Name: Satwik Shirpurwar

Roll No: 52

Batch: A3

Subject: Machine Learning Lab

Sec: A

Experiment No-3

The customer dataset encompasses their purchasing patterns across diverse attributes, intended to aid data scientists and analysts in comprehending the determinants impacting buying choices. It includes demographic details, buying behaviours, and pertinent features. Develop a machine learning model that predicts whether an individual will purchase the product or not.

Perform the EDA

Apply logistic regression

Apply Decision tree algorithm

Apply KNN

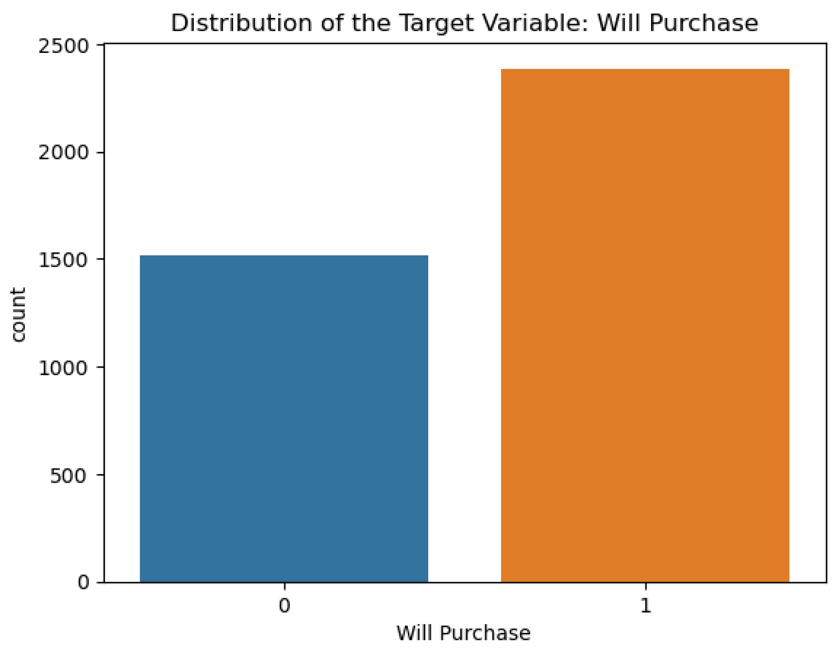
Evaluate the performance using Precision, Recall, F1 score and accuracy.

Apply hyperparameter tuning to improve performance.

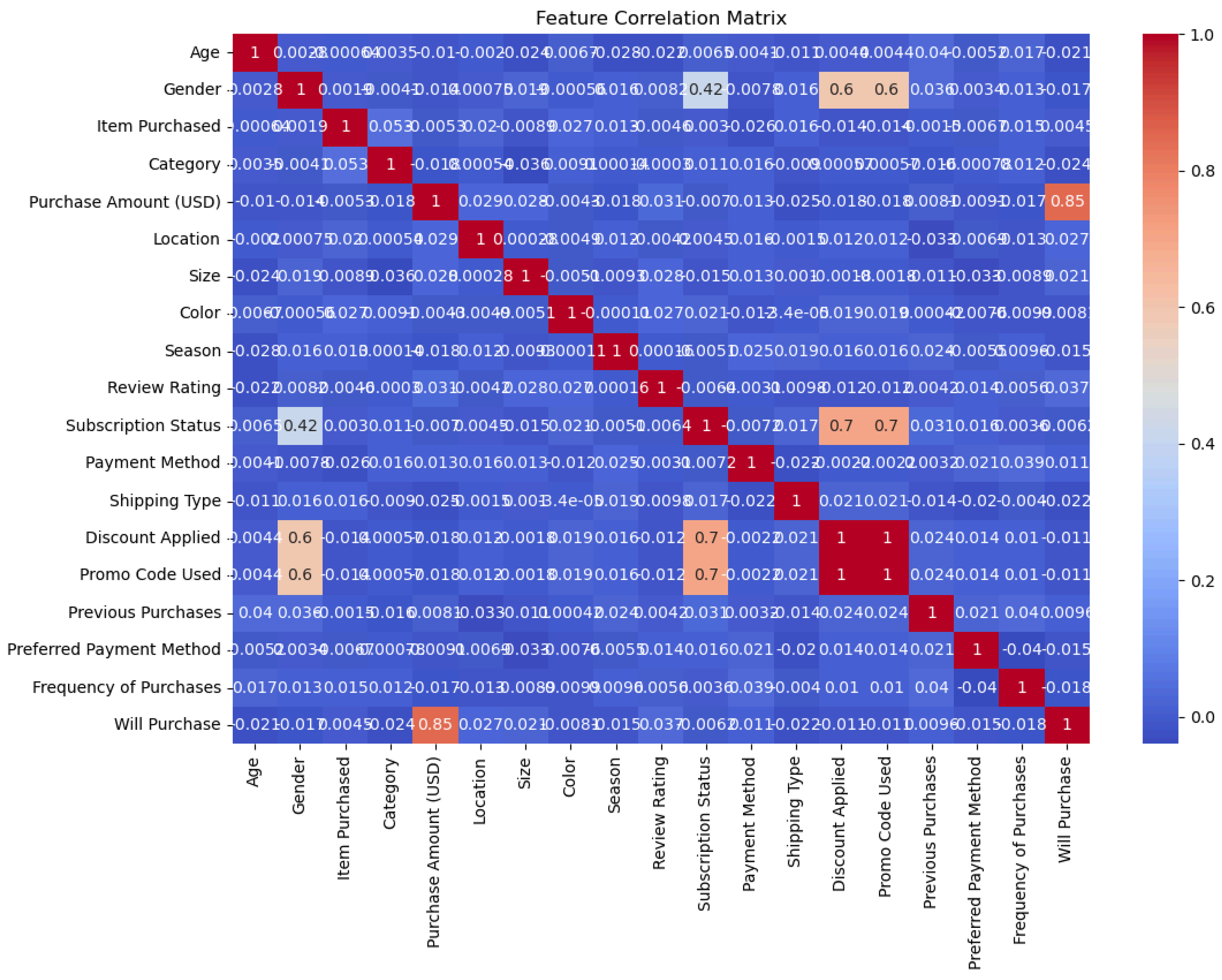
In [19]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the dataset
file_path = 'shopping_trends.csv'  # Replace with the actual file path
data = pd.read_csv(file_path)
```

In [20]:
```
# Step 1: Data Preparation
# Creating a binary target variable based on the 'Purchase Amount (USD)' column
threshold = 50
data['Will Purchase'] = data['Purchase Amount (USD)'].apply(lambda x: 1 if x > threshold else 0)

