Customer Segmentation with Python

1. Understand the problem and Business Case

In this project, we are provided with a dataset containing customer information from a bank over the past six months. The data includes variables such as transaction frequency, transaction amounts, tenure, and other relevant features.

As data scientists, our task is to utilize artificial intelligence and machine learning techniques to classify customers into at least three distinct groups. The marketing team will use the results of this classification to optimize their marketing campaigns and strategies.

2. Import libraries & Load data

```
# Data Manipulation
import pandas as pd
import numpy as np

# Data Visualization
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Machine Learning
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score

# Other useful libraries
import warnings
warnings.filterwarnings("ignore")

# For additional visualization (optional)
import plotly.express as px
import plotly.graph_objs as go

# For display text as Markdown
from IPython.display import display, Markdown

# from jupyterthemes import jtplot
# jtplot.style(theme = 'monokai', context = 'notebook', ticks = True, grid=False )

# Set the display format for floating point numbers
pd.options.display.float_format = '{:,.3f}'.format

# Read data

data = pd.read_csv("data/marketing_data.csv")
data
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTA
0	C10001	40.901	0.818	95.400	0.000	95.400
1	C10002	3,202.467	0.909	0.000	0.000	0.000
2	C10003	2,495.149	1.000	773.170	773.170	0.000
3	C10004	1,666.671	0.636	1,499.000	1,499.000	0.000
4	C10005	817.714	1.000	16.000	16.000	0.000
...	...	...	...	...	...	...
8945	C19186	28.494	1.000	291.120	0.000	291.120
8946	C19187	19.183	1.000	300.000	0.000	300.000
8947	C19188	23.399	0.833	144.400	0.000	144.400
8948	C19189	13.458	0.833	0.000	0.000	0.000
8949	C19190	372.708	0.667	1,093.250	1,093.250	0.000

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   CUST_ID                           8950 non-null   object
 1   BALANCE                           8950 non-null   float64
 2   BALANCE_FREQUENCY                 8950 non-null   float64
 3   PURCHASES                         8950 non-null   float64
 4   ONEOFF_PURCHASES                  8950 non-null   float64
 5   INSTALLMENTS_PURCHASES            8950 non-null   float64
 6   CASH_ADVANCE                      8950 non-null   float64
 7   PURCHASES_FREQUENCY               8950 non-null   float64
 8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 11  CASH_ADVANCE_TRX                  8950 non-null   int64
 12  PURCHASES_TRX                     8950 non-null   int64
 13  CREDIT_LIMIT                      8949 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8637 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

Field Name	Description
CUSTID	Identification of Credit Card holder
BALANCE	Balance amount left in customer's account to make purchases
BALANCE_FREQUENCY	How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated)
PURCHASES	Amount of purchases made from account
ONEOFFPURCHASES	Maximum purchase amount done in one-go
INSTALLMENTS_PURCHASES	Amount of purchase done in installment
CASH_ADVANCE	Cash in advance given by the user
PURCHASES_FREQUENCY	How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
ONEOFF_PURCHASES_FREQUENCY	How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)
PURCHASES_INSTALLMENTS_FREQUENCY	How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)
CASH_ADVANCE_FREQUENCY	How frequently the cash in advance being paid
CASH_ADVANCE_TRX	Number of Transactions made with "Cash in Advance"
PURCHASES_TRX	Number of purchase transactions made
CREDIT_LIMIT	Limit of Credit Card for user

Field Name	Description
PAYMENTS	Amount of Payment done by user
MINIMUM_PAYMENTS	Minimum amount of payments made by user
PRC_FULL_PAYMENT	Percent of full payment paid by user
TENURE	Tenure of credit card service for user

3. Exploratory Data Analysis (EDA)

Let's check the basic statistics and visualize some features:

Descriptive statistics

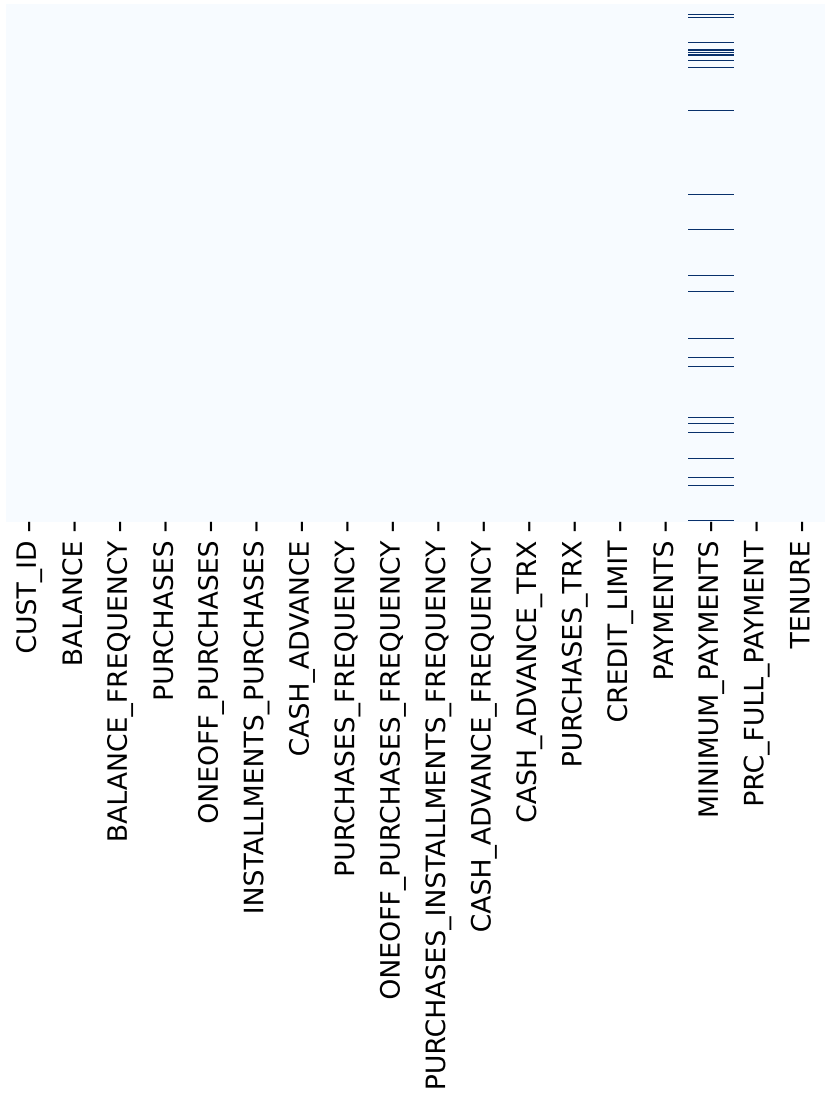
```
# Display basic statistics
data.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
count	8,950.000	8,950.000	8,950.000	8,950.000	8,950.000
mean	1,564.475	0.877	1,003.205	592.437	411.068
std	2,081.532	0.237	2,136.635	1,659.888	904.338
min	0.000	0.000	0.000	0.000	0.000
25%	128.282	0.889	39.635	0.000	0.000
50%	873.385	1.000	361.280	38.000	89.000
75%	2,054.140	1.000	1,110.130	577.405	468.637
max	19,043.139	1.000	49,039.570	40,761.250	22,500.000

