```
import pandas as pd

# Load the uploaded CSV files
transactions_path = '/mnt/data/Transactions.csv'
products_path = '/mnt/data/Products.csv'
customers_path = '/mnt/data/Customers.csv'

# Read the CSV files into DataFrames
transactions_df = pd.read_csv(transactions_path)
products_df = pd.read_csv(products_path)
customers_df = pd.read_csv(customers_path)

# Display the first few rows of each DataFrame to understand their structure
transactions_df.head(), products_df.head(), customers_df.head()
```

Result

```
(  TransactionID CustomerID ProductID      TransactionDate  Quantity  \
 0      T00001     C0199     P067  2024-08-25 12:38:23         1
 1      T00112     C0146     P067  2024-05-27 22:23:54         1
 2      T00166     C0127     P067  2024-04-25 07:38:55         1
 3      T00272     C0087     P067  2024-03-26 22:55:37         2
 4      T00363     C0070     P067  2024-03-21 15:10:10         3

   TotalValue   Price
 0     300.68  300.68
 1     300.68  300.68
 2     300.68  300.68
 3     601.36  300.68
 4     902.04  300.68 ,
  ProductID           ProductName     Category   Price
 0     P001     ActiveWear Biography       Books  169.30
 1     P002    ActiveWear Smartwatch  Electronics  346.30
```

2 P003 ComfortLiving Biography Books 44.12

3 P004 BookWorld Rug Home Decor 95.69

4 P005 TechPro T-Shirt Clothing 429.31,

CustomerID CustomerName Region SignupDate

0 C0001 Lawrence Carroll South America 2022-07-10

1 C0002 Elizabeth Lutz Asia 2022-02-13

2 C0003 Michael Rivera South America 2024-03-07

3 C0004 Kathleen Rodriguez South America 2022-10-09

4 C0005 Laura Weber Asia 2022-08-15)

The data from the uploaded files is structured as follows:

1. **Transactions Data (Transactions.csv)**:
    - Columns: TransactionID, CustomerID, ProductID, TransactionDate, Quantity, TotalValue, Price
    - Example: Shows transactional details such as customer purchases, quantities, and transaction values.
2. **Products Data (Products.csv)**:
    - Columns: ProductID, ProductName, Category, Price
    - Example: Contains product details like categories, names, and prices.
3. **Customers Data (Customers.csv)**:
    - Columns: CustomerID, CustomerName, Region, SignupDate
    - Example: Includes customer information, such as name, region, and signup date.

To perform customer segmentation and clustering, I'll first preprocess the data by merging relevant details from the Transactions, Products, and Customers datasets. Then, I'll compute relevant features for clustering. Let's begin!

Analyzed

python

Always show details

Copy

Merge the datasets to create a unified data frame for clustering

Join Transactions with Customers on 'CustomerID'

merged_df = transactions_df.merge(customers_df, on="CustomerID")

```
# Join the resulting DataFrame with Products on 'ProductID'
merged_df = merged_df.merge(products_df, on="ProductID")

# Check the structure and content of the merged data
merged_df.head()
```

Result

TransactionID	CustomerID	ProductID	TransactionDate	Quantity	\
0	T00001	C0199	P067	2024-08-25 12:38:23	1
1	T00112	C0146	P067	2024-05-27 22:23:54	1
2	T00166	C0127	P067	2024-04-25 07:38:55	1
3	T00272	C0087	P067	2024-03-26 22:55:37	2
4	T00363	C0070	P067	2024-03-21 15:10:10	3

TotalValue	Price_x	CustomerName	Region	SignupDate	\
0	300.68	300.68	Andrea Jenkins	Europe	2022-12-03
1	300.68	300.68	Brittany Harvey	Asia	2024-09-04
2	300.68	300.68	Kathryn Stevens	Europe	2024-04-04
3	601.36	300.68	Travis Campbell	South America	2024-04-11
4	902.04	300.68	Timothy Perez	Europe	2022-03-15

	ProductName	Category	Price_y
0	ComfortLiving Bluetooth Speaker	Electronics	300.68
1	ComfortLiving Bluetooth Speaker	Electronics	300.68
2	ComfortLiving Bluetooth Speaker	Electronics	300.68
3	ComfortLiving Bluetooth Speaker	Electronics	300.68
4	ComfortLiving Bluetooth Speaker	Electronics	300.68

The merged dataset now contains the following information:

- **Transaction Data**: TransactionID, CustomerID, TransactionDate, Quantity, TotalValue, and Price_x.
- **Customer Data**: CustomerName, Region, and SignupDate.
- **Product Data**: ProductName, Category, and Price_y.