# Dropping the 'Customer ID' as it is not useful for prediction
data = data.drop(['Customer ID'], axis=1)
```

In [21]:
```
# Step 2: Exploratory Data Analysis (EDA)
# Check the distribution of the target variable
sns.countplot(x='Will Purchase', data=data)
plt.title('Distribution of the Target Variable: Will Purchase')
plt.show()
```



In [24]:
```
# Visualizing correlations (if numerical features are present)
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Feature Correlation Matrix')
plt.show()

# Step 3: Handling Categorical Variables
categorical_cols = data.select_dtypes(include=['object']).columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
```



In [25]:
```
# Separating features and target
X = data.drop('Will Purchase', axis=1)
y = data['Will Purchase']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Feature scaling for KNN
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [27]:
```
# Step 4: Applying Machine Learning Models

# 4.1 Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred_logreg = log_reg.predict(X_test)

# 4.2 Decision Tree
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
y_pred_tree = decision_tree.predict(X_test)

# 4.3 K-Nearest Neighbors (KNN)
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

In [29]:
```
# Step 5: Model Evaluation
def evaluate_model(y_test, y_pred, model_name):
    print(f"{model_name} Metrics:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
    print(f"Precision: {precision_score(y_test, y_pred)}")
    print(f"Recall: {recall_score(y_test, y_pred)}")
    print(f"F1 Score: {f1_score(y_test, y_pred)}")
    print("-" * 30)

# Evaluate all models
evaluate_model(y_test, y_pred_logreg, "Logistic Regression")
evaluate_model(y_test, y_pred_tree, "Decision Tree")
evaluate_model(y_test, y_pred_knn, "KNN")
```

```
Logistic Regression Metrics:
Accuracy: 0.9965811965811966
Precision: 0.9971223021582734
Recall: 0.9971223021582734
F1 Score: 0.9971223021582734
------------------------------
Decision Tree Metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
------------------------------
KNN Metrics:
Accuracy: 0.8452991452991453
Precision: 0.8454301075268817
Recall: 0.9050359712230216
F1 Score: 0.8742182070882558
------------------------------
```

In [30]:
```
# Step 6: Hyperparameter Tuning (for KNN as an example)
param_grid = {'n_neighbors': np.arange(1, 25), 'weights': ['uniform', 'distance']}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='f1')
grid_search.fit(X_train, y_train)

# Best parameters
print(f"Best Parameters for KNN: {grid_search.best_params_}")

# Using the best KNN model
best_knn = grid_search.best_estimator_
y_pred_best_knn = best_knn.predict(X_test)

# Evaluation of the tuned KNN model
evaluate_model(y_test, y_pred_best_knn, "Tuned KNN")
```

```
Best Parameters for KNN: {'n_neighbors': 24, 'weights': 'uniform'}
Tuned KNN Metrics:
Accuracy: 0.917094017094017
Precision: 0.90625
Recall: 0.9597122302158273
```

```
F1 Score: 0.9322152341020264
------------------------------
```

In [ ]: