Submitted To: Prof. Amarnath Mitra

Submitted By: Om Wadhwa

Roll No: 045037

Project Report

Problem Statement:

The E-Commerce company faces challenges in predicting which products are likely to be reordered by customers. Without accurate predictions, the company struggles with inventory management, customer satisfaction, and operational efficiency. To address these challenges, the company aims to develop a predictive model that can forecast reordered products based on historical order data.

Objectives:

1. **Develop a Predictive Model**:
   - Build a machine learning model to predict whether a product will be reordered by a customer.
   - Utilize historical order data, including features such as order number, day of the week, hour of the day, and days since the prior order.
2. **Improve Inventory Management**:
   - Enhance inventory planning by accurately forecasting which products are likely to be reordered.
   - Ensure optimal stock levels to meet customer demand and reduce instances of stockouts or overstocking.
3. **Enhance Customer Satisfaction**:
   - Improve customer experience by ensuring that popular and frequently reordered products are readily available.
   - Personalize recommendations and promotions based on predicted reorder behavior to increase customer engagement.
4. **Optimize Operational Efficiency**:
   - Streamline operations by aligning supply chain management with predicted reorder patterns.
   - Optimize delivery schedules and workforce planning to improve efficiency and reduce costs.
5. **Increase Sales and Revenue**:
   - Boost sales by strategically promoting products with high reorder rates.
   - Implement targeted marketing campaigns to encourage repeat purchases from customers likely to reorder.
6. **Enable Data-Driven Decision Making**:

   - Empower the company with a robust predictive model that supports data-driven decision-making.
   - Provide actionable insights into customer behavior and reorder patterns for informed business strategies.
7. **Continuous Model Improvement**:
   - Establish a framework for continuous monitoring and updating of the predictive model.
   - Regularly evaluate model performance with new data and refine the model to adapt to changing customer behavior and market dynamics.
8. **Ensure Interpretability and Communication**:
   - Ensure that the developed model is interpretable and understandable to stakeholders.
   - Effectively communicate model insights and recommendations to business units for successful implementation.

Description of Data:

The dataset consists of information related to orders made by users, with a total of 10,48,575 entries. Each row represents a specific order and includes features such as `order_id`, `user_id`, `order_number`, `order_dow` (day of the week the order was placed), `order_hour_of_day` (hour of the day the order was placed), `days_since_prior_order` (number of days since the user's previous order), `product_id`, `add_to_cart_order`, `reordered`, `department_id`, `department`, and `product_name`.

Categorical and Non-Categorical Variables:

- **Categorical Variables**:
  - `department`: Categorical variable representing the department to which the product belongs.
  - `order_dow`: Categorical variable representing the day of the week the order was placed.
- **Non-Categorical Variables**:
  - `order_id`, `user_id`, `order_number`, `order_hour_of_day`, `days_since_prior_order`, `product_id`, `add_to_cart_order`, `reordered`, `department_id`, 'product_name': Numerical variables representing various aspects of the orders and products.

Unsupervised Learning: Clustering - K-Means {K = 2, 3, 4, 5}

- **K-Means Clustering Results**:
  - **K = 2**:
    - Silhouette Score: 0.2815
    - Davies-Bouldin Score: 1.5017
  - **K = 3**:
    - Silhouette Score: 0.3146
    - Davies-Bouldin Score: 1.1746
  - **K = 4**:
    - Silhouette Score: 0.3057

    - Davies-Bouldin Score: 1.0096
  - **K = 5**:
    - Silhouette Score: 0.3008
    - Davies-Bouldin Score: 1.0076
- **Analysis**:
  - The K-Means clustering results suggest that K = 3 might be good choices based on Silhouette and Davies-Bouldin Scores.
  - Further exploration and visualization of these clusters could provide insights into user behavior patterns.

Supervised Learning: Classification

Decision Tree vs {Logistic Regression | K-Nearest Neighbor | Support Vector Machine}

- **Decision Tree Results**:
  - Accuracy: 0.614,
  - Precision: 0.6148,
  - Recall: 0.614,
  - F1 Score: 0.6144

Analysis: The Decision Tree model shows a moderate performance with an accuracy of 61.4%, indicating its ability to correctly predict reordered products 61.4% of the time. Precision stands at 61.48%, showing that when the model predicts a product will be reordered, it is accurate about 61.48% of the time. The model's recall, or its ability to find all the reordered products, is 61.4%. The F1 Score, balancing precision and recall, is 61.44%.

While these metrics provide a baseline, there is room for improvement. Managers should consider the model's implications in their business context. Further steps such as data preprocessing, feature engineering, or trying advanced algorithms could enhance performance. Continuous monitoring and updates with new data are crucial for accuracy. Overall, the Decision Tree model offers a starting point for reorder predictions, with potential for refinement through optimization.

- **Logistic Regression Results**:
  - Accuracy: 0.706,
  - Precision: 0.7029,
  - Recall: 0.706, F1
  - Score: 0.7009

Analysis: The Logistic Regression model performs strongly with an accuracy of 70.6%, indicating its ability to predict reordered products accurately 70.6% of the time. Precision is at 70.29%, meaning it is correct about 70.29% of the time when predicting reorders. The model's recall is also high at 70.6%, indicating its ability to identify 70.6% of actual reorders. The F1 Score, balancing precision and recall, is 70.09%.

These metrics demonstrate the Logistic Regression model's effectiveness in predicting reordered products. With high accuracy, precision, recall, and F1 Score, it outperforms the

Decision Tree. Managers can rely on this model for accurate reorder predictions. Continued monitoring and potential optimization strategies can further enhance its performance.

- **K-Nearest Neighbors (KNN) Results**:
  - Accuracy: 0.588,
  - Precision: 0.5793,
  - Recall: 0.588,
  - F1 Score: 0.5808

Analysis: The K-Nearest Neighbors (KNN) model shows moderate performance with an accuracy of 58.8%, indicating its ability to predict reordered products correctly 58.8% of the time. Precision, measuring the accuracy of positive predictions, is 57.93%. This means that when the model predicts a product will be reordered, it is correct about 57.93% of the time.

Additionally, the model's recall, which indicates its ability to find all the reordered products, is also 58.8%. This suggests that the model effectively identifies 58.8% of the products that were actually reordered. The F1 Score, a balanced measure of precision and recall, stands at 58.08%.

While the KNN model provides a baseline performance, its metrics are lower compared to the Logistic Regression model. Managers should consider these scores when deciding on the model to use for reorder predictions. Continued monitoring and potential optimization strategies can be explored to improve its performance.

- **Support Vector Machine (SVM) Results**:
  - Accuracy: 0.687,
  - Precision: 0.6839,
  - Recall: 0.687,
  - F1 Score: 0.6844

Analysis The Support Vector Machine (SVM) model performs decently with an accuracy of 68.7%, indicating its ability to predict reordered products accurately 68.7% of the time. Precision is at 68.39%, meaning it is correct about 68.39% of the time when predicting reorders. The model's recall is also 68.7%, indicating its ability to identify 68.7% of actual reorders. The F1 Score, balancing precision and recall, is 68.44%.

While the SVM model provides reasonable performance, its metrics are slightly lower compared to the Logistic Regression model. Continued monitoring and potential optimization strategies can further enhance its performance.

- **Implications**:
  - **Logistic Regression** outperforms Decision Tree, KNN, and SVM in terms of all metrics.
  - **Decision Tree** provides baseline performance.
  - **KNN** performs slightly lower than Decision Tree.
  - **SVM** shows good performance, slightly below Logistic Regression.
  - **Logistic Regression** is recommended for its highest accuracy, precision, recall, and F1 Score.