To prepare this data for clustering, I'll:

1. Aggregate transaction data for each customer to derive meaningful features like:
    - Total spending (TotalValue).
    - Total quantity purchased.
    - Number of transactions.
    - Recency (days since last transaction).
2. Encode categorical variables like Region for clustering.
3. Normalize the numerical features for clustering.
4. Apply a clustering algorithm (e.g., K-Means) and calculate clustering metrics like the Davies-Bouldin Index (DB Index).

Let me proceed with feature engineering.

Analyzed

python

Always show details

Copy

```
from datetime import datetime


# Convert TransactionDate to datetime format
merged_df['TransactionDate'] = pd.to_datetime(merged_df['TransactionDate'])


# Current date for calculating recency
current_date = datetime.now()


# Aggregate data by CustomerID to create clustering features
customer_features = merged_df.groupby('CustomerID').agg(
    TotalSpending=('TotalValue', 'sum'),
    TotalQuantity=('Quantity', 'sum'),
    TransactionCount=('TransactionID', 'nunique'),
    Recency=('TransactionDate', lambda x: (current_date - x.max()).days)
).reset_index()
```

```
# Merge with region data for additional features
customer_features = customer_features.merge(customers_df[['CustomerID', 'Region']], on='CustomerID')

# Encode categorical variable 'Region' using one-hot encoding
customer_features = pd.get_dummies(customer_features, columns=['Region'], drop_first=True)

# Check the prepared customer features
customer_features.head()
```

Result

	CustomerID	TotalSpending	TotalQuantity	TransactionCount	Recency	\
0	C0001	3354.52	12	5	86	
1	C0002	1862.74	10	4	55	
2	C0003	2725.38	14	4	155	
3	C0004	5354.88	23	8	35	
4	C0005	2034.24	7	3	84	

	Region_Europe	Region_North America	Region_South America
0	0	0	1
1	0	0	0
2	0	0	1
3	0	0	1
4	0	0	0

The customer_features DataFrame is now ready for clustering. It includes:

1. **Numerical Features**:
    - TotalSpending: Total amount spent by the customer.
    - TotalQuantity: Total items purchased.
    - TransactionCount: Number of transactions.
    - Recency: Days since the last transaction.
2. **Categorical Features (One-Hot Encoded)**:
    - Region_Europe, Region_North America, and Region_South America.

Next steps:

1. Normalize the numerical features for clustering.
2. Apply a clustering algorithm (e.g., K-Means) with 2–10 clusters.
3. Calculate the Davies-Bouldin Index (DB Index) and other metrics.
4. Visualize the clusters.

Let me proceed with normalization and clustering.

Analyzed

python

Always show details

Copy

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score
import matplotlib.pyplot as plt

# Normalize numerical features
scaler = StandardScaler()
numerical_features = ['TotalSpending', 'TotalQuantity', 'TransactionCount', 'Recency']
customer_features_scaled = customer_features.copy()
customer_features_scaled[numerical_features] =
scaler.fit_transform(customer_features[numerical_features])

