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.

Analysis

1. Clustering Analysis

K-Means Clustering Results:

- **k=2**:
  - Silhouette Score: 0.1665
  - Davies-Bouldin Score: 2.1969
- **k=3**:
  - Silhouette Score: 0.1707

    - Davies-Bouldin Score: 1.8782
- **k=4**:
    - Silhouette Score: 0.1660
    - Davies-Bouldin Score: 1.6560
- **k=5**:
    - Silhouette Score: 0.1735
    - Davies-Bouldin Score: 1.5846

2. Classification Modeling Results

Decision Tree:

- **Accuracy**: 0.618
- **Precision**: 0.619
- **Recall**: 0.618
- **F1 Score**: 0.618

Random Forest:

- **Accuracy**: 0.684
- **Precision**: 0.681
- **Recall**: 0.684
- **F1 Score**: 0.675

Observations and Insights

1. **Clustering Analysis**:
    - The Silhouette Scores for all k values (2 to 5) are relatively low, indicating overlapping clusters or uneven cluster sizes.
    - The Davies-Bouldin Scores show improvement as k increases, suggesting better separation between clusters.
    - Cluster assignments for different k values provide groupings of orders based on their day of the week and hour of the day.
    - For k=2, the clustering shows two main groups, possibly separating orders placed during weekdays and weekends.
    - Increasing k to 3 or 4 shows more nuanced clusters, possibly dividing orders by specific times or patterns within weekdays.
2. **Classification Modeling**:
    - Both Decision Tree and Random Forest models were trained to predict "add_to_cart_order" behavior.
    - Random Forest outperformed the Decision Tree in all metrics (Accuracy, Precision, Recall, F1 Score).
    - Random Forest achieved an accuracy of 0.684, indicating it correctly predicted 68.4% of "add_to_cart_order" values.
3. **Recommendations**:
    - **Customer Segmentation**:
        - Utilize the clustering results to segment customers based on their order patterns.

        - Tailor marketing campaigns and promotions for different clusters, such as targeting weekend shoppers separately from weekday shoppers.
    - **Predictive Modeling**:
        - Implement the Random Forest model for predicting "add_to_cart_order" in real-time.
        - Use predictions to optimize inventory management and product recommendations.
    - **Further Analysis**:
        - Explore additional features or combinations of features for clustering to improve cluster quality.
        - Fine-tune the classification models with hyperparameter tuning for better performance.
4. **Conclusion**:
    - The analysis provides valuable insights into customer order patterns and predictive modeling for "add_to_cart_order."
    - Businesses can benefit from segmentation strategies and predictive models to enhance customer satisfaction and operational efficiency.
    - Continuous monitoring and refinement of models based on new data will be crucial for ongoing success.

Managerial Implications

Decision Tree Classifier Visualization:

- **Interpreting Customer Behavior**:
    - Managers can understand how the decision tree classifies customer behavior based on various features like order day, hour, and others.
    - This insight helps in identifying patterns of when and how customers place orders, allowing for targeted marketing strategies.
- **Feature Importance**:
    - Visualizing the decision tree helps in identifying which features (like order day or hour) are most influential in predicting customer behavior.
    - Managers can prioritize resources and promotions based on these important features to maximize sales during peak order times.
- **Customer Segmentation**:
    - Clustering customers based on the decision tree's splits can create distinct customer segments.
    - Managers can tailor marketing campaigns or promotions for each segment, increasing the relevance of offers and improving customer satisfaction.

Single Tree from Random Forest Visualization:

- **Ensemble Model Understanding**:
    - Random Forests are more complex due to multiple decision trees.
    - Visualizing a single tree provides insight into how a particular subset of data is classified, helping managers understand the collective decision-making of the ensemble.
- **Model Validation**:

- Managers can use the visualization to validate the Random Forest's decisions against their domain knowledge.
- This can help in gaining confidence in the model's predictions and making informed decisions based on these predictions.

- **Risk Management**:
  - Identifying specific paths or nodes in the tree can highlight areas where the model might be making risky predictions.
  - Managers can focus on improving these areas to reduce potential errors in predictions.

Confusion Matrix and RMSE Analysis

Confusion Matrix Visualization:

- **Model Performance Evaluation**:
  - The confusion matrix provides a visual representation of the model's performance in classifying different classes.
  - It shows true positives, true negatives, false positives, and false negatives, which are crucial for understanding the model's errors.
- **Identifying Errors**:
  - Managers can identify where the model is making mistakes, such as misclassifying certain types of orders.
  - This helps in understanding the areas where the model needs improvement or where additional data may be required.
- **Validation of Predictions**:
  - By comparing the actual and predicted values in the confusion matrix, managers can validate the model's predictions.
  - This validation ensures that the model is making accurate predictions and can be trusted for decision-making.