Key Insights from Descriptive Statistics:

- The mean balance of customers is approximately $1,564.
- The balance frequency is updated frequently, with an average of around 0.9.
- The average purchase amount is about $1,000.
- The mean one-off purchase amount is roughly $600.
- The average purchases frequency is around 0.5.
- The average frequencies for one-off purchases, installment purchases, and cash advances are generally low.
- The average credit limit for customers is around $4,500.
- The percentage of full payments made is 15%.
- The average tenure of customers is 11 years.

```
# Let's see if we have any missing data
sns.heatmap(data.isnull(), yticklabels=False, cbar=False, cmap="Blues")
```




```
# Check for missing values
data.isnull().sum()
```

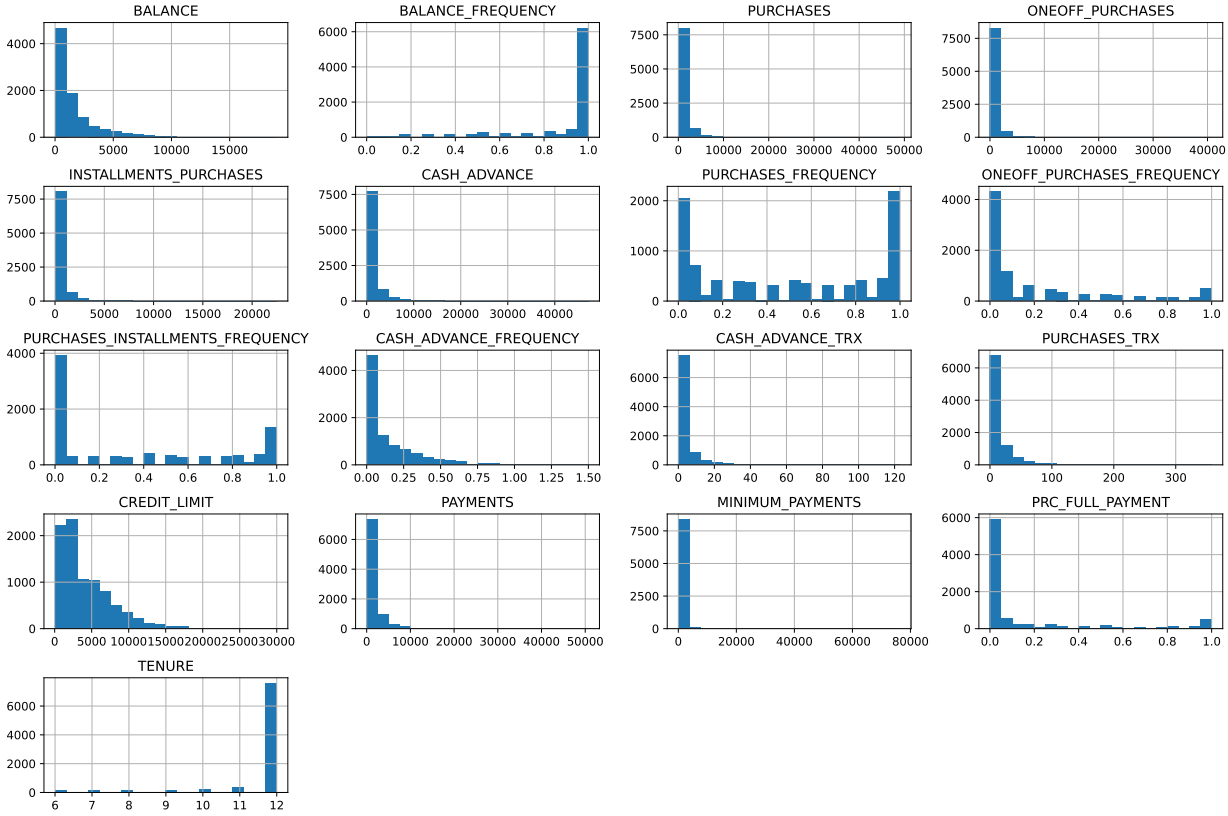
```
CUST_ID                             0
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
```

```
PURCHASES_TRX                     0
CREDIT_LIMIT                      1
PAYMENTS                          0
MINIMUM_PAYMENTS                313
PRC_FULL_PAYMENT                  0
TENURE                            0
dtype: int64
```

There are missing data in `MINIMUM_PAYMENTS` and `CREDIT_LIMIT`.

Distributions of numerical features

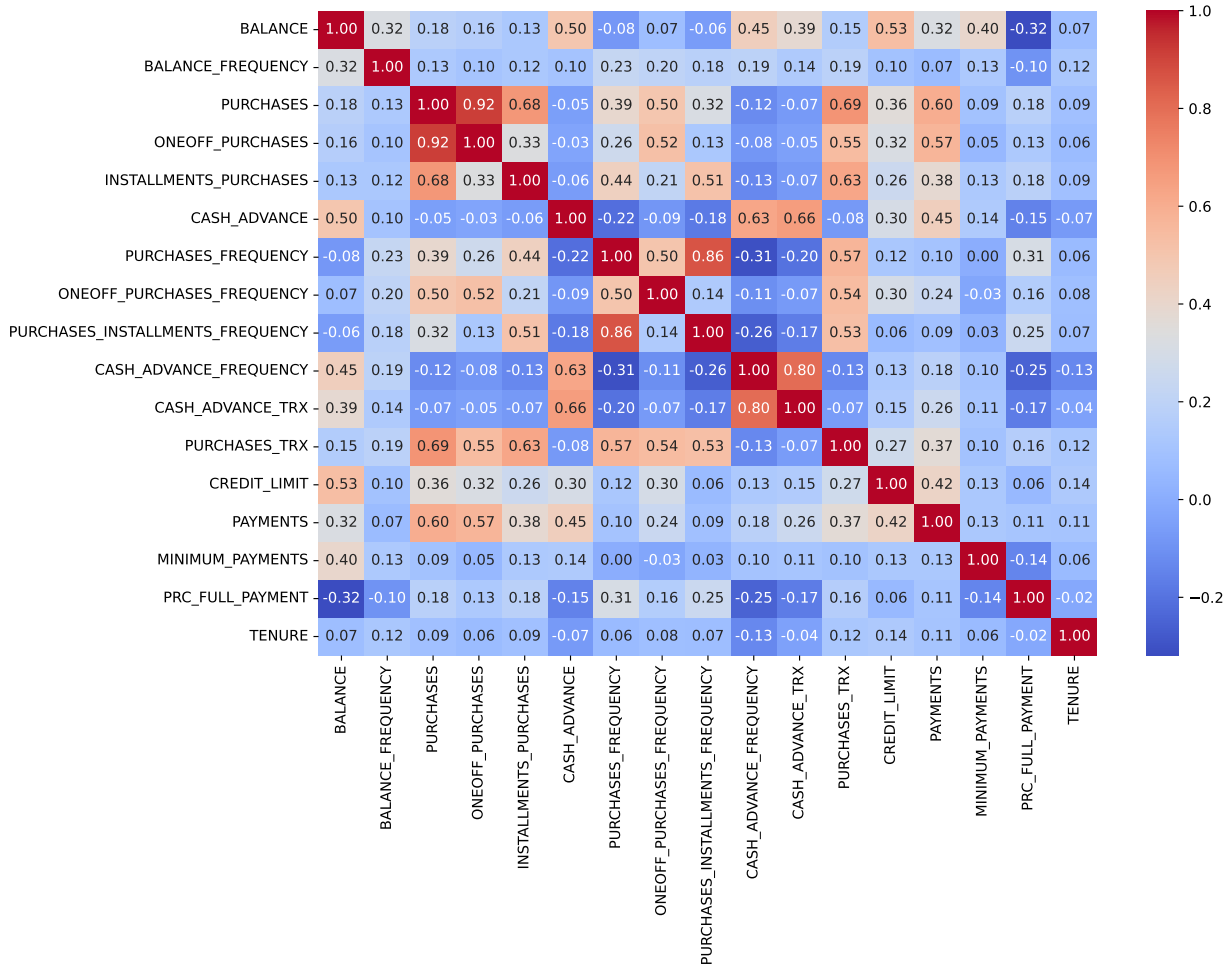
```
# Visualize distributions of numerical features
data.hist(bins=20, figsize=(15, 10))
plt.tight_layout()
plt.show()
```



