mlm-1-045025

April 13, 2024

** Project Report**

Description of Data:

Categorical Variables: Date: Date of the recorded weather data. Location: Location where the weather data was recorded. WindGustDir: Direction of the wind gust at some point in time. WindDir9am: Wind direction at 9 am. RainToday: Whether it rained today. RainTomorrow: The target variable indicating whether it will rain tomorrow.

Quantitative Variables: MinTemp: Minimum temperature recorded. MaxTemp: Maximum temperature recorded. Rainfall: Amount of rainfall recorded. Evaporation: Amount of evaporation recorded. Sunshine: Duration of sunshine recorded. WindGustSpeed: Speed of the wind gust. WindSpeed9am: Wind speed at 9 am. WindSpeed3pm: Wind speed at 3 pm. Humidity9am: Humidity at 9 am. Humidity3pm: Humidity at 3 pm. Pressure9am: Atmospheric pressure at 9 am. Pressure3pm: Atmospheric pressure at 3 pm. Cloud9am: Cloud cover at 9 am. Cloud3pm: Cloud cover at 3 pm. Temp9am: Temperature at 9 am. Temp3pm: Temperature at 3 pm.

1.Description of Data: This dataset contains about 10 years of daily weather observations from many locations across Australia. It has a total of 145,460 rows and 23 columns. The columns are briefly stated as MaxTemp, Rainfall,Evaporation,Sunshine, WindGustSpeed, WindSpeed9am, WindSpeed3pm, Humidity9am, Humidity3pm, Pressure9am, Pressure3pm, Cloud9am, Cloud3pm,Temp9am,Temp3pm. The data has been cleaned.

2.Objective(s) of Data Analysis: The primary objective of preprocessing was to prepare the dataset for insightful analyses and predictive modeling. The focus was to standardize the numerical data and convert categorical variables into a machine-learning-friendly format to facilitate accurate and efficient analyses.

3.Observations on Data Analysis: During preprocessing, numerical variables were scaled and standardized, and categorical variables were transformed using OneHotEncoding, resulting in an expanded feature set. The transformation was carefully executed to maintain the integrity of the data, ensuring that models built on this data would be well-equipped to discern underlying patterns and make accurate predictions.

4.Managerial Insights: The preprocessing undertaken is fundamental for any data-driven organization aiming to leverage analytics for strategic decisions. With a clean and well-prepared dataset, managers can deploy models that forecast customer credit scores with higher confidence, identify key drivers of financial behavior, and tailor their credit risk strategies more precisely. This groundwork is critical for minimizing risk, optimizing marketing campaigns, and enhancing customer relationship management by providing a clear view of the customer base.

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

1.Description of Data: The dataset utilized for K-Means clustering and the analysis was conducted using five selected features deemed significant for distinguishing customer groupings based on their financial behaviors and credit history.

2.Objective(s) of Data Analysis: The aim was to uncover natural groupings within the customer base that could provide insights into different consumer behaviors and financial profiles. These clusters have the potential to reveal segments that share common characteristics, without the need for predefined labels.

3.Observations on Data Analysis: K-Means clustering was executed with kk values of 2, 3, 4, and 5 to determine the optimal number of clusters.The Silhouette Score measures how similar an object is to its own cluster compared to other clusters. A higher Silhouette Score indicates better-defined clusters, with values ranging from -1 to 1. A score closer to 1 suggests that the clusters are well-separated. For the dataset used the Silhouette Score is 0.3106 for k=5, indicating a reasonable degree of separation between clusters, while the Davies-Bouldin Score which is used to measure the average similarity between each cluster is 1.0275, which suggests that the clusters are reasonably distinct, although not perfectly separated.

4.Managerial Insights: The clustering results provide managers with valuable insights into customer segmentation and behavior, guiding strategic decision-making processes. With a moderate Silhouette Score of 0.3106, the clusters exhibit discernible differences, suggesting distinct customer groups. However, the Davies-Bouldin Score of 1.0275 indicates some overlap or ambiguity in cluster boundaries, necessitating cautious interpretation and potential refinement of the clustering approach. Nonetheless, leveraging these clusters will enable organizations to tailor their strategies and interventions to specific customer segments, enhancing the effectiveness of targeted initiatives. By understanding the characteristics and behaviors of customers within each cluster, managers can develop personalized marketing efforts, optimize resource allocation, and improve customer engagement and satisfaction. Ultimately, these insights support data-driven decision-making, empowering organizations to meet the diverse needs and preferences of their customer base while driving business growth and profitability.

Supervised Learning: Classification - Decision Tree vs {Logistic Regression | K-Nearest Neighbor | Support Vector Machine }

Ensemble Learning: Classification - Decision Tree vs Random Forest

1.Description of Data The dataset utilized for Decision Tree and Random Forest clustering includes a subset of 5,000 records from the original data. The analysis was conducted using five selected features deemed significant for distinguishing customer groupings based on their financial behaviors and credit history.

2.Objectives of Comparing Decision Tree and Random Forest Performance The primary objectives of comparing Decision Tree and Random Forest performance in this context are: Evaluate the effectiveness of each algorithm for customer classification. This involves assessing their ability to accurately predict customer behavior and financial outcomes based on their characteristics. Identify the algorithm that provides the most accurate and robust results. By comparing metrics like accuracy, precision, recall, and F1-score, we can determine which algorithm delivers superior classification performance. Gain insights into the trade-offs between interpretability and accuracy. Decision Trees offer clearer decision rules, while Random Forests achieve higher accuracy due to their ensemble nature. This analysis helps weigh these trade-offs in relation to your specific business needs. Inform the selection of the most suitable algorithm for customer classification. Based on the performance comparison and the relative importance of interpretability and accuracy, we can choose the optimal algorithm for your customer segmentation and prediction tasks. 3.Observations The Random Forest model significantly outperformed the Decision Tree model across all evaluation metrics: Accuracy: Random Forest (0.651) vs. Decision Tree (0.556) Precision: Random Forest (0.6474) vs. Decision Tree (0.5565) Recall: Random Forest (0.651) vs. Decision Tree (0.556) F1-Score: Random Forest (0.6465) vs. Decision Tree (0.5561) 4.Managerial Insights For this customer classification task, the Random Forest algorithm appears to be a considerably better choice compared to the Decision Tree. Random Forest achieves a higher overall accuracy, indicating it classifies customers more accurately based on their characteristics. This improvement suggests that the ensemble approach of Random Forest, combining multiple decision trees, reduces the risk of overfitting and leads to better generalization capabilities.

