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 = 50data['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() Distribution of the Target Variable: Will Purchase 2500 2000 1500 1000 500 Will Purchase 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 Feature Correlation Matrix 1 0.00208.0006040035-0.01-0.0020.0240.00670.0280.0220.00615.00410.0110.00469.00440.04-0.00520.017-0.021 Gender -0.0028 1 0.00190.004-D.0140.00076.01-90.00056.01-60.0082-0.42-D.00780.016 0.6 0.6 0.03-60.003-40.013-0.017 Item Purchased -.000640019 1 0.0530.00530.02-0.00890.0270.0130.00460.003-0.0260.016-0.0140.0140.0016.00670.0150.0045 Category - 0.003-0.004 D.053 1 -0.01&0005-0.0360.0090.000194.00030.0110.016-0.0090.0000570005-70.0165.0007&012-0.024 - 0.8 Purchase Amount (USD) -0.01-0.01-0.005-0.018 1 0.0290.0280.004-0.0180.031-0.0070.013-0.0250.0180.0180.008-0.009-0.017 0.85 Location -0.002.000750.020.00054.029 1 0.00028.00490.0120.00402.00450.0160.00150.0120.012-0.0330.00640.0130.027 Size -0.0240.0190.00890.0360.028.00028 1 -0.0050.00930.0280.0150.0130.0010.00180.00180.0110.0330.00890.021 - 0.6 Color - 0.00607.000566.0270.009-D.00469.0049.005 1 1 - 0.000101.0270.021-0.01-23.4e-06.0190.0190.00402.00760.0099.008 Season -0.0280.0160.016.0001-4.0180.0120.009080001110.000106.00510.0250.0190.0160.0160.0240.00505.00960.015 Review Rating -0.0220.00820.0046.00020.0310.00420.0280.0270.00016 1 -0.0064.00301.00980.0120.0120.00420.0140.00560.037 Subscription Status -0.00650.42 0.0030.011-0.0070.00450.0150.0210.0050.0064 1 -0.00720.017 0.7 0.7 0.0310.0160.00360.006 - 0.4 Payment Method - 0.004 D.007 80.0260.0160.0130.0160.013-0.0120.0250.003 0.007 2 1 -0.02 20.002 0.002 0.003 20.0210.0390.011 Shipping Type -0.0110.0160.0160.0090.0250.00150.003.4e-05.0190.00980.017-0.022 1 0.0210.021-0.014-0.02-0.0040.022 Discount Applied -3.0044 0.6 -0.014.0005-0.0180.0120.00180.0190.0160.012 0.7-0.00220.021 1 1 0.0240.014 0.01-0.011 Promo Code Used -0.0044 0.6 -0.014.0005-0.0180.0120.00180.0190.0160.012 0.7 -0.00220.021 1 0.00240.014 0.01-0.011 - 0.2 Previous Purchases - 0.04 0.03@0.00150.01@.00810.0330.010.000410.0240.00420.0310.00320.0140.0240.024 Preferred Payment Method -0.00502.003-0.00607.00070800901.00600.0330.0076.00550.0140.0160.021-0.020.0140.0140.021 1 -0.04-0.015 Frequency of Purchases -0.0170.0130.0150.0120.0170.0130.0089.0099.0096.0056.00360.0390.004 0.01 0.01 0.04 -0.04 1 -0.018 - 0.0 Will Purchase -0.0210.0170.00450.024 0.85 0.0270.0210.00810.0150.0370.00620.011-0.022-0.0110.0110.00960.0150.018 of Purchases Location Amount (USD) Size 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