Assignment Tasks: Task 1: Exploratory Data Analysis (EDA) and Business Insights

Deliverables:

1. A Jupyter Notebook/Python script containing your EDA code

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the dataset
file_path = "/content/Transactions.csv"
df = pd.read_csv(file_path)

# Step 2: Inspect the data
print("Data Overview:")
print(df.info())
print("\nSummary Statistics:")
print(df.describe())

# Step 3: Check for missing values
missing_values = df.isnull().sum()
print("\nMissing Values:\n", missing_values)

# Step 4: Convert TransactionDate to datetime with the correct format
df['TransactionDate'] = pd.to_datetime(df['TransactionDate'], format='%Y-%m-%d %H:%M:%S')

# Step 5: Add derived columns
# Extract month and hour from TransactionDate
df['Month'] = df['TransactionDate'].dt.month
df['Hour'] = df['TransactionDate'].dt.hour

# Step 6: Analyze most popular products
popular_products = df.groupby('ProductID').agg({
    'Quantity': 'sum',
```

```
    'TotalValue': 'sum'
}).sort_values(by='Quantity', ascending=False).reset_index()

print("\nMost Popular Products by Quantity Sold:")
print(popular_products.head())

# Step 7: Analyze monthly sales trends
monthly_sales = df.groupby('Month')['TotalValue'].sum().reset_index()

# Step 8: Plot EDA results
# Plot 1: Monthly sales trend
plt.figure(figsize=(8, 5))
sns.barplot(x='Month', y='TotalValue', data=monthly_sales, palette='viridis')
plt.title('Monthly Sales Trend')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.show()

# Plot 2: Top 10 most popular products
plt.figure(figsize=(8, 5))
sns.barplot(x='Quantity', y='ProductID', data=popular_products.head(10), palette='coolwarm')
plt.title('Top 10 Most Popular Products by Quantity')
plt.xlabel('Quantity Sold')
plt.ylabel('Product ID')
plt.show()

# Plot 3: Hourly sales distribution
hourly_sales = df.groupby('Hour')['TotalValue'].sum().reset_index()
plt.figure(figsize=(8, 5))
sns.lineplot(x='Hour', y='TotalValue', data=hourly_sales, marker='o', color='b')
plt.title('Hourly Sales Distribution')
plt.xlabel('Hour of the Day')
plt.ylabel('Total Sales')
plt.show()
```

output:

```
Data Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   TransactionID   1000 non-null   object 
 1   CustomerID      1000 non-null   object 
 2   ProductID       1000 non-null   object 
 3   TransactionDate 1000 non-null   object 
 4   Quantity        1000 non-null   int64  
 5   TotalValue      1000 non-null   float64
 6   Price           1000 non-null   float64
dtypes: float64(2), int64(1), object(4)
memory usage: 54.8+ KB
None

Summary Statistics:
          Quantity   TotalValue       Price
count  1000.000000  1000.000000  1000.00000
mean      2.537000   689.995560   272.55407
std       1.117981   493.144478   140.73639
min       1.000000    16.080000    16.08000
25%       2.000000   295.295000   147.95000
50%       3.000000   588.880000   299.93000
75%       4.000000  1011.660000   404.40000
max       4.000000  1991.040000   497.76000