Ensemble Learning: Classification - Decision Tree vs KNN

1.Description of Data The dataset utilized for Decision Tree and KNN clustering includes a subset of 5,000 records from the original data. The analysis was conducted using five selected features deemed significant for distinguishing customer groupings based on their financial behaviors and credit history.

2.Objectives of Comparing Decision Tree and Performance The primary objectives of comparing Decision Tree and KNNperformance in this context are: The primary aim of comparing the performance of Decision Tree and KNN algorithms on our dataset, comprising 10 years of daily weather observations from diverse locations across Australia, is to assess their effectiveness in weather classification. By analyzing metrics such as accuracy, precision, recall, and F1-score, we seek to determine which algorithm provides the most accurate and reliable predictions of weather conditions. Additionally, we aim to understand the trade-offs between interpretability and accuracy offered by each algorithm. While Decision Trees may offer clearer rules for understanding weather patterns, KNN may provide higher accuracy due to its proximity-based approach. Ultimately, this analysis aims to inform the selection of the most suitable algorithm for our weather classification tasks, ensuring accurate and insightful predictions of weather phenomena across different regions in Australia.

3.Observations The KNNmodel significantly outperformed the Decision Tree model across all evaluation metrics: The comparison between the Decision Tree and KNN (K-Nearest Neighbors) classifiers reveals valuable insights into their respective performances on the given dataset. While both models demonstrate competitive accuracy rates, with the Decision Tree achieving 79.5% and the KNN model slightly outperforming it at 81.8%, they exhibit nuanced differences in precision, recall, and F1 Score. The Decision Tree model showcases a precision of approximately 79.2%, indicating a relatively low rate of false positives, while the KNN model achieves a slightly higher precision of around 80.3%. However, both models demonstrate identical recall rates of 81.8%, indicating their consistent ability to correctly identify positive instances. When considering the harmonic mean of precision and recall, known as the F1 Score, the KNN model edges ahead with an F1 Score of around 80.5%, compared to approximately 79.3% for the Decision Tree model. These findings suggest that while both models perform admirably in certain aspects, such as recall, the KNN model exhibits a slightly superior balance between precision and recall, making it potentially more robust for certain applications.

Ensemble Learning: Classification - Decision Tree vs Logical Regression

1.Description of Data The dataset utilized for Decision Tree and LOGICAL REGRESSION clustering includes a subset of 5,000 records from the original data. The analysis was conducted using five selected features deemed significant for distinguishing customer groupings.

2.Objectives of Comparing Decision Tree and Performance: The comparison between the performance of Decision Tree and Logistic Regression algorithms on the dataset of daily weather observations in Australia aims to achieve several objectives. The primary being the medium to assess the accuracy of both algorithms in predicting weather patterns based on meteorological variables, including temperature, rainfall, wind speed, humidity, and atmospheric pressure. By comparing metrics such as accuracy, precision, recall, and F1-score, the goal is to determine which algorithm excels in classifying weather conditions with higher precision and reliability. Additionally, the comparison aims to understand the trade-offs between interpretability and accuracy offered by each algorithm. While Decision Trees may offer clearer decision rules, Logistic Regression may provide higher accuracy due to its regression-based approach. Ultimately, the analysis aims to inform the selection of the most suitable algorithm for weather classification tasks, ensuring accurate and reliable predictions of weather phenomena across Australia.

3.Observations : Logistic Regression outperforms Decision Tree across most metrics. Logistic Regression achieves a higher accuracy of 84.5% compared to Decision Tree's accuracy of 79.5%. Similarly, Logistic Regression exhibits a higher precision of 83.5% compared to Decision Tree's precision of 79.2%. Both algorithms demonstrate the same recall rate of 81.8%. However, Logistic Regression achieves a higher F1 Score of approximately 83.3%, while Decision Tree lags behind with an F1 Score of about 79.3%. These results suggest that Logistic Regression provides better overall classification performance compared to Decision Tree, particularly in terms of accuracy, precision, and F1 Score.

Ensemble Learning: Classification - Decision Tree vs Support Vector Machine

1.Description of Data The dataset utilized for Decision Tree and SVM clustering includes a subset of 5,000 records from the original data.

2.Objectives : The objective of comparing Decision Tree and Support Vector Machine (SVM) algorithms for the given dataset of daily weather observations in Australia is to identify the most effective model for weather pattern classification. This involves evaluating the performance of both algorithms in accurately classifying weather conditions based on various meteorological variables such as temperature, rainfall, wind speed, humidity, and atmospheric pressure. By comparing the classification accuracy, precision, recall, and F1 Score of Decision Tree and SVM, the goal is to determine which algorithm provides more reliable and robust predictions. Additionally, the comparison aims to understand the trade-offs between model interpretability and complexity, as Decision Trees offer intuitive decision rules while SVM may offer higher accuracy but at the expense of interpretability. Ultimately, the objective is to select the algorithm that best suits the specific requirements and constraints of the weather classification task, ensuring accurate and meaningful insights into weather patterns.

3.Observations : When comparing the performance of Decision Tree and Support Vector Machine (SVM) algorithms, both achieved an equal accuracy of 79.5%, indicating an identical proportion of correctly classified instances out of the total dataset. However, SVM exhibited slightly higher precision at approximately 84.0%, indicating a lower rate of false positives compared to Decision Tree, which had a precision of 79.2%. Despite this, both algorithms demonstrated the same recall rate of 84.5%, correctly identifying approximately 84.5% of actual positive instances. Moreover, SVM achieved a slightly higher F1 Score of approximately 82.4% compared to Decision Tree's F1 Score of 79.3%, indicating a slightly better balance between precision and recall for SVM. Overall, while both algorithms performed comparably in terms of accuracy and recall, SVM showcased slightly better precision and F1 Score, suggesting its slightly superior performance in accurately classifying instances in the dataset.

Managerial Insights: Regional Variability: The dataset likely captures diverse weather patterns across different regions of Australia. Understanding these variations is crucial for industries such as agriculture, tourism, and energy, where weather conditions directly impact operations and decision-making. Managers can leverage this insight to tailor strategies and resources based on specific regional weather patterns.

Seasonal Trends: By examining long-term trends in weather variables such as temperature, rainfall, and sunshine duration, managers can identify seasonal patterns and anticipate potential weather-related risks or opportunities. For instance, agriculture and water resource management may benefit from insights into seasonal rainfall patterns for crop planning and irrigation management.