Correlation matrix between features

```
# Correlation matrix
plt.figure(figsize=(12, 8))
data_numeric = data.drop(columns=["CUST_ID"])
```

```
sns.heatmap(data_numeric.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.show()
```



The heatmap provides a visual representation of the correlation coefficients, with colors indicating the strength and direction of the correlations

Strong Positive Correlations:

- `PURCHASES` and `ONEOFF_PURCHASES` (0.92): Indicates that one-off purchases are a major component of total purchases.
- `PURCHASES_INSTALLMENTS_FREQUENCY` and `PURCHASES_FREQUENCY` (0.86): Indicates that the frequency of installment purchases is closely related to the overall frequency of purchases.
- `CASH_ADVANCE` and `CASH_ADVANCE_TRX` (0.66): Indicates that the amount of cash advances is strongly related to the number of cash advance transactions.

Moderate Positive Correlations:

- `CREDIT_LIMIT` and `BALANCE` (0.53): Suggests that higher credit limits are associated with higher balances.

- `PURCHASES` and `PURCHASES_TRX` (0.69): Indicates a moderate relationship between total purchases and the number of purchase transactions.
- `PAYMENTS` and `PURCHASES` (0.60): Indicates that higher payments are associated with higher purchase amounts

Strong Negative Correlations:

- `PRC_FULL_PAYMENT` and `BALANCE` (-0.32): Indicates that a higher percentage of full payments is associated with lower balances.

Moderate Negative Correlations:

- `CASH_ADVANCE_FREQUENCY` and `PURCHASES_INSTALLMENTS_FREQUENCY` (-0.31): Suggests that frequent cash advances are associated with less frequent installment purchases.

Analytical Points from EDA:

1. Customer Engagement: The high mean balance ($1,564) and frequent balance (average balance frequency of ~0.9) indicate active account management and regular transactions by customers.

2. Spending Behavior: With an average purchase amount of $1,000 and an average one-off purchase amount of ~$600, customers seem to engage in both frequent smaller purchases and occasional larger one-off purchases.

3. Low Frequency Transactions: The generally low average frequencies for one-off purchases, installment purchases, and cash advances suggest that while customers do make these types of transactions, they do so infrequently.

4. Credit Utilization: The average credit limit of $4,500 suggests that the bank extends a significant amount of credit to its customers. However, the actual usage and payment patterns (such as the 15% full payment rate) indicate varying levels of credit utilization and repayment behavior.

5. Financial Responsibility: The relatively low percentage of full payments (15%) might indicate that a large proportion of customers are not paying off their balances in full each month, which could lead to higher interest income for the bank but also suggests potential financial strain on customers.

6. Customer Loyalty: An average tenure of 11 years highlights a strong customer loyalty and long-term relationship with the bank, which is a positive indicator of customer satisfaction and retention.

7. Marketing Strategy Implications: These insights can help the marketing team tailor their campaigns to different customer segments. For example, they could target customers with high balances and low payment rates with offers for balance transfer or debt consolidation products, or incentivize frequent small purchasers with rewards programs.