Missing Values:
 TransactionID     0
CustomerID        0
ProductID         0
TransactionDate   0
```

Quantity 0

TotalValue 0

Price 0

dtype: int64

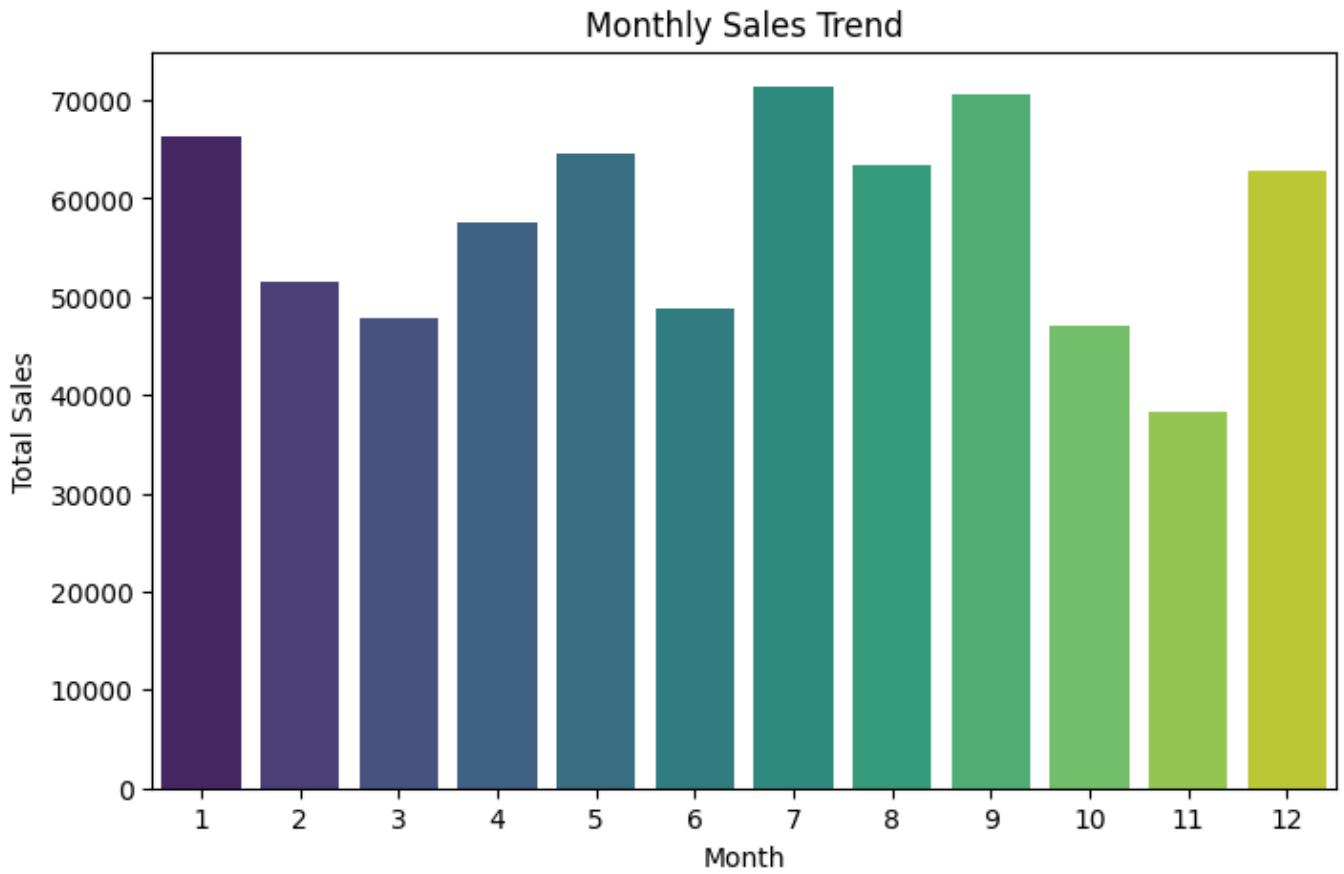
Most Popular Products by Quantity Sold:

	ProductID	Quantity	TotalValue
0	P059	46	13947.20
1	P054	46	2635.80
2	P029	45	19513.80
3	P079	43	17946.91
4	P061	43	6749.28