Root Mean Squared Error (RMSE):

- **Model Accuracy**:
  - RMSE measures the average difference between predicted and actual values.
  - A lower RMSE indicates better model performance in predicting the 'add_to_cart_order'.
- **Prediction Precision**:
  - Managers can use RMSE to gauge how precise the model's predictions are.
  - A high RMSE could indicate that the model's predictions are far from the actual 'add_to_cart_order', while a low RMSE suggests closer predictions.
- **Improvement Focus**:
  - Monitoring RMSE over time helps in tracking the model's performance.
  - A sudden increase in RMSE might indicate changes in customer behavior or data quality issues that need attention.

Insights:

1. **Model Evaluation**:

- The confusion matrix allows managers to evaluate the models' classification performance.
- By observing false positives and false negatives, managers can adjust strategies to reduce errors.

2. **Error Analysis**:
   - Understanding the confusion matrix helps in identifying which types of errors the models are making.
   - Managers can focus on improving the model's performance in specific classes to enhance overall accuracy.
3. **Trust in Predictions**:
   - The validation provided by the confusion matrix builds confidence in the model's predictions.
   - Managers can use these validated predictions for making business decisions with more assurance.
4. **Operational Improvements**:
   - Monitoring RMSE helps in continuous model improvement.
   - Managers can identify trends and patterns in RMSE changes, allowing for proactive adjustments to the model.
5. **Resource Allocation**:
   - Based on RMSE, managers can allocate resources more effectively.
   - If the model's predictions are consistently accurate (low RMSE), resources like inventory and staffing can be optimized accordingly.
6. **Customer Satisfaction**:
   - Improving model accuracy through insights from the confusion matrix can lead to better customer experiences.
   - Accurate predictions of 'add_to_cart_order' mean customers receive the products they want, enhancing satisfaction and loyalty.
7. **Forecasting and Planning**:
   - Reliable predictions from the model support better forecasting and planning.
   - Managers can use these predictions for inventory management, marketing campaigns, and operational planning.

Real Time Managerial Implications:

1. **Marketing Strategies**:
   - Based on the decision tree insights, managers can optimize marketing strategies for different customer segments.
   - Promotions can be timed for peak order days or hours identified by the tree.
2. **Product Recommendations**:
   - Understanding feature importance helps in recommending products at specific times or days when customers are more likely to buy.
   - This can improve cross-selling and upselling opportunities.
3. **Operational Efficiency**:
   - By knowing when orders are likely to peak, managers can allocate resources efficiently.
   - Staffing and inventory management can be optimized for busy periods.

4. **Customer Experience**:
   - Targeted marketing and promotions enhance customer experience by providing relevant offers.
   - Customers are more likely to respond positively to offers that align with their order patterns.
5. **Model Understanding**:
   - Visualization aids in explaining model predictions to stakeholders who may not be familiar with complex algorithms.
   - Managers can confidently explain and defend the model's decisions to higher management or clients.
6. **Continuous Improvement**:
   - Monitoring the model's performance over time with these visualizations helps in continuous improvement.
   - Updates and refinements can be made based on real-world feedback and changing customer behavior patterns.

```
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: 2019501 entries, 0 to 2019500
Data columns (total 12 columns):
 #   Column                  Dtype
---  ------                  -----
 0   order_id                int64
 1   user_id                 int64
 2   order_number            int64
 3   order_dow               int64
 4   order_hour_of_day       int64
 5   days_since_prior_order  float64
 6   product_id              int64
 7   add_to_cart_order       int64
 8   reordered               int64
 9   department_id           int64
 10  department              object
 11  product_name            object
dtypes: float64(1), int64(9), object(2)
memory usage: 184.9+ MB

data.head()
```

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

```python
# 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\":
1199719.248412101,\n        \"min\": 4781.0,\n        \"max\":
3420964.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          1721553.4968,\n          1743609.5,\n          5000.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"user_id\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
71440.77003152356,\n        \"min\": 71.0,\n        \"max\":
206166.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
102402.099,\n          100223.0,\n          5000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"order_number\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1759.1757706432113,\n        \"min\": 1.0,\n        \"max\": 5000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          17.0596,\
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\":
1766.817194552892,\n        \"min\": 0.0,\n        \"max\": 5000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          2.7262,\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\": 1763.7572886418322,\n        \"min\":
0.0,\n        \"max\": 5000.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          13.408,\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\":
1643.62713717923,\n        \"min\": 0.0,\n        \"max\": 4660.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
11.389484978540773,\n          8.0,\n          4660.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"product_id\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1744.7660879372047,\n        \"min\": 1.0,\n        \"max\": 5000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          71.7942,\
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\":
1762.1507267470486,\n        \"min\": 1.0,\n        \"max\": 5000.0,\n
```

\"num_unique_values\": 8,\n   \"samples\": [\n   8.4148,\n   6.0,\n   5000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"reordered\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1767.5612282003613,\n   \"min\": 0.0,\n   \"max\": 5000.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.5806,\n   1.0,\n   0.4935102333995659\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"department_id\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1764.4861508495733,\n   \"min\": 1.0,\n   \"max\": 5000.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   9.8788,\n   7.0,\n   5000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```python
# 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, 1565054 to 111627
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   int64
 3   order_dow               5000 non-null   int64
 4   order_hour_of_day       5000 non-null   int64
 5   days_since_prior_order  5000 non-null   float64
 6   product_id              5000 non-null   int64
 7   add_to_cart_order       5000 non-null   int64
 8   reordered               5000 non-null   int64
 9   department_id           5000 non-null   int64
 10  department              5000 non-null   category
 11  product_name            5000 non-null   category
dtypes: category(2), float64(1), int64(9)
memory usage: 450.1 KB
```