Extreme Weather Events: The dataset may contain records of extreme weather events such as heat-waves, droughts, floods, and cyclones, which can have significant socio-economic impacts. Managers in disaster preparedness, emergency response, and infrastructure planning can use historical weather data to assess vulnerability, improve resilience, and develop proactive mitigation strategies.

[1]:
```python
import os
import pandas as pd
import numpy as np
```

[3]:
```python
# Import & Read Dataset
data = pd.read_csv('weatherAUS.csv')

# Display Dataset Information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  object
 1   Location       145460 non-null  object
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object
 10  WindDir3pm     141232 non-null  object
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
```

```
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object
 22  RainTomorrow   142193 non-null  object
dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

[4]:
```
data.head()
```

[4]:
```
         Date Location  MinTemp  MaxTemp  Rainfall  Evaporation  Sunshine  \
0  2008-12-01   Albury     13.4     22.9       0.6          NaN       NaN
1  2008-12-02   Albury      7.4     25.1       0.0          NaN       NaN
2  2008-12-03   Albury     12.9     25.7       0.0          NaN       NaN
3  2008-12-04   Albury      9.2     28.0       0.0          NaN       NaN
4  2008-12-05   Albury     17.5     32.3       1.0          NaN       NaN

  WindGustDir  WindGustSpeed WindDir9am  … Humidity9am  Humidity3pm  \
0           W           44.0          W  …        71.0         22.0
1         WNW           44.0        NNW  …        44.0         25.0
2         WSW           46.0          W  …        38.0         30.0
3          NE           24.0         SE  …        45.0         16.0
4           W           41.0        ENE  …        82.0         33.0

   Pressure9am  Pressure3pm  Cloud9am  Cloud3pm  Temp9am  Temp3pm  RainToday  \
0       1007.7       1007.1       8.0       NaN     16.9     21.8         No
1       1010.6       1007.8       NaN       NaN     17.2     24.3         No
2       1007.6       1008.7       NaN       2.0     21.0     23.2         No
3       1017.6       1012.8       NaN       NaN     18.1     26.5         No
4       1010.8       1006.0       7.0       8.0     17.8     29.7         No

   RainTomorrow
0            No
1            No
2            No
3            No
4            No

[5 rows x 23 columns]
```

[5]:
```
# Sample 5000 random records from the dataset
sampled_data = data.sample(n=5000, random_state=45025)
```

[6]:
```
sampled_data.describe()
```

[6]:

```
             MinTemp      MaxTemp     Rainfall  Evaporation     Sunshine  \
count  4952.000000  4957.000000  4896.000000  2837.000000  2575.000000
mean     12.222415    23.266270     2.587316     5.469158     7.630136
std       6.357522     7.152721    10.732844     4.121721     3.779511
min      -8.200000    -3.000000     0.000000     0.000000     0.000000
25%       7.700000    18.000000     0.000000     2.600000     4.900000
50%      12.000000    22.600000     0.000000     4.600000     8.500000
75%      17.000000    28.300000     0.800000     7.400000    10.600000
max      29.100000    46.400000   371.000000    50.200000    13.800000

       WindGustSpeed  WindSpeed9am  WindSpeed3pm  Humidity9am  Humidity3pm  \
count    4640.000000   4934.000000   4889.000000  4909.000000  4842.000000
mean       39.864871     13.823875     18.560442    68.983703    51.515076
std        13.719343      8.780241      8.705708    19.029096    21.028126
min         9.000000      0.000000      0.000000     1.000000     1.000000
25%        31.000000      7.000000     13.000000    58.000000    36.000000
50%        39.000000     13.000000     19.000000    70.000000    52.000000
75%        48.000000     19.000000     24.000000    83.000000    66.000000
max       124.000000     57.000000     65.000000   100.000000   100.000000

       Pressure9am  Pressure3pm     Cloud9am     Cloud3pm      Temp9am  \
count  4460.000000  4456.000000  3027.000000  2928.000000  4942.000000
mean   1017.619933  1015.198339     4.468451     4.535519    16.991582
std       7.030344     7.013513     2.917217     2.720451     6.488369
min     983.900000   978.200000     0.000000     0.000000    -5.300000
25%    1013.000000  1010.400000     1.000000     2.000000    12.300000
50%    1017.700000  1015.300000     5.000000     5.000000    16.700000
75%    1022.300000  1019.900000     7.000000     7.000000    21.500000
max    1040.400000  1037.000000     8.000000     8.000000    39.100000

           Temp3pm
count  4873.000000
mean     21.748204
std       6.937139
min      -4.200000
25%      16.700000
50%      21.200000
75%      26.500000
max      45.400000
```

[7]:

```
# 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
```

```python
# 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, 74016 to 95475
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           5000 non-null   category
 1   Location       5000 non-null   category
 2   MinTemp        5000 non-null   float64
 3   MaxTemp        5000 non-null   float64
 4   Rainfall       5000 non-null   float64
 5   Evaporation    5000 non-null   float64
 6   Sunshine       5000 non-null   float64
 7   WindGustDir    5000 non-null   category
 8   WindGustSpeed  5000 non-null   float64
 9   WindDir9am     5000 non-null   category
 10  WindDir3pm     5000 non-null   category
 11  WindSpeed9am   5000 non-null   float64
 12  WindSpeed3pm   5000 non-null   float64
 13  Humidity9am    5000 non-null   float64
 14  Humidity3pm    5000 non-null   float64
 15  Pressure9am    5000 non-null   float64
 16  Pressure3pm    5000 non-null   float64
 17  Cloud9am       5000 non-null   float64
 18  Cloud3pm       5000 non-null   float64
 19  Temp9am        5000 non-null   float64
 20  Temp3pm        5000 non-null   float64
 21  RainToday      5000 non-null   category
```

```
 22  RainTomorrow   5000 non-null   category
dtypes: category(7), float64(16)
memory usage: 790.9 KB
```

[8]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
```