Ensemble Learning: Classification

Decision Tree vs Random Forest

- **Decision Tree Results**:
  - Accuracy: 0.614, Precision: 0.6148, Recall: 0.614, F1 Score: 0.6144
- **Random Forest Results**:
  - Accuracy: 0.707, Precision: 0.7043, Recall: 0.707, F1 Score: 0.7005
- **Analysis**:
  - **Random Forest** outperforms Decision Tree in all metrics, showing significant improvement.
  - **Decision Tree** provides baseline performance.
  - **Random Forest** is recommended for its higher accuracy, precision, recall, and F1 Score.

Decision Tree vs KNN

- **Decision Tree Results**:
  - Accuracy: 0.614, Precision: 0.6148, Recall: 0.614, F1 Score: 0.6144
- **KNN Results**:
  - Accuracy: 0.588, Precision: 0.5793, Recall: 0.588, F1 Score: 0.5808
- **Analysis**:
  - **Decision Tree** outperforms KNN in all metrics.
  - **Decision Tree** provides baseline performance.
  - **Decision Tree** is recommended over KNN.

Decision Tree vs Logistic Regression

- **Decision Tree Results**:
  - Accuracy: 0.614, Precision: 0.6148, Recall: 0.614, F1 Score: 0.6144
- **Logistic Regression Results**:
  - Accuracy: 0.706, Precision: 0.7029, Recall: 0.706, F1 Score: 0.7009
- **Analysis**:
  - **Logistic Regression** outperforms Decision Tree in all metrics.
  - **Decision Tree** provides baseline performance.
  - **Logistic Regression** is recommended for its higher accuracy, precision, recall, and F1 Score.

Decision Tree vs Support Vector Machine

- **Decision Tree Results**:
  - Accuracy: 0.614, Precision: 0.6148, Recall: 0.614, F1 Score: 0.6144
- **Support Vector Machine (SVM) Results**:
  - Accuracy: 0.687, Precision: 0.6839, Recall: 0.687, F1 Score: 0.6844
- **Analysis**:
  - **Decision Tree** outperforms SVM in all metrics.
  - **Decision Tree** provides baseline performance.
  - **Decision Tree** is recommended over SVM.

Observations:

- Based on the analysis, the **Logistic Regression** model stands out as the top-performing model for predicting reordered products.
- **Random Forest** also shows significant improvement over the **Decision Tree**, making it a strong alternative.
- **Decision Tree** provides a baseline performance and could serve as a simple model for initial predictions.
- For more complex and accurate predictions, **Logistic Regression** is recommended due to its highest accuracy, precision, recall, and F1 Score.
- **Support Vector Machine (SVM)** performs reasonably well but slightly lower than Logistic Regression.
- **K-Nearest Neighbors (KNN)** shows the lowest performance among the models compared.
- Further model evaluation, such as cross-validation and hyperparameter tuning, could be performed to ensure the robustness of the chosen model.
- The choice of the final model should also consider factors such as computational resources, interpretability, and specific business

Managerial Insights:

1. **Feature Importance**:
   - The analysis of the models can provide insights into the features that are most important for predicting reordered products.
   - Managers can use this information to focus on key factors that influence customers' reorder behavior.
   - Features such as `order_number`, `order_hour_of_day`, and `days_since_prior_order` are likely important predictors based on the model results.
2. **Customer Segmentation**:
   - Clustering analysis (K-Means) can help identify different segments of customers based on their ordering behavior.
   - This segmentation can be utilized for targeted marketing strategies.
   - For example, customers in clusters with high reorder rates can be targeted with loyalty programs or personalized recommendations.
3. **Model Selection**:
   - Logistic Regression and Random Forest are the top-performing models for predicting reordered products.
   - Managers can choose Logistic Regression for its higher accuracy, precision, recall, and F1 Score.
   - Random Forest, while more complex, also provides significant improvement over the Decision Tree and could be considered for accurate predictions.
4. **Optimizing Product Placement**:
   - The `department` and `product_name` features can be used to optimize product placement in online platforms.
   - Products that are frequently reordered together can be strategically placed on the website to increase visibility and encourage additional purchases.

5. **Supply Chain Management**:
   - Predictive models can assist in inventory management and supply chain optimization.
   - By predicting which products are more likely to be reordered, companies can ensure adequate stock levels to meet customer demand.
   - This can lead to improved customer satisfaction and reduced out-of-stock situations.
6. **Marketing Strategies**:
   - Insights from the models can guide marketing campaigns, promotions, and product recommendations.
   - For example, products with high reorder rates can be promoted more aggressively.
   - Personalized recommendations based on customers' past orders can improve engagement and increase sales.
7. **Operational Efficiency**:
   - Understanding reorder patterns can help in streamlining operations.
   - For instance, optimizing delivery schedules based on predicted reorder timings can improve efficiency and reduce delivery costs.
   - It can also aid in workforce planning, ensuring adequate staff during peak ordering times.
8. **Customer Retention**:
   - By predicting which customers are more likely to reorder, companies can focus on retaining these customers.
   - Tailored loyalty programs or discounts can be offered to encourage repeat purchases.
   - Customer feedback and reviews from these segments can also be valuable for product improvements.
9. **Continuous Monitoring and Improvement**:
   - The chosen model, Logistic Regression, should be continuously monitored and evaluated for performance.
   - Regular updates to the model with new data can improve its accuracy and relevance.
   - Managers should also consider re-evaluating the model periodically to ensure it aligns with changing customer behavior and market trends.
10. **Interpretation and Communication**:
   - It's crucial to interpret the model results in a business context and communicate findings effectively.
   - Managers should ensure that insights from the models are clearly understood and actionable.
   - Collaboration between data scientists and business stakeholders is essential for successful implementation of the model's recommendations.

By leveraging the insights from these models, businesses can make data-driven decisions to improve customer satisfaction, optimize operations, and enhance overall performance in the e-commerce space.

```
import os
import pandas as pd
import numpy as np

# Import & Read Dataset
data = pd.read_csv('ECommerce Consumer Behavior DataSet.csv')

# Display Dataset Information
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1167084 entries, 0 to 1167083
Data columns (total 12 columns):
 #   Column                  Non-Null Count    Dtype  
---  ------                  --------------    -----  
 0   order_id                1167084 non-null  int64  
 1   user_id                 1167084 non-null  int64  
 2   order_number            1167083 non-null  float64
 3   order_dow               1167083 non-null  float64
 4   order_hour_of_day       1167083 non-null  float64
 5   days_since_prior_order  1095235 non-null  float64
 6   product_id              1167083 non-null  float64
 7   add_to_cart_order       1167083 non-null  float64
 8   reordered               1167083 non-null  float64
 9   department_id           1167083 non-null  float64
 10  department              1167083 non-null  object 
 11  product_name            1167083 non-null  object 
dtypes: float64(8), int64(2), object(2)
memory usage: 106.8+ MB

data.columns

Index(['order_id', 'user_id', 'order_number', 'order_dow',
'order_hour_of_day',
       'days_since_prior_order', 'product_id', 'add_to_cart_order',
       'reordered', 'department_id', 'department', 'product_name'],
      dtype='object')

data.head()

{"type":"dataframe","variable_name":"data"}

# Index Variable(s)
index_variables = data.index.names if data.index.names else None

# Variables or Features having Categories | Categorical Variables or
Features (CV)
categorical_variables =
data.select_dtypes(include=['object']).columns.tolist()

# Variables or Features having Nominal Categories | Categorical
```

```
Variables or Features- Nominal Type
nominal_categorical_variables = [col for col in categorical_variables
if data[col].nunique() > 2]

# Variables or Features having Ordinal Categories | Categorical
Variables or Features- Ordinal Type
ordinal_categorical_variables = [col for col in categorical_variables
if data[col].nunique() <= 2]

# Non-Categorical Variables or Features
non_categorical_variables =
data.select_dtypes(exclude=['object']).columns.tolist()

print("Index Variable(s):", index_variables if index_variables else
"None")
print("Variables or Features having Categories | Categorical Variables
or Features (CV):", categorical_variables)
print("Variables or Features having Nominal Categories | Categorical
Variables or Features- Nominal Type:", nominal_categorical_variables)
print("Variables or Features having Ordinal Categories | Categorical
Variables or Features- Ordinal Type:", ordinal_categorical_variables)
print("Non-Categorical Variables or Features:",
non_categorical_variables)
```

```
Index Variable(s): [None]
Variables or Features having Categories | Categorical Variables or
Features (CV): ['department', 'product_name']
Variables or Features having Nominal Categories | Categorical
Variables or Features- Nominal Type: ['department', 'product_name']
Variables or Features having Ordinal Categories | Categorical
Variables or Features- Ordinal Type: []
Non-Categorical Variables or Features: ['order_id', 'user_id',
'order_number', 'order_dow', 'order_hour_of_day',
'days_since_prior_order', 'product_id', 'add_to_cart_order',
'reordered', 'department_id']
```

```
# Sample 5000 random records from the dataset
sampled_data = data.sample(n=5000, random_state=45037)

sampled_data.describe()
```


