Predicting Customer Purchase

In this assignment we are going to classify the customer according to his purchasing pattern. We will study the customer behaviour patterns and will categorize them into categories based on within how many days they make their purchases.Then we will train a classification model that will classify which customer belongs to which category.

In [41]:
```
#loading the required packages

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
```

In [2]:
```
# Load the datasets
online_sales = pd.read_csv('C:\\Users\\sujoydutta\\Desktop\\Data analysis\\Projects\\Marketing insights\\Online_
customer_data = pd.read_excel('C:\\Users\\sujoydutta\\Desktop\\Data analysis\\Projects\\Marketing insights\\Cus
discount_coupons = pd.read_csv('C:\\Users\\sujoydutta\\Desktop\\Data analysis\\Projects\\Marketing insights\\Di
marketing_spend= pd.read_csv('C:\\Users\\sujoydutta\\Desktop\\Data analysis\\Projects\\Marketing insights\\Mark
tax_amount = pd.read_excel('C:\\Users\\sujoydutta\\Desktop\\Data analysis\\Projects\\Marketing insights\\Tax_am
```

In [3]:
```
# Merging datasets using 'CustomerID' and 'Product Category as the common key
merged_data = pd.merge(online_sales, customer_data, on='CustomerID', how='left')
merged_data = pd.merge(merged_data, discount_coupons, on='Product_Category', how='left')
merged_data = pd.merge(merged_data, tax_amount, on='Product_Category', how='left')
```

In [4]:
```
# Convert 'Date' column in the marketing_spend DataFrame to datetime data type
marketing_spend['Date'] = pd.to_datetime(marketing_spend['Date'])
merged_data['Transaction_Date'] = pd.to_datetime(merged_data['Transaction_Date'], format='%Y%m%d')

# Merge datasets using 'Transaction_Date' as the common key
merged_data = pd.merge(merged_data, marketing_spend, left_on='Transaction_Date', right_on='Date', how='left')


# Dropping the duplicate 'Date' column if needed
merged_data.drop(columns=['Date'], inplace=True)
```

In [5]:
```
# Calculating Invoice Value for each transaction in the merged dataset
merged_data['Invoice_Value'] = (
    (merged_data['Quantity'] * merged_data['Avg_Price']) *
    (1 - merged_data['Discount_pct']) *
    (1 + merged_data['GST']) +
    merged_data['Delivery_Charges']
).abs()
```

In [6]:
```
#examining the merged dataset
merged_data.head()
```

Out[6]:

	CustomerID	Transaction_ID	Transaction_Date	Product_SKU	Product_name	Product_Category	Quantity	Avg_Price	Delivery_Charges	Cc
0	17850	16679	2019-01-01	GGOENEBJ079499	Nest Learning Thermostat 3rd Gen-USA - Stainle...	Nest-USA	1	153.71	6.5	
1	17850	16679	2019-01-01	GGOENEBJ079499	Nest Learning Thermostat 3rd Gen-USA - Stainle...	Nest-USA	1	153.71	6.5	
2	17850	16679	2019-01-01	GGOENEBJ079499	Nest Learning Thermostat 3rd Gen-USA - Stainle...	Nest-USA	1	153.71	6.5	
3	17850	16679	2019-01-01	GGOENEBJ079499	Nest Learning Thermostat 3rd Gen-USA - Stainle...	Nest-USA	1	153.71	6.5	
4	17850	16679	2019-01-01	GGOENEBJ079499	Nest Learning Thermostat 3rd Gen-USA - Stainle...	Nest-USA	1	153.71	6.5	

5 rows × 21 columns

In [7]:
```
#correcting the format
merged_data['Transaction_Date'] = pd.to_datetime(merged_data['Transaction_Date'])
merged_data['Transaction_Date']
```

Out[7]:
```
0        2019-01-01
1        2019-01-01
2        2019-01-01
3        2019-01-01
4        2019-01-01
            ...
630683   2019-12-31
630684   2019-12-31
630685   2019-12-31
630686   2019-12-31
630687   2019-12-31
Name: Transaction_Date, Length: 630688, dtype: datetime64[ns]
```

In [18]:
```
#  Calculating Recency for Each Transaction
merged_data['Transaction_Date'] = pd.to_datetime(merged_data['Transaction_Date'])
merged_data['Recency'] = merged_data.groupby('CustomerID')['Transaction_Date'].diff().dt.days
merged_data['Recency'] = merged_data['Recency'].fillna(0).astype(int)
merged_data['Recency']
```