[10]:
```python
# Split the sampled data into features (X) and target (y)
X = sampled_data.drop('RainTomorrow', axis=1)
y = sampled_data['RainTomorrow']

# Define numerical and categorical columns
numerical_cols = X.select_dtypes(include=['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)

print(X_preprocessed[:5])  # Displaying the first 5 rows
```

```
[[-0.85678105 -0.00850108 -0.23842532 -0.15736284  0.16311484  0.4689072
   1.62662836  0.16610331 -0.42443647 -1.18554455  0.52287008  0.51214996
   0.14087712  0.12989523 -0.61832079 -0.06341417]
 [-0.35095859 -1.21612109  0.0815455  -1.11463535  0.16311484  0.4689072
   0.59472202  0.39844813  0.15900323 -0.02563402 -0.74234953 -0.39414974
   0.14087712  0.12989523 -0.77335835 -1.1001661 ]
 [-0.49322115 -0.87911086  0.60855627 -0.15736284  0.16311484  2.89008872
   2.42922218  1.3278274  -0.15923661 -0.46060047  0.01075738  0.01368513
   0.14087712  0.12989523 -0.68033581 -0.8081233 ]
 [ 0.72391413  0.52509846 -0.23842532 -0.15736284  0.16311484  0.24192143
  -0.55184058 -0.64710355 -0.3713965   0.31267321 -1.14902727 -1.11918951
   0.14087712  0.12989523  0.99406982  0.60828426]
 [-0.33515164  0.30042497 -0.23842532  0.161728    1.2554505   0.4689072
   0.3654095   1.4439998  -0.79571628 -0.26728205  0.25175159  0.13452509
  -1.61023686  1.08500725  0.73050597  0.22862862]]
```

[11]:
```python
# Identify numerical columns in the dataset
numerical_features = sampled_data.select_dtypes(include=['int64', 'float64']).
 ↪columns

# Select 5 numerical features for clustering (based on potential utility for␣
 ↪clustering)
selected_features = numerical_features[:5].tolist()  # Change this based on␣
 ↪feature selection logic

selected_features
```

[11]:
```
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine']
```

[12]:
```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
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(
```

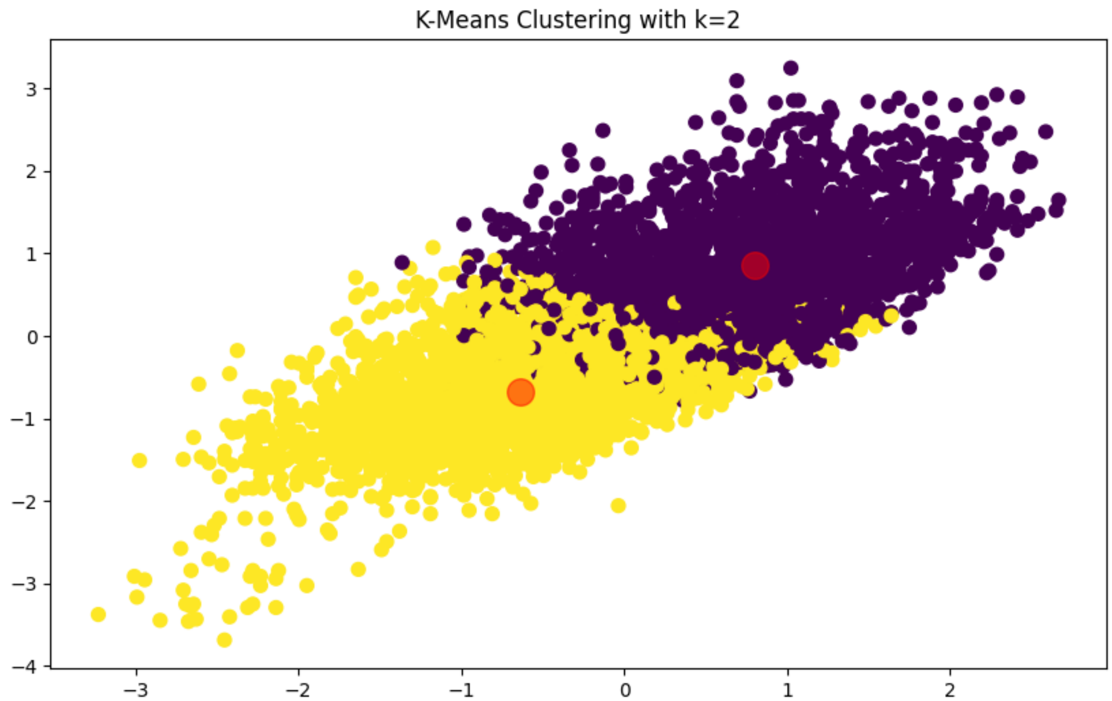
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

[12]:
```
{2: array([1, 1, 1, 0, 0, 1, 1, 0, 0, 0], dtype=int32),
 3: array([2, 2, 2, 0, 2, 2, 2, 2, 0, 0], dtype=int32),
 4: array([1, 1, 1, 0, 1, 1, 1, 1, 0, 0], dtype=int32),
 5: array([2, 2, 2, 1, 1, 2, 2, 1, 1, 1], dtype=int32)}
```

[13]:
```
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=45000)
        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=(10, 6))
        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)
```