4. Data Preprocessing

Handle missing values and prepare the data for clustering:

```
# Drop non-numeric columns if necessary (e.g., CUST_ID)
data = data.drop(columns=["CUST_ID"])

# Fill missing values with mean
data = data.fillna(data.mean())

# Check if any missing value again
data.isnull().sum()
```

```
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
CREDIT_LIMIT                        0
PAYMENTS                            0
MINIMUM_PAYMENTS                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

```
# Let's see if we have duplicated entries in the data
data.duplicated().sum()
```

```
0
```

5. Feature Scalling

Standardize the features before clustering:

```
# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

scaled_data.shape
```

```
(8950, 17)
```

```
scaled_data
```

```
array([[-0.73198937, -0.24943448, -0.42489974, ..., -0.31096755,
        -0.52555097,  0.36067954],
       [ 0.78696085,  0.13432467, -0.46955188, ...,  0.08931021,
         0.2342269 ,  0.36067954],
       [ 0.44713513,  0.51808382, -0.10766823, ..., -0.10166318,
        -0.52555097,  0.36067954],
       ...,
       [-0.7403981 , -0.18547673, -0.40196519, ..., -0.33546549,
         0.32919999, -4.12276757],
       [-0.74517423, -0.18547673, -0.46955188, ..., -0.34690648,
         0.32919999, -4.12276757],
       [-0.57257511, -0.88903307,  0.04214581, ..., -0.33294642,
        -0.52555097, -4.12276757]])
```

Why do we need scaling data before clustering ?

Scaling data before clustering is important because many clustering algorithms rely on distance measures, such as Euclidean distance, to determine the similarity between data points. If the data is not scaled, features with larger ranges can dominate the distance calculations, potentially skewing the results.
Common methods for scaling data include:

- **Standardization**: Transforming the data to have a mean of zero and a standard deviation of one.
- **Normalization**: Scaling the data to a specific range, usually [0, 1] or [-1, 1].

By scaling the data, we ensure that the clustering results are more reliable and meaningful.

6. Model Training

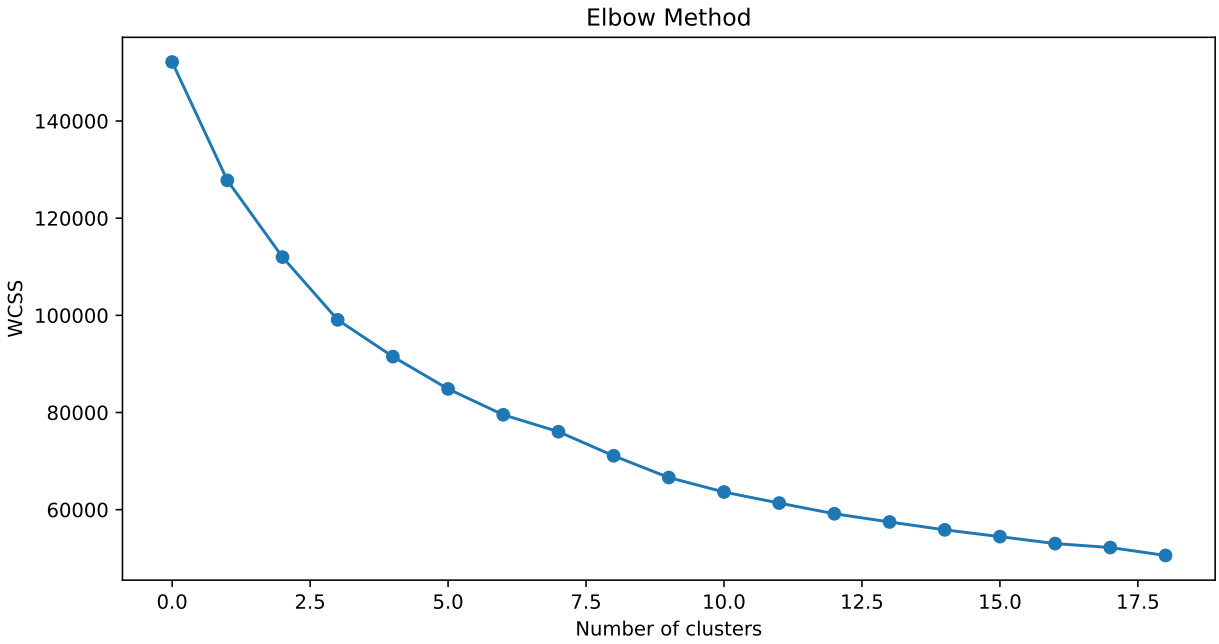
Apply K-Means clustering

Apply K-Means clustering and determine the optimal number of clusters

```
# Determine the optimal number of clusters using the elbow method
wcss = []
for i in range(1, 20):
    kmeans = KMeans(
        n_clusters=i, init="k-means++", max_iter=300, n_init=10, random_state=42
    )
    kmeans.fit(scaled_data)
```

```
    wcss.append(kmeans.inertia_)