Out[18]:
```
1057        0
2393        0
3281        0
4866        0
4926        1
         ...
628492    163
629152      1
629224      0
629884      0
629920     12
Name: Recency, Length: 1740, dtype: int32
```

In [19]:
```
#  Dropping rows where Recency is less than 1
merged_data = merged_data[merged_data['Recency'] > 0]
```

In [20]:
```
#Identify Repeat Customers
repeat_customers = merged_data.groupby('CustomerID')['Transaction_ID'].count() > 1
merged_data['IsRepeatCustomer'] = merged_data['CustomerID'].map(repeat_customers)
merged_data['IsRepeatCustomer']
```

```
C:\Users\sujoydutta\AppData\Local\Temp\ipykernel_14228\2896614341.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  merged_data['IsRepeatCustomer'] = merged_data['CustomerID'].map(repeat_customers)
```

Out[20]:
```
4926       True
23505      True
23601      True
27156      True
31550      True
          ...
627160    False
628348     True
628492     True
629152     True
629920     True
Name: IsRepeatCustomer, Length: 1006, dtype: bool
```

In [21]:
```
#seeing how many repeat customers
merged_data['IsRepeatCustomer'].value_counts()
```

Out[21]:
```
True     845
False    161
Name: IsRepeatCustomer, dtype: int64
```

In [22]:
```
#Calculate Average Days per Transaction for Repeat Customers
avg_days_per_transaction = merged_data[merged_data['IsRepeatCustomer']].groupby('CustomerID')['Recency'].mean()
avg_days_per_transaction
```

Out[22]:
```
CustomerID
12383     35.500000
12431    107.333333
12471     26.444444
12474     15.750000
12481     56.000000
            ...
18092     27.000000
18109     34.500000
18116     19.750000
18118     70.666667
18223     28.333333
Name: Recency, Length: 209, dtype: float64
```

In [23]:
```
# Creating Categorical Labels (0-30 days, 30-60 days, 60-90 days, 90+ days)
bins = [0, 30, 60, 90,np.inf]
labels = ['0-30 days', '30-60 days', '60-90 days', '90+ days']
AvgDaysPerTransactionCategory = pd.cut(avg_days_per_transaction, bins=bins, labels=labels)
```

In [24]:
```
#viewing the new variable
AvgDaysPerTransactionCategory.value_counts()
```

Out[24]:
```
30-60 days    86
0-30 days     55
60-90 days    48
90+ days      20
Name: Recency, dtype: int64
```

In [25]:
```
# Create a new variable 'RecencyCategory' based on Recency
merged_data['RecencyCategory'] = pd.cut(merged_data['Recency'], bins=bins, labels=labels, right=False)
merged_data['RecencyCategory']
```

```
C:\Users\sujoydutta\AppData\Local\Temp\ipykernel_14228\3764021990.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  merged_data['RecencyCategory'] = pd.cut(merged_data['Recency'], bins=bins, labels=labels, right=False)
```

Out[25]:
```
4926      0-30 days
23505     0-30 days
23601     0-30 days
27156     0-30 days
31550     0-30 days
            ...
627160     90+ days
628348    0-30 days
628492     90+ days
629152    0-30 days
629920    0-30 days
Name: RecencyCategory, Length: 1006, dtype: category
Categories (4, object): ['0-30 days' < '30-60 days' < '60-90 days' < '90+ days']
```

In [26]:
```
#examining the new dataset
merged_data.head()
```

Out[26]:

	CustomerID	Transaction_ID	Transaction_Date	Product_SKU	Product_name	Product_Category	Quantity	Avg_Price	Delivery_Charge
4926	17850	16984	2019-01-04	GGOENEBJ079499	Nest Learning Thermostat 3rd Gen-USA - Stainle...	Nest-USA	1	153.71	6
23505	17850	18017	2019-01-16	GGOEAAEB031614	Android Men's 3/4 Sleeve Raglan Henley Black	Apparel	1	20.62	6
23601	17850	18021	2019-01-17	GGOENEBJ079499	Nest Learning Thermostat 3rd Gen-USA - Stainle...	Nest-USA	1	153.71	6
27156	17850	18208	2019-01-18	GGOEGOCT019199	Red Spiral Google Notebook	Office	155	9.97	6
31550	17850	18492	2019-01-21	GGOENEBQ079099	Nest Protect Smoke + CO White Battery Alarm-USA	Nest-USA	4	81.50	6