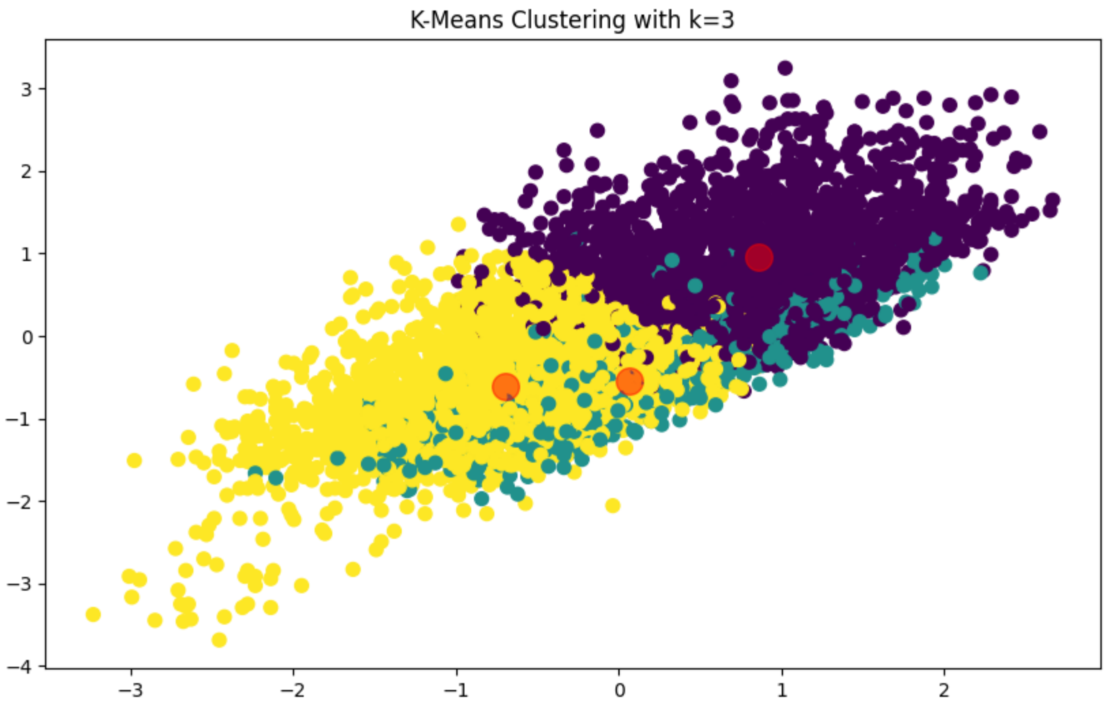
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
For k=2, the Silhouette Score is: 0.3059
For k=2, the Davies-Bouldin Score is: 1.2860
```



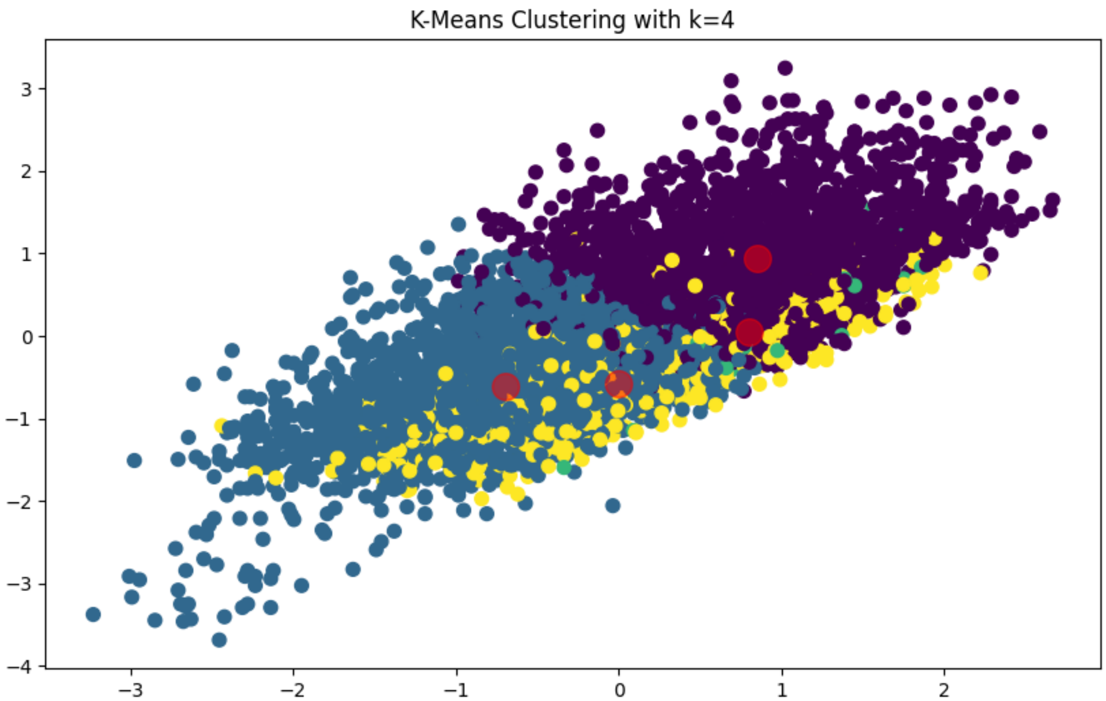

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
For k=3, the Silhouette Score is: 0.3190
For k=3, the Davies-Bouldin Score is: 1.2056
```



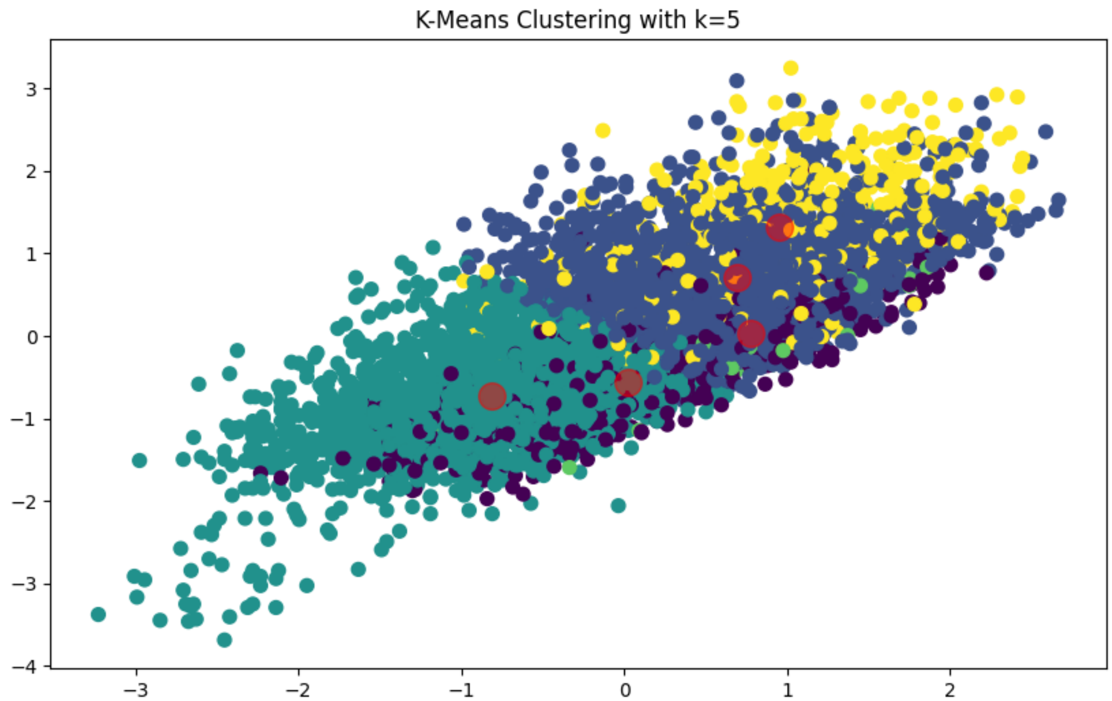

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
For k=4, the Silhouette Score is: 0.3243
For k=4, the Davies-Bouldin Score is: 0.9997
```