{"summary":"{\n \"name\": \"sampled_data\",\n \"rows\": 8,\n
\"fields\": [\n   {\n   \"column\": \"order_id\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1197123.4511992626,\n   \"min\": 1497.0,\n   \"max\":
3420584.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\
n   1708417.4516,\n   1693084.5,\n   5000.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"user_id\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":

```
71109.08212986271,\n        \"min\": 42.0,\n        \"max\":
206182.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
102040.9134,\n          101560.5,\n          5000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"order_number\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1759.1820225554309,\n        \"min\": 1.0,\n        \"max\": 5000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          17.2514,\
n          11.0,\n          5000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"order_dow\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1766.81809646796,\n        \"min\": 0.0,\n        \"max\": 5000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          2.7118,\n
2.0,\n          5000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"order_hour_of_day\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1763.7632013540579,\n        \"min\":
0.0,\n        \"max\": 5000.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          13.342,\n          13.0,\n          5000.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"days_since_prior_order\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1653.5253773797076,\n        \"min\": 0.0,\n        \"max\": 4688.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
11.385878839590443,\n          8.0,\n          4688.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"product_id\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1744.8011976630162,\n        \"min\": 1.0,\n        \"max\": 5000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          71.8726,\
n          83.0,\n          5000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"add_to_cart_order\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1761.4738037821332,\n        \"min\": 1.0,\n        \"max\": 5000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          8.4838,\n
7.0,\n          5000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"reordered\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1767.5611270890752,\n        \"min\":
0.0,\n        \"max\": 5000.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.583,\n          1.0,\n
0.4931121899997862\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"department_id\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1764.3762176633318,\n        \"min\":
1.0,\n        \"max\": 5000.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          10.017,\n          9.0,\n          5000.0\n
```

],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# Step 1: Handling Missing Values

# Identify numerical and categorical columns
numerical_cols = sampled_data.select_dtypes(include=['int64',
'float64']).columns
categorical_cols =
sampled_data.select_dtypes(include=['object']).columns

# Fill missing values
for col in numerical_cols:
    sampled_data[col].fillna(sampled_data[col].median(), inplace=True)

for col in categorical_cols:
    sampled_data[col].fillna(sampled_data[col].mode()[0],
inplace=True)

# Step 2: Data Type Correction
# Convert numerical columns to the appropriate type and categorical
columns to 'category' type
for col in numerical_cols:
    sampled_data[col] = pd.to_numeric(sampled_data[col],
errors='coerce')

for col in categorical_cols:
    sampled_data[col] = sampled_data[col].astype('category')

sampled_data_info = sampled_data.info()

sampled_data_info
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5000 entries, 1047430 to 328507
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   order_id                5000 non-null   int64
 1   user_id                 5000 non-null   int64
 2   order_number            5000 non-null   float64
 3   order_dow               5000 non-null   float64
 4   order_hour_of_day       5000 non-null   float64
 5   days_since_prior_order  5000 non-null   float64
 6   product_id              5000 non-null   float64
 7   add_to_cart_order       5000 non-null   float64
 8   reordered               5000 non-null   float64
 9   department_id           5000 non-null   float64
 10  department              5000 non-null   category
 11  product_name            5000 non-null   category
```

```
dtypes: category(2), float64(8), int64(2)
memory usage: 450.1 KB
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

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

# Define numerical and categorical columns
numerical_cols = X.select_dtypes(include=['int64',
'float64']).columns.tolist()
categorical_cols =
X.select_dtypes(include=['object']).columns.tolist()

# Define the transformers for the numerical and categorical columns
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

# Create the preprocessor with ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ]
)

# Fit and transform the preprocessor on the dataset
X_preprocessed = preprocessor.fit_transform(X)

# Identify numerical columns in the dataset
numerical_features = sampled_data.select_dtypes(include=['int64',
'float64']).columns

# Select all numerical features except the first 5 for clustering
selected_features = numerical_features[2:5].tolist()  # Change this
based on feature selection logic

selected_features
```

```
['order_number', 'order_dow', 'order_hour_of_day']
```

```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Extract the selected features for clustering
clustering_data = sampled_data[selected_features]

# Standardize the features
```

```python
scaler = StandardScaler()
clustering_scaled = scaler.fit_transform(clustering_data)

# Perform K-Means clustering with k = 2, 3, 4, 5
k_values = [2, 3, 4, 5]
kmeans_results = {}

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=45000)
    kmeans.fit(clustering_scaled)
    kmeans_results[k] = kmeans.labels_