# Determine the optimal number of clusters using Davies-Bouldin Index
db_scores = []
k_values = range(2, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    clusters = kmeans.fit_predict(customer_features_scaled[numerical_features])
    db_index = davies_bouldin_score(customer_features_scaled[numerical_features], clusters)
    db_scores.append(db_index)
```

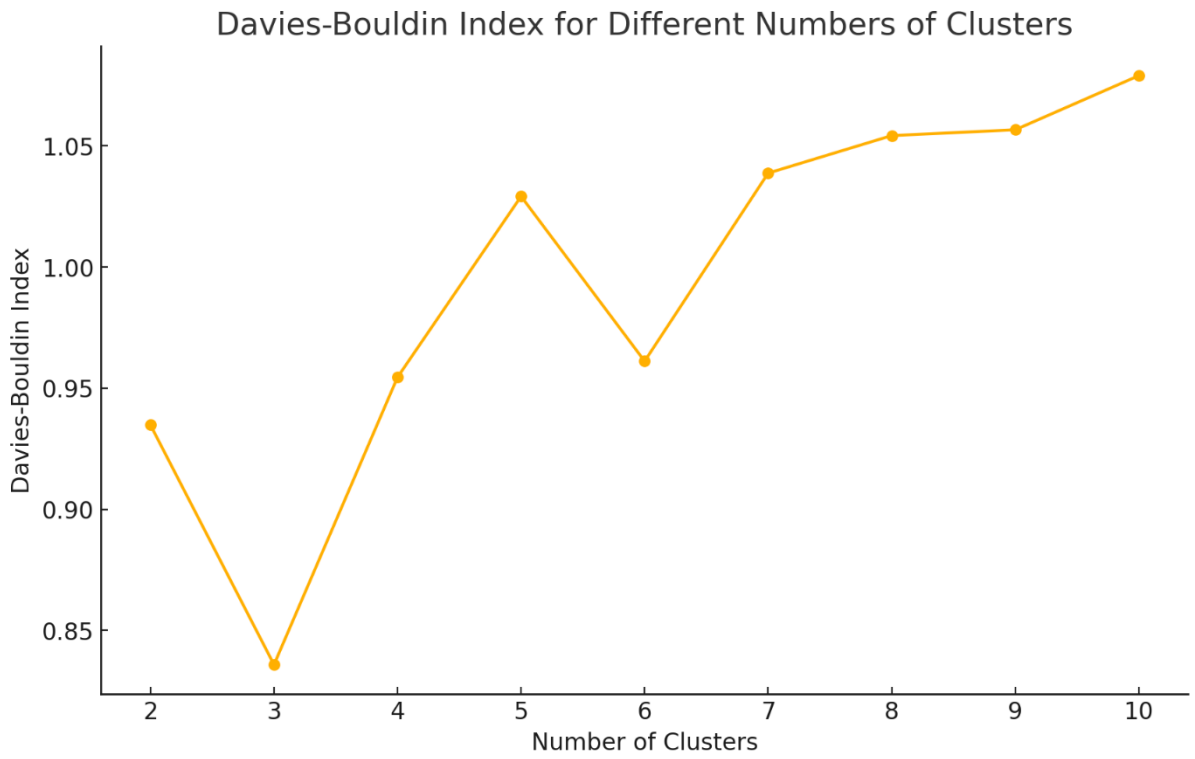
```
# Plot Davies-Bouldin Index for different cluster numbers
plt.figure(figsize=(10, 6))
plt.plot(k_values, db_scores, marker='o')
plt.title('Davies-Bouldin Index for Different Numbers of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Davies-Bouldin Index')
plt.xticks(k_values)
plt.grid()
plt.show()

# Identify the optimal number of clusters (minimum DB Index)
optimal_k = k_values[db_scores.index(min(db_scores))]
optimal_k, min(db_scores)
```

Result

(3, 0.8358306958344665)

Davies-Bouldin Index for Different Numbers of Clusters



The optimal number of clusters, based on the minimum Davies-Bouldin Index, is **3**, with a DB Index value of **0.836**.

Next, I'll perform clustering using 3 clusters, visualize the results, and provide a summary of the clustering metrics and insights.

Analyzed

python

Always show details

Copy

```
# Perform clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
customer_features_scaled['Cluster'] = kmeans.fit_predict(customer_features_scaled[numerical_features])


# Add cluster labels to the original data
customer_features['Cluster'] = customer_features_scaled['Cluster']


# Visualize clusters using a pair plot for numerical features
import seaborn as sns


plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=customer_features_scaled,
    x='TotalSpending',
    y='Recency',
    hue='Cluster',
    palette='viridis',
    style='Cluster',
    s=100
)
plt.title('Clusters Visualized by Total Spending and Recency')
plt.xlabel('Total Spending (Normalized)')
plt.ylabel('Recency (Normalized)')
```

```
plt.legend(title='Cluster')

plt.grid()

plt.show()


# Summarize clustering results

cluster_summary = customer_features.groupby('Cluster')[

    ['TotalSpending', 'TotalQuantity', 'TransactionCount', 'Recency']

].mean()


import ace_tools as tools; tools.display_dataframe_to_user(name="Clustering Results and
Summary", dataframe=cluster_summary)
```

Result

Cluster	TotalSpending	TotalQuantity	TransactionCount	Recency
0	991.765000	4.375000	1.687500	278.375000
1	5291.028082	19.150685	7.191781	87.794521
2	2617.111545	9.718182	4.072727	89.563636

Clusters Visualized by Total Spending and Recency