```
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 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
```

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

```
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=45036)
    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()}
```

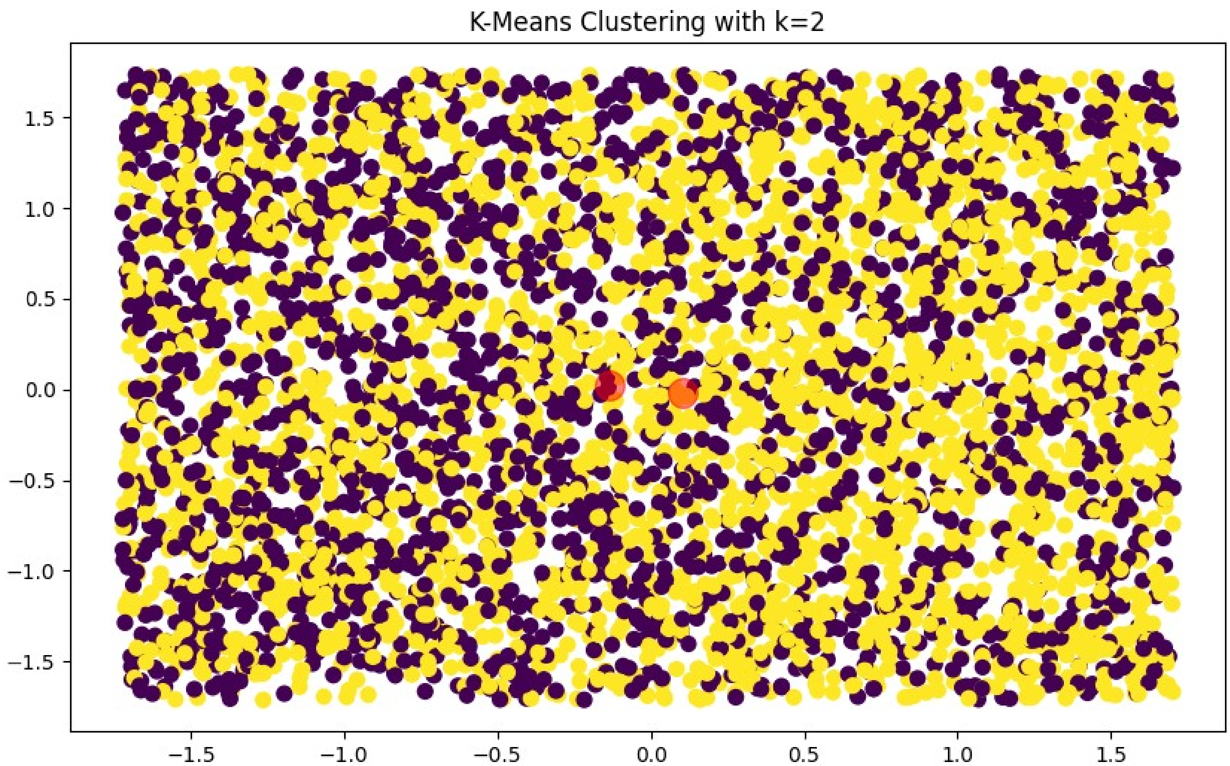
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: 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([0, 1, 0, 1, 1, 1, 0, 0, 0, 0], dtype=int32),
 3: array([2, 1, 2, 1, 1, 1, 2, 2, 2, 2], dtype=int32),
 4: array([3, 2, 0, 2, 2, 2, 0, 3, 0, 0], dtype=int32),
 5: array([4, 3, 4, 3, 3, 3, 4, 1, 4, 4], dtype=int32)}
```