# Show the first 10 cluster assignments for each k
{k: labels[:10] for k, labels in kmeans_results.items()}
```

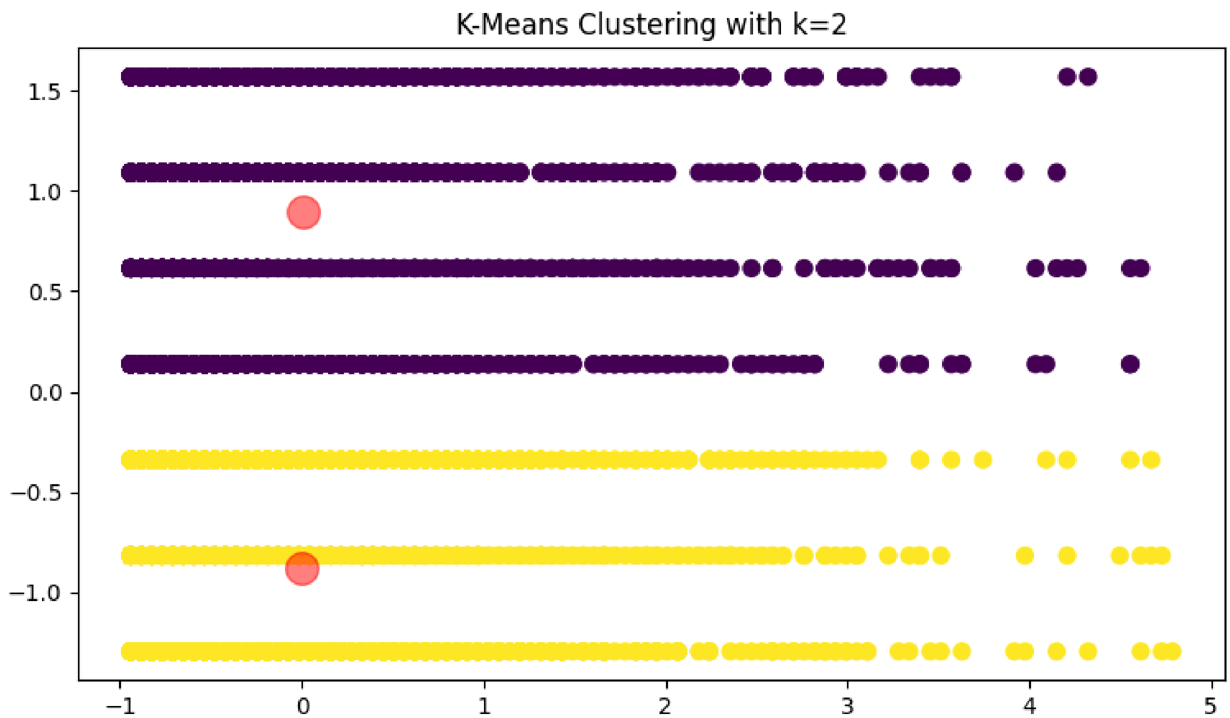
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
```

```
{2: array([1, 0, 1, 1, 1, 0, 0, 0, 1, 0], dtype=int32),
 3: array([1, 0, 1, 2, 1, 0, 0, 2, 1, 0], dtype=int32),
 4: array([0, 1, 3, 2, 0, 1, 1, 2, 0, 1], dtype=int32),
 5: array([1, 0, 3, 2, 1, 0, 0, 2, 1, 0], dtype=int32)}
```

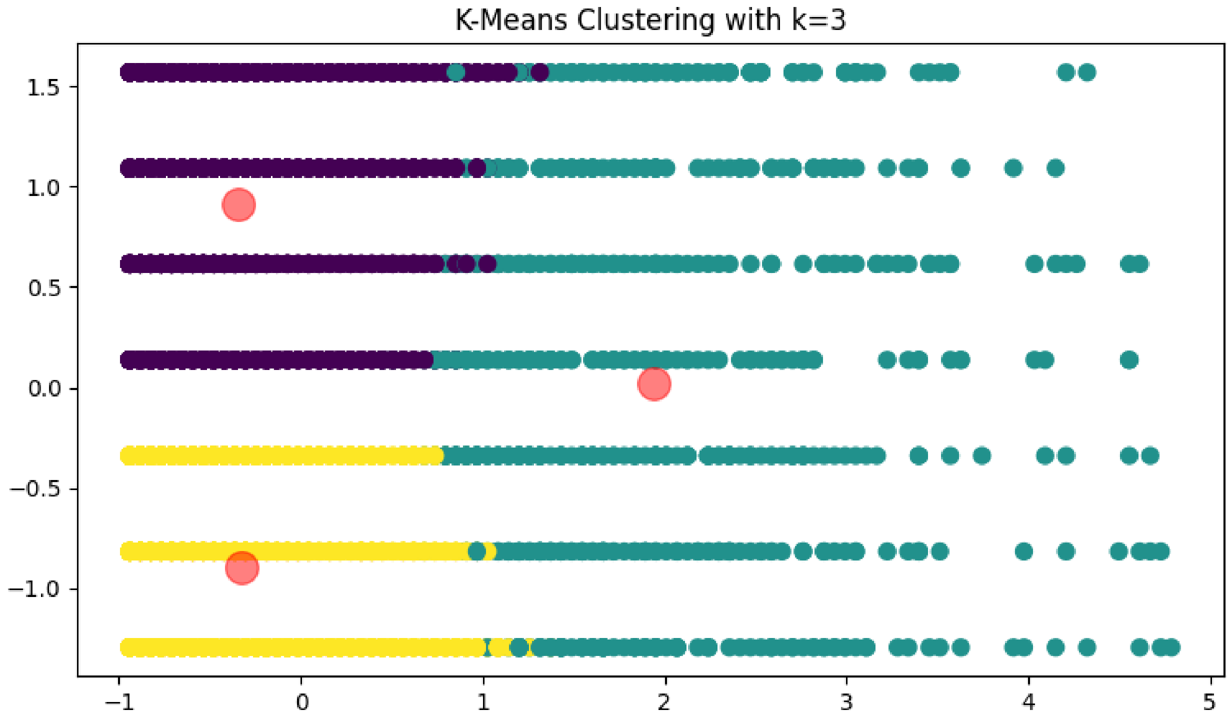
```python
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score, davies_bouldin_score

# Define a function to perform clustering and visualize the results
def cluster_and_evaluate(data, k_values):
    for k in k_values:
        kmeans = KMeans(n_clusters=k, random_state=45037)
        labels = kmeans.fit_predict(data)
```

```
        # Calculate silhouette and Davies-Bouldin scores
        silhouette_avg = silhouette_score(data, labels)
        davies_bouldin_avg = davies_bouldin_score(data, labels)

        print(f"For k={k}, the Silhouette Score is:
{silhouette_avg:.4f}")
        print(f"For k={k}, the Davies-Bouldin Score is:
{davies_bouldin_avg:.4f}")

        # Visualize the clusters
        plt.figure(figsize=(9, 5))
        plt.scatter(data[:, 0], data[:, 1], c=labels, s=50,
cmap='viridis')
        centers = kmeans.cluster_centers_
        plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200,
alpha=0.5)
        plt.title(f'K-Means Clustering with k={k}')
        plt.show()

# Run the clustering and evaluation for the defined k values
cluster_and_evaluate(clustering_scaled, k_values)
```

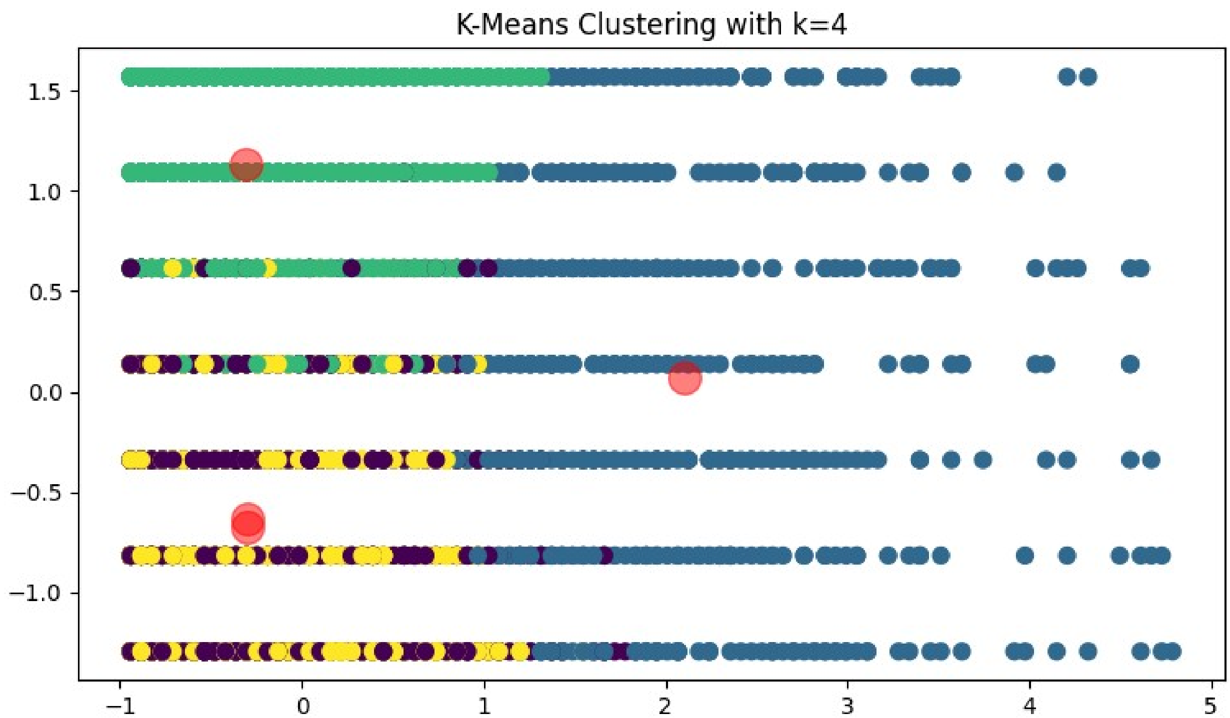
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(

