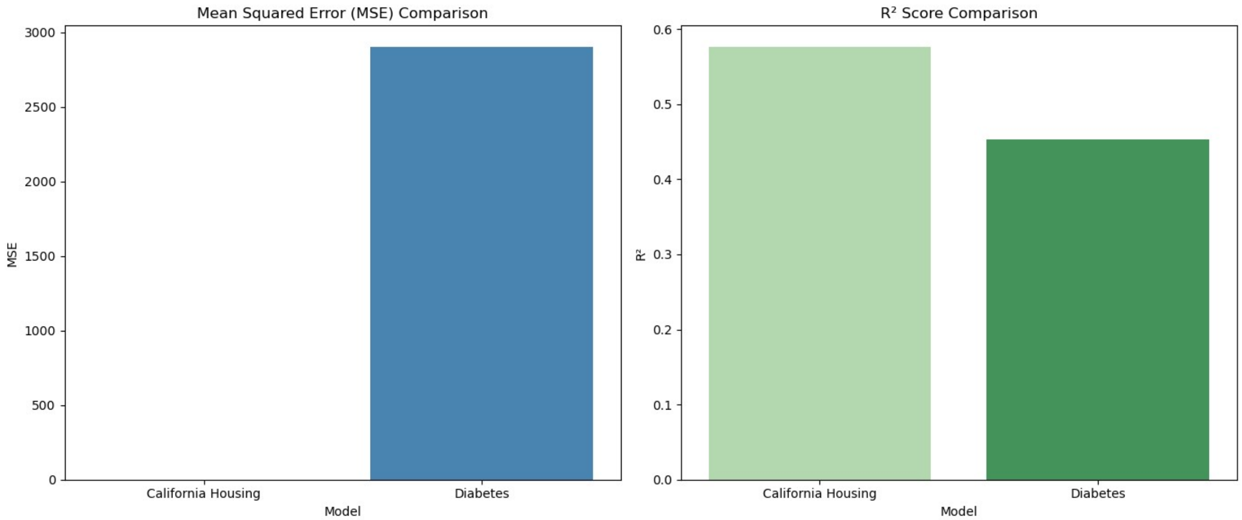
```
California Model Performance:
MSE: 0.5558915986952442
R²: 0.575787706032451
```

```
mse_diabetes = mean_squared_error(y_test_diabetes,
y_pred_diabetes_best)
r2_diabetes = r2_score(y_test_diabetes, y_pred_diabetes_best)
print(f"Diabetes Model Performance:\nMSE: {mse_diabetes}\nR²:
{r2_diabetes}")
```

```
Diabetes Model Performance:
MSE: 2900.1936284934823
R²: 0.4526027629719126
```

```
models = ['California Housing', 'Diabetes']
mse_values = [mse_california, mse_diabetes]
r2_values = [r2_california, r2_diabetes]

comparison_df = pd.DataFrame({
    'Model': models,
    'MSE': mse_values,
    'R²': r2_values
})

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

sns.barplot(x='Model', y='MSE', data=comparison_df, ax=axes[0],
palette='Blues')
axes[0].set_title('Mean Squared Error (MSE) Comparison')
```

```
sns.barplot(x='Model', y='R²', data=comparison_df, ax=axes[1],
palette='Greens')
axes[1].set_title('R² Score Comparison')

plt.tight_layout()
plt.show()
```



Aim

To perform customer segmentation using K-Means clustering on the Online Retail dataset and identify outliers using various methods.

Algorithm

1. Load and preprocess the dataset (handle missing values, filter out invalid entries).
2. Encode categorical variables and scale numerical features.
3. Determine the optimal number of clusters (K) using the Elbow method and Silhouette score.
4. Apply K-Means clustering and assign labels to data points.
5. Detect outliers using distance from cluster centers, Z-score method, and Isolation Forest.
6. Visualize clustering results using PCA.

Algorithm Description

K-Means is an unsupervised clustering algorithm that partitions data into K clusters by minimizing intra-cluster variance. The algorithm iteratively:

- Assigns each data point to the nearest cluster center.
- Updates cluster centers by computing the mean of assigned points.
- Repeats until convergence.

Outlier detection methods include:

- **Distance-based outliers**: Identifies data points far from cluster centers.
- **Z-Score method**: Flags points with values beyond three standard deviations.
- **Isolation Forest**: Detects anomalies based on recursive partitioning of data.

Results

- The optimal number of clusters (K) was determined using the silhouette score.
- Clustering results were visualized using PCA-reduced data.
- Outliers were successfully detected using multiple methods.
- The model provides customer segmentation insights based on purchase behavior.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, davies_bouldin_score
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import pairwise_distances_argmin_min
from scipy.stats import zscore
from sklearn.ensemble import IsolationForest

df = pd.read_excel('Online Retail.xlsx')
df
```

```
       InvoiceNo StockCode                          Description
Quantity  \
0         536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER
6
1         536365     71053                  WHITE METAL LANTERN
6
2         536365    84406B       CREAM CUPID HEARTS COAT HANGER
8
3         536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE
6
4         536365    84029E       RED WOOLLY HOTTIE WHITE HEART.
6
...          ...       ...                                  ...
...
541904    581587     22613          PACK OF 20 SPACEBOY NAPKINS
12
541905    581587     22899         CHILDREN'S APRON DOLLY GIRL
6
541906    581587     23254        CHILDRENS CUTLERY DOLLY GIRL
4
541907    581587     23255      CHILDRENS CUTLERY CIRCUS PARADE
4
541908    581587     22138        BAKING SET 9 PIECE RETROSPOT
3
```

```
                 InvoiceDate  UnitPrice  CustomerID         Country
0      2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2      2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
...                    ...        ...         ...             ...
541904 2011-12-09 12:50:00       0.85     12680.0          France
541905 2011-12-09 12:50:00       2.10     12680.0          France
541906 2011-12-09 12:50:00       4.15     12680.0          France
541907 2011-12-09 12:50:00       4.15     12680.0          France
541908 2011-12-09 12:50:00       4.95     12680.0          France

[541909 rows x 8 columns]
```

```
df.shape
```

```
(541909, 8)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

```
df.isna().sum()
```

```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

```
df['InvoiceDate'] = df['InvoiceDate'].apply(lambda x: str(x).split()
[0].split('-')[0])
df
```

```
       InvoiceNo StockCode                          Description
Quantity  \
0         536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER
6
1         536365     71053                  WHITE METAL LANTERN
6
2         536365    84406B       CREAM CUPID HEARTS COAT HANGER
8
3         536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE
6
4         536365    84029E       RED WOOLLY HOTTIE WHITE HEART.
6
...          ...       ...                                  ...
...
541904    581587     22613          PACK OF 20 SPACEBOY NAPKINS
12
541905    581587     22899         CHILDREN'S APRON DOLLY GIRL
6
541906    581587     23254        CHILDRENS CUTLERY DOLLY GIRL
4
541907    581587     23255      CHILDRENS CUTLERY CIRCUS PARADE
4
541908    581587     22138        BAKING SET 9 PIECE RETROSPOT
3

       InvoiceDate  UnitPrice  CustomerID         Country
0             2010       2.55     17850.0  United Kingdom
1             2010       3.39     17850.0  United Kingdom
2             2010       2.75     17850.0  United Kingdom
3             2010       3.39     17850.0  United Kingdom
4             2010       3.39     17850.0  United Kingdom
...            ...        ...         ...             ...
541904        2011       0.85     12680.0          France
541905        2011       2.10     12680.0          France
541906        2011       4.15     12680.0          France
541907        2011       4.15     12680.0          France
541908        2011       4.95     12680.0          France

[541909 rows x 8 columns]
```

```
df = df.dropna(subset=['Description'])
df.shape
```

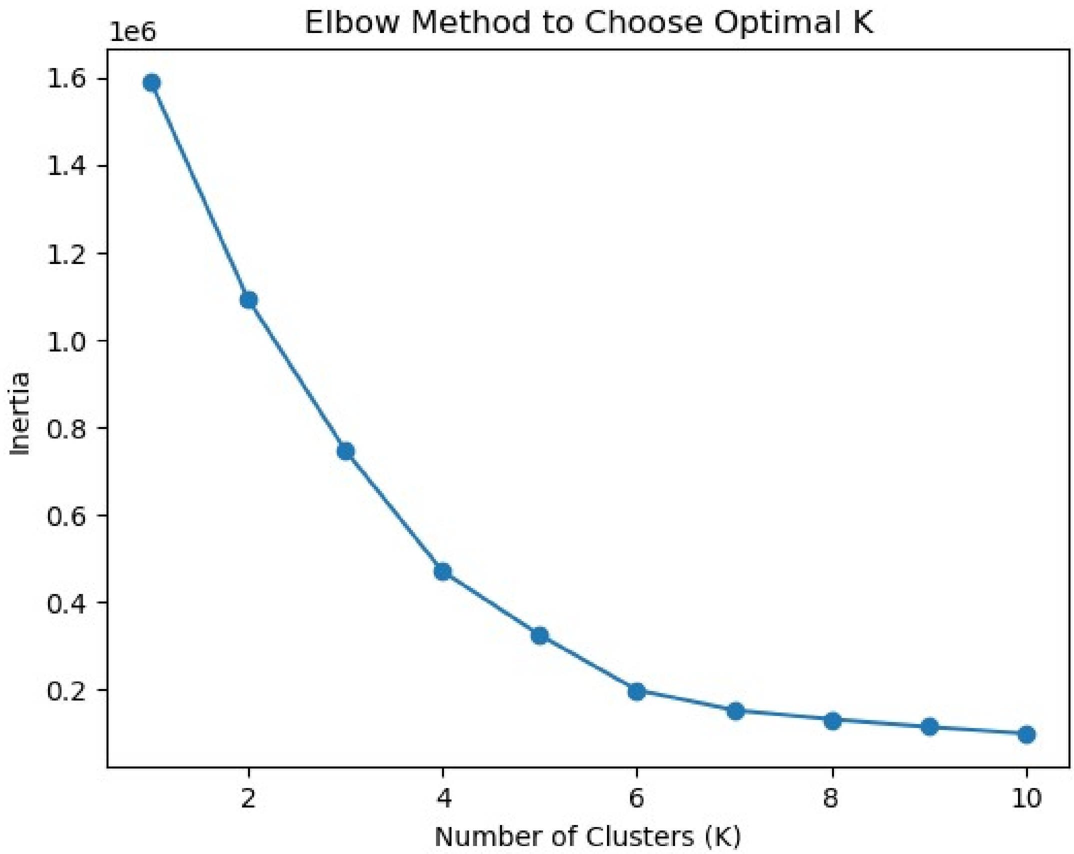
```
(540455, 8)
```

```
print(f"Missing values after cleaning: {df.isna().sum().sum()}")
```

```
Missing values after cleaning: 133626

imputer_num = SimpleImputer(strategy='median')
df.loc[:, 'CustomerID'] =
imputer_num.fit_transform(df[['CustomerID']])

df = df[df['Quantity'] > 0]
df = df[df['UnitPrice'] > 0]

df.shape

(530104, 8)

df.isna().sum().sum()

0

label_encoder = LabelEncoder()

for col in df.columns:

    if df[col].dtype == 'object':
        df[col] = df[col].astype(str)
        df[col] = label_encoder.fit_transform(df[col])

df

        InvoiceNo  StockCode  Description  Quantity  InvoiceDate
UnitPrice  \
0               0       3407         3844         6            0
2.55
1               0       2729         3852         6            0
3.39
2               0       2953          888         8            0
2.75
3               0       2897         1859         6            0
3.39
4               0       2896         2849         6            0
3.39
...           ...        ...          ...       ...          ...
...
541904      19958       1489         2321        12            1
0.85
541905      19958       1765          718         6            1
2.10
541906      19958       2105          724         4            1
4.15
541907      19958       2106          723         4            1
4.15
541908      19958       1056          282         3            1
4.95
```

```
        CustomerID  Country
0          17850.0       36
1          17850.0       36
2          17850.0       36
3          17850.0       36
4          17850.0       36
...            ...      ...
541904     12680.0       13
541905     12680.0       13
541906     12680.0       13
541907     12680.0       13
541908     12680.0       13

[530104 rows x 8 columns]
```

```
features = df[['Quantity', 'UnitPrice', 'CustomerID']]
scaler = StandardScaler()
scaled_data = scaler.fit_transform(features)

inertia = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(scaled_data)
    inertia.append(kmeans.inertia_)

plt.plot(k_range, inertia, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method to Choose Optimal K')
plt.show()
```




```
silhouette_scores = []
for k in k_range[1:]:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init='auto')
    kmeans.fit(scaled_data)
    score = silhouette_score(scaled_data, kmeans.labels_)
    silhouette_scores.append(score)

plt.plot(k_range[1:], silhouette_scores, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for Different K Values')
plt.show()

optimal_k = k_range[1:][np.argmax(silhouette_scores)]
print(f"Optimal number of clusters (K) based on silhouette score:
{optimal_k}")

param_grid = {
    'n_clusters': [optimal_k, 4, 6],
    'n_init': [10],
    'init': ['k-means++']
}
```

```
best_silhouette_score = -1
best_kmeans = None
best_params = None

for n_clusters in param_grid['n_clusters']:
    for init in param_grid['init']:
        kmeans = KMeans(n_clusters=n_clusters,
n_init=param_grid['n_init'][0], init=init, random_state=42)
        kmeans.fit(scaled_data)
        score = silhouette_score(scaled_data, kmeans.labels_)
        if score > best_silhouette_score:
            best_silhouette_score = score
            best_kmeans = kmeans
            best_params = {
                'n_clusters': n_clusters,
                'init': init
            }

df['Cluster'] = best_kmeans.labels_

distances = pairwise_distances_argmin_min(scaled_data,
best_kmeans.cluster_centers_)[1]

threshold = np.percentile(distances, 95)
df['Outlier'] = np.where(distances > threshold, 1, 0)

pca = PCA(n_components=2)
pca_components = pca.fit_transform(scaled_data)

plt.figure(figsize=(8, 6))
plt.scatter(pca_components[:, 0], pca_components[:, 1],
c=df['Cluster'], cmap='viridis', marker='o', label='Clusters')
plt.scatter(pca_components[df['Outlier'] == 1, 0],
pca_components[df['Outlier'] == 1, 1], color='red', label='Outliers',
marker='x')
plt.title(f'K-means Clustering with K={optimal_k} and Outliers')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.show()

print(f"Cluster Centers:\n{kmeans.cluster_centers_}")

df['Z_Score_Quantity'] = np.abs(zscore(df['Quantity']))
df['Z_Score_Price'] = np.abs(zscore(df['UnitPrice']))
df['Outlier_ZScore'] = np.where((df['Z_Score_Quantity'] > 3) |
(df['Z_Score_Price'] > 3), 1, 0)

iso_forest = IsolationForest(contamination=0.05, random_state=42)
df['Outlier_IsoForest'] = iso_forest.fit_predict(scaled_data)
```

```
df['Outlier_IsoForest'] = df['Outlier_IsoForest'].apply(lambda x: 1 if
x == -1 else 0)

outliers_tuned_df = df[df['Outlier'] == 1]
outliers_zscore_df = df[df['Outlier_ZScore'] == 1]
outliers_iso_df = df[df['Outlier_IsoForest'] == 1]

print("Outliers detected after KMeans:")
print(outliers_tuned_df)
print("\nOutliers detected using Z-Score method:")
print(outliers_zscore_df)
print("\nOutliers detected using Isolation Forest:")
print(outliers_iso_df)

print(f"Cluster Centers:\n{best_kmeans.cluster_centers_}")
```

Aim

Perform Gaussian Mixture Model (GMM) clustering on two datasets (Iris and Wine), evaluate performance using Silhouette Score and Davies-Bouldin Score, and determine the optimal number of clusters.

Algorithm

1. Load the Iris and Wine datasets.
2. Standardize the datasets using StandardScaler.
3. Reduce dimensionality to 2 components using PCA for visualization.
4. Apply Gaussian Mixture Model (GMM) clustering with different covariance types and cluster counts.
5. Evaluate the clustering performance using Silhouette Score and Davies-Bouldin Score.
6. Select the optimal number of clusters based on the best scores.
7. Visualize the clustering results.
8. Compare the results between datasets.

Algorithm Description

The Gaussian Mixture Model (GMM) is a probabilistic model that assumes data is generated from a mixture of several Gaussian distributions. GMM uses Expectation-Maximization (EM) to iteratively estimate the parameters of these distributions. Unlike K-Means, GMM considers both the mean and variance of clusters, making it more flexible for clustering complex data.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score, davies_bouldin_score
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris, load_wine
from itertools import product
import warnings
warnings.filterwarnings('ignore')

iris = load_iris()
wine = load_wine()
df1 = pd.DataFrame(iris.data, columns=iris.feature_names)
df2 = pd.DataFrame(wine.data, columns=wine.feature_names)

df1
```