# Plot the elbow method graph
plt.figure(figsize=(10, 5))
plt.plot(wcss, marker="o")
plt.title("Elbow Method")
plt.xlabel("Number of clusters")
plt.ylabel("WCSS")
plt.show()
```



Choose the optimal number of clusters (e.g., 7 based on the elbow method) and fit the K-Means model:

```
# Fit K-Means model
optimal_clusters = 7  # Based on the elbow method
kmeans = KMeans(
    n_clusters=optimal_clusters,
    init="k-means++",
    max_iter=300,
    n_init=10,
    random_state=42,
)

kmeans.fit(scaled_data)
kmeans
```

```
KMeans(n_clusters=7, n_init=10, random_state=42)
```

```
# Cluster_center shape
kmeans.cluster_centers_.shape
```

```
(7, 17)
```

```
# Cluster_center scaled values
cluster_centers = pd.DataFrame(data=kmeans.cluster_centers_, columns=[data.columns])
cluster_centers
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PU
0	1.430	0.419	6.915	6.083	5.172
1	1.666	0.392	-0.205	-0.150	-0.210
2	0.127	0.430	0.939	0.896	0.574
3	-0.368	0.331	-0.040	-0.235	0.337
4	-0.336	-0.348	-0.285	-0.209	-0.288
5	0.008	0.403	-0.344	-0.225	-0.399
6	-0.702	-2.134	-0.307	-0.230	-0.303

Cluster Centers

```
# In order to understand what these numbers mean, let's perform inverse transformation
cluster_centers = scaler.inverse_transform(cluster_centers)
cluster_centers = pd.DataFrame(data=cluster_centers, columns=[data.columns])
cluster_centers
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PU
0	4,541.394	0.977	15,777.311	10,689.028	5,088.284
1	5,033.097	0.970	564.520	343.613	221.021
2	1,828.400	0.979	3,009.455	2,079.428	930.501
3	799.440	0.956	918.086	202.469	716.053
4	866.148	0.795	395.312	245.586	150.203
5	1,580.736	0.973	268.426	218.629	49.971
6	103.587	0.372	347.456	210.200	137.507

Classify data into clusters

```
markdown_text = f"Our account data is now classified into `{optimal_clusters}` clusters"
Markdown(markdown_text)
```

Our account data is now classified into 7 clusters

```
# Predict cluster from scaled_data
clusters = kmeans.fit_predict(scaled_data)

# Add the cluster labels to the original data
data["Cluster"] = clusters
data
```

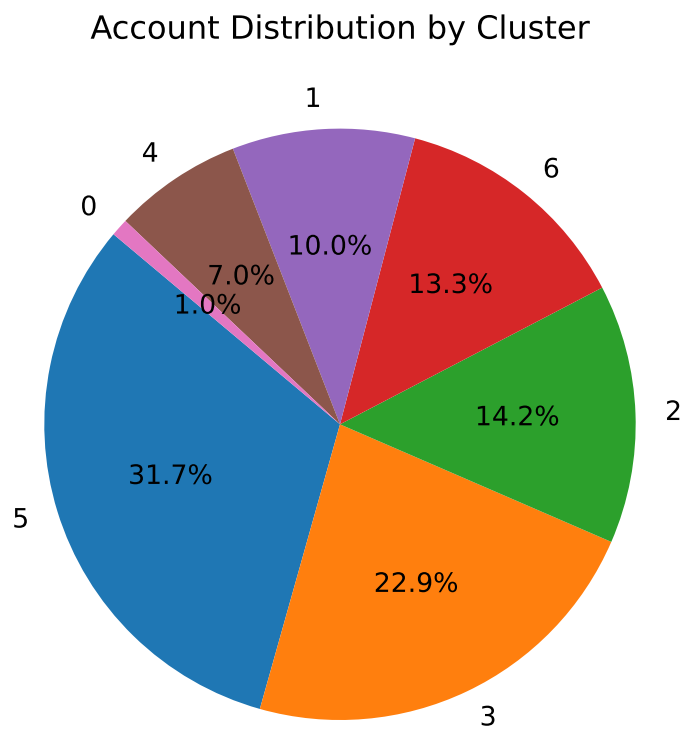
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	40.901	0.818	95.400	0.000	95.400
1	3,202.467	0.909	0.000	0.000	0.000
2	2,495.149	1.000	773.170	773.170	0.000
3	1,666.671	0.636	1,499.000	1,499.000	0.000
4	817.714	1.000	16.000	16.000	0.000
...	...	...	...	...	...
8945	28.494	1.000	291.120	0.000	291.120
8946	19.183	1.000	300.000	0.000	300.000
8947	23.399	0.833	144.400	0.000	144.400
8948	13.458	0.833	0.000	0.000	0.000
8949	372.708	0.667	1,093.250	1,093.250	0.000

Let's view account distribution by 7 clusters

```
# Total account by cluster
cluster_counts = data["Cluster"].value_counts()
print(cluster_counts)