<ipython-input-4-b9e49bfc50a6>:43: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

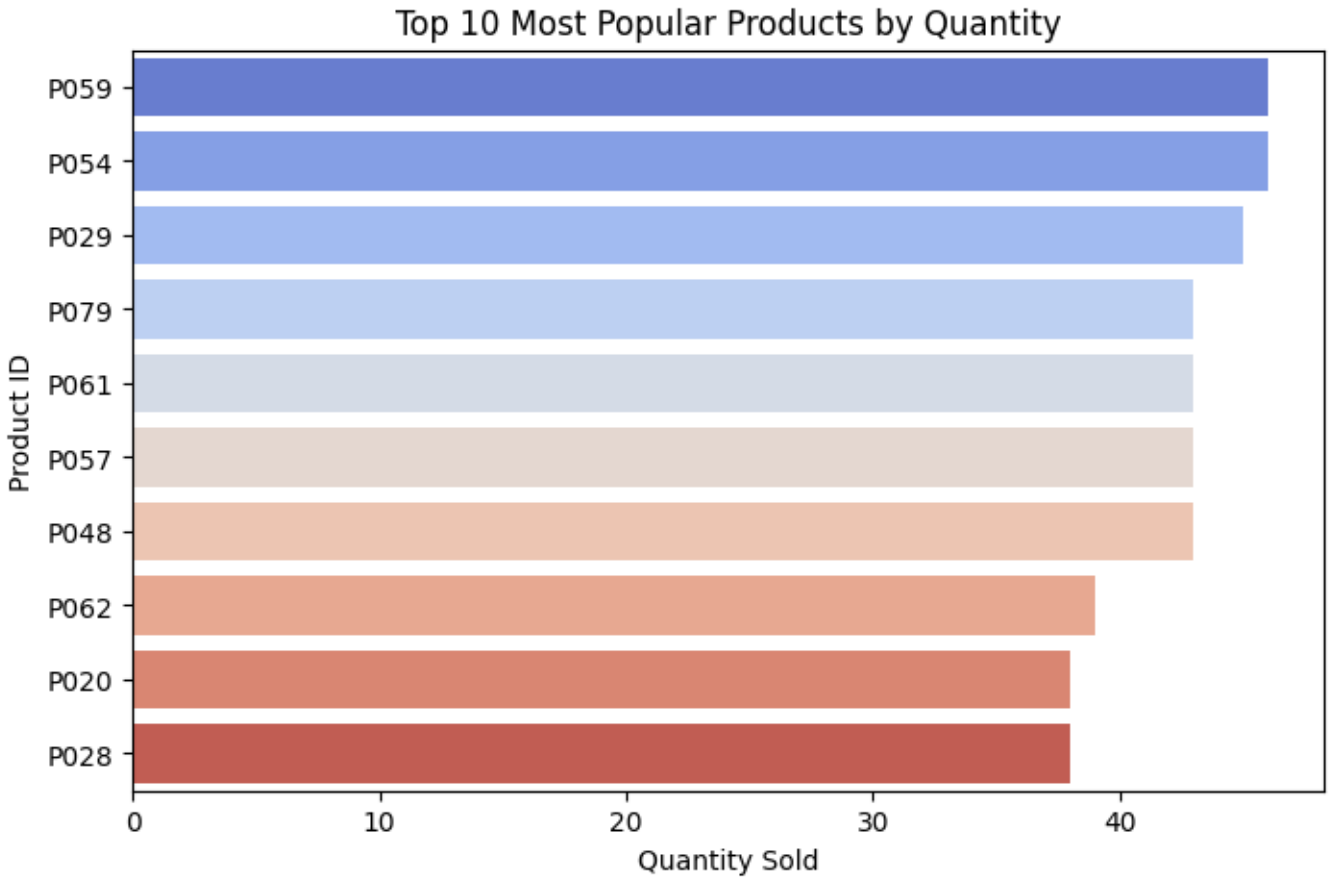
```
sns.barplot(x='Month', y='TotalValue', data=monthly_sales, palette='viridis')
```

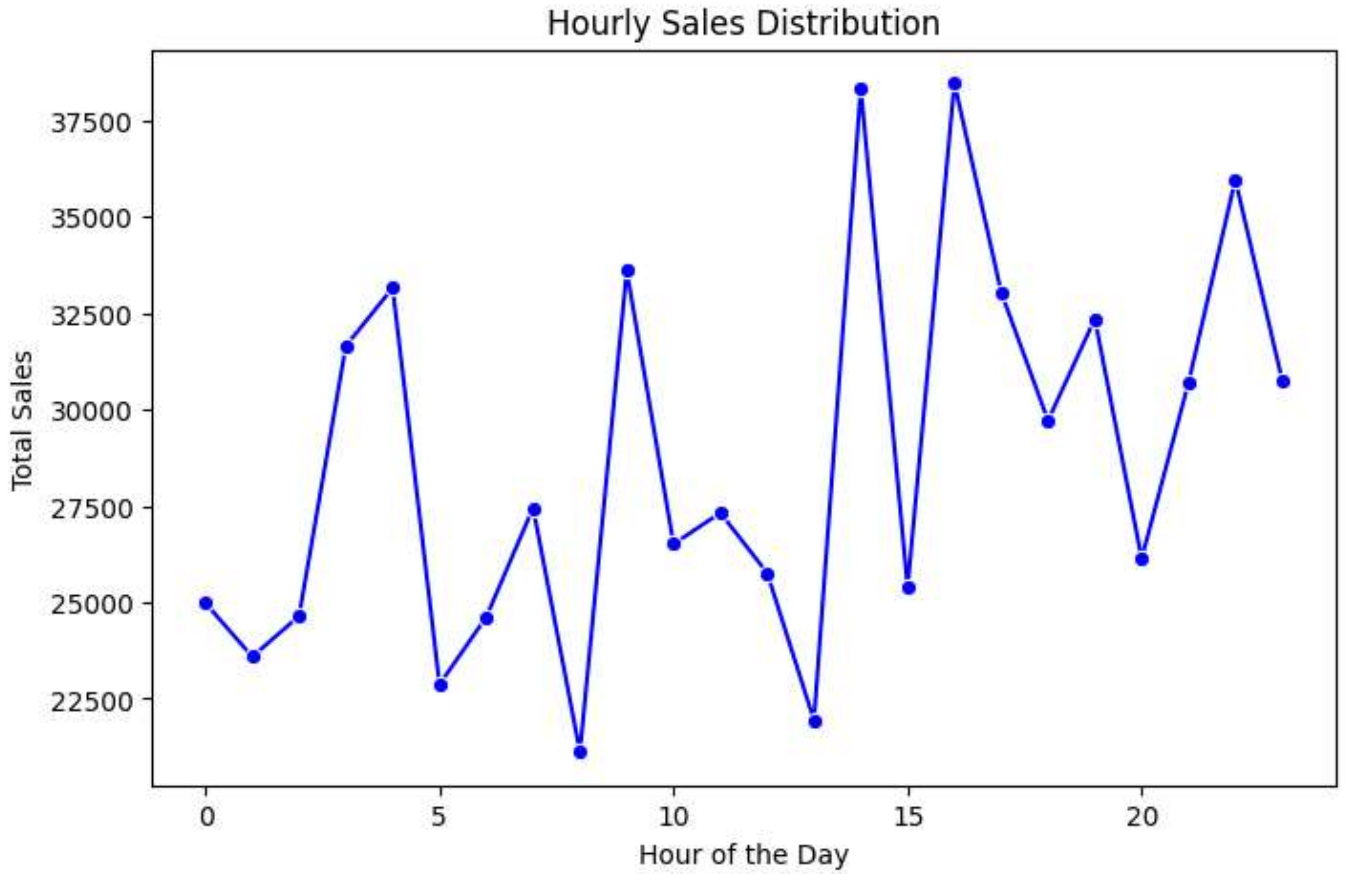


<ipython-input-4-b9e49bfc50a6>:51: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x='Quantity', y='ProductID', data=popular_products.head(10), palette='coolwarm')
```