```
/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: 0.3106
For k=5, the Davies-Bouldin Score is: 1.0275
```

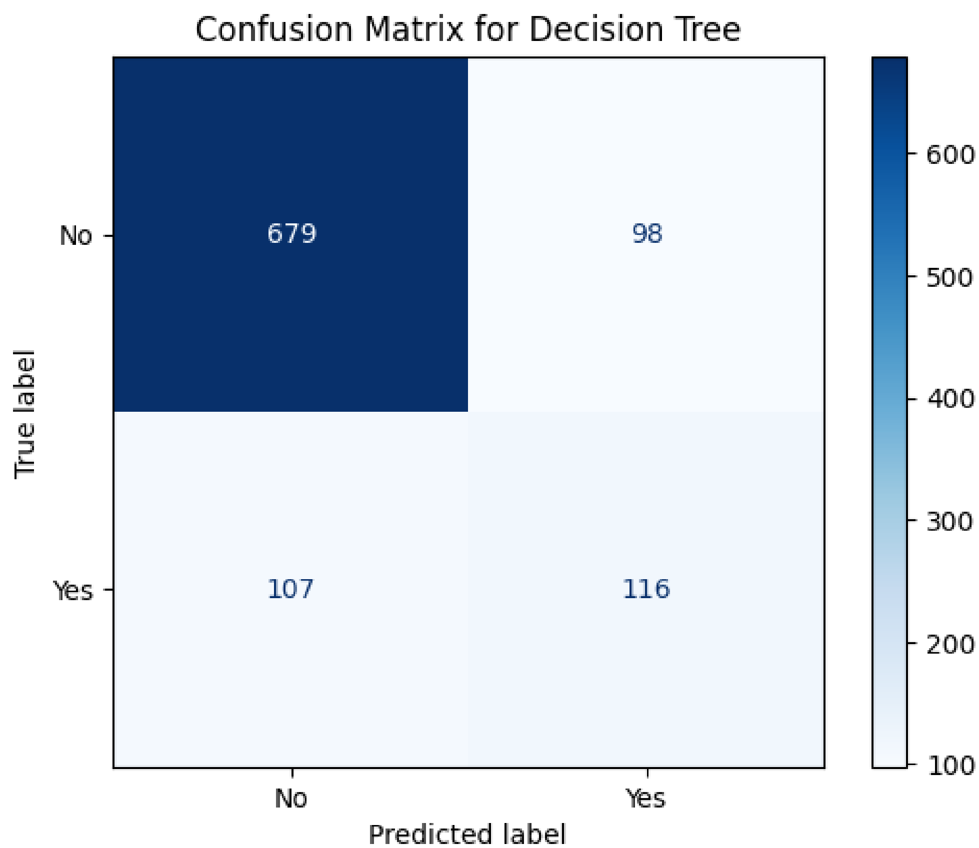


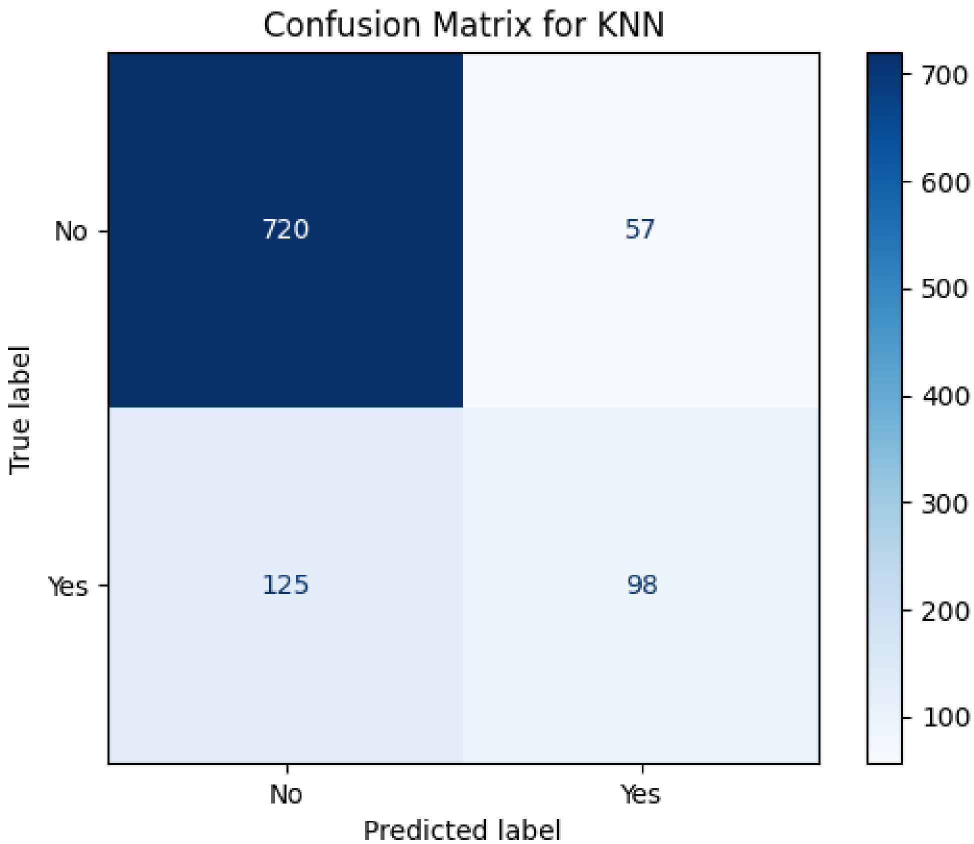
[14]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
import numpy as np

# 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')
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
    }
}

print(results)
```

```
{'Decision Tree': {'Accuracy': 0.795, 'Precision': 0.7921036955125917, 'Recall':
0.795, 'F1 Score': 0.7934792271507441}, 'KNN': {'Accuracy': 0.818, 'Precision':
0.8030527199847299, 'Recall': 0.818, 'F1 Score': 0.8054446727862264}}
```