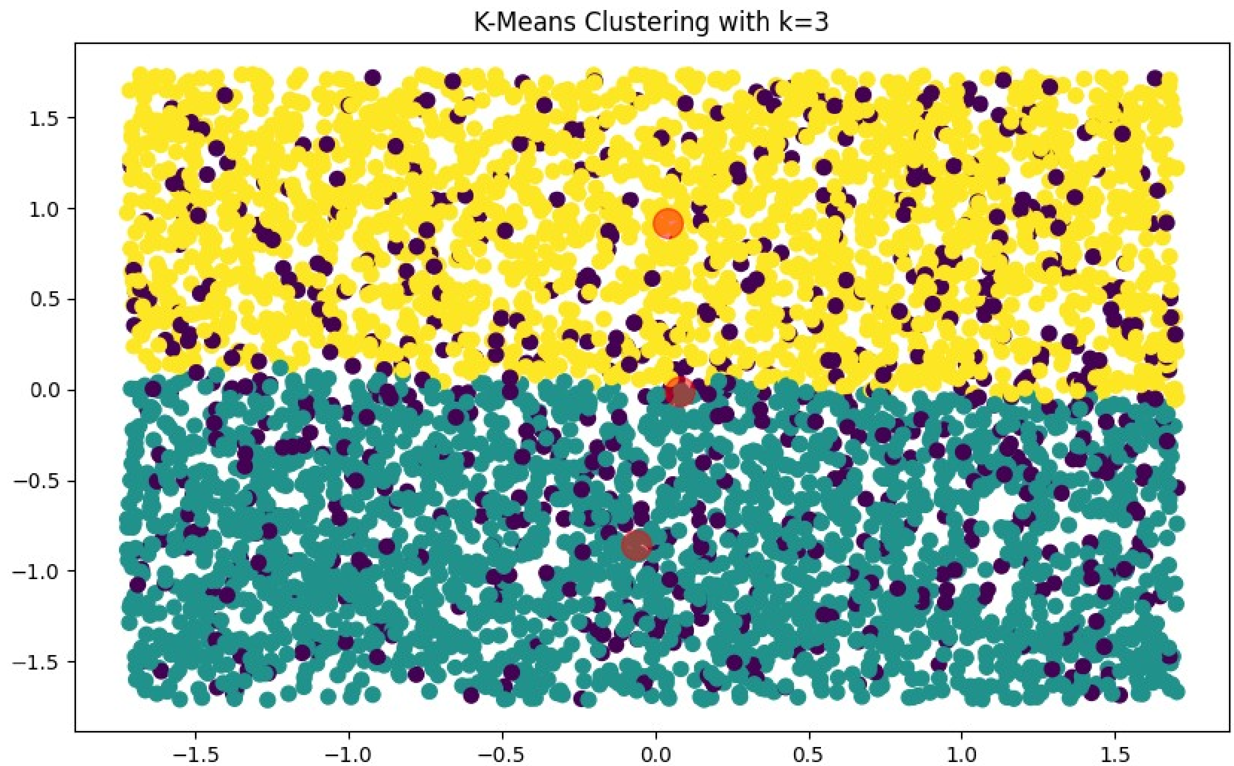
```
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score, davies_bouldin_score

# 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=(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(
```

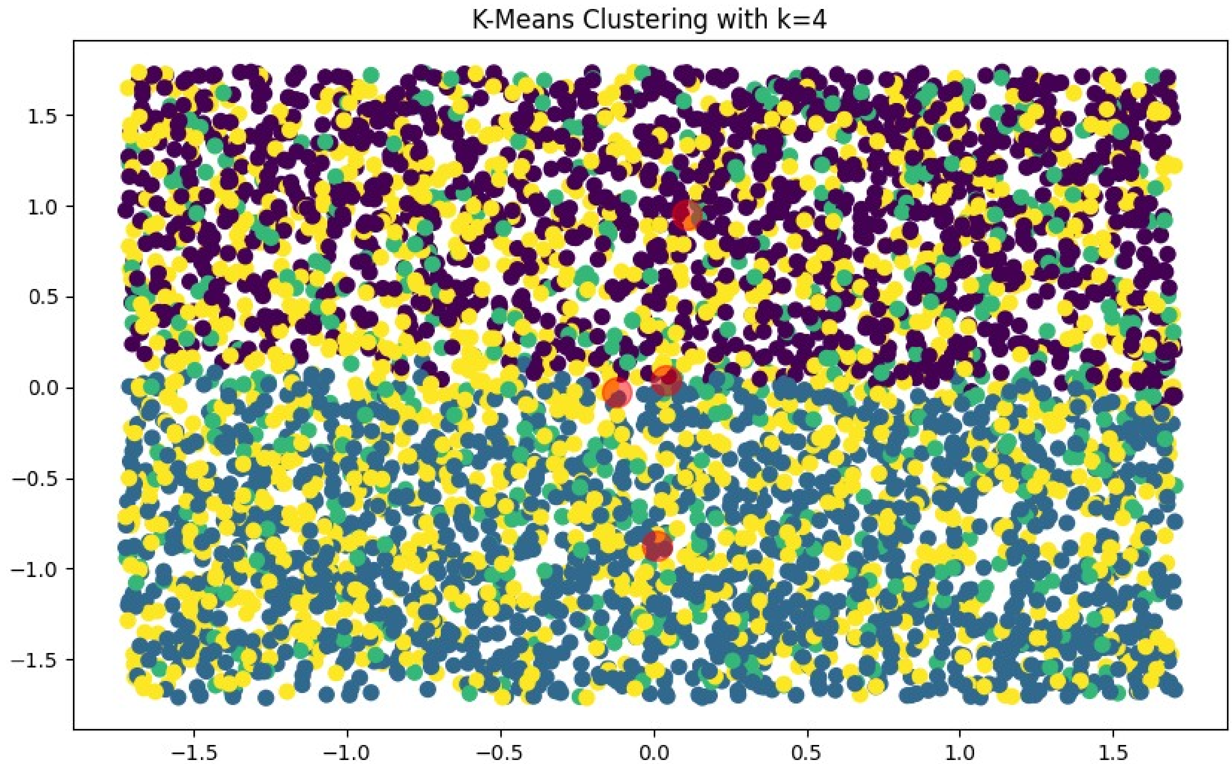
```
For k=2, the Silhouette Score is: 0.1665
For k=2, the Davies-Bouldin Score is: 2.1969
```