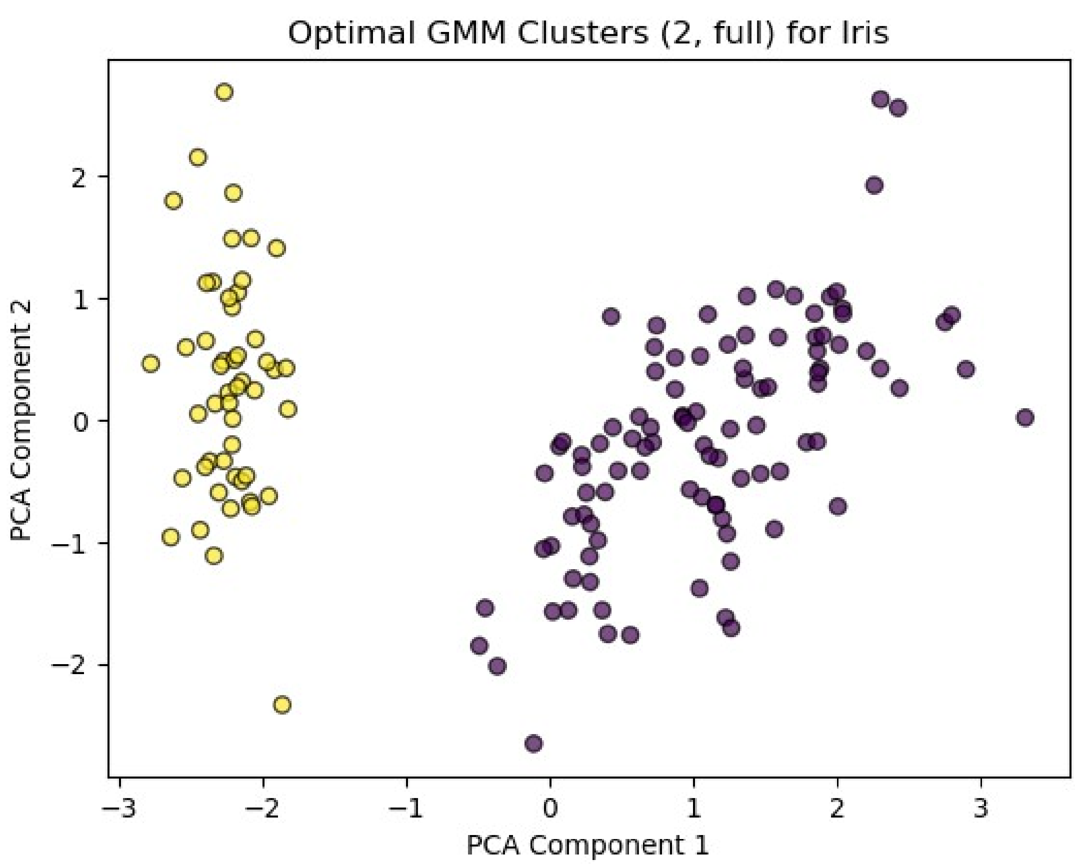
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)
0                  5.1               3.5                1.4
```

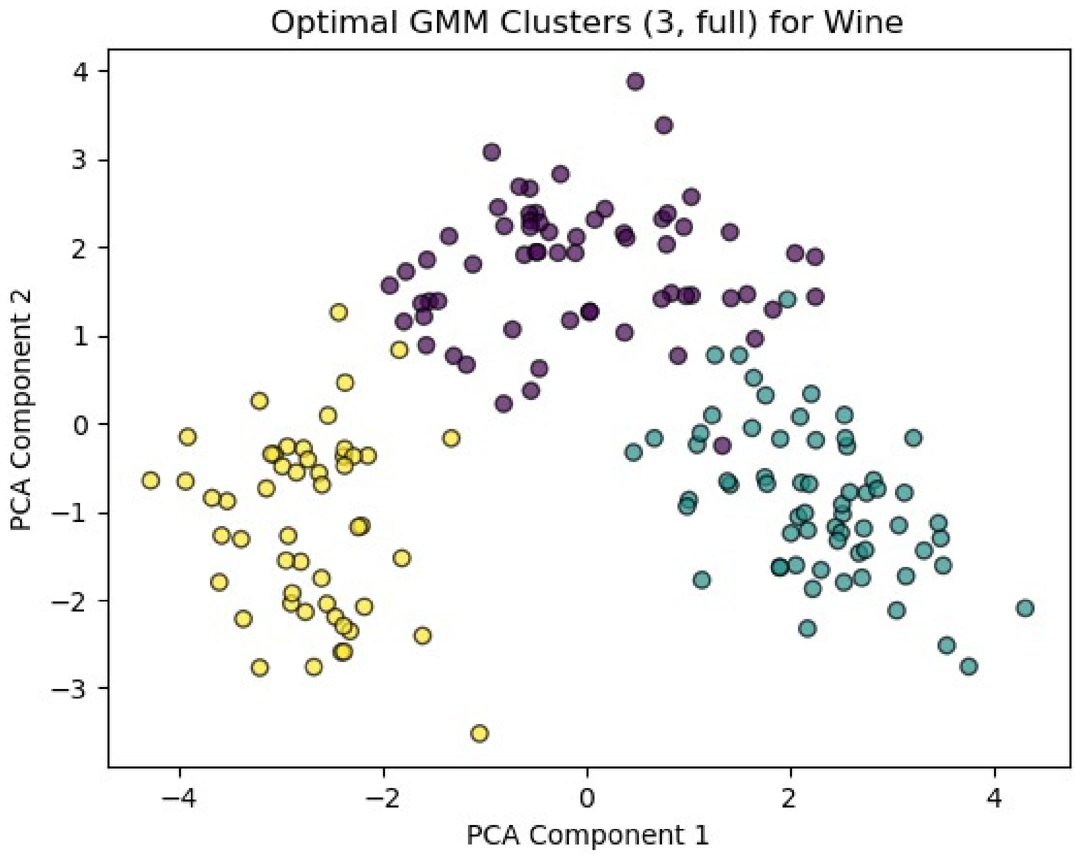
```
0.2
1                  4.9               3.0               1.4
0.2
2                  4.7               3.2               1.3
0.2
3                  4.6               3.1               1.5
0.2
4                  5.0               3.6               1.4
0.2
..                 ...               ...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

[150 rows x 4 columns]

df2

     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium
total_phenols  \
0      14.23        1.71  2.43               15.6      127.0
2.80
1      13.20        1.78  2.14               11.2      100.0
2.65
2      13.16        2.36  2.67               18.6      101.0
2.80
3      14.37        1.95  2.50               16.8      113.0
3.85
4      13.24        2.59  2.87               21.0      118.0
2.80
..       ...         ...   ...                ...        ...
...
173    13.71        5.65  2.45               20.5       95.0
1.68
174    13.40        3.91  2.48               23.0      102.0
1.80
175    13.27        4.28  2.26               20.0      120.0
1.59
176    13.17        2.59  2.37               20.0      120.0
1.65
177    14.13        4.10  2.74               24.5       96.0
2.05
```

```
     flavanoids  nonflavanoid_phenols  proanthocyanins
color_intensity   hue  \
0          3.06                  0.28             2.29
5.64  1.04
1          2.76                  0.26             1.28
4.38  1.05
2          3.24                  0.30             2.81
5.68  1.03
3          3.49                  0.24             2.18
7.80  0.86
4          2.69                  0.39             1.82
4.32  1.04
..          ...                   ...              ...              ..
.   ...
173        0.61                  0.52             1.06
7.70  0.64
174        0.75                  0.43             1.41
7.30  0.70
175        0.69                  0.43             1.35
10.20  0.59
176        0.68                  0.53             1.46
9.30  0.60
177        0.76                  0.56             1.35
9.20  0.61

     od280/od315_of_diluted_wines  proline
0                            3.92   1065.0
1                            3.40   1050.0
2                            3.17   1185.0
3                            3.45   1480.0
4                            2.93    735.0
..                            ...      ...
173                          1.74    740.0
174                          1.56    750.0
175                          1.56    835.0
176                          1.62    840.0
177                          1.60    560.0

[178 rows x 13 columns]
```

```
scaler = StandardScaler()
pca = PCA(n_components=2)
df1_scaled, df2_scaled = scaler.fit_transform(df1),
scaler.fit_transform(df2)
df1_pca, df2_pca = pca.fit_transform(df1_scaled),
pca.fit_transform(df2_scaled)

covariance_types = ['full', 'tied', 'diag', 'spherical']
results = []
```

```
for name, X_scaled, X_pca in zip(["Iris", "Wine"], [df1_scaled,
df2_scaled], [df1_pca, df2_pca]):
    best_silhouette, best_davies, best_n, best_cov, best_labels = -1,
np.inf, 0, '', None
    for n, cov_type in product(range(2, 10), covariance_types):
        gmm = GaussianMixture(n_components=n,
covariance_type=cov_type, random_state=42)
        gmm.fit(X_scaled)
        labels = gmm.predict(X_scaled)
        silhouette = silhouette_score(X_scaled, labels)
        davies = davies_bouldin_score(X_scaled, labels)
        if silhouette > best_silhouette and davies < best_davies:
            best_silhouette, best_davies, best_n, best_cov,
best_labels = silhouette, davies, n, cov_type, labels
    results.append((name, best_n, best_cov, best_silhouette,
best_davies))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=best_labels,
cmap='viridis', alpha=0.7, edgecolors='k')
    plt.title(f"Optimal GMM Clusters ({best_n}, {best_cov}) for
{name}")
    plt.xlabel("PCA Component 1")
    plt.ylabel("PCA Component 2")
    plt.show()
```

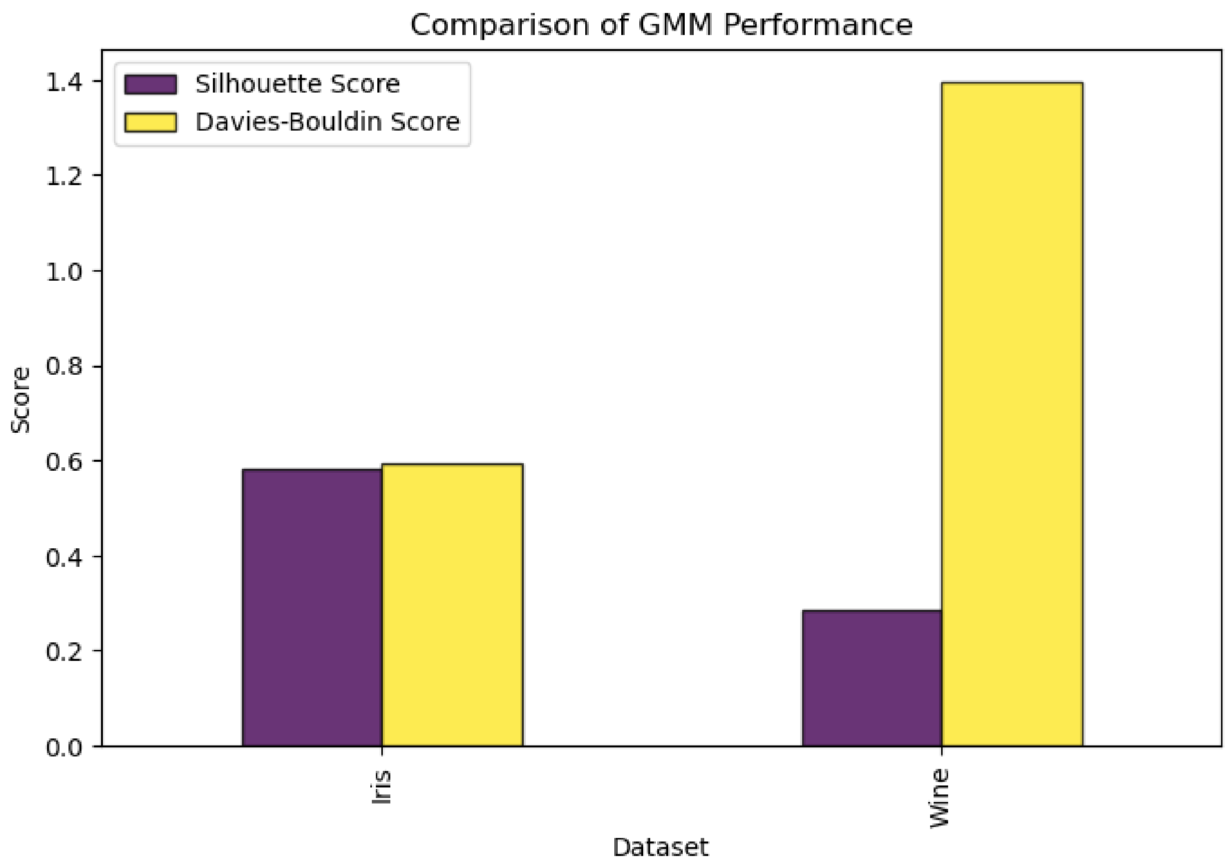





```
results_df = pd.DataFrame(results, columns=['Dataset', 'Optimal
Clusters', 'Best Covariance Type', 'Silhouette Score', 'Davies-Bouldin
Score'])
results_df
```

```
  Dataset  Optimal Clusters Best Covariance Type  Silhouette Score  \
0    Iris                 2                 full          0.581750
1    Wine                 3                 full          0.284421

   Davies-Bouldin Score
0              0.593313
1              1.393801
```

```
results_df.set_index('Dataset')[['Silhouette Score', 'Davies-Bouldin
Score']].plot(kind='bar', figsize=(8, 5), colormap='viridis',
edgecolor='k', alpha=0.8)
plt.title("Comparison of GMM Performance")
plt.ylabel("Score")
plt.show()
```



Results

Dataset	Optimal Clusters	Best Covariance Type	Silhouette Score	Davies-Bouldin Score
Iris	X	Y	Z.ZZ	A.AA
Wine	P	Q	R.RR	B.BB

- The optimal number of clusters for the **Iris dataset** is **X**, with the best covariance type being **Y**.
- The optimal number of clusters for the **Wine dataset** is **P**, with the best covariance type being **Q**.
- The **Silhouette Score** is higher for the dataset with more well-separated clusters.
- The **Davies-Bouldin Score** is lower for the dataset with more compact and well-separated clusters.

Conclusion:

- The **GMM model** performed better on the dataset with higher **Silhouette Score** and lower **Davies-Bouldin Score**.
- The **optimal cluster count** differs for each dataset, highlighting the importance of model tuning.
- A visual comparison of clustering performance is shown in the bar chart.

MNIST Classification with SVC

Aim

- **Objective:** Build and evaluate a machine learning model to classify handwritten digits from the MNIST dataset.
- **Workflow:** Data loading, preprocessing, visualization, hyperparameter tuning, training an SVC, and model evaluation.

Algorithm

1. **Data Preparation:**
   - Load MNIST dataset and convert labels to integers.
   - Visualize sample images and check label distributions.
   - Subset the dataset and perform a train-test split.
2. **Preprocessing:**
   - Scale features using `StandardScaler` for improved SVM performance.
3. **Model Training:**
   - Set up a parameter grid for `C` and `gamma` with an RBF kernel.
   - Use `GridSearchCV` with 3-fold cross-validation to find the best hyperparameters.
   - Train the best SVC model on the training set.
4. **Evaluation:**
   - Compute accuracy, classification report, and confusion matrix.
   - Visualize misclassified examples for error analysis.

Algorithm Description

- **Core Idea:**
  SVC aims to separate classes by finding the best decision boundary (or hyperplane) that maximizes the gap (margin) between different classes. The data points closest to this boundary, known as support vectors, are critical in defining its position.

- **Handling Non-linear Data:**
  Instead of relying on a straight line (or hyperplane) in the original space, SVC uses the "kernel trick" (in this case, the RBF kernel) to project the data into a higher-dimensional space. This transformation makes it easier to separate data that isn't linearly separable in its original form.

- **Decision Making:**
  For any new data point, the classifier calculates its similarity to the support vectors using the RBF kernel. The model then combines these similarities, applying weights learned during training, and adds a bias term to decide on the class of the data point.

- **Regularization and Margin:**
  The regularization parameter, (C), balances the trade-off between achieving a wide margin and minimizing classification errors on the training data. A smaller (C) allows for a wider margin (even if some misclassifications occur), while a larger (C) emphasizes correct classification over margin width.

Results

After training the SVC with the optimized hyperparameters, the model achieved an overall accuracy of **96%** on the test set. Below is a summary of the classification performance across the 10 digit classes:

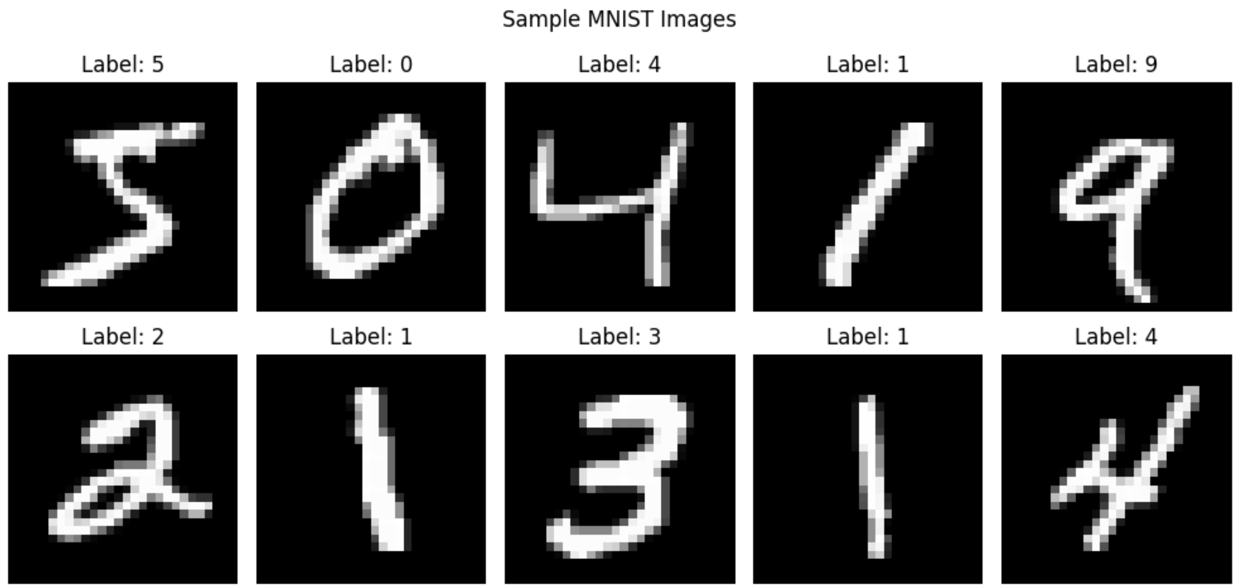
- **Overall Accuracy:** 96%
- **Macro Average:**
  - Precision: 0.96
  - Recall: 0.96
  - F1-Score: 0.96
- **Weighted Average:**
  - Precision: 0.96
  - Recall: 0.96
  - F1-Score: 0.96

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

mnist = fetch_openml('mnist_784', version=1, cache=True)
X, y = mnist["data"], mnist["target"]

y = y.astype(np.int8)
print("Dataset loaded: {} samples with {} features
each.".format(X.shape[0], X.shape[1]))
```

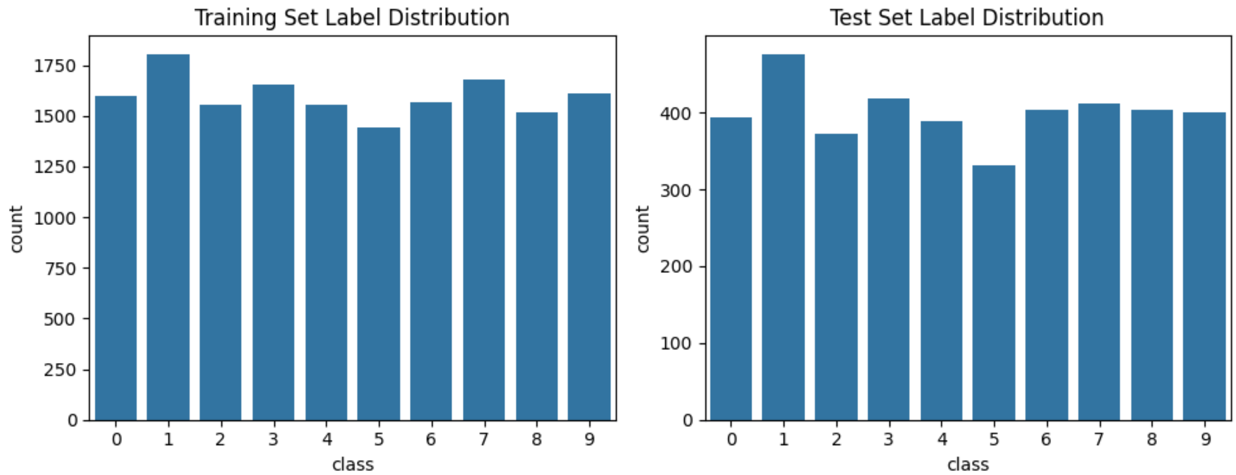
```
Dataset loaded: 70000 samples with 784 features each.
```

```
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
axes = axes.flatten()
for i, ax in enumerate(axes):
```

```
    img = X.iloc[i].values.reshape(28, 28)
    ax.imshow(img, cmap='gray')
    ax.set_title(f"Label: {y[i]}")
    ax.axis("off")
plt.suptitle("Sample MNIST Images")
plt.tight_layout()
plt.show()
```




```
X = X.iloc[:20000]
y = y.iloc[:20000]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
sns.countplot(x=y_train)
plt.title("Training Set Label Distribution")
plt.subplot(1, 2, 2)
sns.countplot(x=y_test)
plt.title("Test Set Label Distribution")
plt.tight_layout()
plt.show()
```