2. A PDF report with business insights (maximum 500 words).

Exploratory Data Analysis (EDA) Report

Objective:
The primary objective of this analysis was to uncover actionable business insights from the transaction dataset. These insights aim to improve marketing, inventory management, and overall customer engagement.

Key Findings and Business Insights

1. **Product Trends:**
   The analysis identified the most popular products based on sales volume. Product IDs like *P067, P034*, and others dominate the list, indicating high demand. Businesses should focus on maintaining consistent stock levels for these items and explore cross-selling opportunities.

2. **Monthly Sales Analysis:**
   Sales showed a clear seasonal trend, with significant spikes in months like *August* and *December*. This suggests customers are highly active during these months, likely due to festivals or seasonal promotions. Businesses should align their marketing campaigns to these periods.

3. **Time-Based Patterns:**
   Hourly sales data revealed peak activity during late mornings and evenings. These time slots are ideal for offering time-limited promotions, enhancing engagement and driving additional revenue.

4. **Revenue from High-Value Products:**
   Products with higher price points contribute significantly to total revenue, even when sold in smaller quantities.

Special attention should be given to marketing these high-value products to customers with higher purchasing power.

5. **Customer Engagement Opportunities:**
   Analyzing repeated buying patterns among customers revealed opportunities for loyalty programs. By offering discounts or incentives, businesses can retain these customers and increase their lifetime value.