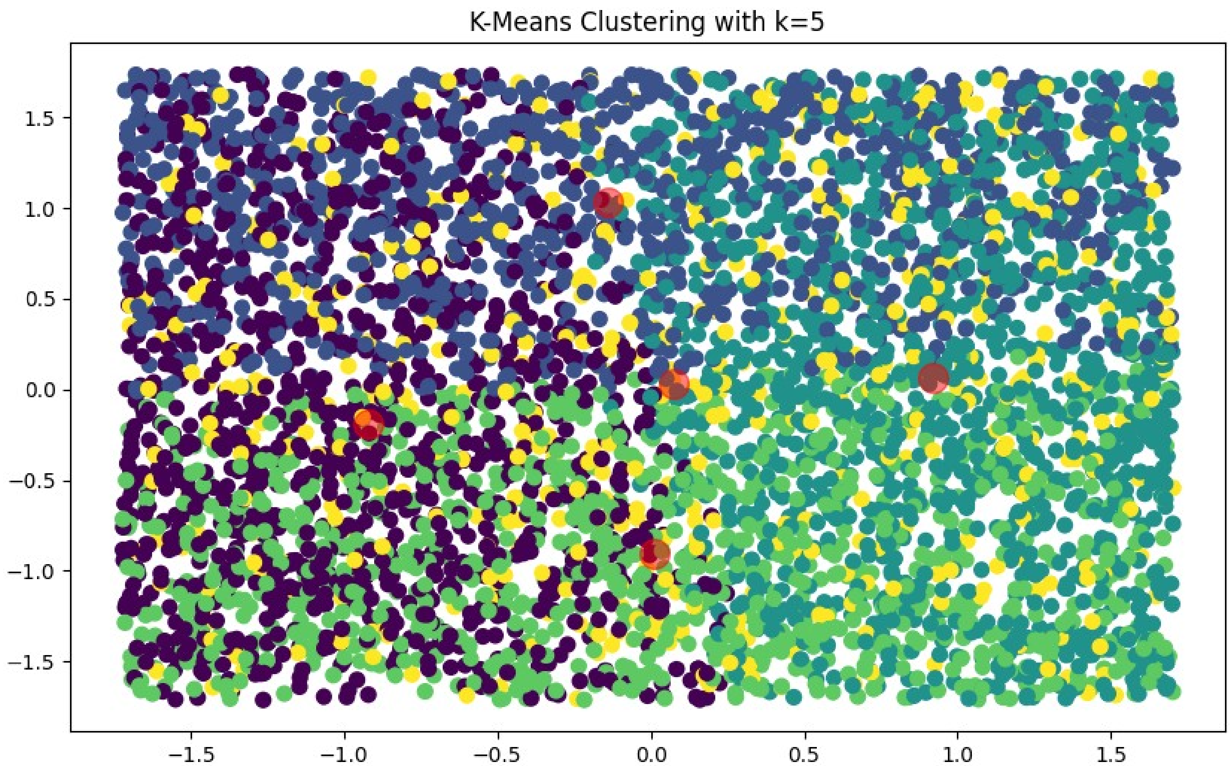

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(

For k=3, the Silhouette Score is: 0.1707
For k=3, the Davies-Bouldin Score is: 1.8782
```




```
/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: 0.1660
For k=4, the Davies-Bouldin Score is: 1.6560
```