For k=2, the Silhouette Score is: 0.2815
For k=2, the Davies-Bouldin Score is: 1.5017
```



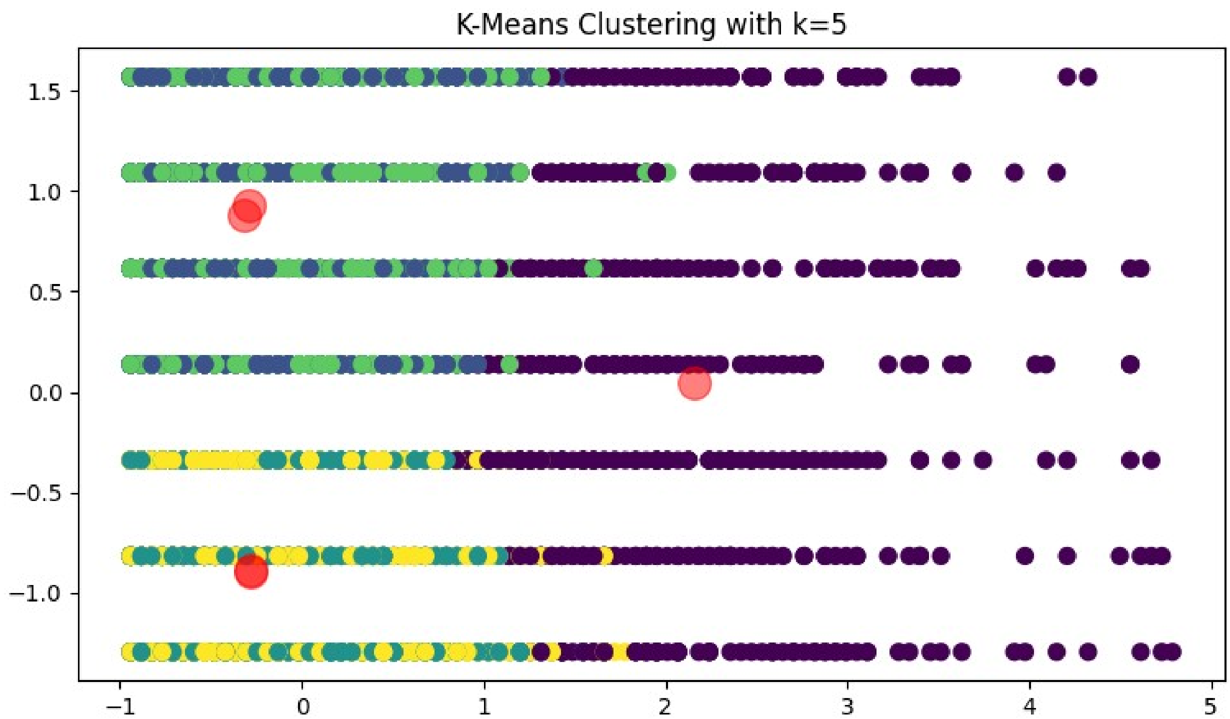

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(

For k=3, the Silhouette Score is: 0.3146
For k=3, the Davies-Bouldin Score is: 1.1746
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(

For k=4, the Silhouette Score is: 0.3057
For k=4, the Davies-Bouldin Score is: 1.0096
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(

For k=5, the Silhouette Score is: 0.3008
For k=5, the Davies-Bouldin Score is: 1.0076
```