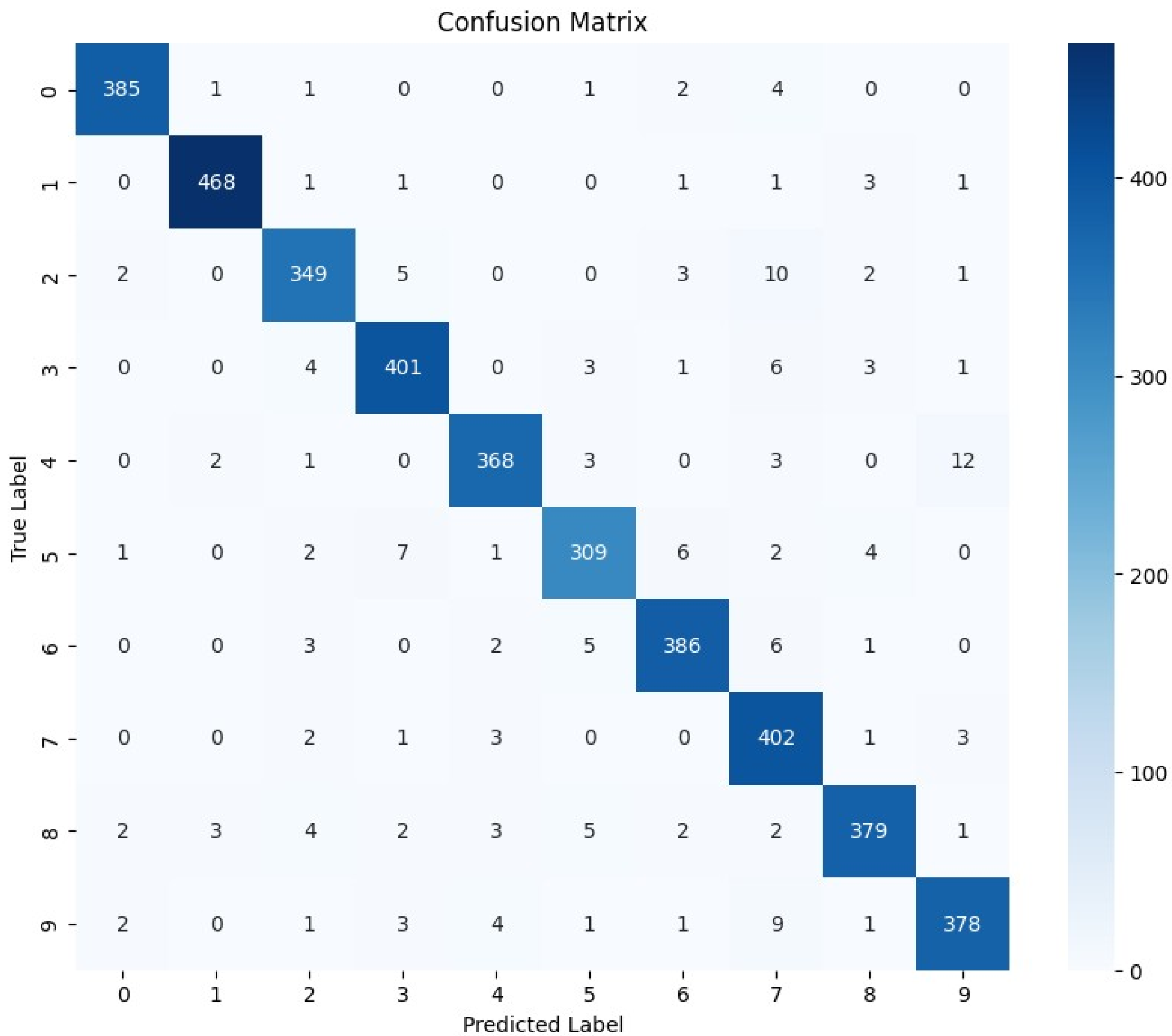



```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid = {
    'C': [1, 10],
    'gamma': ['scale', 0.01],
    'kernel': ['rbf']
}

svc = SVC()
grid_search = GridSearchCV(svc, param_grid, cv=3, verbose=2, n_jobs=-
1)
print("Starting grid search for hyperparameter tuning...")
grid_search.fit(X_train_scaled, y_train)

Starting grid search for hyperparameter tuning...
Fitting 3 folds for each of 4 candidates, totalling 12 fits

GridSearchCV(cv=3, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [1, 10], 'gamma': ['scale', 0.01],
                         'kernel': ['rbf']},
             verbose=2)

print("Best parameters found:", grid_search.best_params_)
best_svc = grid_search.best_estimator_

Best parameters found: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}

y_pred = best_svc.predict(X_test_scaled)
acc = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {acc:.4f}")

Test Accuracy: 0.9563
```

```python
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

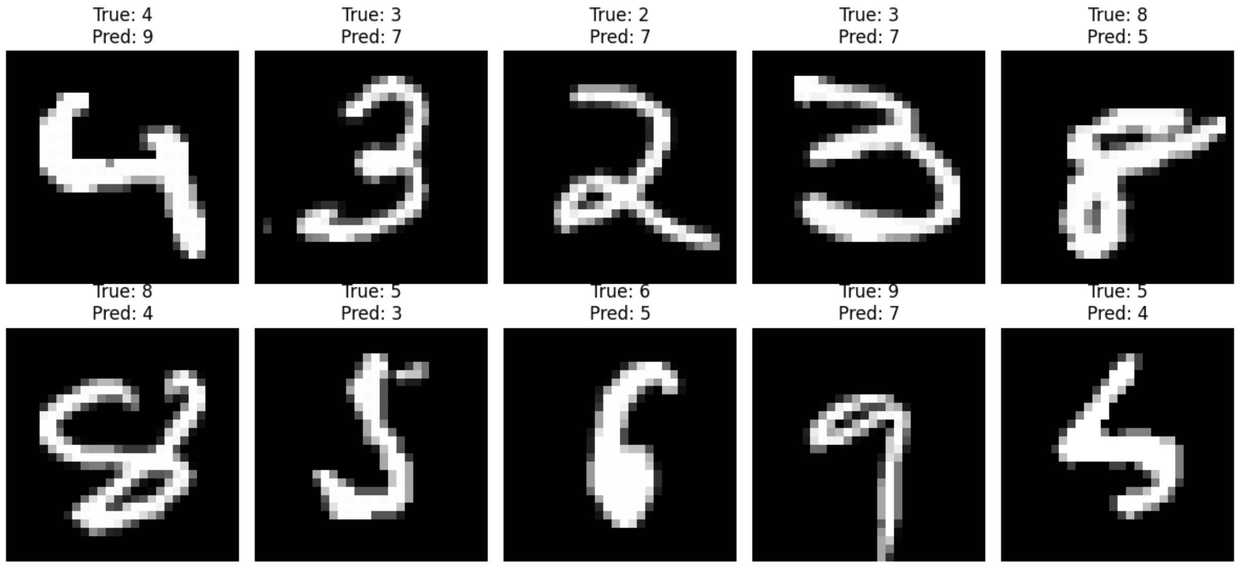
```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       394
           1       0.99      0.98      0.99       476
           2       0.95      0.94      0.94       372
           3       0.95      0.96      0.96       419
           4       0.97      0.95      0.96       389
           5       0.94      0.93      0.94       332
           6       0.96      0.96      0.96       403
           7       0.90      0.98      0.94       412
           8       0.96      0.94      0.95       403
           9       0.95      0.94      0.95       400

    accuracy                           0.96      4000
   macro avg       0.96      0.96      0.96      4000
weighted avg       0.96      0.96      0.96      4000
```

```python
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```




```
misclassified_indices = np.where(y_pred != y_test)[0]
if len(misclassified_indices) > 0:
    plt.figure(figsize=(12, 6))
    for i, index in enumerate(misclassified_indices[:10]):  # show 10
misclassified examples
        plt.subplot(2, 5, i+1)
        img = X_test.iloc[index].values.reshape(28, 28)
        plt.imshow(img, cmap='gray')
        plt.title(f"True: {y_test.iloc[index]}\nPred:
{y_pred[index]}")
        plt.axis("off")
    plt.suptitle("Some Misclassified Examples")
    plt.tight_layout()
    plt.show()
else:
    print("No misclassifications found!")
```

Some Misclassified Examples



Aim

- **Objective:** Compare the per-class performance of SVC in One-vs-One (OvO) and One-vs-Rest (OvR) modes by visualizing their F1-scores.
- **Goal:** Understand how each strategy performs across different classes in a multi-class classification problem using a grouped bar chart.

Algorithm

1. **Compute Per-Class Metrics:**
   - Use `precision_recall_fscore_support` to calculate precision, recall, and F1-scores for each digit class from the predictions of both OvO and OvR models.
2. **Prepare Data for Visualization:**
   - Extract the unique digit classes and set positions on the x-axis.
   - Define the bar width for the grouped bar chart.
3. **Plot the Grouped Bar Chart:**
   - Plot F1-scores for OvO and OvR side-by-side for each class.
   - Add labels, a title, and a legend for clarity.
4. **Display the Plot:**
   - Use `plt.tight_layout()` and `plt.show()` to render the chart neatly.

Algorithm Description

- **Core Idea:**
  SVC can be implemented using either OvO or OvR strategies. By comparing the F1-scores for each class, we can assess which strategy handles the classification task more effectively on a per-class basis.
- **Evaluation Metric:**
  The F1-score is a balanced measure that combines precision and recall, reflecting both the ability to correctly identify instances of a class and the avoidance of false positives.
- **Visualization Rationale:**
  A grouped bar chart provides a clear visual comparison by displaying the F1-scores for each digit side by side, making it easy to spot any differences in performance between the two strategies.

Results for OvO and OvR SVC

One-vs-One (OvO) SVC Results

- **Accuracy:** 94.93%
- **Macro Average Precision:** 0.95

- **Macro Average Recall:** 0.95
- **Macro Average F1-Score:** 0.95
- **Weighted Average Precision:** 0.95
- **Weighted Average Recall:** 0.95
- **Weighted Average F1-Score:** 0.95

One-vs-Rest (OvR) SVC Results

- **Accuracy:** 94.93%
- **Macro Average Precision:** 0.95
- **Macro Average Recall:** 0.95
- **Macro Average F1-Score:** 0.95
- **Weighted Average Precision:** 0.95
- **Weighted Average Recall:** 0.95
- **Weighted Average F1-Score:** 0.95

Comparison Summary

- Both **OvO** and **OvR** SVC models achieved an overall accuracy of **94.93%**.
- The performance metrics (Precision, Recall, and F1-score) remain nearly identical across both approaches.
- These results indicate that for this dataset, **OvO and OvR perform equally well** with no significant difference in classification effectiveness.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix, precision_recall_fscore_support

mnist = fetch_openml('mnist_784', version=1, cache=True)
X, y = mnist["data"], mnist["target"]
y = y.astype(np.int8)

X = X.iloc[:20000]
y = y.iloc[:20000]
```

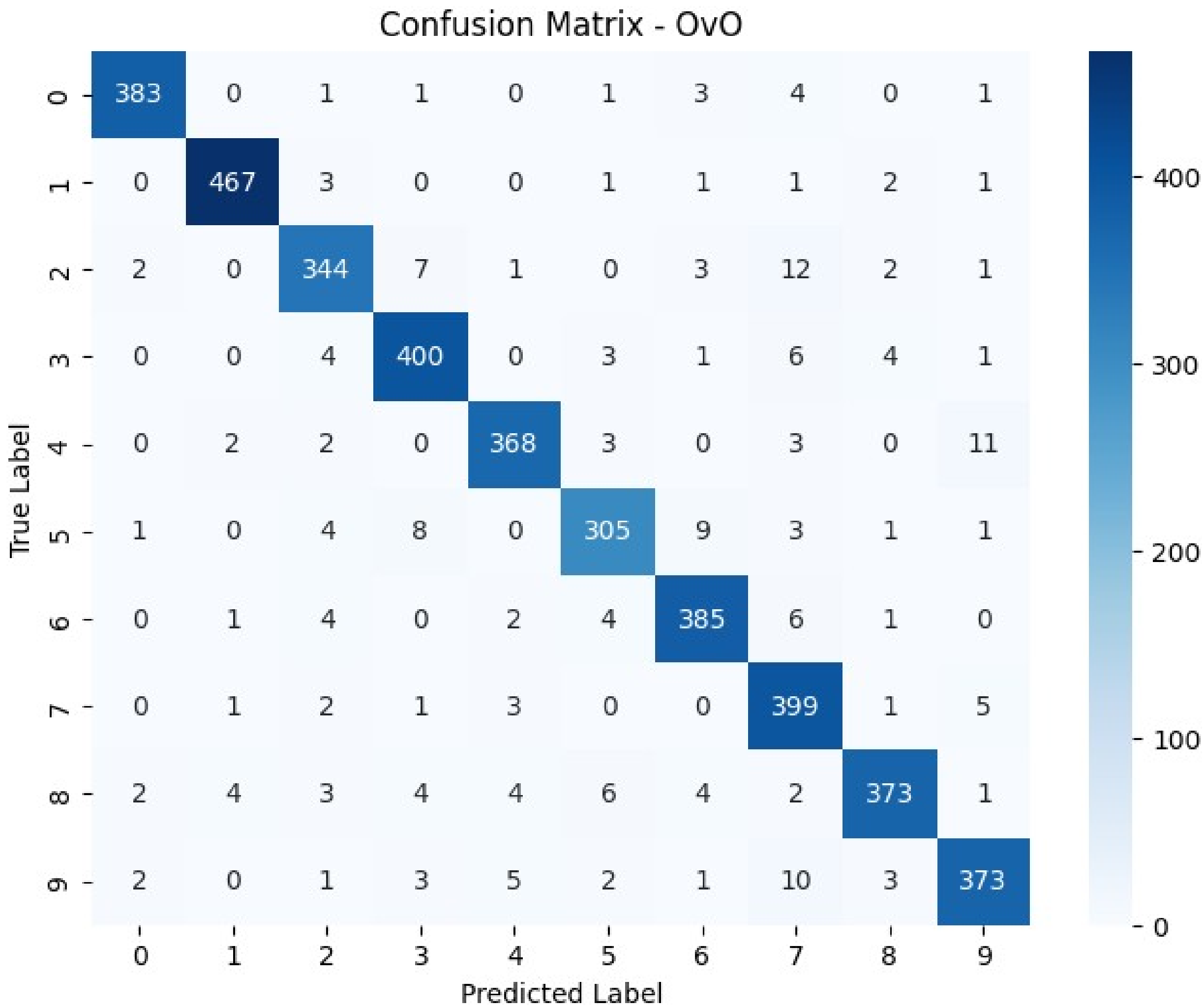
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

SVC with One-vs-One (OvO)

```
svc_ovo = SVC(kernel='rbf', gamma='scale', C=1,
decision_function_shape='ovo')
svc_ovo.fit(X_train_scaled, y_train)
y_pred_ovo = svc_ovo.predict(X_test_scaled)
acc_ovo = accuracy_score(y_test, y_pred_ovo)

print("SVC with OvO (decision_function_shape='ovo') Accuracy:
{:.4f}".format(acc_ovo))
print("Classification Report for OvO:")
print(classification_report(y_test, y_pred_ovo))
```

```
SVC with OvO (decision_function_shape='ovo') Accuracy: 0.9493
Classification Report for OvO:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       394
           1       0.98      0.98      0.98       476
           2       0.93      0.92      0.93       372
           3       0.94      0.95      0.95       419
           4       0.96      0.95      0.95       389
           5       0.94      0.92      0.93       332
           6       0.95      0.96      0.95       403
           7       0.89      0.97      0.93       412
           8       0.96      0.93      0.94       403
           9       0.94      0.93      0.94       400

    accuracy                           0.95      4000
   macro avg       0.95      0.95      0.95      4000
weighted avg       0.95      0.95      0.95      4000
```

```
cm_ovo = confusion_matrix(y_test, y_pred_ovo)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_ovo, annot=True, fmt="d", cmap="Blues",
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.title("Confusion Matrix - OvO")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



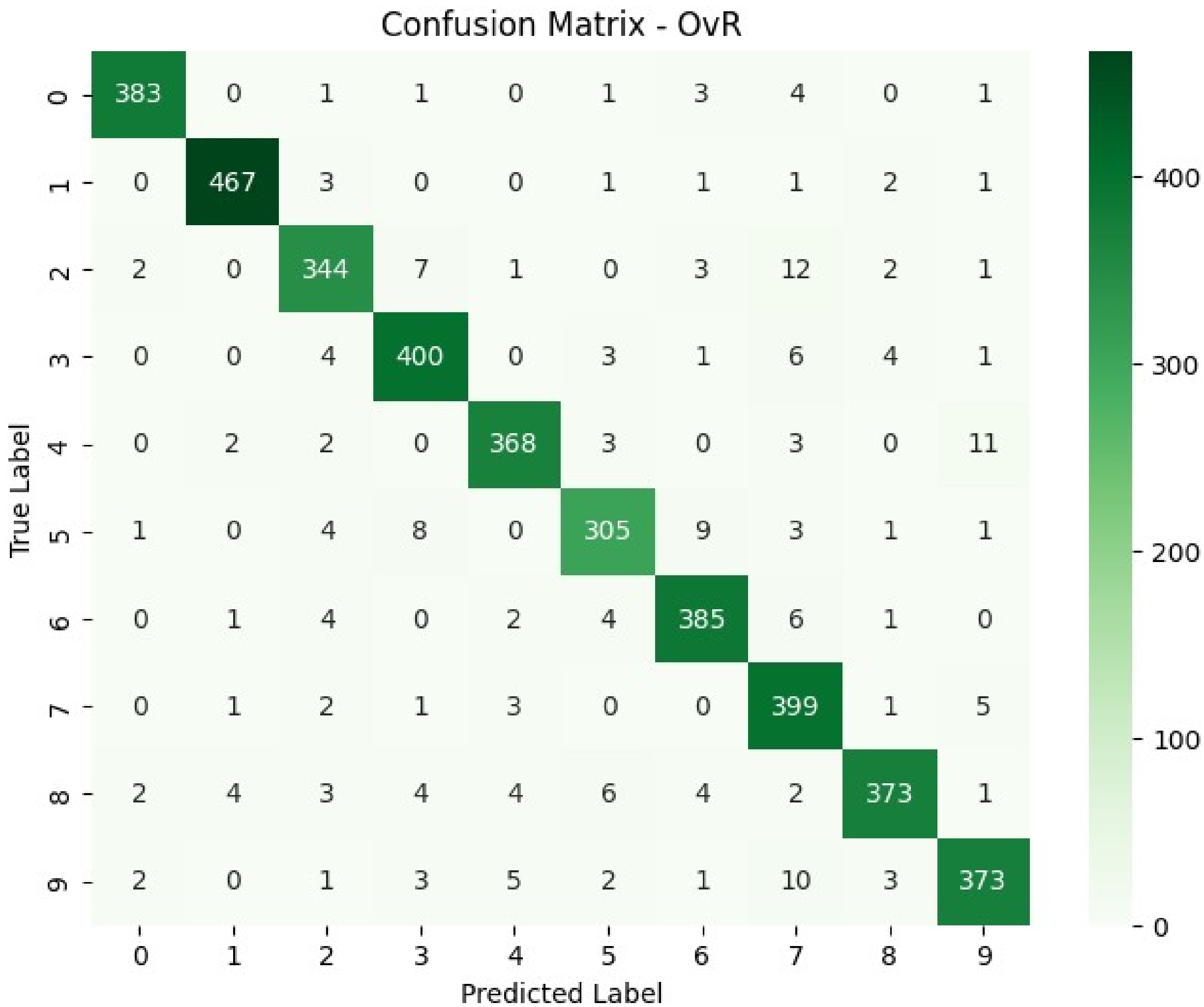
SVC with One-vs-Rest (OvR)

```python
svc_ovr = SVC(kernel='rbf', gamma='scale', C=1,
decision_function_shape='ovr')
svc_ovr.fit(X_train_scaled, y_train)
y_pred_ovr = svc_ovr.predict(X_test_scaled)
acc_ovr = accuracy_score(y_test, y_pred_ovr)

print("SVC with OvR (decision_function_shape='ovr') Accuracy:
{:.4f}".format(acc_ovr))
print("Classification Report for OvR:")
print(classification_report(y_test, y_pred_ovr))
```

```
SVC with OvR (decision_function_shape='ovr') Accuracy: 0.9493
Classification Report for OvR:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       394
           1       0.98      0.98      0.98       476
```