# Plot the pie chart
plt.figure(figsize=(8, 5))
plt.pie(cluster_counts, labels=cluster_counts.index, autopct="%1.1f%%", startangle=140)
plt.title("Account Distribution by Cluster")
plt.show()
```

```
Cluster
5    2841
3    2046
2    1267
6    1187
1     894
4     629
0      86
Name: count, dtype: int64
```



Apply Principal Component Analysis (PCA)

```
# Apply PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(scaled_data)
pc_df = pd.DataFrame(data=principal_components, columns=["PC1", "PC2"])
pc_df["Cluster"] = clusters

# Create DataFrames for the different cluster groups
pc_df_all = pc_df.copy()
pc_df_2356 = pc_df[pc_df["Cluster"].isin([2, 3, 5, 6])].copy()
pc_df_014 = pc_df[pc_df["Cluster"].isin([0, 1, 4])].copy()

# Set up the figure with GridSpec
fig = plt.figure(figsize=(12, 12))
gs = fig.add_gridspec(2, 2, height_ratios=[1, 1])

# Plot for all clusters in the first row spanning both columns
ax1 = fig.add_subplot(gs[0, :])
sns.scatterplot(
    x="PC1", y="PC2", hue="Cluster", data=pc_df_all,
    palette=["red", "green", "blue", "pink", "yellow", "gray", "purple", "black"],
    s=70, alpha=0.6, ax=ax1
```

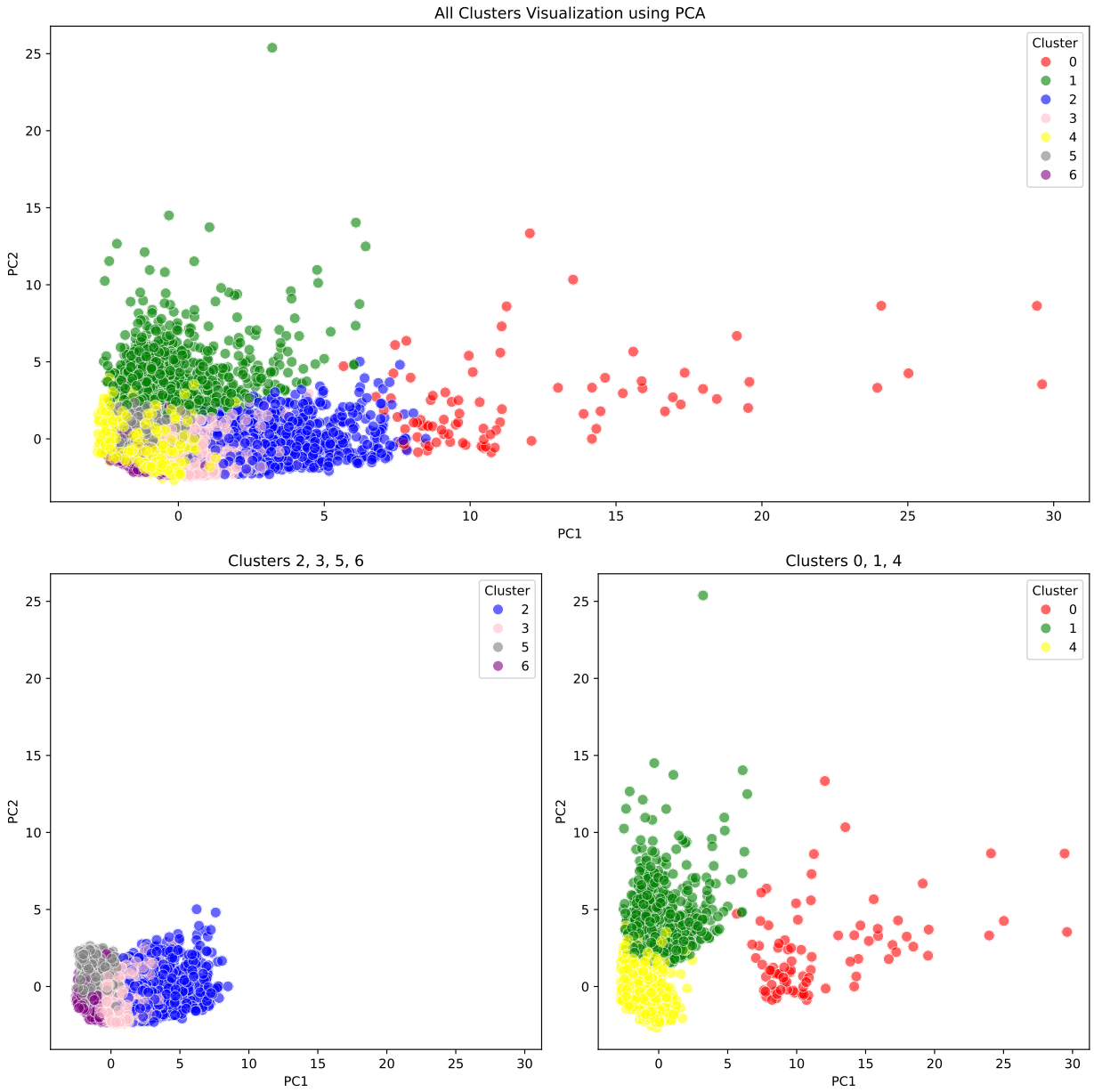
```
)
ax1.set_title("All Clusters Visualization using PCA")
##ax1.legend(loc='center left', bbox_to_anchor=(1, 0.5))