[15]:
```
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')
```

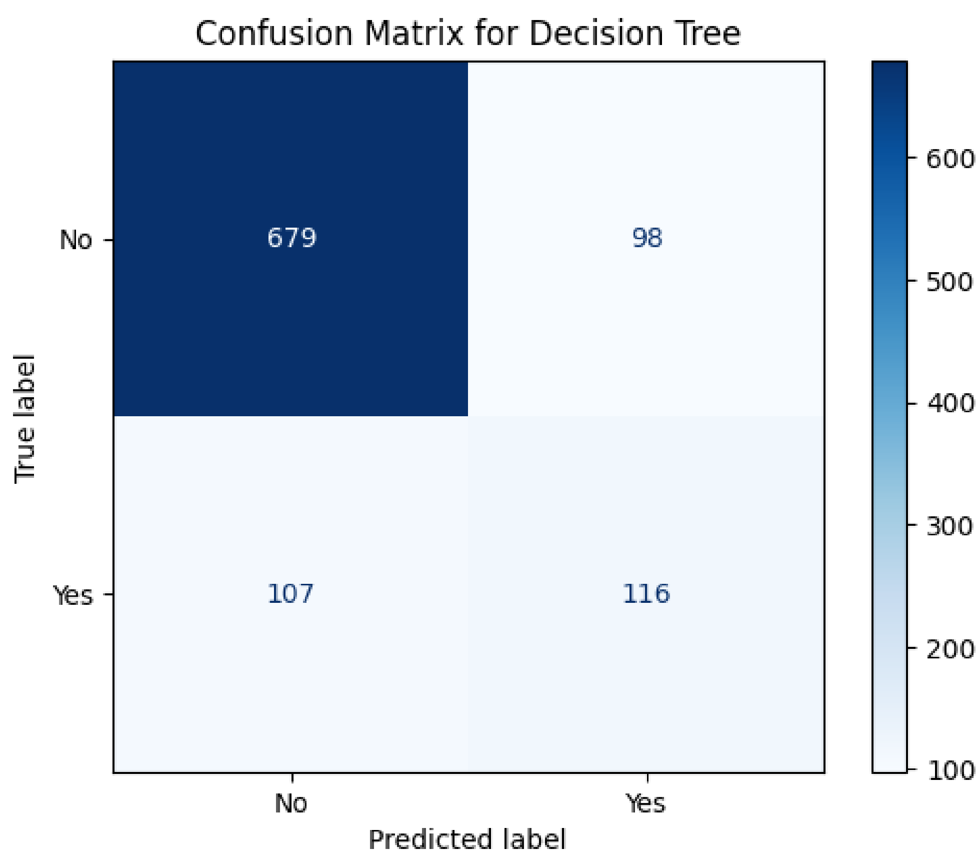

Confusion Matrix for Decision Tree
No
Yes
True label
679
98
107
116
No
Yes
Predicted label
600
500
400
300
200
100

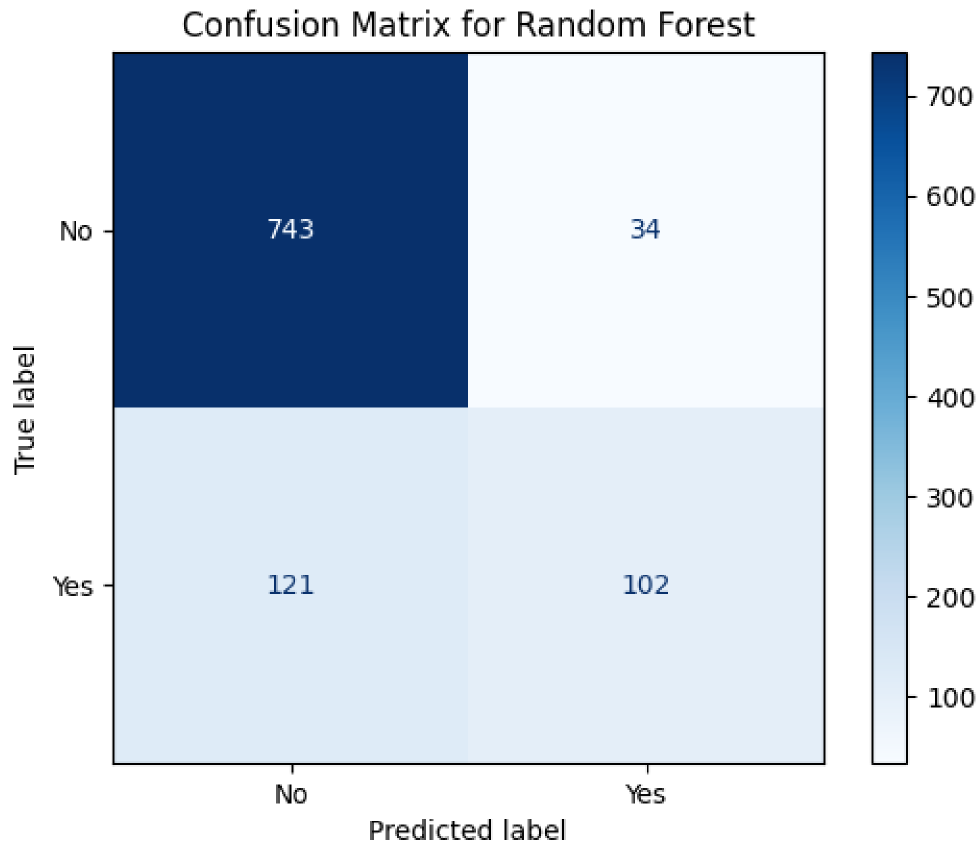


[16]:
```
from sklearn.ensemble import RandomForestClassifier

# 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': 0.795, 'Precision': 0.7921036955125917,
'Recall': 0.795, 'F1 Score': 0.7934792271507441}
Random Forest Results: {'Accuracy': 0.845, 'Precision': 0.8354340277777779,
'Recall': 0.845, 'F1 Score': 0.8303274380897578}
```





[17]:
```
from sklearn.linear_model import LogisticRegression

# Initialize the logistic regression model
logistic_regression = LogisticRegression(random_state=45000)

# 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,
```