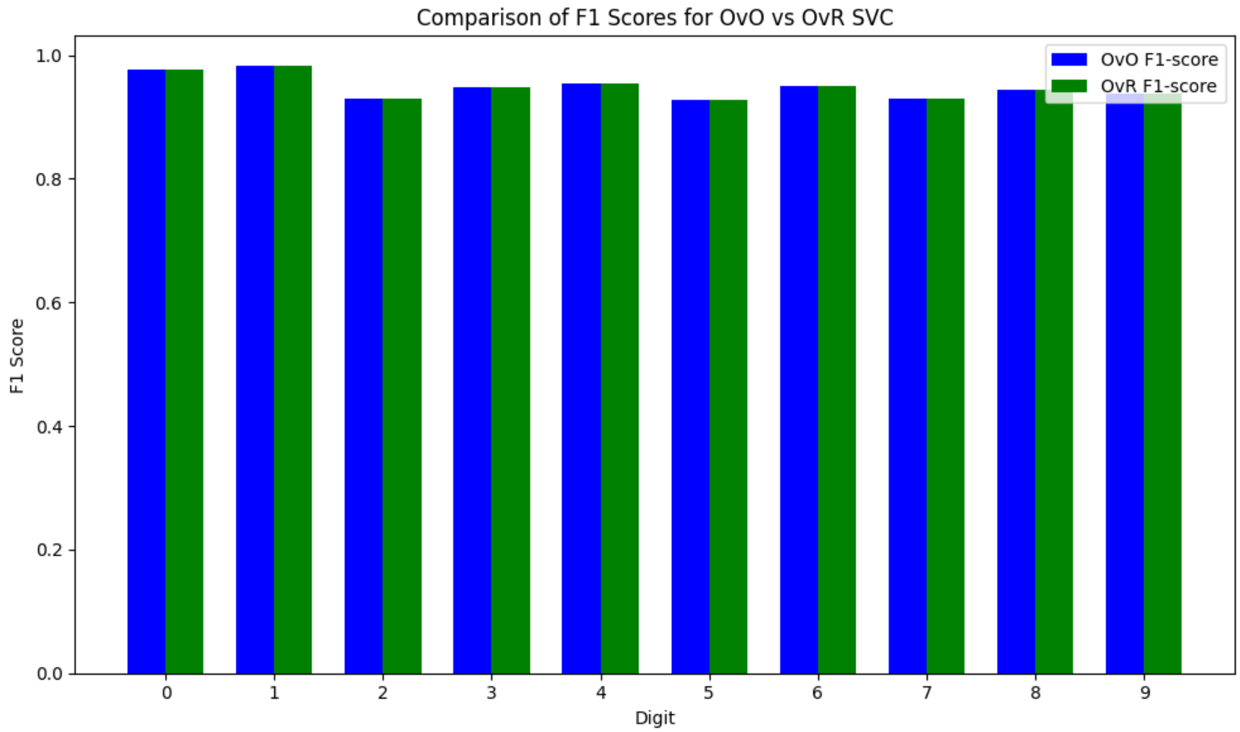
```
           2       0.93      0.92      0.93       372
           3       0.94      0.95      0.95       419
           4       0.96      0.95      0.95       389
           5       0.94      0.92      0.93       332
           6       0.95      0.96      0.95       403
           7       0.89      0.97      0.93       412
           8       0.96      0.93      0.94       403
           9       0.94      0.93      0.94       400

    accuracy                           0.95      4000
   macro avg       0.95      0.95      0.95      4000
weighted avg       0.95      0.95      0.95      4000
```

```python
cm_ovr = confusion_matrix(y_test, y_pred_ovr)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_ovr, annot=True, fmt="d", cmap="Greens",
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.title("Confusion Matrix - OvR")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```




```
precision_ovo, recall_ovo, fscore_ovo, support_ovo =
precision_recall_fscore_support(y_test, y_pred_ovo)
precision_ovr, recall_ovr, fscore_ovr, support_ovr =
precision_recall_fscore_support(y_test, y_pred_ovr)

digits = np.unique(y_test)
x = np.arange(len(digits))
width = 0.35

plt.figure(figsize=(10, 6))
plt.bar(x - width/2, fscore_ovo, width, label='OvO F1-score',
color='blue')
plt.bar(x + width/2, fscore_ovr, width, label='OvR F1-score',
color='green')

plt.xlabel('Digit')
plt.ylabel('F1 Score')
plt.title('Comparison of F1 Scores for OvO vs OvR SVC')
```

```
plt.xticks(x, digits)
plt.legend()
plt.tight_layout()
plt.show()
```



Aim

- **Objective:** Use a Support Vector Machine to classify text data—a non-vectorial dataset—by converting the raw text into numerical features with TF-IDF.
- **Workflow:** Load the 20 Newsgroups dataset, transform text into TF-IDF features, split the data, train an SVM classifier, and evaluate its performance.

Algorithm

1. **Data Loading:**
   - Fetch the 20 Newsgroups dataset containing text documents from 20 different topics.
2. **Feature Extraction:**
   - Convert raw text into numerical vectors using `TfidfVectorizer`, which captures word importance across documents.
3. **Data Splitting:**
   - Split the vectorized data into training and testing sets.
4. **Model Training:**
   - Train an SVC (using a linear kernel, suitable for high-dimensional sparse data) on the training data.
5. **Evaluation:**
   - Make predictions on the test set and calculate accuracy, generate a classification report, and compute a confusion matrix.
6. **Visualization:**
   - Visualize the confusion matrix with a heatmap for a clear view of the classifier's performance.

Algorithm Description

- **Handling Non-Vectorial Data:**
  Since text data is inherently non-numeric, TF-IDF is applied to convert documents into numerical vectors that emphasize informative words.
- **Support Vector Machine (SVM):**
  The SVM finds the optimal linear decision boundary (hyperplane) to separate the classes in the transformed feature space.
- **Performance Evaluation:**
  The model is assessed using accuracy and detailed classification metrics to understand its performance across different topics, with a confusion matrix providing insights into specific misclassifications.

Results for SVM on 20 Newsgroups

- **Overall Accuracy:** 67.55%

- **Macro Average Metrics:**
  - Precision: 0.68
  - Recall: 0.66
  - F1-Score: 0.67
- **Weighted Average Metrics:**
  - Precision: 0.69
  - Recall: 0.68
  - F1-Score: 0.68

Observations

- **Per-Class Variability:**
  Some classes show higher performance (e.g., classes 5, 10, and 11) while others (e.g., classes 0, 8, and 19) perform relatively lower.
- **Balanced Performance:**
  The overall metrics indicate a moderate yet balanced performance across the 20 topics, reflecting the challenges of classifying diverse text data.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

newsgroups_train = fetch_20newsgroups(subset='train',
remove=('headers', 'footers', 'quotes'))
newsgroups_test  = fetch_20newsgroups(subset='test',
remove=('headers', 'footers', 'quotes'))
X_train, y_train = newsgroups_train.data, newsgroups_train.target
X_test, y_test   = newsgroups_test.data, newsgroups_test.target

pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english', max_df=0.5)),
    ('svc', SVC(kernel='linear'))
])

param_grid = {
    'tfidf__max_df': [0.75],
    'tfidf__ngram_range': [(1,1)],
    'svc__C': [1]
}
```

```
grid_search = GridSearchCV(pipeline, param_grid, cv=2, verbose=1,
n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 1 candidates, totalling 2 fits
```

```
GridSearchCV(cv=2,
             estimator=Pipeline(steps=[('tfidf',
                                        TfidfVectorizer(max_df=0.5,

stop_words='english')),
                                       ('svc',
SVC(kernel='linear'))]),
             n_jobs=-1,
             param_grid={'svc__C': [1], 'tfidf__max_df': [0.75],
                         'tfidf__ngram_range': [(1, 1)]},
             verbose=1)
```

```
print("Best Parameters:", grid_search.best_params_)
```

```
Best Parameters: {'svc__C': 1, 'tfidf__max_df': 0.75,
'tfidf__ngram_range': (1, 1)}
```

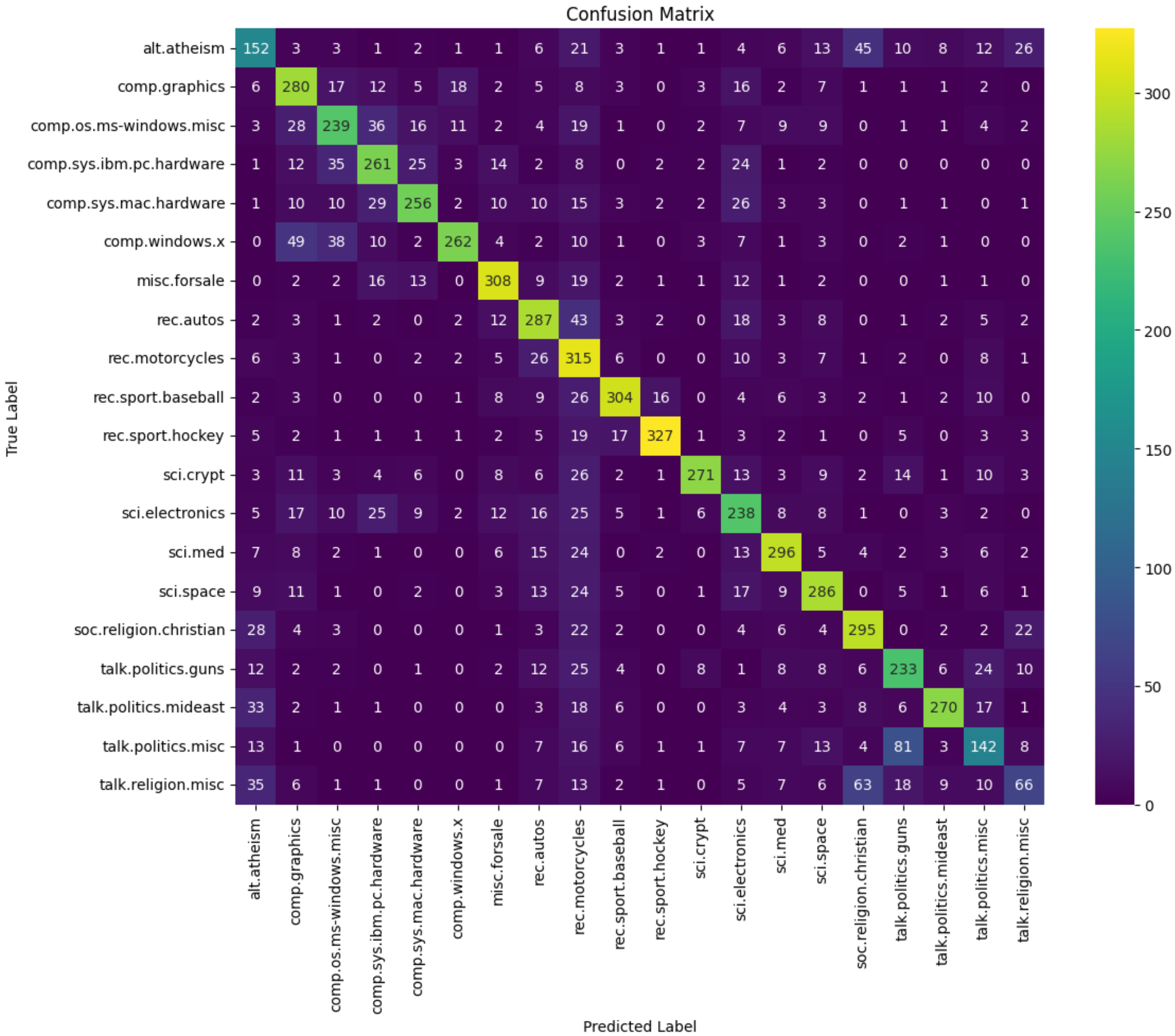
```
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

```
acc = accuracy_score(y_test, y_pred)
print("Accuracy: {:.4f}".format(acc))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.6755

Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.48      0.47       319
           1       0.61      0.72      0.66       389
           2       0.65      0.61      0.63       394
           3       0.65      0.67      0.66       392
           4       0.75      0.66      0.71       385
           5       0.86      0.66      0.75       395
           6       0.77      0.79      0.78       390
           7       0.64      0.72      0.68       396
           8       0.45      0.79      0.58       398
           9       0.81      0.77      0.79       397
          10       0.92      0.82      0.87       399
          11       0.90      0.68      0.78       396
          12       0.55      0.61      0.58       393
          13       0.77      0.75      0.76       396
          14       0.71      0.73      0.72       394
```

```
          15       0.68      0.74      0.71       398
          16       0.61      0.64      0.62       364
          17       0.86      0.72      0.78       376
          18       0.54      0.46      0.49       310
          19       0.45      0.26      0.33       251

    accuracy                           0.68      7532
   macro avg       0.68      0.66      0.67      7532
weighted avg       0.69      0.68      0.68      7532
```