```
/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: 0.1735
For k=5, the Davies-Bouldin Score is: 1.5846
```



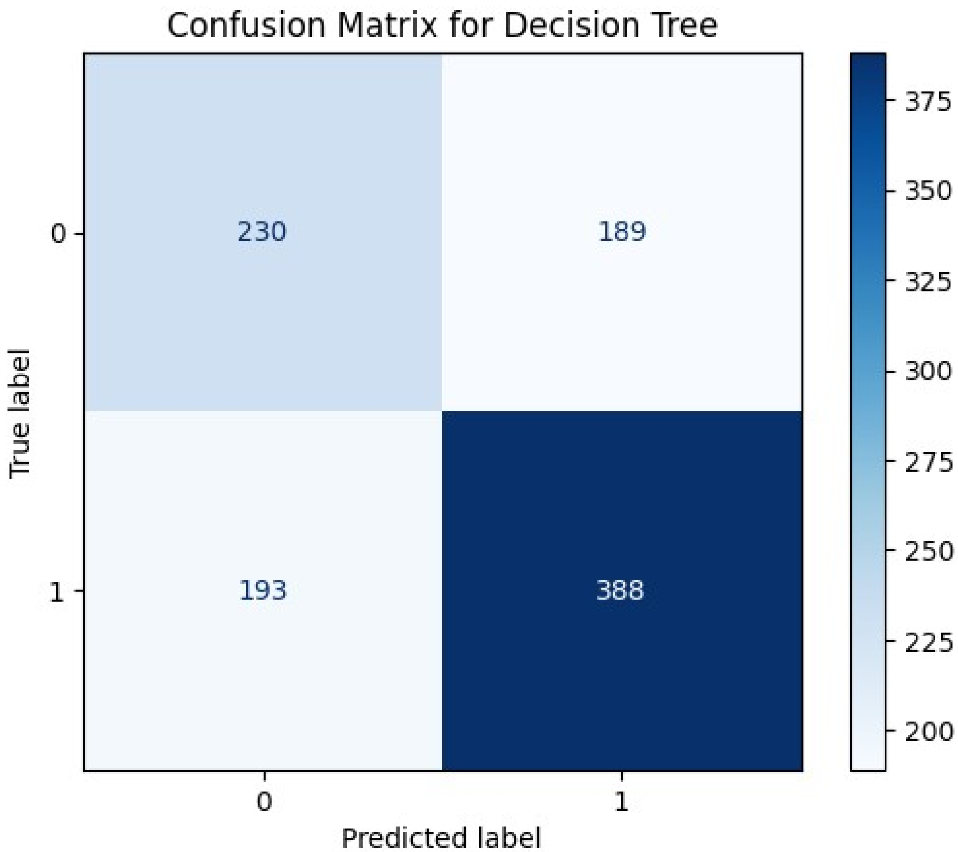

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve, auc
import numpy as np

# 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=45037, stratify=y)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

# Assuming you have X_train, X_test, y_train, y_test from your data
preparation step

# Initialize the models
decision_tree = DecisionTreeClassifier(random_state=45037)
random_forest = RandomForestClassifier(random_state=45037)
```

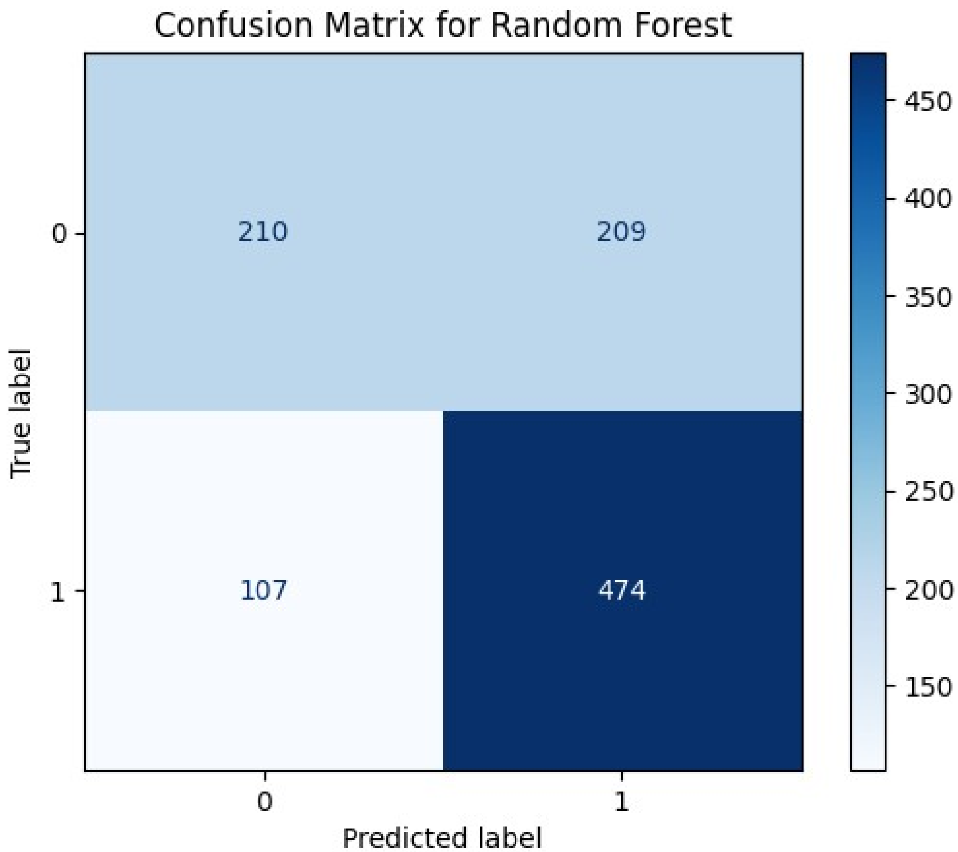
```python
# Train the models
decision_tree.fit(X_train, y_train)
random_forest.fit(X_train, y_train)

# Predict on the testing set
y_pred_dt = decision_tree.predict(X_test)
y_pred_rf = random_forest.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_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
results = {
    'Decision Tree': {
        'Accuracy': accuracy_dt,
        'Precision': precision_dt,
        'Recall': recall_dt,
        'F1 Score': f1_dt
    },
    'Random Forest': {
        'Accuracy': accuracy_rf,
        'Precision': precision_rf,
        'Recall': recall_rf,
        'F1 Score': f1_rf
    }
}

results
```

```
{'Decision Tree': {'Accuracy': 0.618,
  'Precision': 0.6185148337983619,
  'Recall': 0.6179999999999999,
  'F1 Score': 0.6182476052166279},
 'Random Forest': {'Accuracy': 0.684,
  'Precision': 0.680783276600265,
  'Recall': 0.684,
  'F1 Score': 0.6748532608695652}}
```

```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(random_forest, X_test, y_test, 'Random 
Forest')
```