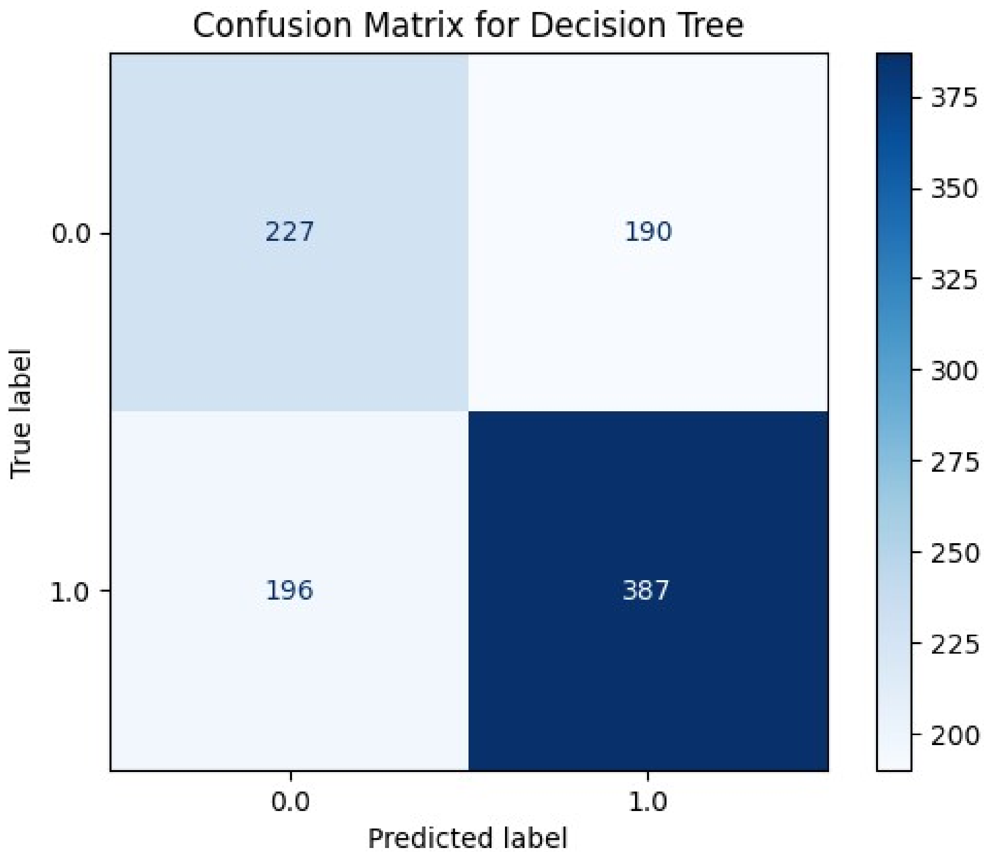


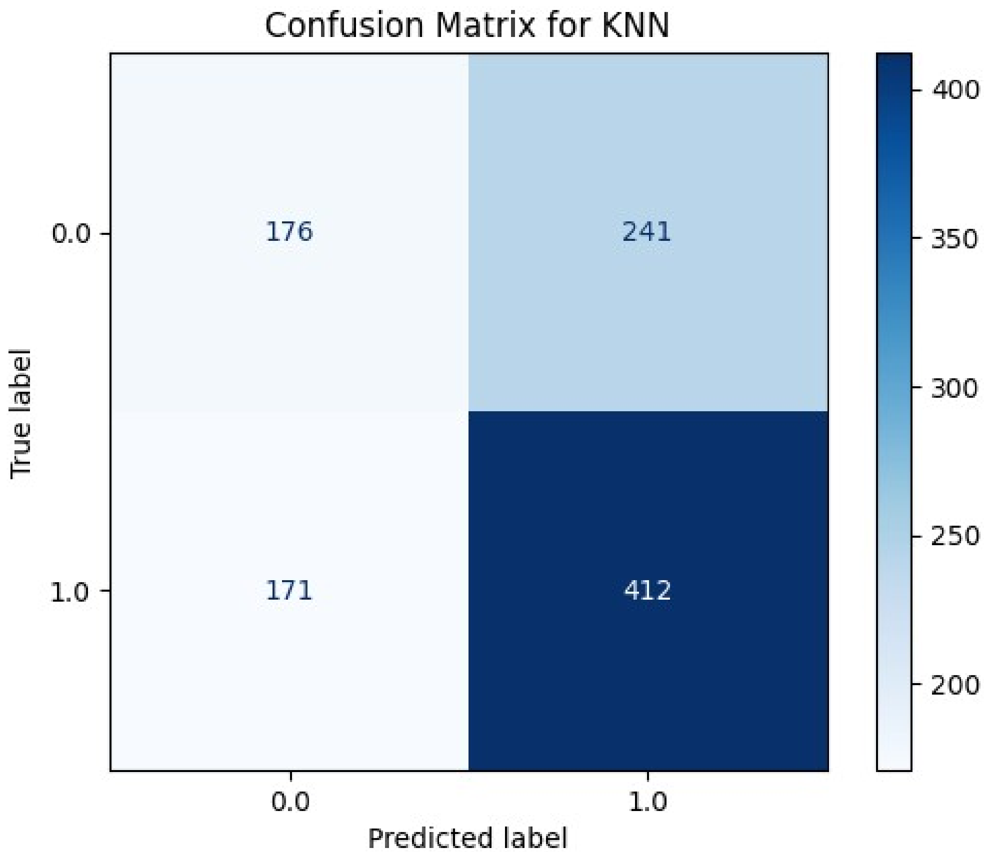

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve, auc
import numpy as np

# Split the preprocessed data into training and testing sets with
stratified sampling
X_train, X_test, y_train, y_test = train_test_split(
    X_preprocessed, y, test_size=0.20, random_state=45000, stratify=y)

# Initialize the models
decision_tree = DecisionTreeClassifier(random_state=45000)
knn = KNeighborsClassifier()

# Train the models
decision_tree.fit(X_train, y_train)
knn.fit(X_train, y_train)

# Predict on the testing set
y_pred_dt = decision_tree.predict(X_test)
y_pred_knn = knn.predict(X_test)

# Calculate the metrics
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt, average='weighted')
```

```python
recall_dt = recall_score(y_test, y_pred_dt, average='weighted')
f1_dt = f1_score(y_test, y_pred_dt, average='weighted')

accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn,
average='weighted')
recall_knn = recall_score(y_test, y_pred_knn, average='weighted')
f1_knn = f1_score(y_test, y_pred_knn, average='weighted')

# Prepare the results
results = {
    'Decision Tree': {
        'Accuracy': accuracy_dt,
        'Precision': precision_dt,
        'Recall': recall_dt,
        'F1 Score': f1_dt
    },
    'KNN': {
        'Accuracy': accuracy_knn,
        'Precision': precision_knn,
        'Recall': recall_knn,
        'F1 Score': f1_knn
    }
}

results
```

```
{'Decision Tree': {'Accuracy': 0.614,
  'Precision': 0.6148044052755142,
  'Recall': 0.614,
  'F1 Score': 0.6143802955665024},
 'KNN': {'Accuracy': 0.588,
  'Precision': 0.5793389322612108,
  'Recall': 0.588,
  'F1 Score': 0.5807923211169284}}
```

```python
from sklearn.metrics import ConfusionMatrixDisplay

# Function to plot confusion matrix using ConfusionMatrixDisplay
def plot_confusion_matrix_for_model(model, X_test, y_test, title):
    disp = ConfusionMatrixDisplay.from_estimator(model, X_test,
y_test, cmap=plt.cm.Blues)
    disp.ax_.set_title(f'Confusion Matrix for {title}')
    plt.show()