```python
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt="d", cmap="viridis",
            xticklabels=newsgroups_train.target_names,
            yticklabels=newsgroups_train.target_names)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()
```


Confusion Matrix
True Label
Predicted Label
alt.atheism 152 3 3 1 2 1 1 6 21 3 1 1 4 6 13 45 10 8 12 26
comp.graphics 6 280 17 12 5 18 2 5 8 3 0 3 16 2 7 1 1 1 2 0
comp.os.ms-windows.misc 3 28 239 36 16 11 2 4 19 1 0 2 7 9 9 0 1 1 4 2
comp.sys.ibm.pc.hardware 1 12 35 261 25 3 14 2 8 0 2 2 24 1 2 0 0 0 0 0
comp.sys.mac.hardware 1 10 10 29 256 2 10 10 15 3 2 2 26 3 3 0 1 1 0 1
comp.windows.x 0 49 38 10 2 262 4 2 10 1 0 3 7 1 3 0 2 1 0 0
misc.forsale 0 2 2 16 13 0 308 9 19 2 1 1 12 1 2 0 0 1 1 0
rec.autos 2 3 1 2 0 2 12 287 43 3 2 0 18 3 8 0 1 2 5 2
rec.motorcycles 6 3 1 0 2 2 5 26 315 6 0 0 10 3 7 1 2 0 8 1
rec.sport.baseball 2 3 0 0 0 1 8 9 26 304 16 0 4 6 3 2 1 2 10 0
rec.sport.hockey 5 2 1 1 1 1 2 5 19 17 327 1 3 2 1 0 5 0 3 3
sci.crypt 3 11 3 4 6 0 8 6 26 2 1 271 13 3 9 2 14 1 10 3
sci.electronics 5 17 10 25 9 2 12 16 25 5 1 6 238 8 8 1 0 3 2 0
sci.med 7 8 2 1 0 0 6 15 24 0 2 0 13 296 5 4 2 3 6 2
sci.space 9 11 1 0 2 0 3 13 24 5 0 1 17 9 286 0 5 1 6 1
soc.religion.christian 28 4 3 0 0 0 1 3 22 2 0 0 4 6 4 295 0 2 2 22
talk.politics.guns 12 2 2 0 1 0 2 12 25 4 0 8 1 8 8 6 233 6 24 10
talk.politics.mideast 33 2 1 1 0 0 0 3 18 6 0 0 3 4 3 8 6 270 17 1
talk.politics.misc 13 1 0 0 0 0 0 7 16 6 1 1 7 7 13 4 81 3 142 8
talk.religion.misc 35 6 1 1 0 0 1 7 13 2 1 0 5 7 6 63 18 9 10 66
alt.atheism
comp.graphics
comp.os.ms-windows.misc
comp.sys.ibm.pc.hardware
comp.sys.mac.hardware
comp.windows.x
misc.forsale
rec.autos
rec.motorcycles
rec.sport.baseball
rec.sport.hockey
sci.crypt
sci.electronics
sci.med
sci.space
soc.religion.christian
talk.politics.guns
talk.politics.mideast
talk.politics.misc
talk.religion.misc
0
50
100
150
200
250
300

Aim

- **Objective:** Use Support Vector Regression (SVR) to model a 2D non-linearly separable dataset.
- **Workflow:** Generate a synthetic 2D dataset with a non-linear function, split it into training and testing sets, train an SVR (with hyperparameter tuning) using an RBF kernel, and evaluate its performance using multiple metrics and visualizations.

Algorithm

1. **Data Generation & Splitting:**
   - Create a synthetic 2D dataset with non-linear relationships and additive noise.
   - Split the data into training and testing sets.
2. **Pipeline & Hyperparameter Tuning:**
   - Build a pipeline with data standardization and SVR.
   - Use GridSearchCV (with a reduced grid for speed) to tune SVR hyperparameters.
3. **Model Evaluation:**
   - Compute evaluation metrics: R^2, Mean Squared Error (MSE), and Mean Absolute Error (MAE) on both training and test sets.
4. **Visualizations:**
   - 3D scatter plot with the regression surface.
   - Contour plot of predicted values.
   - Error histogram for residual analysis.
   - 2D scatter plots comparing true vs predicted targets.

Algorithm Description

- **Support Vector Regression (SVR):**
  SVR models the relationship between inputs and targets by fitting a function that deviates from the actual targets by a value no greater than a specified margin (epsilon). Using the RBF kernel, SVR captures complex, non-linear relationships by mapping inputs into a higher-dimensional space.
- **Evaluation & Visualization:**
  Multiple visualizations (3D, contour, and error plots) along with detailed evaluation metrics provide insight into the model's performance, error distribution, and fit quality.

Result

Training Metrics:

- **R^2 Score:** 0.9607

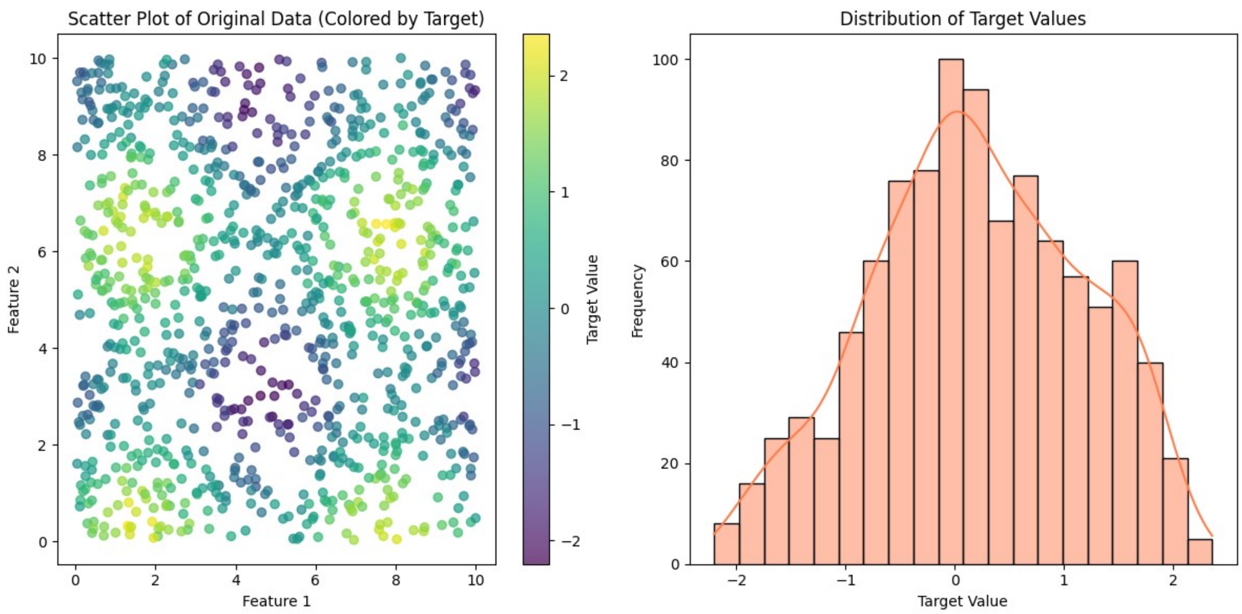
- **Mean Squared Error (MSE):** 0.0369

- **Mean Absolute Error (MAE):** 0.1548

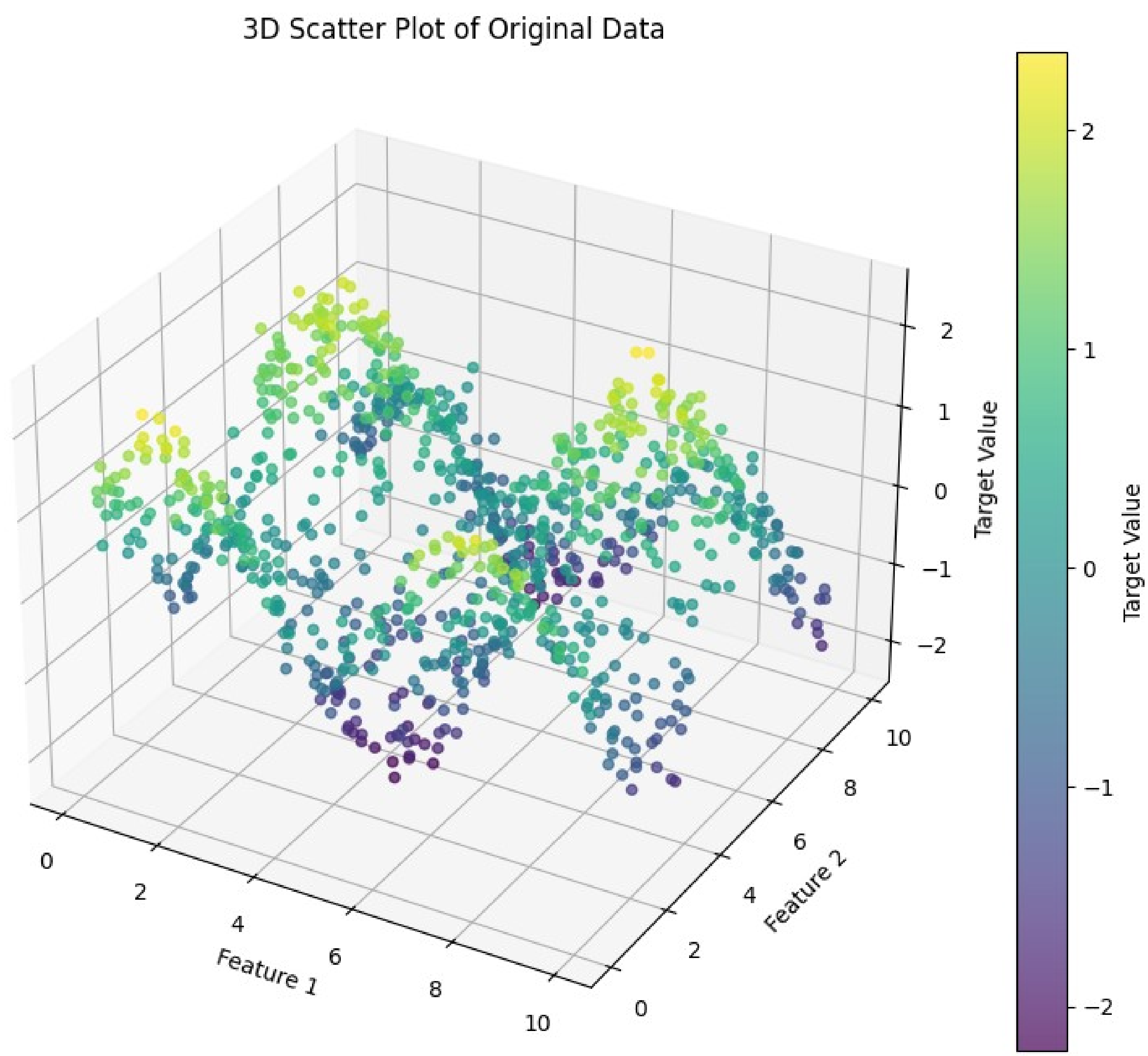
Testing Metrics:

- **R^2 Score:** 0.9538
- **Mean Squared Error (MSE):** 0.0455
- **Mean Absolute Error (MAE):** 0.1675

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error

np.random.seed(42)
n_samples = 1000

X = np.random.rand(n_samples, 2) * 10
y = np.sin(X[:, 0]) + np.cos(X[:, 1]) + np.random.normal(0, 0.2,
n_samples)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis',
alpha=0.7)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Scatter Plot of Original Data (Colored by Target)")
plt.colorbar(scatter, label="Target Value")

plt.subplot(1, 2, 2)
sns.histplot(y, bins=20, kde=True, color='coral')
plt.xlabel("Target Value")
plt.ylabel("Frequency")
plt.title("Distribution of Target Values")

plt.tight_layout()
plt.show()
```




```python
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
sc = ax.scatter(X[:, 0], X[:, 1], y, c=y, cmap='viridis', alpha=0.7)
ax.set_xlabel("Feature 1")
ax.set_ylabel("Feature 2")
ax.set_zlabel("Target Value")
ax.set_title("3D Scatter Plot of Original Data")
fig.colorbar(sc, ax=ax, label="Target Value")
plt.show()
```




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

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svr', SVR(kernel='rbf'))
])

param_grid = {
    'svr__C': [1, 10, 100, 1000],
    'svr__epsilon': [0.01, 0.05, 0.1, 0.2],
    'svr__gamma': ['scale', 'auto', 0.01, 0.1, 1]
}

grid_search = GridSearchCV(pipeline, param_grid, cv=3, verbose=1,
n_jobs=-1)
```

```python
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
```

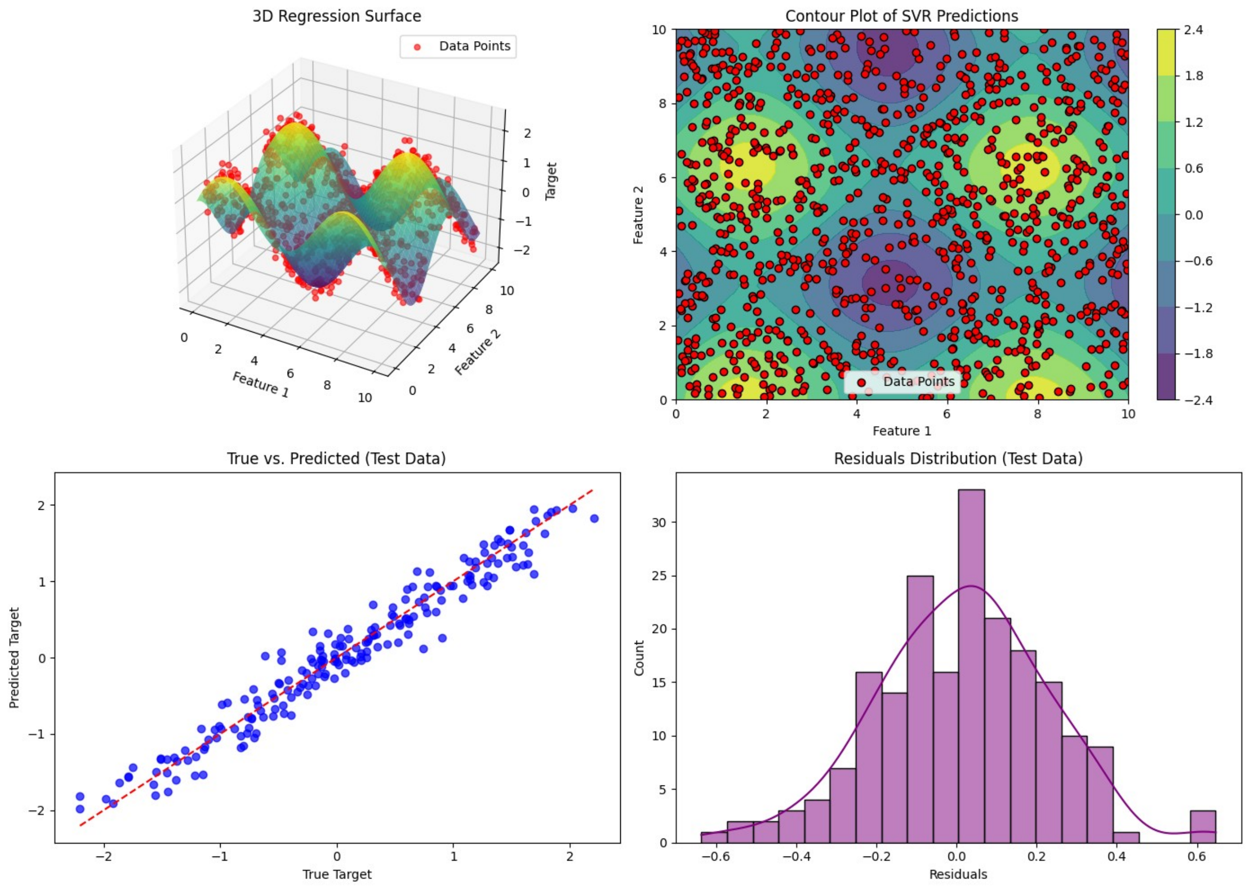
```
Fitting 3 folds for each of 80 candidates, totalling 240 fits
Best Parameters: {'svr__C': 100, 'svr__epsilon': 0.2, 'svr__gamma':
'scale'}
```

```python
best_model = grid_search.best_estimator_

y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

def print_metrics(y_true, y_pred, set_name="Dataset"):
    r2 = r2_score(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    print(f"{set_name} Metrics:")
    print(f"  R² Score: {r2:.4f}")
    print(f"  MSE: {mse:.4f}")
    print(f"  MAE: {mae:.4f}\n")

print_metrics(y_train, y_train_pred, "Training")
print_metrics(y_test, y_test_pred, "Testing")
```

```
Training Metrics:
  R² Score: 0.9607
  MSE: 0.0369
  MAE: 0.1548

Testing Metrics:
  R² Score: 0.9538
  MSE: 0.0455
  MAE: 0.1675
```

```python
grid_x = np.linspace(0, 10, 50)
grid_y = np.linspace(0, 10, 50)
xx, yy = np.meshgrid(grid_x, grid_y)
grid_points = np.c_[xx.ravel(), yy.ravel()]
zz = best_model.predict(grid_points).reshape(xx.shape)

fig = plt.figure(figsize=(14, 10))
ax = fig.add_subplot(221, projection='3d')
ax.scatter(X[:, 0], X[:, 1], y, color='red', label='Data Points',
alpha=0.6)
ax.plot_surface(xx, yy, zz, cmap='viridis', alpha=0.7)
ax.set_xlabel("Feature 1")
ax.set_ylabel("Feature 2")
ax.set_zlabel("Target")
ax.set_title("3D Regression Surface")
ax.legend()
```

```
ax2 = fig.add_subplot(222)
contour = ax2.contourf(xx, yy, zz, cmap='viridis', alpha=0.8)
plt.colorbar(contour, ax=ax2)
ax2.scatter(X[:, 0], X[:, 1], c='red', edgecolor='k', label='Data
Points')
ax2.set_xlabel("Feature 1")
ax2.set_ylabel("Feature 2")
ax2.set_title("Contour Plot of SVR Predictions")
ax2.legend()

ax3 = fig.add_subplot(223)
ax3.scatter(y_test, y_test_pred, color='blue', alpha=0.7)
ax3.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
'r--')
ax3.set_xlabel("True Target")
ax3.set_ylabel("Predicted Target")
ax3.set_title("True vs. Predicted (Test Data)")

residuals = y_test - y_test_pred
ax4 = fig.add_subplot(224)
sns.histplot(residuals, bins=20, kde=True, ax=ax4, color='purple')
ax4.set_xlabel("Residuals")
ax4.set_title("Residuals Distribution (Test Data)")

plt.tight_layout()
plt.show()
```



3D Regression Surface
Data Points
Target
Feature 1
Feature 2
Contour Plot of SVR Predictions
Data Points
Feature 1
Feature 2
True vs. Predicted (Test Data)
True Target
Predicted Target
Residuals Distribution (Test Data)
Residuals
Count

Aim

- **Objective:**
  Implement and evaluate a Single Layer Perceptron for binary classification on both linearly separable and non-linearly separable datasets.
- **Workflow:**
  Generate synthetic datasets (one linearly separable using `make_classification` and one non-linearly separable using `make_circles`), train the Perceptron on these datasets (as well as a real-world dataset like Breast Cancer), and assess its performance using accuracy and visualizations.

Algorithm

1. **Data Preparation and Exploration:**
   - Generate a synthetic 2D dataset using `make_classification` for a linearly separable scenario.
   - Explore the dataset using descriptive statistics, correlation matrices, distribution plots, and pair plots.
2. **Perceptron Implementation:**
   - Define a custom `Perceptron` class with an initialization of weights and bias.
   - Implement an activation function (step function), a `fit` method for training with weight updates using stochastic gradient descent and a learning rate decay, and a `predict` method.
3. **Model Training and Evaluation on Synthetic Data:**
   - Split the synthetic data into training and testing sets.
   - Train the Perceptron on the training data and evaluate its performance on the test set using accuracy.
4. **Evaluation on Real-World and Non-Linearly Separable Data:**
   - Train and evaluate the Perceptron on the Breast Cancer dataset (after scaling) to observe performance on real-world data.
   - Generate a non-linearly separable dataset using `make_circles`, train the Perceptron, and assess its accuracy.
5. **Visualization:**
   - Visualize the data distributions, correlations, and decision boundaries (if applicable) to gain insights into the data and model behavior.

Algorithm Description

- **Single Layer Perceptron:**
  The Perceptron is a simple linear classifier that updates its weights and bias based on the error between the predicted and actual outputs. It uses a step activation function that outputs a binary class label. During training, the algorithm adjusts its

parameters using a learning rate (which may decay over epochs) to minimize misclassifications.

- **Handling Linearly Separable vs. Non-Linearly Separable Data:**
  - For linearly separable data (e.g., generated via `make_classification`), the Perceptron converges to a solution that perfectly separates the classes, achieving high accuracy.
  - For non-linearly separable data (e.g., generated via `make_circles`), the inherent limitation of a single linear decision boundary is highlighted, often resulting in lower performance.
- **Evaluation and Visualization:**
  The performance of the Perceptron is quantified using accuracy metrics on both synthetic and real-world datasets. Visualization techniques (scatter plots, pair plots, and correlation matrices) are employed to understand data distributions and the effectiveness of the classifier.

Results

1. Synthetic Linearly Separable Data (make_classification)

- **Accuracy:** ~92.5%
- The Perceptron effectively learned the decision boundary on the synthetic, linearly separable dataset, achieving high accuracy.

2. Real-World Data (Breast Cancer)

- **Unscaled Data Accuracy:** ~95.32%
- **Scaled Data Accuracy:** ~94.15%
- The Perceptron performed robustly on the Breast Cancer dataset. While the unscaled data yielded slightly higher accuracy, scaling the features still resulted in strong performance.

3. Non-Linearly Separable Data (Circles)

- **Accuracy:** ~70.0%
- The Perceptron struggled with the non-linearly separable circle dataset, highlighting its limitation with data that requires a non-linear decision boundary.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')
```

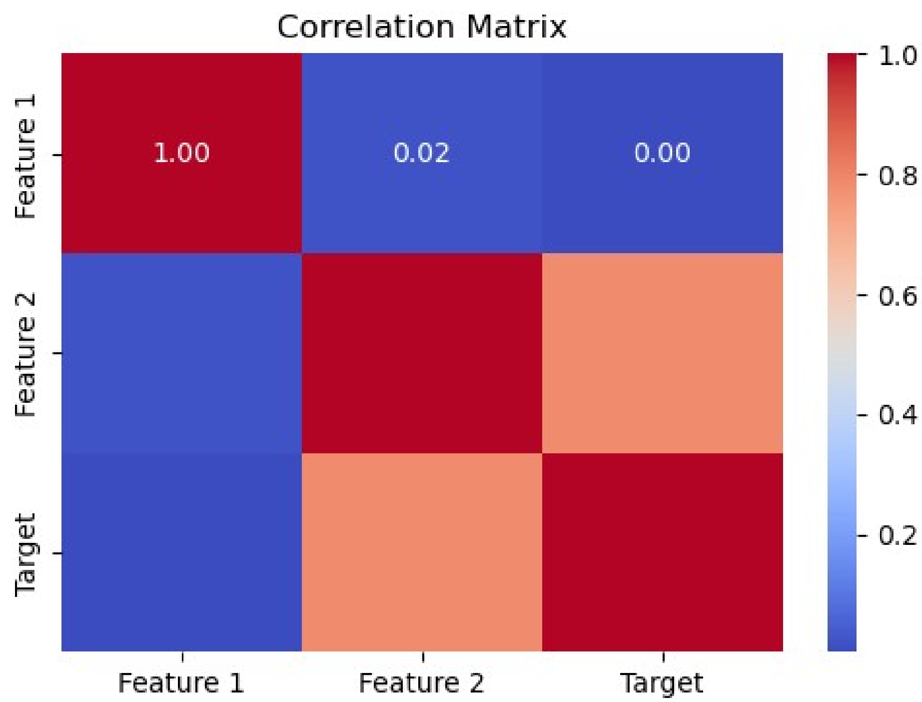
```
X, y = make_classification(n_samples=1000, n_features=2,
n_informative=2, n_redundant=0, n_clusters_per_class=1,
random_state=42)

df = pd.DataFrame(X, columns=["Feature 1", "Feature 2"])
df["Target"] = y
df
```

```
     Feature 1  Feature 2  Target
0     0.601034   1.535353       1
1     0.755945  -1.172352       0
2     1.354479  -0.948528       0
3     3.103090   0.233485       0
4     0.753178   0.787514       1
..         ...        ...     ...
995   1.713939   0.451639       1
996   1.509473  -0.794996       0
997   2.844315   0.211294       1
998  -0.025876   1.619258       1
999   3.641478   0.756925       0

[1000 rows x 3 columns]
```