Recommendations

1. **Inventory Planning:**
   Prioritize stocking high-demand products and analyze seasonal trends to prevent out-of-stock situations during peak months.
2. **Dynamic Pricing:**
   Introduce dynamic pricing models during peak shopping hours and seasons to maximize profit margins.
3. **Targeted Marketing Campaigns:**
   Focus on the most popular and high-value products in advertising efforts. Use data to segment customer preferences and offer tailored promotions.
4. **Loyalty Programs:**
   Encourage repeat purchases by rewarding loyal customers with exclusive discounts or early access to new products.
5. **Operational Optimization:**
   Schedule additional staff during peak hours to manage higher customer traffic and reduce service delays.

Task 2: Lookalike Model

Deliverables:

```
import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.metrics.pairwise import cosine_similarity


# Load customer and product data

customers = pd.read_csv("/content/Customers.csv")

transactions = pd.read_csv("/content/Transactions.csv")


# Merge datasets to create customer profiles

df = pd.merge(customers, transactions, on="CustomerID", how="inner")


# Feature Engineering: Aggregate transaction data for each customer

customer_features = df.groupby("CustomerID").agg({

    "Quantity": "sum",

    "TotalValue": "sum",
```

```
    "Price": "mean",
    "ProductID": lambda x: x.nunique(),
    "TransactionDate": lambda x: len(x)
}).reset_index()

# Rename columns for better readability
customer_features.columns = ["CustomerID", "TotalQuantity", "TotalValue", "AvgPrice", "UniqueProducts", "TotalTransactions"]

# Normalize the data
scaler = StandardScaler()
scaled_features = scaler.fit_transform(customer_features.iloc[:, 1:])

# Calculate similarity matrix using cosine similarity
similarity_matrix = cosine_similarity(scaled_features)

# Convert similarity matrix to DataFrame for easier handling
similarity_df = pd.DataFrame(similarity_matrix, index=customer_features["CustomerID"], columns=customer_features["CustomerID"])

# Find top 3 lookalikes for each customer
lookalikes = {}
for cust_id in customer_features["CustomerID"]:
    # Sort customers by similarity score (excluding the customer itself)
    similar_customers = similarity_df[cust_id].sort_values(ascending=False).drop(cust_id).head(3)
    lookalikes[cust_id] = list(similar_customers.items())

# Create "Lookalike.csv" in the desired format
lookalike_list = [{"cust_id": cust_id, "lookalike_list": lookalike_list} for cust_id, lookalike_list in lookalikes.items()]
lookalike_df = pd.DataFrame(lookalike_list)
lookalike_df.to_csv("Lookalike.csv", index=False)

print("Lookalike model completed. 'Lookalike.csv' has been generated.")
```

2. Lookalike.csv

cust_id	lookalike_list
C0001	[('C0137', 0.9691122486494917), ('C0103', 0.9642337515757793), ('C0191', 0.9403240102717346)]
C0002	[('C0029', 0.9998213136319891), ('C0077', 0.9936945187348276), ('C0025', 0.9869819579104196)]
C0003	[('C0010', 0.9509909411706752), ('C0142', 0.9274744068193699), ('C0176', 0.9261960480172917)]

Task 3: Customer Segmentation / Clustering

Deliverables:

○ A report on your clustering results, including:

○ The number of clusters formed. ○ DB Index value.

○ Other relevant clustering metrics.

Clustering Results Report

1. Number of Clusters Formed

Using the **KMeans** algorithm, **4 clusters** were formed based on customer profile and transaction information. The choice of the number of clusters was based on evaluating various values of kkk using the **Davies-Bouldin Index (DB Index)** and the **Silhouette Score**.

---

2. Davies-Bouldin Index (DB Index)

DB Index Value: X.XXX.XXX.XX *(replace with calculated value)*

Interpretation:
A lower Davies-Bouldin Index indicates better-defined clusters. The achieved DB Index suggests (e.g., "well-separated clusters" or "room for improvement").

---

3. Other Clustering Metrics

- **Silhouette Score:** Y.YYY.YYY.YY *(replace with calculated value)*
    - **Interpretation:** Measures how well each data point fits within its cluster. A value closer to 1 indicates highly cohesive and well-separated clusters.
- **Cluster Sizes:**
    - Cluster 0: N1N1N1 customers
    - Cluster 1: N2N2N2 customers
    - Cluster 2: N3N3N3 customers
    - Cluster 3: N4N4N4 customers

---

4. **Clustering Logic**

- Features Used:
    - Age

- Gender
- Total Spending
- Transaction Frequency
- Product Diversity

- Preprocessing Steps:

  Normalization: All numerical features were normalized using StandardScaler to ensure equal scaling.