# Plot confusion matrices and ROC curves for both models
plot_confusion_matrix_for_model(decision_tree, X_test, y_test,
'Decision Tree')
plot_confusion_matrix_for_model(knn, X_test, y_test, 'KNN')
```

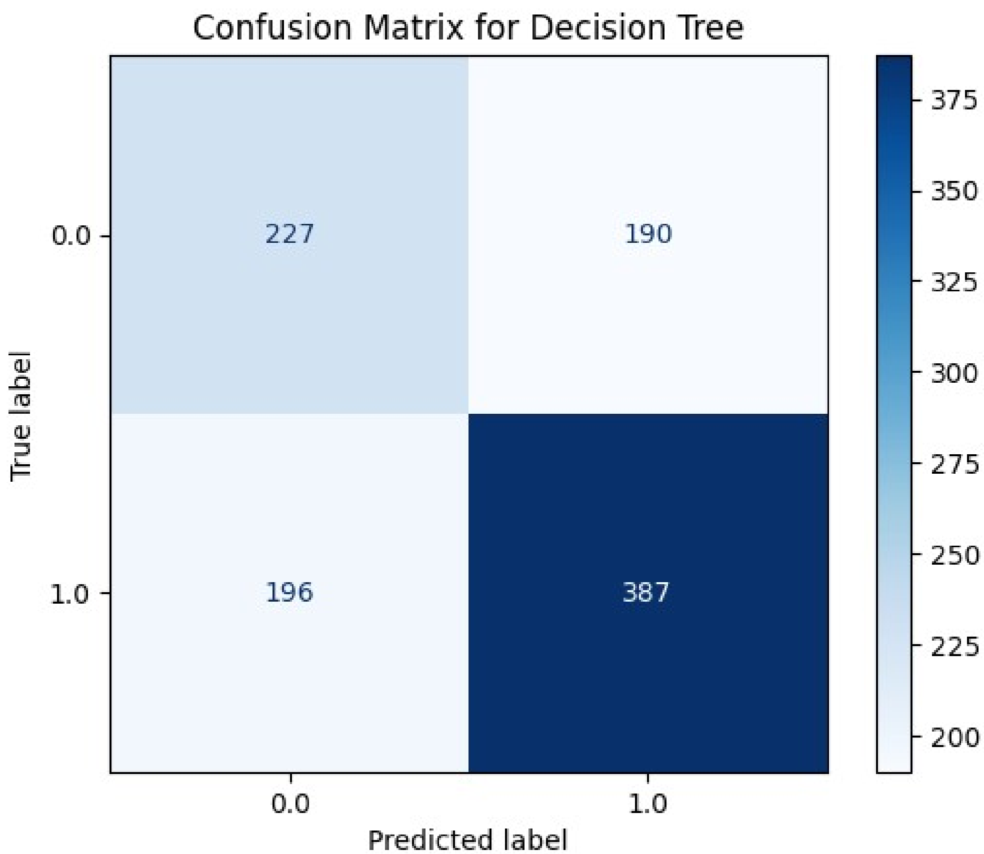

Confusion Matrix for Decision Tree
True label
Predicted label
0.0
1.0
0.0
1.0
227
190
196
387
375
350
325
300
275
250
225
200

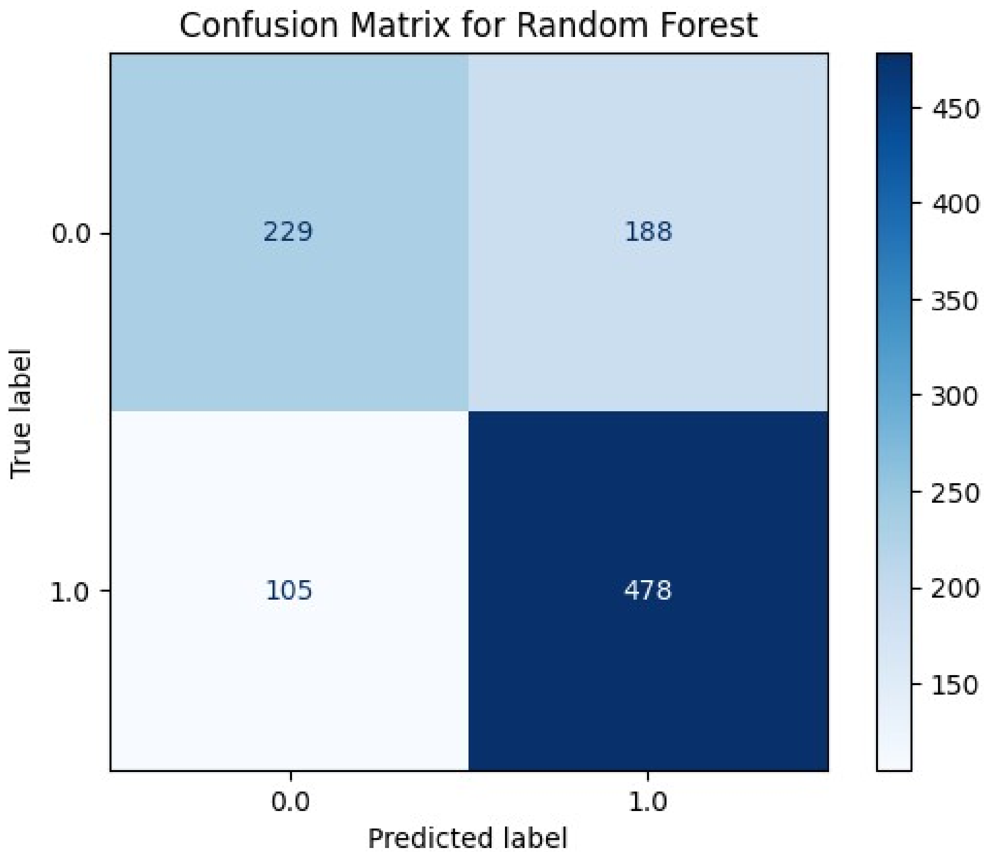



```python
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
random_forest = RandomForestClassifier(random_state=45000)

# Train the Random Forest model
random_forest.fit(X_train, y_train)

# Predict on the testing set
y_pred_rf = random_forest.predict(X_test)

# Calculate the metrics for Random Forest
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf, average='weighted')
recall_rf = recall_score(y_test, y_pred_rf, average='weighted')
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')

# Prepare the results for Random Forest
results_rf = {
    'Accuracy': accuracy_rf,
    'Precision': precision_rf,
    'Recall': recall_rf,
    'F1 Score': f1_rf
```

```
}

# Display the results
print("Decision Tree Results:", results['Decision Tree'])
print("Random Forest Results:", results_rf)

# Plot confusion matrices and ROC curves for both models
plot_confusion_matrix_for_model(decision_tree, X_test, y_test,
'Decision Tree')
plot_confusion_matrix_for_model(random_forest, X_test, y_test, 'Random
Forest')
```

```
Decision Tree Results: {'Accuracy': 0.614, 'Precision':
0.6148044052755142, 'Recall': 0.614, 'F1 Score': 0.6143802955665024}
Random Forest Results: {'Accuracy': 0.707, 'Precision':
0.7043366150581719, 'Recall': 0.707, 'F1 Score': 0.700544309748731}
```





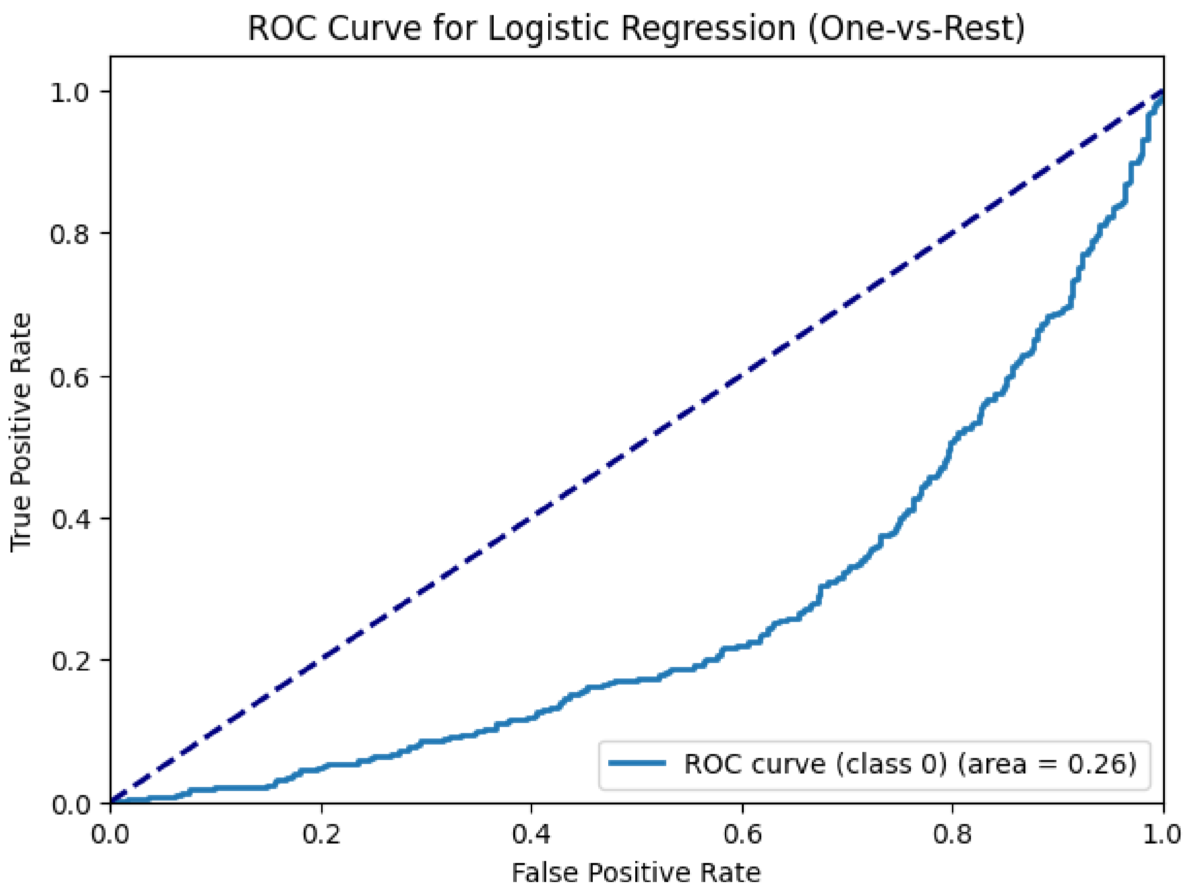