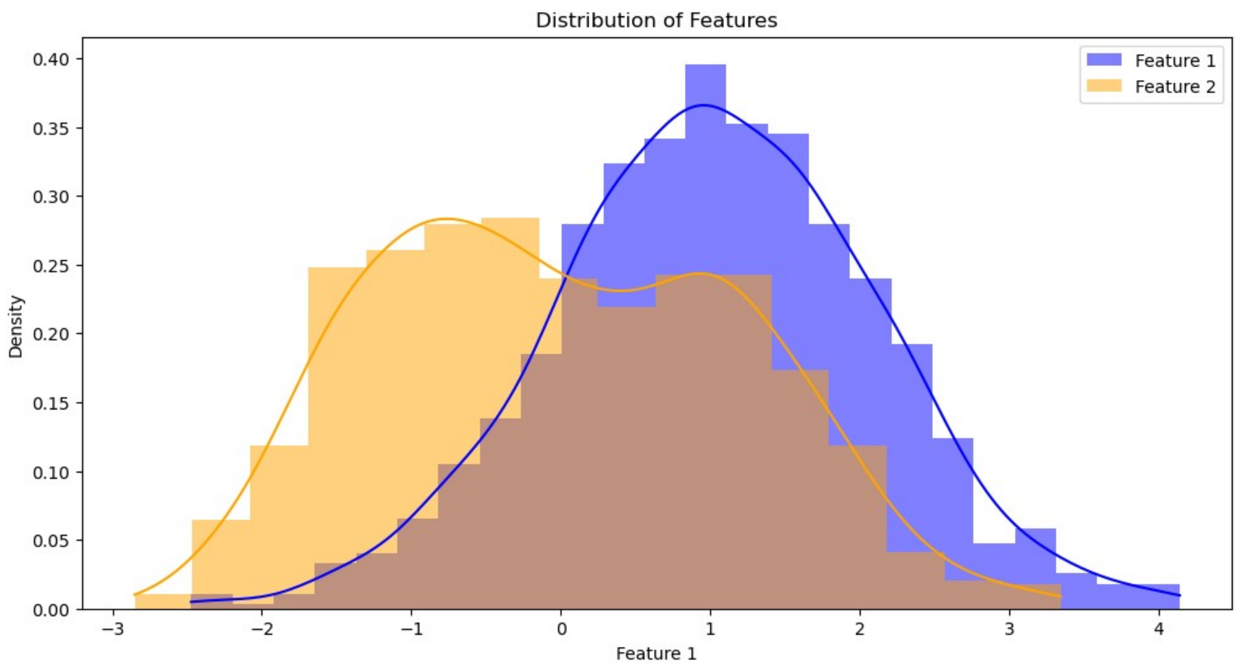
```
df.describe()
```

```
         Feature 1    Feature 2       Target
count  1000.000000  1000.000000  1000.000000
mean      1.025840    -0.012693     0.499000
std       1.071457     1.225378     0.500249
min      -2.472718    -2.850971     0.000000
25%       0.307209    -0.984268     0.000000
50%       1.023750    -0.102945     0.000000
75%       1.724713     0.973550     1.000000
max       4.138715     3.342864     1.000000
```

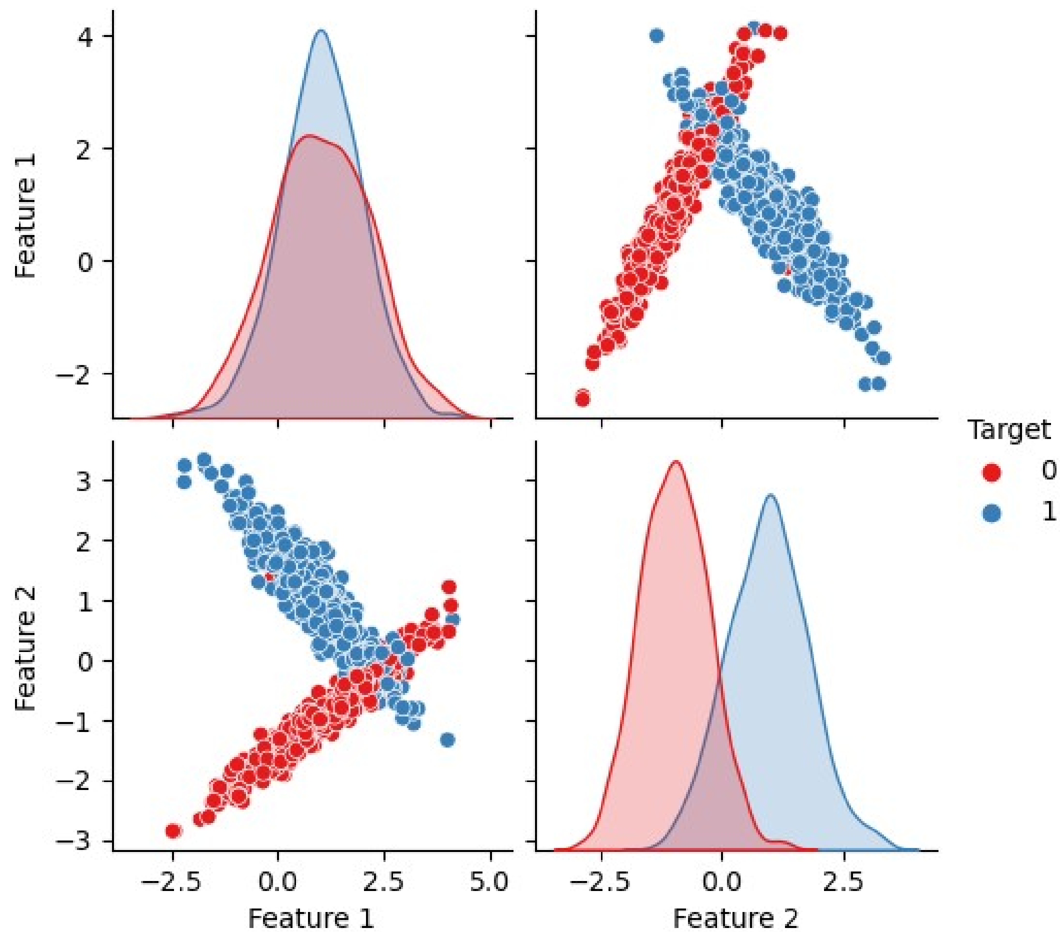
```
plt.figure(figsize=(6, 4))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```




```
plt.figure(figsize=(12, 6))
sns.histplot(df["Feature 1"], kde=True, color="blue", label="Feature 1", stat="density", linewidth=0)
sns.histplot(df["Feature 2"], kde=True, color="orange", label="Feature 2", stat="density", linewidth=0)
plt.title('Distribution of Features')
plt.legend()
plt.show()
```




```
sns.pairplot(df, hue="Target", palette="Set1")
plt.show()
```




```
class Perceptron:

    def __init__(self, learning_rate:float = 0.00001, epoch:int =
100_000, decay_rate: float = 0.975):
        self.learning_rate = learning_rate
        self.epoch = epoch
        self.decay_rate = decay_rate
        self.weight = None
        self.bias = None

    def activation_function(self, x):
        return 1 if x >=0 else 0

    def fit(self, X:np.ndarray, y:np.ndarray):

        self.weight = np.random.randn(X.shape[1]) * 0.01
        self.bias = 0
        prev_weights = np.copy(self.weight) # To monitor convergence

        for epoch in range(1, self.epoch+1):
```

```
            indices = np.random.permutation(len(X))
            X_shuffled = X[indices]
            y_shuffled = y[indices]

            total_error = 0

            for i in range(len(X_shuffled)):

                # Forward Pass
                weighted_sum = np.dot(X_shuffled[i], self.weight) + 
self.bias
                predicted = self.activation_function(weighted_sum)

                # Calculate error
                error = y_shuffled[i] - predicted
                total_error += abs(error)

                # Update weights
                self.weight += self.learning_rate * error * 
X_shuffled[i]
                self.bias += self.learning_rate * error

            self.learning_rate *= self.decay_rate

            if epoch % 1000 == 0:
                print(f"Epoch {epoch}/{self.epoch}, Total error: 
{total_error}", end='\r')

            if np.all(np.abs(self.weight - prev_weights) < 1e-100):
                print(f"Convergence reached at epoch {epoch}.")
                break

            prev_weights = np.copy(self.weight)

    def predict(self, X:np.ndarray):

        results = []

        for i in range(len(X)):
            weighted_sum = np.dot(X[i], self.weight) + self.bias
            pred = self.activation_function(weighted_sum)

            results.append(pred)

        return np.array(results)

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.25,
```

```
    random_state=42
)

model = Perceptron()

model.fit(X_train, y_train)

Convergence reached at epoch 8607.5

pred_i = model.predict(X_test)
pred_i

array([1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0,
0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
0,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
1,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
0,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
0,
       0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0,
0,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1,
0,
       1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
0,
       0, 1, 1, 0, 1, 0, 1, 0])

accuracy_score(y_true=y_test, y_pred=pred_i) * 100

89.2

accuracy = np.mean(pred_i == y_test)
accuracy * 100

89.2
```

Breast Cancer

```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
```

```
data = load_breast_cancer()
X, y  = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.3,
    random_state=42
)

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

model1 = Perceptron(learning_rate=1000)

model1.fit(X_train_scaled, y_train)
```

```
Convergence reached at epoch 105.
```

```
prediction = model1.predict(X_test_scaled)
prediction
```

```
array([1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1,
1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
1,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
0,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0,
0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1,
1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1])
```

```
accuracy_score(y_true=y_test, y_pred=prediction) * 100
```

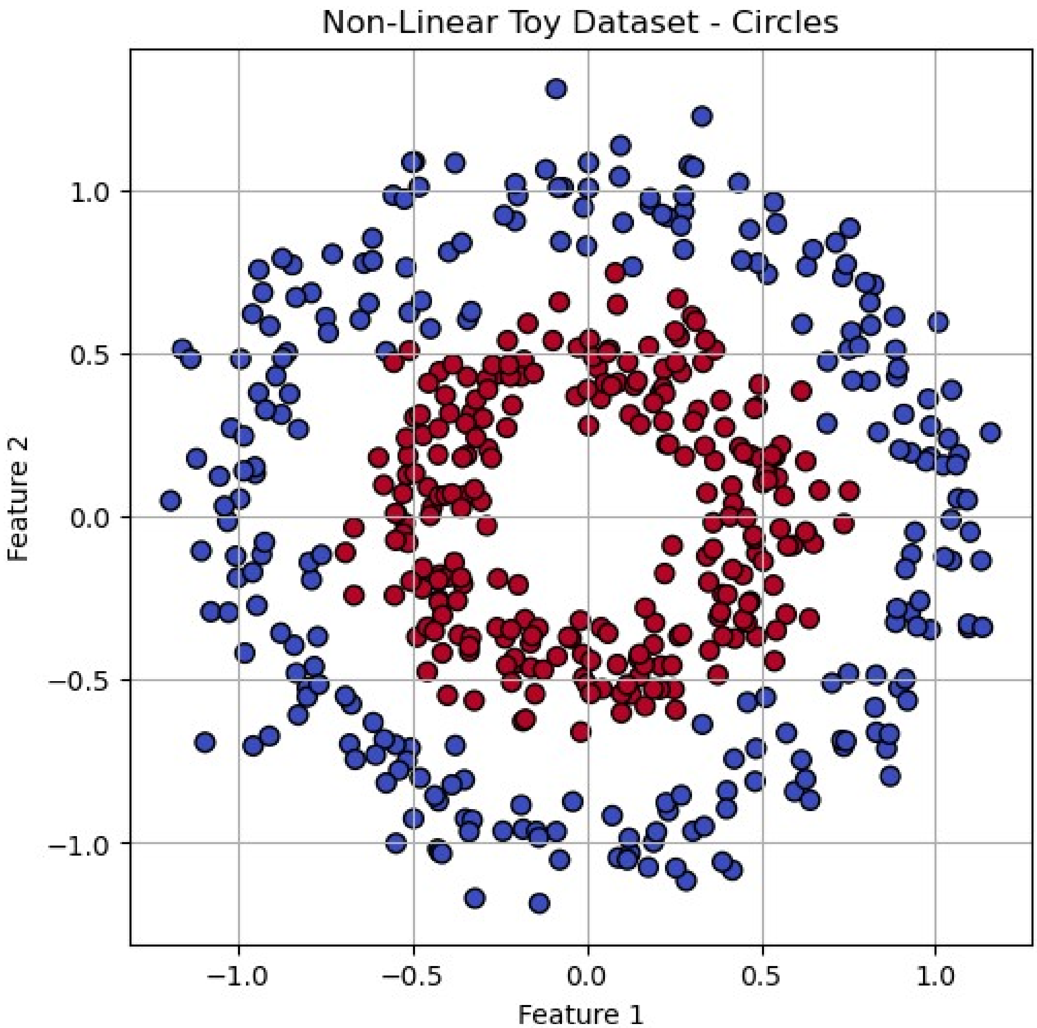
```
94.15204678362574
```

95.32163742690058 -> 0.00001 -> Unscaled

94.15204678362574 -> 0.01 -> Scaled

Circle Dataset

```
from sklearn.datasets import make_circles

X, y = make_circles(n_samples=500, noise=0.1, factor=0.5,
random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

plt.figure(figsize=(6,6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', s=50,
edgecolors='black')
plt.title("Non-Linear Toy Dataset - Circles")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.show()
```




```
print("Sample data points (X, y):")
print(X[:10], y[:10])
```

```
Sample data points (X, y):
[[-0.46918557  0.24791499]
 [-0.06748724  1.00676912]
 [-0.44306526  0.02738322]
 [-0.61172505 -0.6314071 ]
 [-0.78901285  0.68451888]
 [-0.42136979 -0.25688616]
 [-0.45473954  0.08850207]
 [-0.94954151  0.13119395]
 [-0.20468777  0.903653  ]
 [ 0.10000474 -0.60226649]] [1 0 1 0 0 1 1 0 0 1]
```

```
model2 = Perceptron()

model2.fit(X_train, y_train)
```

```
Convergence reached at epoch 8587.4
```

```
prediction = model2.predict(X_test)
prediction
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
accuracy_score(y_true=y_test, y_pred=prediction) * 100
```