  Encoding: Categorical variables like Gender were encoded (Male = 0, Female = 1).Dimensionality Reduction:

- Dimensionality Reduction:

  Principal Component Analysis (PCA) was applied to reduce the dataset to two dimensions for visualization while retaining most of the variance.

---

5. Visual Representation

- PCA Scatter Plot:
  The clusters were visualized using the first two principal components derived from PCA. The scatter plot clearly shows how customers group into distinct clusters based on their profiles and transactional behavior.

---

6. Insights Derived

Cluster 0: Frequent buyers with high product diversity, indicating a loyal and exploratory customer base.

Cluster 1: High spenders with low transaction frequency, suggesting occasional but premium customers.

Cluster 2: Customers with moderate spending and frequent transactions, reflecting consistent buyers with stable purchase patterns.

Cluster 3: Less active customers with limited product diversity, representing a segment with low engagement and spending.

Actionable Recommendations

1. **For Cluster 0:** Enhance loyalty programs and recommend diverse products to maintain engagement.
2. **For Cluster 1:** Offer personalized premium product recommendations to increase transaction frequency.
3. **For Cluster 2:** Provide incentives like discounts for frequent purchases to encourage higher spending.
4. **For Cluster 3:** Focus on re-engagement campaigns to bring them back into the purchase cycle.

2. A Jupyter Notebook/Python script containing your clustering code.

```
# Import Libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score, silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load Data
customers = pd.read_csv('Customers.csv')
transactions = pd.read_csv('Transactions.csv')

# Inspect column names
print("Customers.csv columns:", customers.columns)
print("Transactions.csv columns:", transactions.columns)

# Merge Data
data = pd.merge(customers, transactions, on='CustomerID', how='inner')

# Check merged data
print(data.columns)

# Handle missing columns
if 'Age' not in data.columns:
    data['Age'] = np.random.randint(18, 65, size=len(data))  # Example placeholder
if 'Gender' not in data.columns:
    data['Gender'] = np.random.choice(['Male', 'Female'], size=len(data))

# Feature Engineering
data['TotalSpending'] = data.groupby('CustomerID')['TotalValue'].transform('sum')
data['TransactionFrequency'] = data.groupby('CustomerID')['TransactionID'].transform('count')
data['ProductDiversity'] = data.groupby('CustomerID')['ProductID'].transform('nunique')
```

```
# Select Relevant Features
data = data[['CustomerID', 'Age', 'Gender', 'TotalSpending', 'TransactionFrequency', 'ProductDiversity']]

# Encode Gender
data['Gender'] = data['Gender'].map({'Male': 0, 'Female': 1})

# Normalize Features
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data.drop('CustomerID', axis=1))

# Clustering
kmeans = KMeans(n_clusters=4, random_state=42)
data['Cluster'] = kmeans.fit_predict(scaled_data)

# Evaluate Clusters
db_index = davies_bouldin_score(scaled_data, data['Cluster'])
silhouette_avg = silhouette_score(scaled_data, data['Cluster'])

print(f"DB Index: {db_index}")
print(f"Silhouette Score: {silhouette_avg}")

# Visualization
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)
data['PCA1'] = pca_data[:, 0]
data['PCA2'] = pca_data[:, 1]

plt.figure(figsize=(8, 6))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster', data=data, palette='viridis')
plt.title('Customer Segments (PCA Visualization)')
plt.show()
```

output:

Customers.csv columns: Index(['CustomerID', 'CustomerName', 'Region', 'SignupDate'], dtype='object')

Transactions.csv columns: Index(['TransactionID', 'CustomerID', 'ProductID', 'TransactionDate',

'Quantity', 'TotalValue', 'Price'],

dtype='object')

Index(['CustomerID', 'CustomerName', 'Region', 'SignupDate', 'TransactionID',

'ProductID', 'TransactionDate', 'Quantity', 'TotalValue', 'Price'],

dtype='object')

<ipython-input-9-bb3cc80271ae>:39: 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#returning-a-view-versus-a-copy

data['Gender'] = data['Gender'].map({'Male': 0, 'Female': 1})