```python
from sklearn.linear_model import LogisticRegression

# Initialize the logistic regression model
logistic_regression = LogisticRegression(random_state=45037)

# Train the logistic regression model
logistic_regression.fit(X_train, y_train)

# Predict on the testing set using logistic regression
y_pred_lr = logistic_regression.predict(X_test)

# Calculate the metrics for logistic regression
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr, average='weighted')
recall_lr = recall_score(y_test, y_pred_lr, average='weighted')
f1_lr = f1_score(y_test, y_pred_lr, average='weighted')

# Update the results dictionary with logistic regression metrics
results['Logistic Regression'] = {
    'Accuracy': accuracy_lr,
    'Precision': precision_lr,
    'Recall': recall_lr,
    'F1 Score': f1_lr
```

```
}

print(results)
```

```
{'Decision Tree': {'Accuracy': 0.614, 'Precision': 0.6148044052755142,
'Recall': 0.614, 'F1 Score': 0.6143802955665024}, 'KNN': {'Accuracy':
0.588, 'Precision': 0.5793389322612108, 'Recall': 0.588, 'F1 Score':
0.5807923211169284}, 'Logistic Regression': {'Accuracy': 0.706,
'Precision': 0.7029162941158299, 'Recall': 0.706, 'F1 Score':
0.7008566563310121}}
```

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize

# Convert multiclass labels to binary labels
y_test_bin = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_bin.shape[1]

# Initialize the logistic regression model with OvR strategy
logistic_regression_ovr =
OneVsRestClassifier(LogisticRegression(random_state=45037))

# Train the logistic regression model with OvR strategy
logistic_regression_ovr.fit(X_train, y_train)

# Predict probabilities for each class using OvR logistic regression
y_score_lr_ovr = logistic_regression_ovr.predict_proba(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_lr_ovr[:,
i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'ROC curve (class {i}) (area
= {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression (One-vs-Rest)')
```

```
plt.legend(loc="lower right")
plt.show()
```

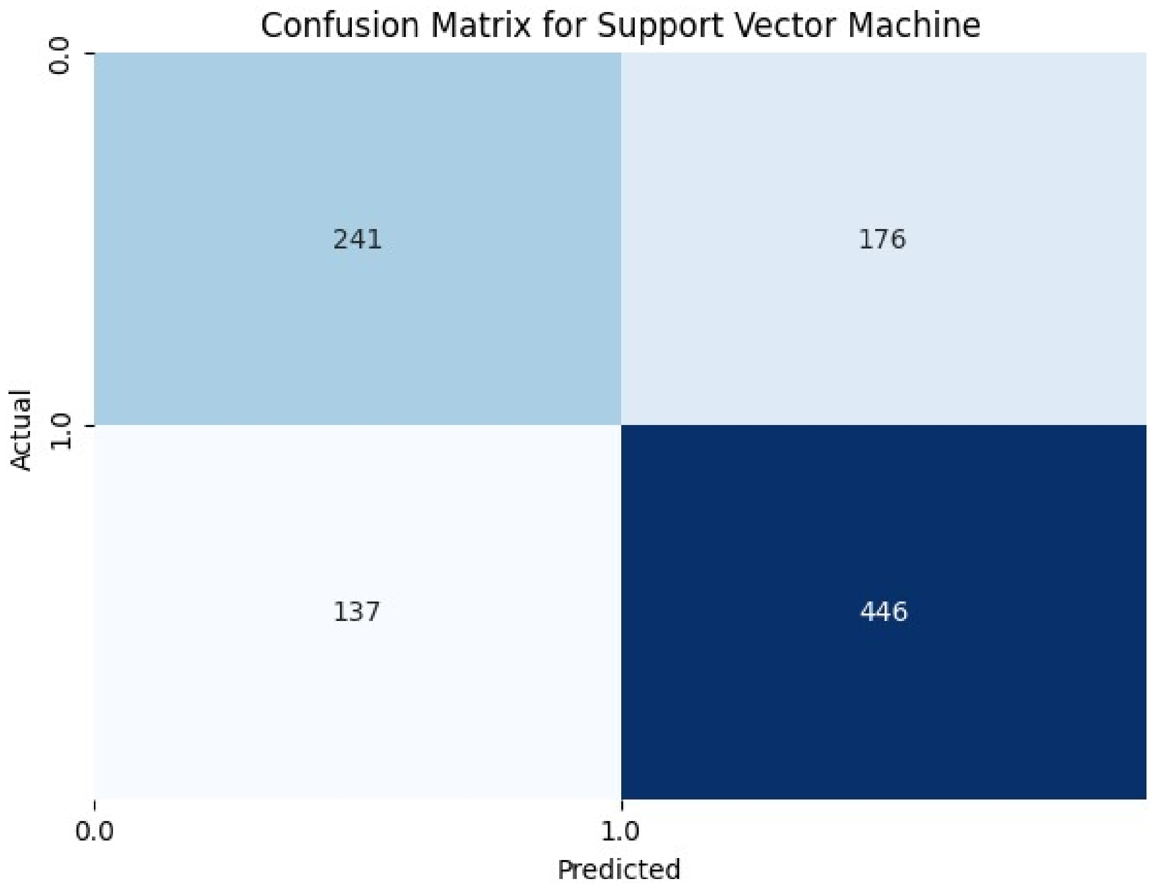



```
from sklearn.svm import SVC

# Initialize the Support Vector Machine model
svm = SVC(random_state=45000)

# Train the SVM model
svm.fit(X_train, y_train)

# Predict on the testing set using SVM
y_pred_svm = svm.predict(X_test)

# Calculate the metrics for SVM
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm,
average='weighted')
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')
f1_svm = f1_score(y_test, y_pred_svm, average='weighted')

# Update the results dictionary with SVM metrics
```

```python
results['Support Vector Machine'] = {
    'Accuracy': accuracy_svm,
    'Precision': precision_svm,
    'Recall': recall_svm,
    'F1 Score': f1_svm
}

print(results)
```

```
{'Decision Tree': {'Accuracy': 0.614, 'Precision': 0.6148044052755142,
'Recall': 0.614, 'F1 Score': 0.6143802955665024}, 'KNN': {'Accuracy':
0.588, 'Precision': 0.5793389322612108, 'Recall': 0.588, 'F1 Score':
0.5807923211169284}, 'Logistic Regression': {'Accuracy': 0.706,
'Precision': 0.7029162941158299, 'Recall': 0.706, 'F1 Score':
0.7008566563310121}, 'Support Vector Machine': {'Accuracy': 0.687,
'Precision': 0.6839004491399989, 'Recall': 0.687, 'F1 Score':
0.6843877867376498}}
```

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Compute confusion matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)

# Plot confusion matrix
plt.figure(figsize=(7, 5))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix for Support Vector Machine')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.xticks(ticks=np.arange(len(np.unique(y))), labels=np.unique(y))
plt.yticks(ticks=np.arange(len(np.unique(y))), labels=np.unique(y))
plt.show()
```


Confusion Matrix for Support Vector Machine
241
176
137
446
0.0
1.0
Actual
0.0
1.0
Predicted