Customer Segmentation and Transaction Analysis for Business Insights

Part 1

Welcome to the first part of our project on Customer Segmentation and Transaction Analysis for Business Insights. These kinds of things are commonly built into software for users. There are many software that provide this kind of elaborate analysis, and nowadays, people are using ML to advance that.

In this project, we aim to transform transaction data to identify distinct customer segments and provide actionable insights for business decision-making. Understanding customer behavior is essential for businesses to tailor their marketing strategies, improve customer retention, and increase overall profitability.

In this part of the project, we will focus on using K-Means clustering to group customers based on their total spending. This technique will help us identify different customer segments and understand their purchasing patterns. By analyzing these clusters, businesses can create targeted marketing campaigns and personalized offers, enhancing customer satisfaction and loyalty.

We will be given invoice data, so in addition to standard preprocessing, we will also have to create our own customer level data. Simply put, we are given rows where each row is one purchase or one transaction, but we need one data point or one row to be one customer, and his/her details about purchases he/she made.

Now, let's walk through the skeleton code step by step.

Skeleton Code Explanation

In

Skeleton Code and Task Walkthrough

provided URL.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# URL of the dataset
data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online%20Retail.xlsx'

# TASK 1: Load the data into a DataFrame. Provide the correct method and arguments.
df = pd.read_excel(data_url)
```

Copy

Explanation: Load the dataset into a DataFrame using the appropriate method. This task ensures that the data is ready for processing.

Convert 'InvoiceDate' to Datetime Format

The InvoiceDate column needs to be converted to datetime format for accurate time-based calculations.

```
# TASK 2: Convert 'InvoiceDate' to datetime format.
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
```

Copy

Explanation: Convert the InvoiceDate column to a datetime format to facilitate calculations involving dates, such as purchase intervals.

Calculate the Total Bill for Each Transaction

Create a new column Total_Bill by multiplying the Quantity and UnitPrice columns.

```
# TASK 3: Calculate the total bill for each transaction.
df['Total_Bill'] = df['Quantity'] * df['UnitPrice']
```

Copy

Skeleton Code and Task Walkthrough

Group the data by CustomerID and aggregate to calculate total spending, first and last purchase dates, most common location, and top item for each customer.

```
customer_df = df.groupby('CustomerID').agg(
    Total_Bill_Size=('Total_Bill', 'sum'),
    First_Purchase=('InvoiceDate', 'min'),
    Last_Purchase=('InvoiceDate', 'max'),
    Most_Common_Location=('Country', lambda x: x.mode()[0]),
    Top_Item=('StockCode', lambda x: x.value_counts().idxmax())
)
```

Copy

Explanation: Aggregate the data at the customer level to summarize each customer's purchasing behavior, including total spending, purchase dates, and most common purchase locations and items.

Calculate the Purchase Interval in Days

Calculate the purchase interval by subtracting the first purchase date from the last purchase date.

```
# TASK 4: Calculate the purchase interval in days.
customer_df['Purchase_Interval_Days'] = (customer_df['Last_Purchase'] - customer_df['First_Purchase']).
```

Copy

Explanation: Calculate the number of days between the first and last purchases to understand the frequency of purchases for each customer.

Standardize the 'Total_Bill_Size' Feature

Use StandardScaler to standardize the Total_Bill_Size feature.

Skeleton Code and Task Walkthrough

this feature appropriately, preventing any bias due to scale differences.

Implement K-Means Clustering

Apply K-Means clustering to the standardized Total_Bill_Size feature. Choose an appropriate number of clusters and set a random state for reproducibility.

```
# TASK 5: Implement K-Means clustering. Choose an appropriate number of clusters and set a rando
kmeans = KMeans(n_clusters=5, random_state=42)
customer_df['Cluster'] = kmeans.fit_predict(customer_df[['Total_Bill_Size_Scaled']])
```

Explanation: Implement K-Means clustering to group customers based on their spending patterns. The number of clusters should be chosen based on analysis and business needs.

Summarize Cluster Information

Define a function to summarize cluster information, including customer count, average spend, top locations, and top items for each cluster.

```
# Function to summarize cluster information with improved readability
def summarize_cluster_info(clustered_df):
    for i in range(kmeans.n_clusters):
        cluster_data = clustered_df[clustered_df['Cluster'] == i]
        # TASK 6: Print a summary of each cluster. Include customer count, average spend, top locations, and top
        print(f"\nCluster {i} Summary:")
        print(f"Customer Count: {len(cluster_data)}")
        print(f"Average Spend: {cluster_data['Total_Bill_Size'].mean():.2f}")
        print(f"Most Common Location: {cluster_data['Most_Common_Location'].mode()[0]}")
        print(f"Top Item: {cluster_data['Top_Item'].mode()[0]}")
```

Skeleton Code and Task Walkthrough