# Determine limits for x and y axes from the first plot
x_limits = ax1.get_xlim()
y_limits = ax1.get_ylim()

# Plot for clusters 2, 3, 5, and 6 in the second row, first column
ax2 = fig.add_subplot(gs[1, 0])
sns.scatterplot(
    x="PC1", y="PC2", hue="Cluster", data=pc_df_2356,
    palette=["blue","pink", "gray", "purple"],
    s=70, alpha=0.6, ax=ax2
)
ax2.set_title("Clusters 2, 3, 5, 6")
ax2.set_xlim(x_limits)
ax2.set_ylim(y_limits)
##ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))

# Plot for clusters 0, 1, and 4 in the second row, second column
ax3 = fig.add_subplot(gs[1, 1])
sns.scatterplot(
    x="PC1", y="PC2", hue="Cluster", data=pc_df_014,
    palette=["red", "green", "yellow"],
    s=70, alpha=0.6, ax=ax3
)
ax3.set_title("Clusters 0, 1, 4")
ax3.set_xlim(x_limits)
ax3.set_ylim(y_limits)
#ax3.legend(loc='center left', bbox_to_anchor=(1, 0.5))

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



7. Model Evaluation and Conclusions

Evaluate the clustering result using silhouette score:

```
# Calculate silhouette score
sil_score = silhouette_score(scaled_data, clusters)
print(f"Silhouette Score: {sil_score}")
```

```
Silhouette Score: 0.21503808606365246
```

! Sihoutte score

A silhouette score of 0.215 ***suggests that the clusters might be overlapping or not distinctly separated and indicates room for improvement.***

Conclusion: The Four Significant Groups

To analyze the cluster centers into four significant groups, we will examine various attributes and categorize the rows based on similarities and differences. Here is a detailed breakdown:

1. **High Spending and High Balance**
   - This group has high balances and high spending on purchases, including both one-off and installment purchases. They also have high credit limits and good payment histories.
   - Examples:
     - **Row 0**: Balance of $4541.39, high purchases ($15777.31), and a substantial credit limit ($12493.02).
     - **Row 2**: Balance of $1828.40, significant purchases ($3009.46), and a high credit limit ($7047.42).
   - These customers are actively using their credit and have significant balances. They are ***likely to be valuable clients due to their high engagement and spending.***
2. **High Balance but Low Spending**
   - This group has high balances and credit limits but low purchase amounts and frequencies, indicating lower transaction activity.
   - Examples:
     - **Row 1**: Balance of $5033.10, but low purchases ($564.52) and high cash advance ($5153.57).
     - **Row 5**: Balance of $1580.74, with lower purchases ($268.43) and a relatively high cash advance ($760.33).
   - These customers have high credit limits but are not utilizing them much. They may be ***less engaged or could be saving their credit for future use.***
3. **Low Balance and High Cash Advance**
   - This group has lower balances but high cash advances with modest purchase amounts. They show a higher frequency of cash advances relative to purchases.
   - Examples:
     - **Row 3**: Balance of $799.44, with high cash advance ($206.95) and low purchase activity ($918.09).
     - **Row 6**: Balance of $103.59, with high cash advance ($301.36) and low purchases ($347.46).
   - These customers frequently use cash advances despite having lower balances. They might be ***facing financial difficulties or prefer cash over credit.***
4. **Low Balance and Low Spending**

- This group has low balances and spending across all purchase categories, indicating minimal activity and expenditure.
- Examples:
  - **Row 6**: Balance of $103.59, with minimal purchases ($347.46) and low credit limit ($3865.96).
  - **Row 4**: Balance of $866.15, with low purchases ($395.31) and a lower credit limit ($2468.23).
    - Additionally, the accounts in Row 4 have a tenure of 7.2, the lowest among all others. This suggests these customers have used their credit accounts for a shorter period compared to the rest.
- This group shows minimal engagement with their credit accounts. They have low balances and low spending, suggesting limited usage or disinterest. This group might ***include new accounts that have not fully explored the features and benefits of their credit accounts***.

This classification can help the company target different customer segments and personalize their financial products or interventions.