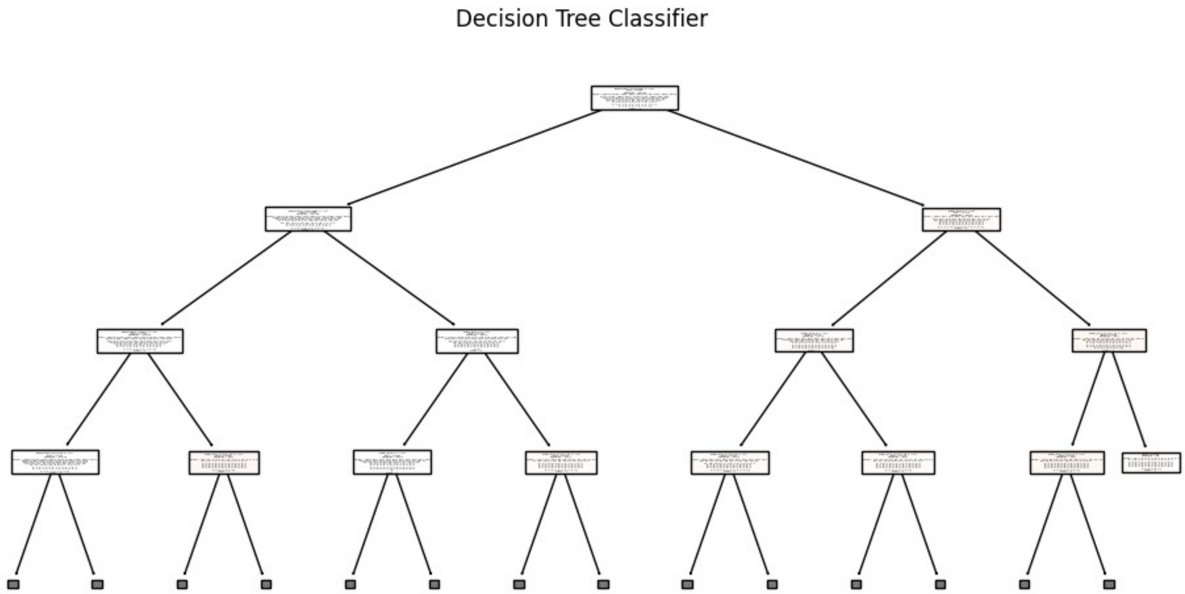


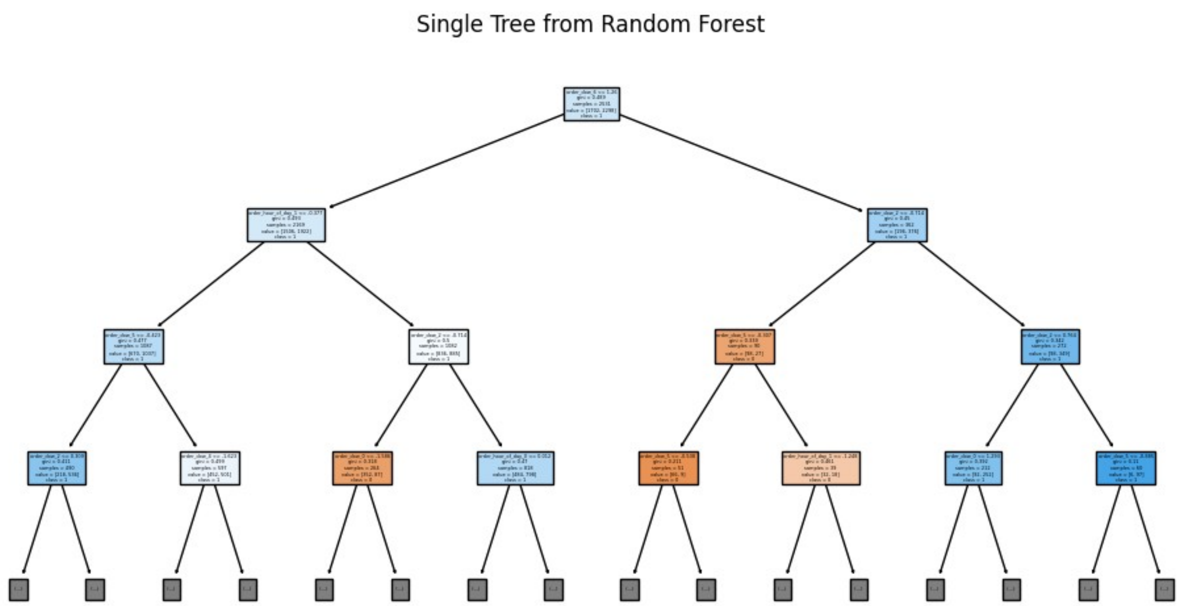

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error

# Assuming you have your data loaded into a DataFrame called 'data'

# Selecting features and target
X = data.drop(['add_to_cart_order', 'user_id', 'order_number'],
axis=1)  # Drop non-useful and target variable
y = data['add_to_cart_order']