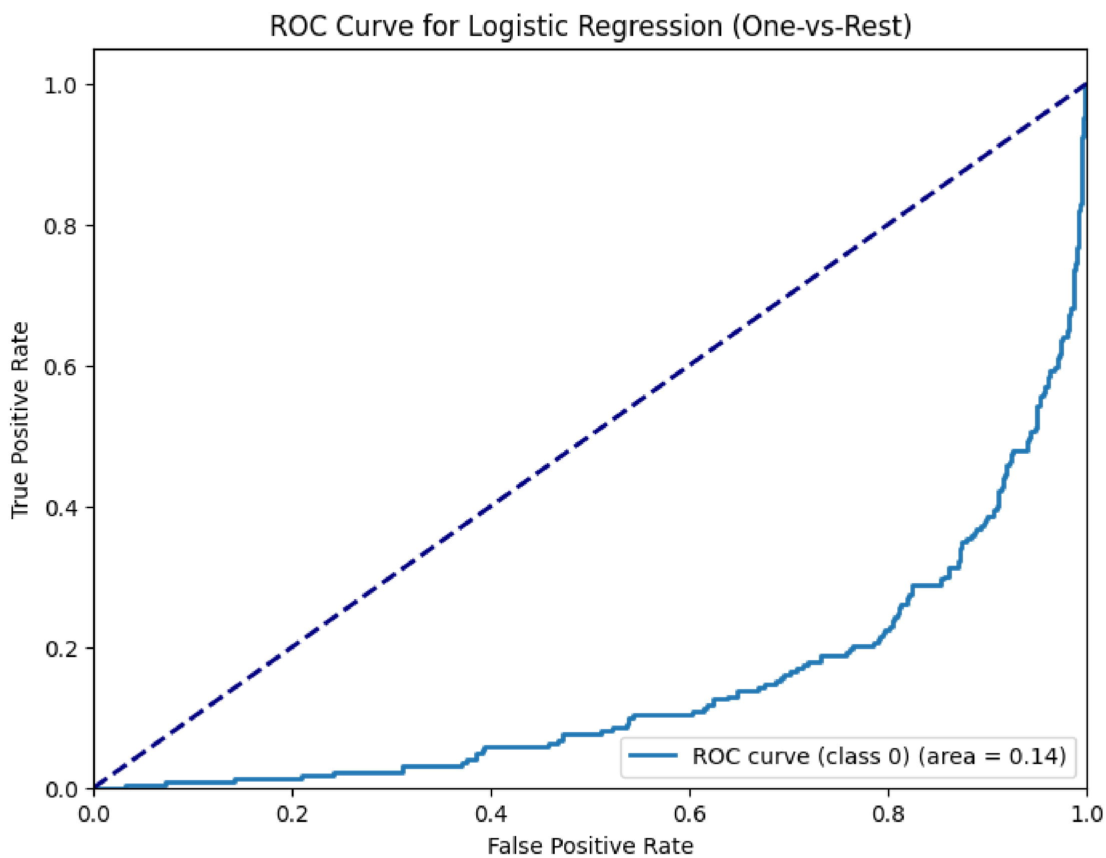
```
    'Precision': precision_lr,
    'Recall': recall_lr,
    'F1 Score': f1_lr
}

print(results)
```

{'Decision Tree': {'Accuracy': 0.795, 'Precision': 0.7921036955125917, 'Recall': 0.795, 'F1 Score': 0.7934792271507441}, 'KNN': {'Accuracy': 0.818, 'Precision': 0.8030527199847299, 'Recall': 0.818, 'F1 Score': 0.8054446727862264}, 'Logistic Regression': {'Accuracy': 0.845, 'Precision': 0.8348372549019607, 'Recall': 0.845, 'F1 Score': 0.8333096984367354}}

[19]:

```
from sklearn.metrics import roc_curve, auc



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=45000))

# 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=(8, 6))
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()
```

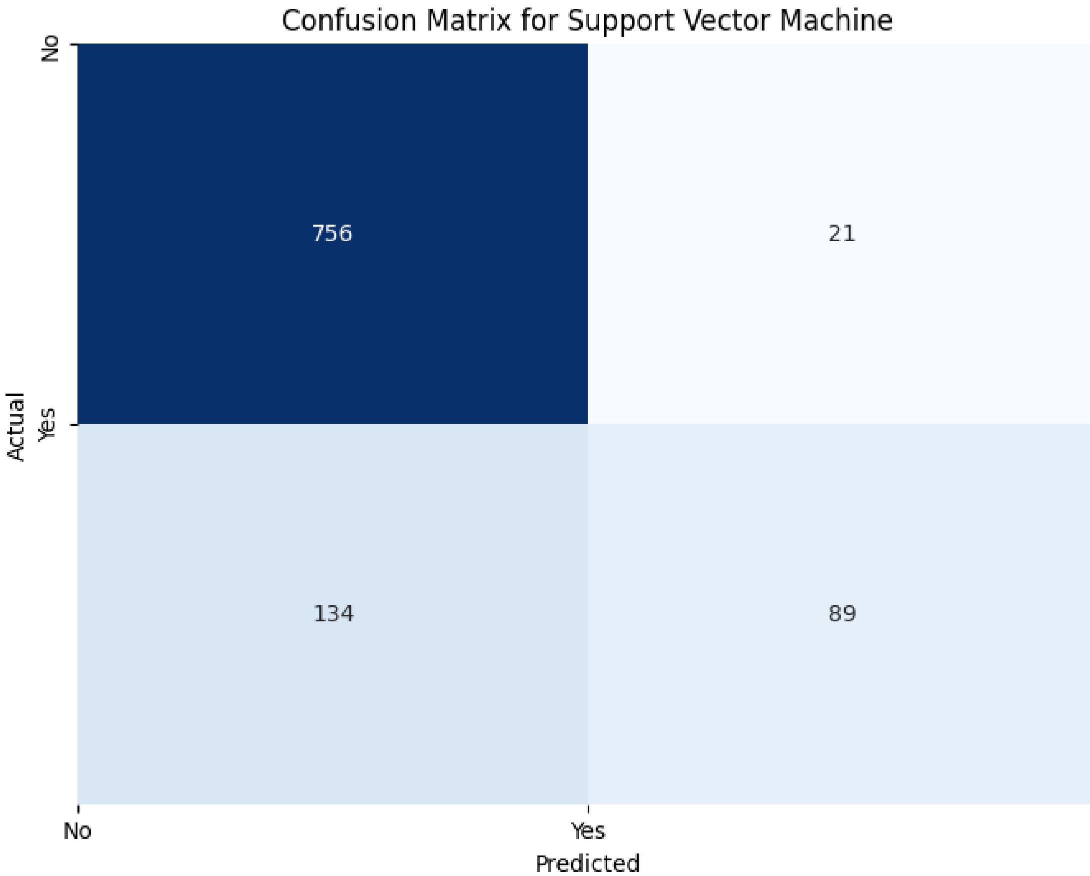


[20]:
```
from sklearn.svm import SVC

# 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)
```

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

print(results)
```

```
{'Decision Tree': {'Accuracy': 0.795, 'Precision': 0.7921036955125917, 'Recall':
0.795, 'F1 Score': 0.7934792271507441}, 'KNN': {'Accuracy': 0.818, 'Precision':
0.8030527199847299, 'Recall': 0.818, 'F1 Score': 0.8054446727862264}, 'Logistic
Regression': {'Accuracy': 0.845, 'Precision': 0.8348372549019607, 'Recall':
0.845, 'F1 Score': 0.8333096984367354}, 'Support Vector Machine': {'Accuracy':
0.845, 'Precision': 0.8404407558733401, 'Recall': 0.845, 'F1 Score':
0.8239546505113392}}
```

[21]:
```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=(8, 6))
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
756
21
134
89
No
Yes
Actual
No
Yes
Predicted