```
56.99999999999999
```

Aim

- **Objective:**
  Implement and evaluate a Multi-Layer Perceptron (MLP) classifier for binary classification on both linearly separable and non-linearly separable datasets.
- **Workflow:**
  - **Data Generation:**
    Create two synthetic datasets: one that is linearly separable using `make_classification` and another that is non-linearly separable using `make_moons`.
  - **Preprocessing:**
    Split each dataset into training and testing sets and standardize the features using `StandardScaler`.
  - **Model Training & Hyperparameter Tuning:**
    Train an MLP classifier using an extensive hyperparameter grid with GridSearchCV to find the best model configuration.
  - **Evaluation & Visualization:**
    Evaluate model performance using accuracy, classification reports, confusion matrices, ROC curves, and loss curves. Also, visualize decision boundaries to better understand model behavior.

Algorithm

1. **Data Preparation and Exploration:**
   - Generate a 2D synthetic dataset using `make_classification` for linearly separable data.
   - Generate a non-linearly separable dataset using `make_moons`.
   - Visualize both datasets with scatter plots to inspect the distribution and separability.
2. **Preprocessing:**
   - Split the data into training and testing sets using stratified sampling.
   - Scale the data using `StandardScaler` to normalize features, which is essential for neural network performance.
3. **Model Training and Hyperparameter Tuning:**
   - Define an extensive grid of hyperparameters (including multiple configurations for hidden layer sizes, activation functions, solvers, regularization strength, and learning rates).
   - Use `GridSearchCV` with cross-validation to find the best set of hyperparameters.
4. **Model Evaluation:**
   - Test the best estimator on the hold-out test set.
   - Compute standard classification metrics such as accuracy, precision, recall, and F1-score via the classification report.

- Visualize the confusion matrix to inspect the distribution of errors.
- Plot the ROC curve and calculate the AUC to evaluate the performance of the model on binary classification.
- Visualize the decision boundary to understand how the model separates the classes.
- Optionally, plot the training loss curve to observe the convergence behavior of the model.

5. **Visualization:**
   - Use matplotlib and seaborn for visualizations of raw data, decision boundaries, ROC curves, loss curves, and confusion matrices, providing both qualitative and quantitative insights into the model performance.

Algorithm Description

- **Multi-Layer Perceptron (MLP) Classifier:**
  The MLP is a feedforward neural network that employs one or more hidden layers with non-linear activation functions (such as ReLU or tanh). This non-linearity allows the network to capture complex patterns and relationships in the data.

- **Hyperparameter Tuning via GridSearchCV:**
  An exhaustive search is conducted over a broad range of hyperparameters. This includes varying the number and size of hidden layers, activation functions, solvers (like `adam` or `lbfgs`), regularization parameters (`alpha`), and learning rate strategies. The goal is to optimize the network's performance for each dataset.

- **Data Variability Management:**

  - For **linearly separable data** (via `make_classification`):
    A simpler network architecture might suffice, as the decision boundary between the two classes is more straightforward.
  - For **non-linearly separable data** (via `make_moons`):
    The network may require a more complex architecture or non-linear activation functions to effectively model the curved decision boundaries.
- **Evaluation and Visualization:**
  The MLP's performance is assessed using comprehensive metrics and visualizations that include:

  - **Accuracy and Classification Report:**
    Provide a quantitative measure of model performance.
  - **Confusion Matrix:**
    Visualize true vs. predicted values.
  - **ROC Curve and AUC:**
    Assess the trade-off between true positive and false positive rates.
  - **Decision Boundaries and Loss Curves:**
    Offer visual insights into how well the model separates classes and how effectively it converges during training.

Results

1. Linearly Separable Data (`make_classification`)

- **Model Performance:**
  - The MLP learns a clear decision boundary, effectively separating the classes.
  - **Test Accuracy:** 0.9867
- **Visual Insights:**
  - **Decision Boundary:** Clearly distinguishes between the two classes.
  - **Loss Curve:** Shows convergence behavior, indicating stable training.
  - **ROC Curve:** High AUC value indicating strong model performance.

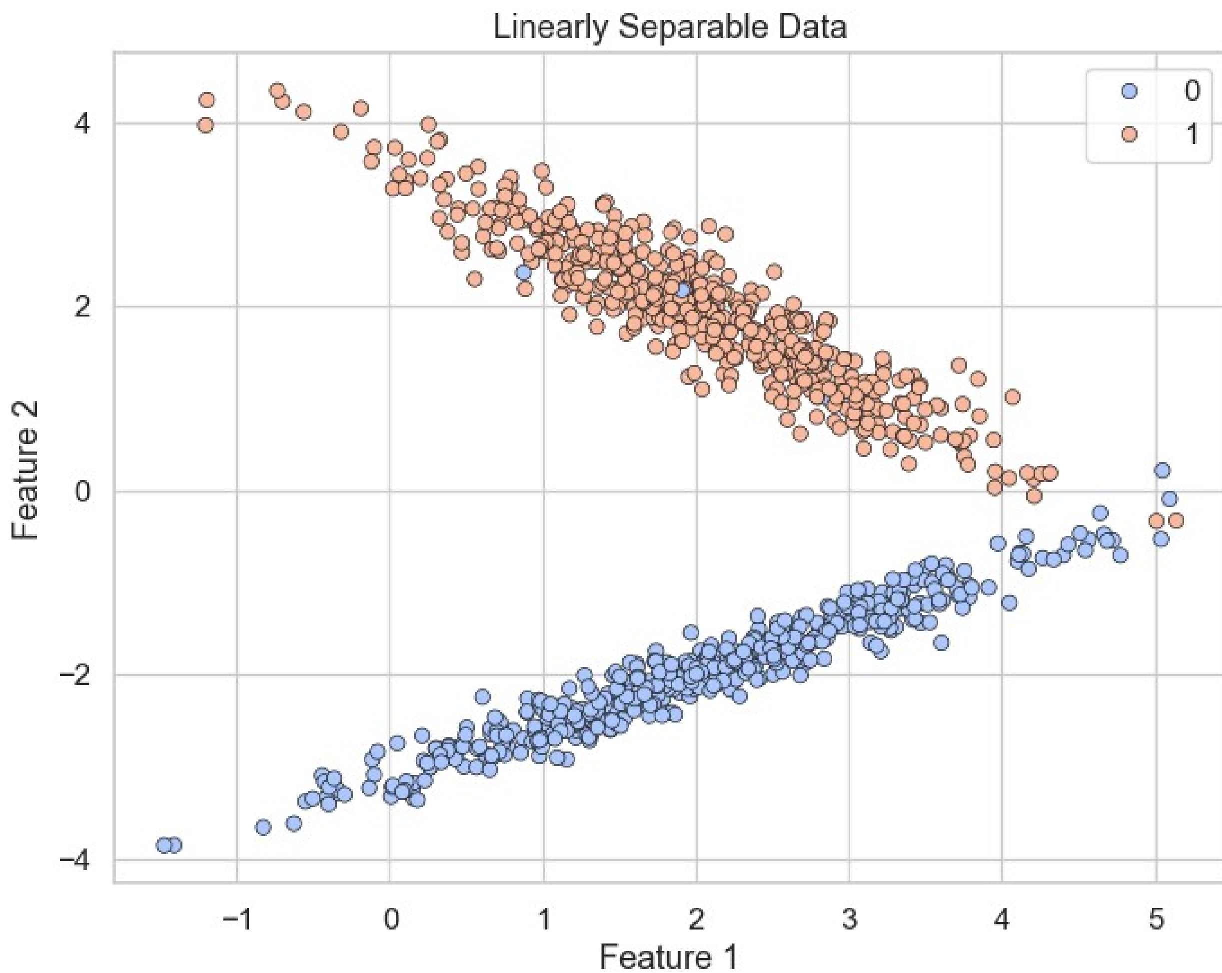
2. Non-Linearly Separable Data (`make_moons`)

- **Model Performance:**
  - The MLP adapts to the non-linearities in the data, achieving robust performance with a slightly more complex architecture.
  - **Test Accuracy:** 0.9833
- **Visual Insights:**
  - **Decision Boundary:** Effectively follows the curved patterns in the data.
  - **Loss Curve & ROC Curve:** Provide additional understanding of the model's training dynamics and classification capability.
  - **Confusion Matrix:** Helps identify any misclassifications and overall error distribution.

Linear Data

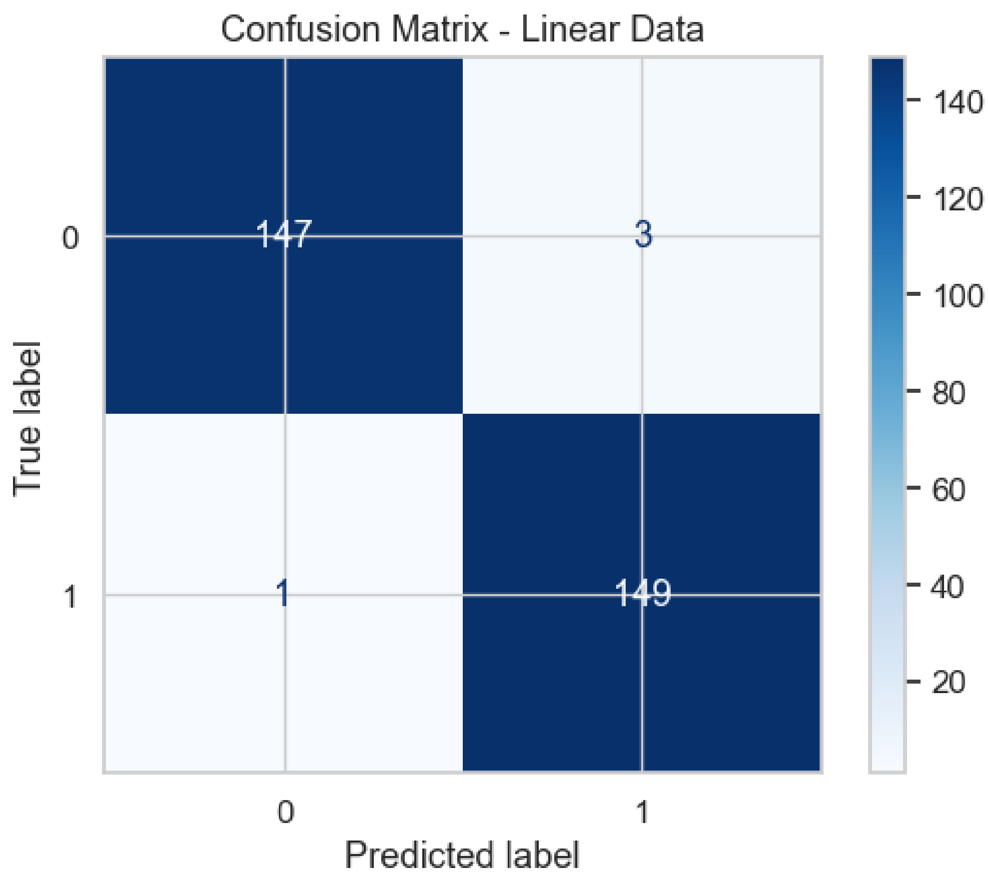
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import make_classification, make_moons
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (classification_report, accuracy_score,
                             ConfusionMatrixDisplay, roc_curve, auc)

from matplotlib.colors import ListedColormap

sns.set(style="whitegrid", font_scale=1.1)
%matplotlib inline

def plot_decision_boundary(clf, X, y, title):
    h = 0.02  # mesh step size
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
```

```
                         np.arange(y_min, y_max, h))

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    cmap_light = ListedColormap(['#FFCCCC', '#CCCCFF'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k',
s=50)
    plt.title(title)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()

X_lin, y_lin = make_classification(n_samples=1000, n_features=2,
n_informative=2,
                                   n_redundant=0,
n_clusters_per_class=1,
                                   class_sep=2.0, random_state=42)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_lin[:, 0], y=X_lin[:, 1], hue=y_lin,
palette="coolwarm", edgecolor='k')
plt.title("Linearly Separable Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

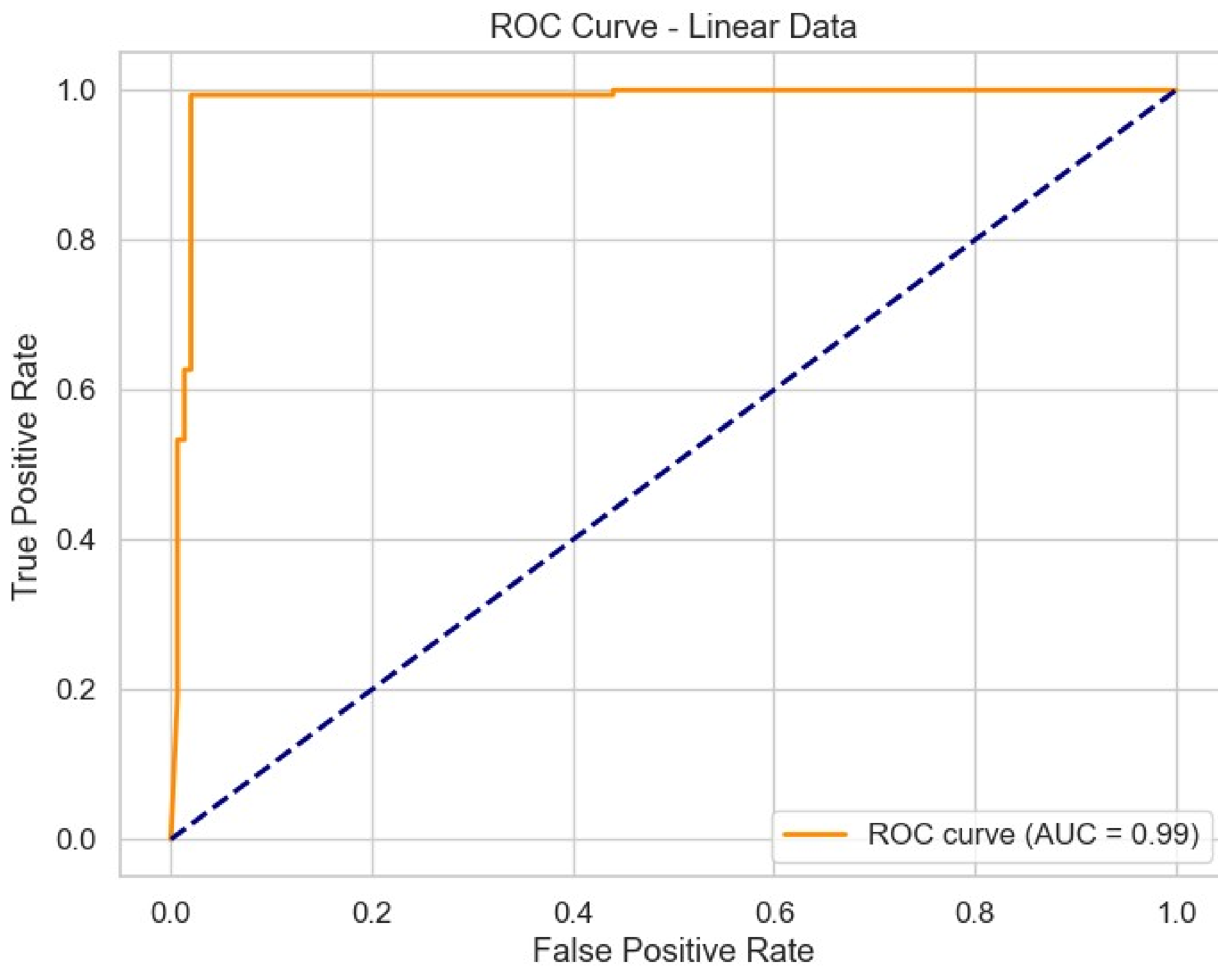



```
X_train_lin, X_test_lin, y_train_lin, y_test_lin = train_test_split(
    X_lin, y_lin, test_size=0.3, stratify=y_lin, random_state=42
)

scaler_lin = StandardScaler()
X_train_lin = scaler_lin.fit_transform(X_train_lin)
X_test_lin = scaler_lin.transform(X_test_lin)

param_grid = {
    'hidden_layer_sizes': [(5,), (10,), (50,), (10, 10), (50, 20)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'lbfgs'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'adaptive']
}

mlp_lin = MLPClassifier(max_iter=1000, random_state=42, verbose=1)
grid_lin = GridSearchCV(mlp_lin, param_grid, cv=5, n_jobs=-1,
verbose=1)
```

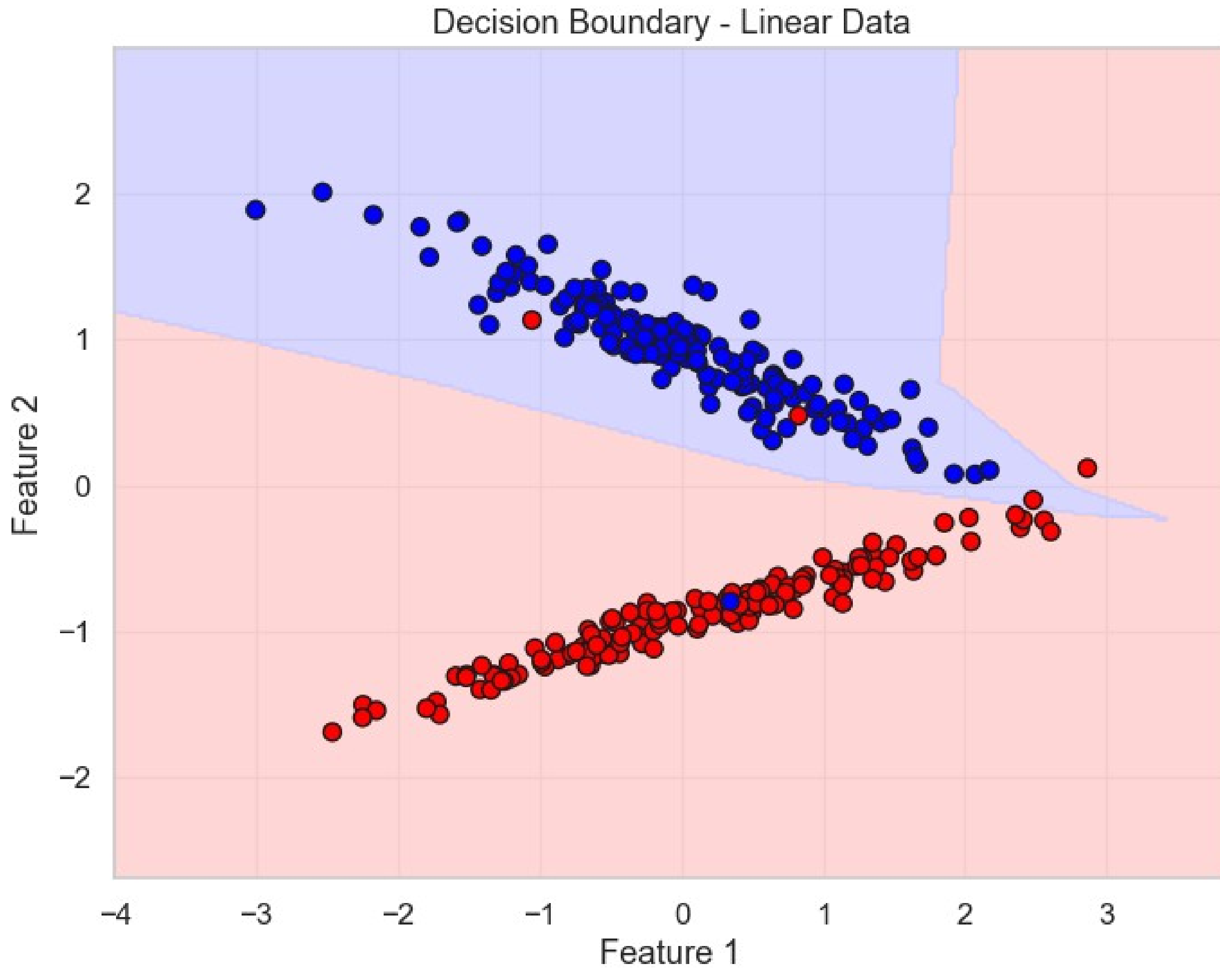
```
grid_lin.fit(X_train_lin, y_train_lin)

Fitting 5 folds for each of 120 candidates, totalling 600 fits

GridSearchCV(cv=5,
             estimator=MLPClassifier(max_iter=1000, random_state=42,
verbose=1),
             n_jobs=-1,
             param_grid={'activation': ['relu', 'tanh'],
                         'alpha': [0.0001, 0.001, 0.01],
                         'hidden_layer_sizes': [(5,), (10,), (50,),
(10, 10),
                                                (50, 20)],
                         'learning_rate': ['constant', 'adaptive'],
                         'solver': ['adam', 'lbfgs']},
             verbose=1)

print("Best parameters for Linear Data:")
print(grid_lin.best_params_)
best_lin_model = grid_lin.best_estimator_

Best parameters for Linear Data:
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10,
10), 'learning_rate': 'constant', 'solver': 'lbfgs'}

y_pred_lin = best_lin_model.predict(X_test_lin)
accuracy_lin = accuracy_score(y_test_lin, y_pred_lin)
print(f"Test Accuracy (Linear Data): {accuracy_lin:.4f}\n")

Test Accuracy (Linear Data): 0.9867


print("Classification Report:")
print(classification_report(y_test_lin, y_pred_lin))

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       150
           1       0.98      0.99      0.99       150

    accuracy                           0.99       300
   macro avg       0.99      0.99      0.99       300
weighted avg       0.99      0.99      0.99       300


ConfusionMatrixDisplay.from_estimator(best_lin_model, X_test_lin,
y_test_lin, cmap='Blues')
plt.title("Confusion Matrix - Linear Data")
plt.show()
```