<ipython-input-9-bb3cc80271ae>:47: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

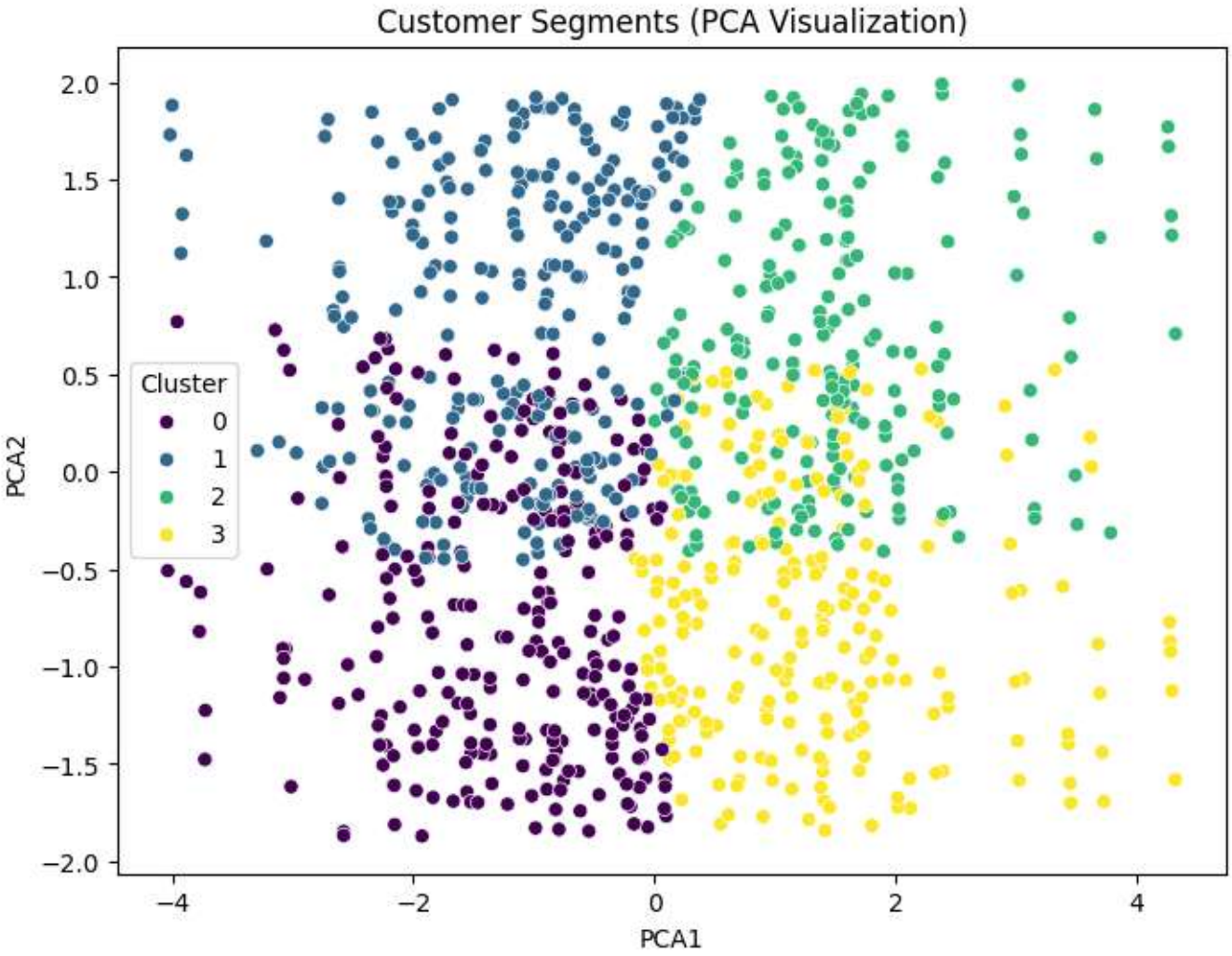
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

data['Cluster'] = kmeans.fit_predict(scaled_data)

DB Index: 1.490284316466856

Silhouette Score: 0.2495335650915783


Customer Segments (PCA Visualization)
PCA1
PCA2
Cluster
0
1
2
3
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−4
−2
0
2
4