5 rows × 25 columns

In [27]:
```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1006 entries, 4926 to 629920
Data columns (total 25 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   CustomerID                     1006 non-null   int64
 1   Transaction_ID                 1006 non-null   int64
 2   Transaction_Date               1006 non-null   datetime64[ns]
 3   Product_SKU                    1006 non-null   object
 4   Product_name                   1006 non-null   object
 5   Product_Category               1006 non-null   object
 6   Quantity                       1006 non-null   int64
 7   Avg_Price                      1006 non-null   float64
 8   Delivery_Charges               1006 non-null   float64
 9   Coupon_Status                  1006 non-null   object
 10  Gender                         1006 non-null   object
 11  Location                       1006 non-null   object
 12  Tenure_Months                  1006 non-null   int64
 13  Month                          1002 non-null   object
 14  Coupon_Code                    1002 non-null   object
 15  Discount_pct                   1002 non-null   float64
 16  GST                            1006 non-null   float64
 17  Offline_Spend                  1006 non-null   int64
 18  Online_Spend                   1006 non-null   float64
 19  Total                          1006 non-null   float64
 20  Invoice_Value                  1002 non-null   float64
 21  Recency                        1006 non-null   int32
 22  IsRepeatCustomer               1006 non-null   bool
 23  AvgDaysPerTransactionCategory  0 non-null      category
 24  RecencyCategory                1006 non-null   category
dtypes: bool(1), category(2), datetime64[ns](1), float64(7), int32(1), int64(5), object(8)
memory usage: 180.2+ KB
```

In [28]:
```
#dropping useless columns
subset=merged_data.drop(['Product_SKU','Coupon_Status','Month','Total','AvgDaysPerTransactionCategory','Product_
subset
```

Out[28]:

	CustomerID	Transaction_ID	Transaction_Date	Product_Category	Quantity	Avg_Price	Delivery_Charges	Gender	Location	Tenure_Mo
4926	17850	16984	2019-01-04	Nest-USA	1	153.71	6.5	M	Chicago	
23505	17850	18017	2019-01-16	Apparel	1	20.62	6.5	M	Chicago	
23601	17850	18021	2019-01-17	Nest-USA	1	153.71	6.5	M	Chicago	
27156	17850	18208	2019-01-18	Office	155	9.97	6.5	M	Chicago	
31550	17850	18492	2019-01-21	Nest-USA	4	81.50	6.5	M	Chicago	
...	...	...	...	...	...	...	...	...	...	
627160	14810	48258	2019-12-28	Nest-USA	1	151.88	6.5	M	California	
628348	16525	48340	2019-12-29	Apparel	1	61.15	6.0	F	California	
628492	15808	48349	2019-12-29	Nest-USA	2	121.30	6.5	F	California	
629152	15808	48393	2019-12-30	Nest	5	49.95	6.5	F	California	
629920	14606	48447	2019-12-31	Drinkware	1	10.59	6.5	F	Chicago	

1006 rows × 18 columns

In [37]:
```
# Creating an instance of LabelEncoder
label_encoder = LabelEncoder()

# List of categorical columns to encode
categorical_columns = ['Product_Category', 'Gender', 'Location']

# Apply Label Encoding to each categorical column
for col in categorical_columns:
    subset[col] = label_encoder.fit_transform(subset[col])
```

In [42]:
```
# Selecting Features and Target Variable
features = ['CustomerID', 'Transaction_ID', 'Product_Category', 'Quantity', 'Avg_Price', 'Delivery_Charges', 'G
X = subset[features]
y = subset['RecencyCategory']
```

In [43]:
```
# Creating an imputer instance
imputer = SimpleImputer(strategy='mean')

# Fit and transform the imputer on your feature data
X = imputer.fit_transform(X)
```