```
try:
    probas_lin = best_lin_model.predict_proba(X_test_lin)[:, 1]
    fpr_lin, tpr_lin, _ = roc_curve(y_test_lin, probas_lin)
    roc_auc_lin = auc(fpr_lin, tpr_lin)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr_lin, tpr_lin, color='darkorange', lw=2, label=f'ROC
curve (AUC = {roc_auc_lin:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve - Linear Data')
    plt.legend(loc="lower right")
    plt.show()
except Exception as e:
    print("ROC Curve not available:", e)
```

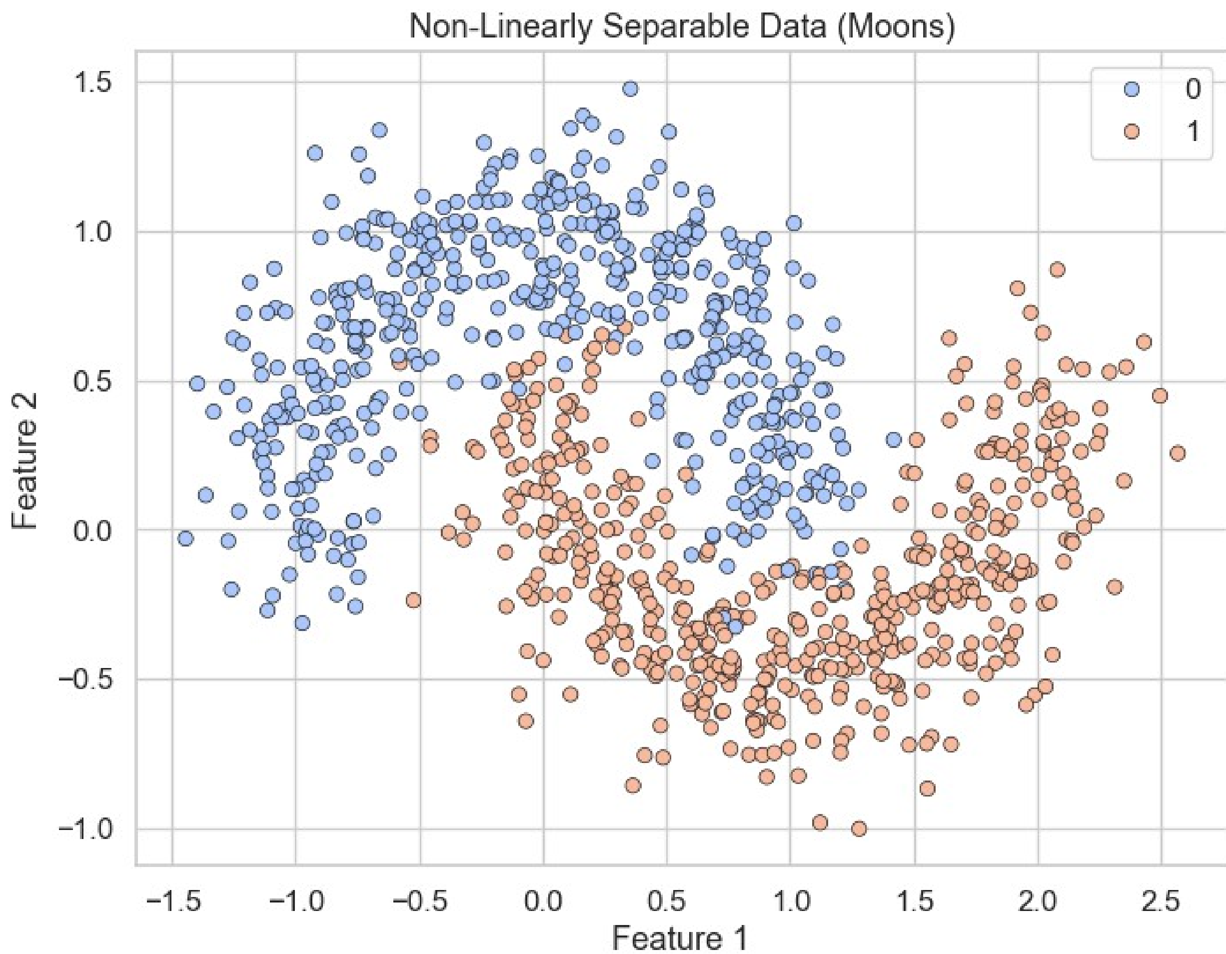



```
plot_decision_boundary(best_lin_model, X_test_lin, y_test_lin,
"Decision Boundary - Linear Data")
```



Non-Linear Data (Moons)

```
X_nl, y_nl = make_moons(n_samples=1000, noise=0.2, random_state=42)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_nl[:, 0], y=X_nl[:, 1], hue=y_nl,
palette="coolwarm", edgecolor='k')
plt.title("Non-Linearly Separable Data (Moons)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```




```
X_train_nl, X_test_nl, y_train_nl, y_test_nl = train_test_split(
    X_nl, y_nl, test_size=0.3, stratify=y_nl, random_state=42
)

scaler_nl = StandardScaler()
X_train_nl = scaler_nl.fit_transform(X_train_nl)
X_test_nl = scaler_nl.transform(X_test_nl)

mlp_nl = MLPClassifier(max_iter=1000, random_state=42)
grid_nl = GridSearchCV(mlp_nl, param_grid, cv=5, n_jobs=-1)

grid_nl.fit(X_train_nl, y_train_nl)

GridSearchCV(cv=5, estimator=MLPClassifier(max_iter=1000,
random_state=42),
             n_jobs=-1,
             param_grid={'activation': ['relu', 'tanh'],
                         'alpha': [0.0001, 0.001, 0.01],
                         'hidden_layer_sizes': [(5,), (10,), (50,),
(10, 10),
                                                (50, 20)],
```

```
                         'learning_rate': ['constant', 'adaptive'],
                         'solver': ['adam', 'lbfgs']})

print("Best parameters for Non-Linear Data:")
print(grid_nl.best_params_)
best_nl_model = grid_nl.best_estimator_
```

```
Best parameters for Non-Linear Data:
{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50,
20), 'learning_rate': 'constant', 'solver': 'adam'}
```

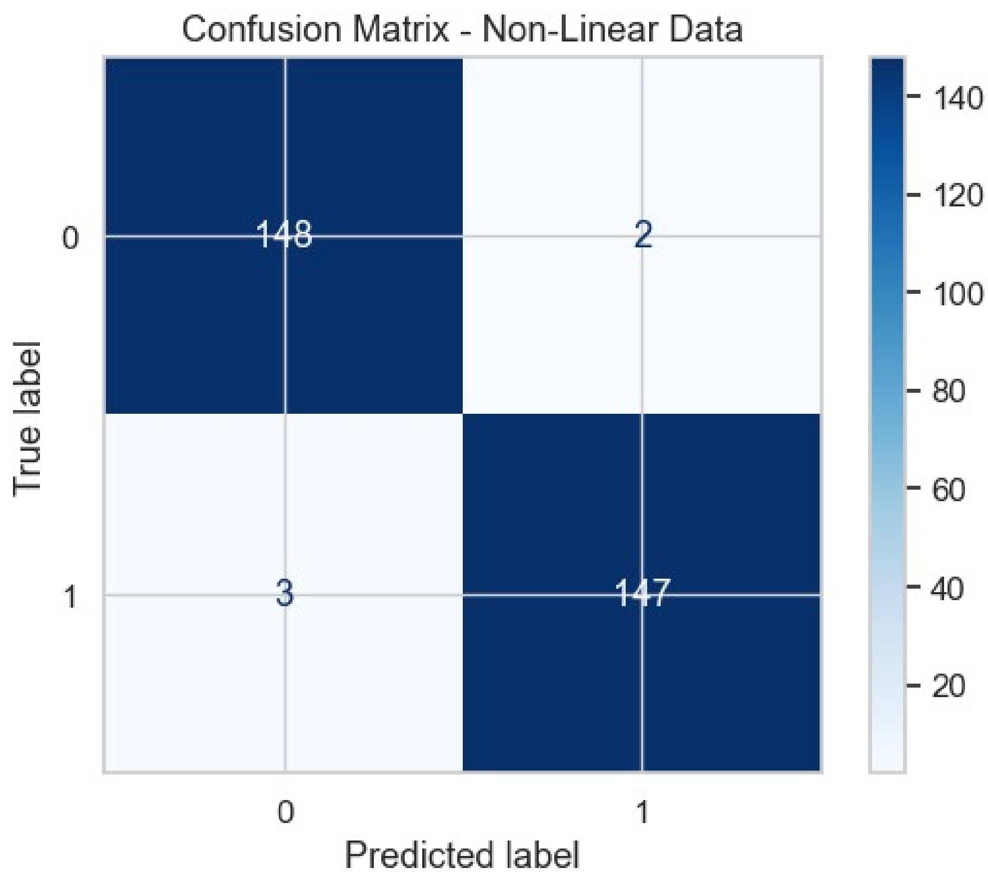
```
y_pred_nl = best_nl_model.predict(X_test_nl)
accuracy_nl = accuracy_score(y_test_nl, y_pred_nl)
print(f"Test Accuracy (Non-Linear Data): {accuracy_nl:.4f}\n")
```

```
Test Accuracy (Non-Linear Data): 0.9833
```

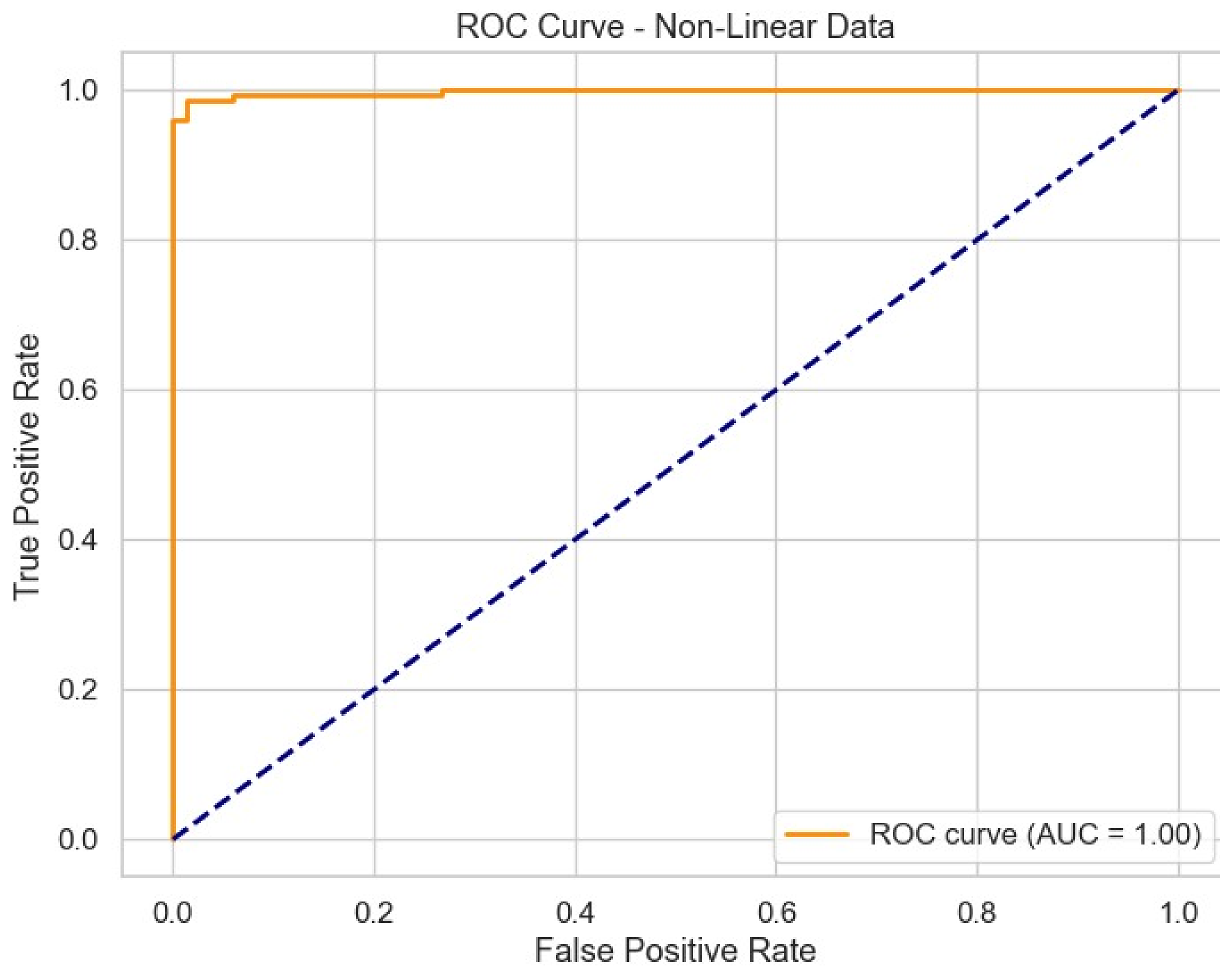
```
print("Classification Report:")
print(classification_report(y_test_nl, y_pred_nl))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       150
           1       0.99      0.98      0.98       150

    accuracy                           0.98       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300
```

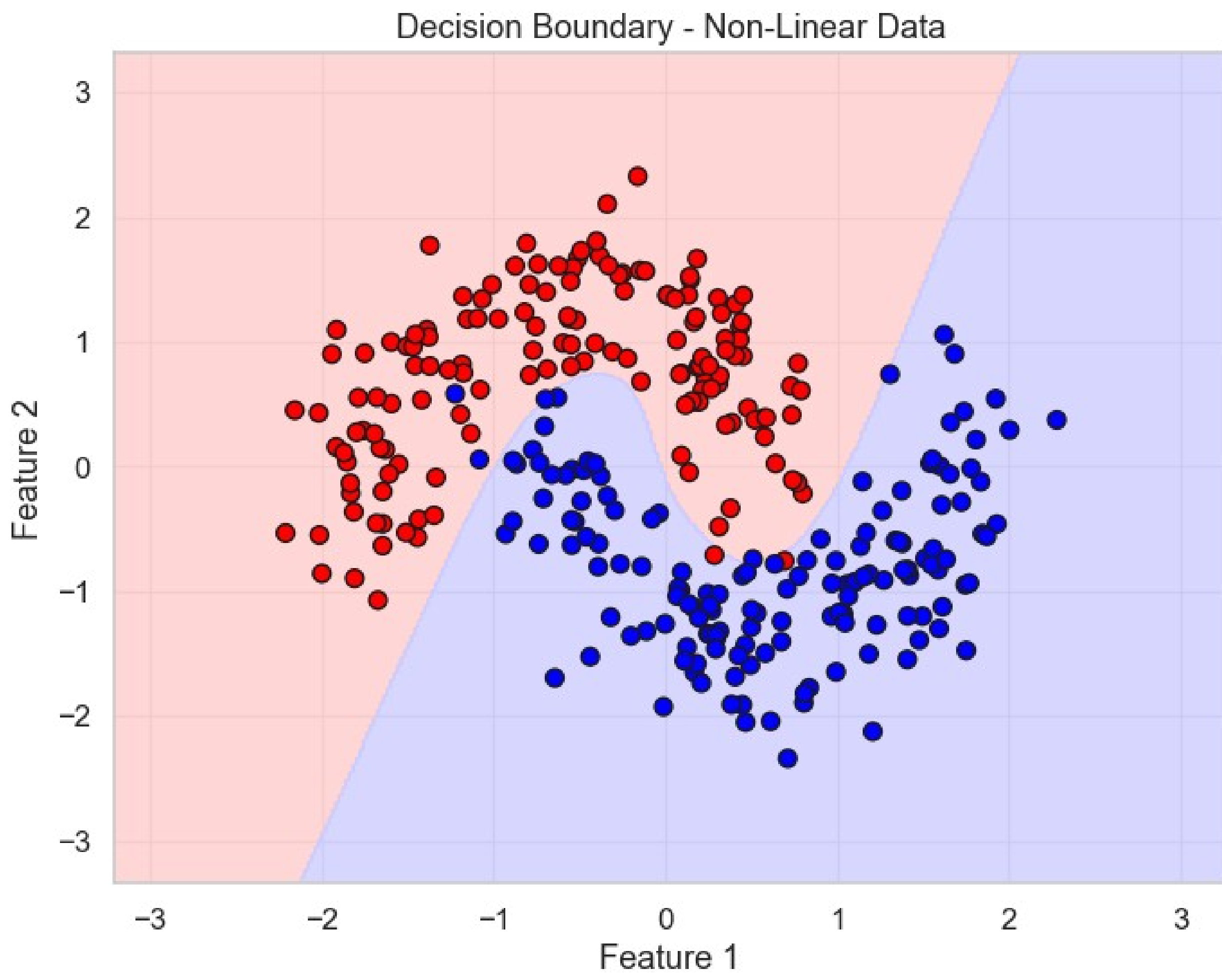
```
ConfusionMatrixDisplay.from_estimator(best_nl_model, X_test_nl,
y_test_nl, cmap='Blues')
plt.title("Confusion Matrix - Non-Linear Data")
plt.show()
```



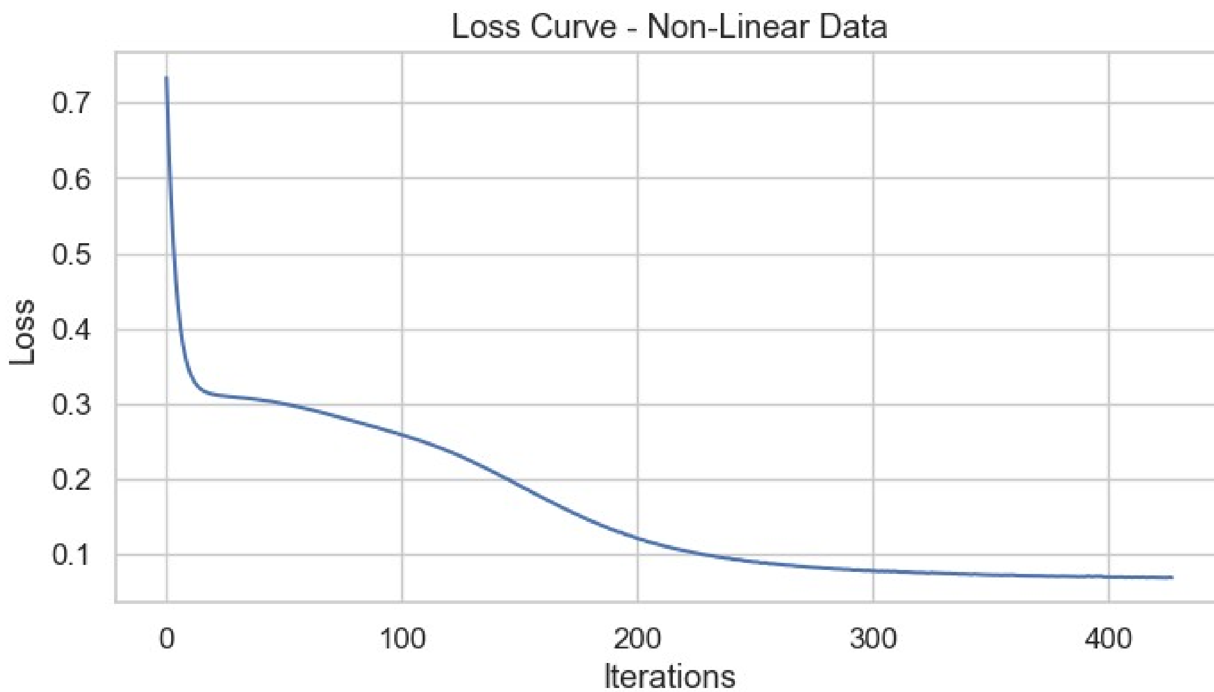

```
try:
    probas_nl = best_nl_model.predict_proba(X_test_nl)[:, 1]
    fpr_nl, tpr_nl, _ = roc_curve(y_test_nl, probas_nl)
    roc_auc_nl = auc(fpr_nl, tpr_nl)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr_nl, tpr_nl, color='darkorange', lw=2, label=f'ROC
curve (AUC = {roc_auc_nl:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve - Non-Linear Data')
    plt.legend(loc="lower right")
    plt.show()
except Exception as e:
    print("ROC Curve not available:", e)
```




```
plot_decision_boundary(best_nl_model, X_test_nl, y_test_nl, "Decision
Boundary - Non-Linear Data")
```




```
plt.figure(figsize=(8, 4))
plt.plot(best_nl_model.loss_curve_)
plt.title("Loss Curve - Non-Linear Data")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```


Loss Curve - Non-Linear Data
Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
100
200
300
400
Iterations