# Defining categorical columns
categorical_cols = ['order_dow', 'order_hour_of_day', 'department']

# Creating a preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'),
categorical_cols)
```

```
    ])

# Creating a preprocessing and training pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier',
DecisionTreeClassifier(random_state=45037))])

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=45037)

# Fitting the model
pipeline.fit(X_train, y_train)

# Predicting on the test set
y_pred = pipeline.predict(X_test)

# Calculate RMSE
rmse = mean_squared_error(y_test, y_pred, squared=False)

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

```
Root Mean Squared Error: 9.563820309321484
```

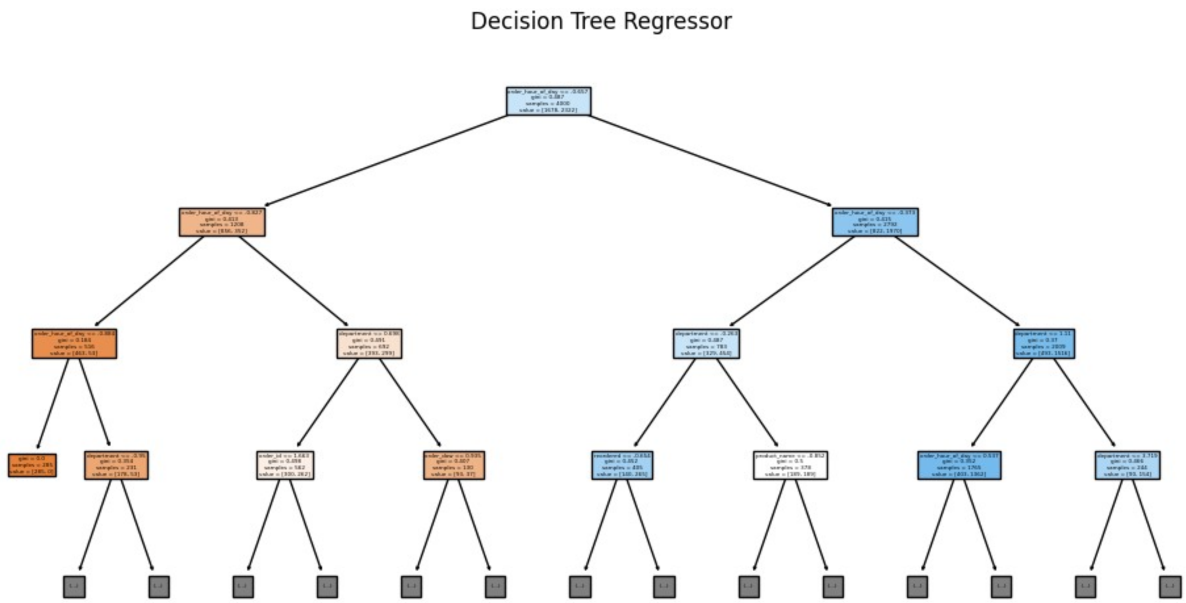
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Get feature names after one-hot encoding
encoded_features = preprocessor.transformers_[0]
[1].get_feature_names_out()

# Plot the Decision Tree from DecisionTreeClassifier
plt.figure(figsize=(12, 6))
plot_tree(decision_tree_model, filled=True,
feature_names=encoded_features, class_names=[str(i) for i in
decision_tree_model.classes_], max_depth=3)
plt.title("Decision Tree Classifier")
plt.show()

# Plot a single tree from the Random Forest
plt.figure(figsize=(12, 6))
rf_tree = random_forest.estimators_[0]  # Select the first tree in the
Random Forest
plot_tree(rf_tree, filled=True, feature_names=encoded_features,
class_names=[str(i) for i in random_forest.classes_], max_depth=3)
plt.title("Single Tree from Random Forest")
plt.show()
```






```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Assume X_train is defined in your notebook and contains the feature
names
features = X_train.columns.tolist()  # Make sure X_train is your
feature DataFrame

# Plot the Decision Tree
```

```
plt.figure(figsize=(12, 6))
plot_tree(decision_tree, filled=True, feature_names=features,
class_names=None, max_depth=3)
plt.title("Decision Tree Regressor")
plt.show()

# Plot a single tree from the Random Forest
plt.figure(figsize=(20, 10))
rf_tree = random_forest.estimators_[0]  # Select the first tree in the
Random Forest
plot_tree(rf_tree, filled=True, feature_names=features,
class_names=None, max_depth=3)
plt.title("Single Tree from Random Forest Regressor")
plt.show()
```



Single Tree from Random Forest Regressor