```
# TASK 7: Look at the results, write a report around what you would conclude if you were working as the business ow
# Your business report here.
```

Explanation: Analyze the clusters to identify actionable insights and write a report that outlines these findings, along with recommendations for improving business strategies based on customer segmentation.

Part 2

Welcome to the second part of our project on Customer Segmentation and Transaction Analysis for Business Insights. In this part, we will use DBSCAN (Density-Based Spatial Clustering of Applications with Noise) to identify clusters and outliers based on customer spending and purchase intervals. DBSCAN is a powerful clustering technique that can handle clusters of varying shapes and sizes and is particularly effective at identifying outliers.

Unlike K-Means, which requires the number of clusters to be specified beforehand, DBSCAN automatically determines the number of clusters based on the density of points. This makes it a useful tool for discovering natural groupings in the data and identifying customers with unusual purchasing patterns.

Now, let's walk through the skeleton code step by step.

Skeleton Code Explanation

Import Libraries and Load Dataset

We start by importing the necessary libraries, including pandas for data manipulation, StandardScaler from sklearn.preprocessing for feature scaling, DBSCAN from sklearn.cluster for clustering, and matplotlib.pyplot for visualization. The dataset will be lo

Skeleton Code and Task Walkthrough

```
import matplotlib.pyplot as plt

# Load the data
data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online%20Retail.xlsx'
df = pd.read_excel(data_url)  # Modify if different method needed based on data format
```

Explanation: Load the dataset into a DataFrame using the appropriate method. This task ensures that the data is ready for processing.

Preprocess the Data

Convert the InvoiceDate column to datetime format. Calculate the Total_Bill for each transaction by multiplying the Quantity and UnitPrice columns.

Copy

```
# Preprocess the data
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
df['Total_Bill'] = df['Quantity'] * df['UnitPrice']
```

Explanation: Preprocess the data to ensure that it is in the correct format for aggregation and clustering.

Aggregate Data at the Customer Level

Group the data by CustomerID and aggregate to calculate total spending and purchase intervals for each customer.

Copy

```
# TASK 1: Aggregate data at the customer level to calculate 'Total_Bill_Size' and 'Purchase_Interval_Days'
customer_df = df.groupby('CustomerID').agg(
    Total_Bill_Size=('Total_Bill', 'sum'),
    First_Purchase=('InvoiceDate', 'min'),
    Last_Purchase=('InvoiceDate', 'max')
)
```

se'

Skeleton Code and Task Walkthrough

Use StandardScaler to standardize the Total_Bill_Size and Purchase_Interval_Days features.

```
# Normalize the features
scaler = StandardScaler()
# TASK 2: Standardize the 'Total_Bill_Size' and 'Purchase_Interval_Days' features using StandardScaler.
customer_df[['Total_B

ill_Size', 'Purchase_Interval_Days']] = scaler.fit_transform(customer_df[['Total_Bill_Size', 'Purchase_Int
```

Explanation: Standardize the features to ensure that the clustering algorithm treats these features appropriately, preventing any bias due to scale differences.

Apply DBSCAN

Initialize and fit the DBSCAN model with appropriate eps and min_samples parameters.

```
# Apply DBSCAN
# TASK 3: Initialize and fit the DBSCAN model with appropriate 'eps' and 'min_samples'.
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters = dbscan.fit_predict(customer_df[['Total_Bill_Size', 'Purchase_Interval_Days']])
```

Explanation: Apply DBSCAN to identify clusters and outliers based on the standardized features. The eps parameter controls the maximum distance between points in a cluster, and min_samples specifies the minimum number of points required to form a cluster.

Add Clusters Back to the DataFrame

Add the cluster labels back to the customer DataFrame.

```
# Add clusters back to the dataframe
```

Skeleton Code and Task Walkthrough

Print the count of customers in each cluster and identify noise points (marked as -1).

```
# Display clustering results
print("Cluster counts:")
print(customer_df['Cluster'].value_counts())

# Identify and display noise points
# TASK 4: Identify noise points (marked as -1), and count them.
noise_points = customer_df[customer_df['Cluster'] == -1]
print(f"Number of noise points: {len(noise_points)}")
```

Explanation: Display the results of the clustering, including the number of customers in each cluster and the identification of noise points.

Visualize the Results

Create a scatter plot to visualize the clusters.

```
# Visualizing the results
plt.scatter(customer_df['Total_Bill_Size'], customer_df['Purchase_Interval_Days'], c=clusters, cmap='v
plt.title('DBSCAN Clustering')
plt.xlabel('Scaled Total Bill Size')
plt.ylabel('Scaled Purchase Interval Days')
plt.colorbar(label='Cluster Label')
plt.show()
```

Explanation: Visualize the clustering results using a scatter plot to understand the distribution of clusters and outliers.

Summarize Clusters

Skeleton Code and Task Walkthrough

```
    for key, group in grouped:
        if key == -1:
            continue  # Skip the noise points for detailed summary
        print(f"\nCluster {key} Summary:")
        print(f"Number of Customers: {len(group)}")
        avg_bill = group['Total_Bill_Size'].mean()
        avg_interval = group['Purchase_Interval_Days'].mean()
        print(f"Average Total Bill: {avg_bill:.2f}")
        print(f"Average Purchase Interval: {avg_interval:.2f}")

# Call the summary function
summarize_clusters(customer_df)
```

Explanation: Summarize the clusters to provide insights into the characteristics of each customer segment.

Analyze and Discuss Results

Analyze the plot results and write a discussion on the distribution of clusters, any outliers identified, and potential reasons for the clustering pattern observed.

```
# TASK 5: Analyze the plot results and include a discussion in your final report.
# Your analysis and discussion here.
```

Copy

Explanation: Analyze the clustering results and write a discussion on the findings, providing insights and recommendations for business strategies based on the identified customer segments.

Next Item

Skeleton Code and Task Walkthrough

Submit

All Comments

Skeleton Code and Task Walkthrough

No comments yet