In [44]:
```
#Splitting the Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [45]:
```
#  Training the Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
```

Out[45]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [46]:
```
#Making the Predictions
y_pred = clf.predict(X_test)
y_pred
```

Out[46]:
```
array(['0-30 days', '30-60 days', '30-60 days', '0-30 days', '30-60 days',
       '90+ days', '0-30 days', '60-90 days', '0-30 days', '0-30 days',
       '0-30 days', '30-60 days', '0-30 days', '0-30 days', '0-30 days',
       '30-60 days', '0-30 days', '90+ days', '0-30 days', '0-30 days',
       '30-60 days', '90+ days', '0-30 days', '0-30 days', '30-60 days',
       '0-30 days', '0-30 days', '60-90 days', '90+ days', '0-30 days',
       '0-30 days', '60-90 days', '0-30 days', '30-60 days', '30-60 days',
       '0-30 days', '90+ days', '0-30 days', '0-30 days', '0-30 days',
       '30-60 days', '90+ days', '0-30 days', '90+ days', '30-60 days',
       '0-30 days', '0-30 days', '90+ days', '90+ days', '30-60 days',
       '90+ days', '30-60 days', '90+ days', '30-60 days', '90+ days',
       '0-30 days', '0-30 days', '0-30 days', '0-30 days', '30-60 days',
       '90+ days', '60-90 days', '0-30 days', '90+ days', '0-30 days',
       '0-30 days', '0-30 days', '60-90 days', '0-30 days', '30-60 days',
       '90+ days', '30-60 days', '90+ days', '0-30 days', '60-90 days',
       '60-90 days', '0-30 days', '90+ days', '60-90 days', '0-30 days',
       '90+ days', '90+ days', '30-60 days', '0-30 days', '0-30 days',
       '90+ days', '90+ days', '0-30 days', '90+ days', '60-90 days',
       '0-30 days', '90+ days', '60-90 days', '0-30 days', '0-30 days',
       '30-60 days', '30-60 days', '0-30 days', '0-30 days', '0-30 days',
       '60-90 days', '90+ days', '60-90 days', '0-30 days', '30-60 days',
       '90+ days', '0-30 days', '0-30 days', '0-30 days', '90+ days',
       '60-90 days', '30-60 days', '90+ days', '0-30 days', '0-30 days',
       '0-30 days', '30-60 days', '60-90 days', '0-30 days', '30-60 days',
       '30-60 days', '30-60 days', '0-30 days', '90+ days', '30-60 days',
       '0-30 days', '0-30 days', '0-30 days', '0-30 days', '0-30 days',
       '90+ days', '30-60 days', '0-30 days', '0-30 days', '90+ days',
       '0-30 days', '60-90 days', '30-60 days', '0-30 days', '0-30 days',
       '0-30 days', '60-90 days', '30-60 days', '0-30 days', '0-30 days',
       '0-30 days', '0-30 days', '0-30 days', '90+ days', '90+ days',
       '0-30 days', '30-60 days', '0-30 days', '0-30 days', '30-60 days',
       '90+ days', '30-60 days', '90+ days', '0-30 days', '0-30 days',
       '0-30 days', '30-60 days', '0-30 days', '0-30 days', '60-90 days',
       '30-60 days', '0-30 days', '0-30 days', '0-30 days', '0-30 days',
       '0-30 days', '0-30 days', '0-30 days', '90+ days', '0-30 days',
       '30-60 days', '60-90 days', '90+ days', '60-90 days', '30-60 days',
       '90+ days', '0-30 days', '0-30 days', '0-30 days', '0-30 days',
       '60-90 days', '0-30 days', '60-90 days', '0-30 days', '90+ days',
       '30-60 days', '60-90 days', '0-30 days', '0-30 days', '60-90 days',
       '0-30 days', '30-60 days', '0-30 days', '30-60 days', '0-30 days',
       '60-90 days', '0-30 days'], dtype=object)
```

In [47]:
```
#  Model Performance Evaluation
accuracy = accuracy_score(y_test, y_pred)
classification_report_result = classification_report(y_test, y_pred)

# Printing the evaluation metrics
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report_result)
```

```
Accuracy: 1.0

Classification Report:
               precision    recall  f1-score   support

   0-30 days       1.00      1.00      1.00       100
  30-60 days       1.00      1.00      1.00        40
  60-90 days       1.00      1.00      1.00        24
    90+ days       1.00      1.00      1.00        38

    accuracy                           1.00       202
   macro avg       1.00      1.00      1.00       202
weighted avg       1.00      1.00      1.00       202
```

Remark: The model is 100% accurate and we can say that it can classify customers into recency categories perfectly.

In [ ]: