Market Segmentation Customer Data Project

Madhumita Mondal

11/14/2023

In marketing, market segmentation is the process of dividing a broad consumer or business market, normally consisting of existing and potential customers, into sub-groups of consumers based on some type of shared characteristics.

Task:

This project focuses on customer segmentation, a technique commonly used in marketing and business strategy. The primary goal is to categorize customers into distinct groups based on their behavior, preferences, or characteristics. The project involves both unsupervised learning, specifically K-Means clustering, and supervised learning using a Decision Tree classifier.

Type of Learning/Algorithms:

Unsupervised Learning (Clustering):

Algorithm: K-Means Clustering Purpose: Group customers into clusters based on similarities in their features without the need for labeled target variables.

Supervised Learning (Classification):

Algorithm: Decision Tree Classifier Purpose: Train a model to predict the cluster (previously identified through K-Means) of a customer based on their features. This step involves a labeled dataset where the clusters serve as the target variable.

Goal:

The project seeks to leverage data-driven insights to enhance business strategies and decision-making by understanding and categorizing customer behavior. The combination of unsupervised learning for segmentation and supervised learning for prediction contributes to a holistic approach in addressing business challenges related to customer engagement and satisfaction.A case requires to develop a customer segmentation to give recommendations like saving plans, loans, wealth management, etc. on target customers groups.

Dataset:

The sample Dataset summarizes the usage behavior of about 9000 active credit card holders during the last 6 months. The file is at a customer level with 18 behavioral variables. Dataset is collected from Kaggle. It has 8950 rows × 18 columns.

Variables of Dataset

Balance Balance Frequency Purchases One-off Purchases Installment Purchases Cash Advance Purchases Frequency One-off Purchases Frequency Purchases Installments Frequency Cash Advance Frequency Cash Advance TRX Purchases TRX Credit Limit Payments Minimum Payments PRC Full payment Tenure Cluster

Importing Libraries

In [34]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
scalar=StandardScaler()
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans,AgglomerativeClustering,DBSCAN,SpectralClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn import tree
from sklearn import metrics

import warnings
warnings.filterwarnings("ignore")
```

Loading the dataset

In [35]:

```
df = pd.read_csv("Customer Data.csv")
df
```

Out[35]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENT
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	
...	...	...	...	...	...	
8945	C19186	28.493517	1.000000	291.12	0.00	
8946	C19187	19.183215	1.000000	300.00	0.00	
8947	C19188	23.398673	0.833333	144.40	0.00	
8948	C19189	13.457564	0.833333	0.00	0.00	
8949	C19190	372.708075	0.666667	1093.25	1093.25	

8950 rows × 18 columns

EDA

In [36]:

```
df.shape
```

Out[36]:

```
(8950, 18)
```

In [37]:

```
df.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
```

In [38]:
```
df.describe()
```

Out[38]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURC
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.
mean	1564.474828	0.877271	1003.204834	592.437371	411.
std	2081.531879	0.236904	2136.634782	1659.887917	904.
min	0.000000	0.000000	0.000000	0.000000	0.
25%	128.281915	0.888889	39.635000	0.000000	0.
50%	873.385231	1.000000	361.280000	38.000000	89.
75%	2054.140036	1.000000	1110.130000	577.405000	468.
max	19043.138560	1.000000	49039.570000	40761.250000	22500.

In [39]:
```
df.isnull().sum()
```

Out[39]:
```
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
```

Data Cleaning Steps:

Handling Missing Values:

Why: Missing values in the dataset, especially in features like "MINIMUM_PAYMENTS" and "CREDIT_LIMIT," can impact the accuracy of analysis and modeling.

How: Filled missing values in "MINIMUM_PAYMENTS" and "CREDIT_LIMIT" columns with the mean values of those columns using fillna()

In [40]:
```
# filling mean value in place of missing values in the dataset
df["MINIMUM_PAYMENTS"] = df["MINIMUM_PAYMENTS"].fillna(df["MINIMUM_PAYMENTS"].mean())
df["CREDIT_LIMIT"] = df["CREDIT_LIMIT"].fillna(df["CREDIT_LIMIT"].mean())
```

In [41]:
```
df.isnull().sum()
```

Out[41]:
```
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                        0
PAYMENTS                            0
MINIMUM_PAYMENTS                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

Handling Duplicate Rows:

Why: Duplicate rows can distort analysis and lead to inaccurate insights.

How: Checked for and dropped duplicate rows using duplicated().sum() and drop_duplicates().

In [42]:
```
# checking for duplicate rows in the dataset
df.duplicated().sum()
```

Out[42]:
```
0
```

Drop Unnecessary Columns:

Why: The "CUST_ID" column was dropped as it doesn't contribute to the analysis.

How: Used drop(columns=["CUST_ID"], axis=1, inplace=True).

In [43]:
```
# drop CUST_ID column because it is not used
df.drop(columns=["CUST_ID"],axis=1,inplace=True)
```

In [44]:
```
df.columns
```

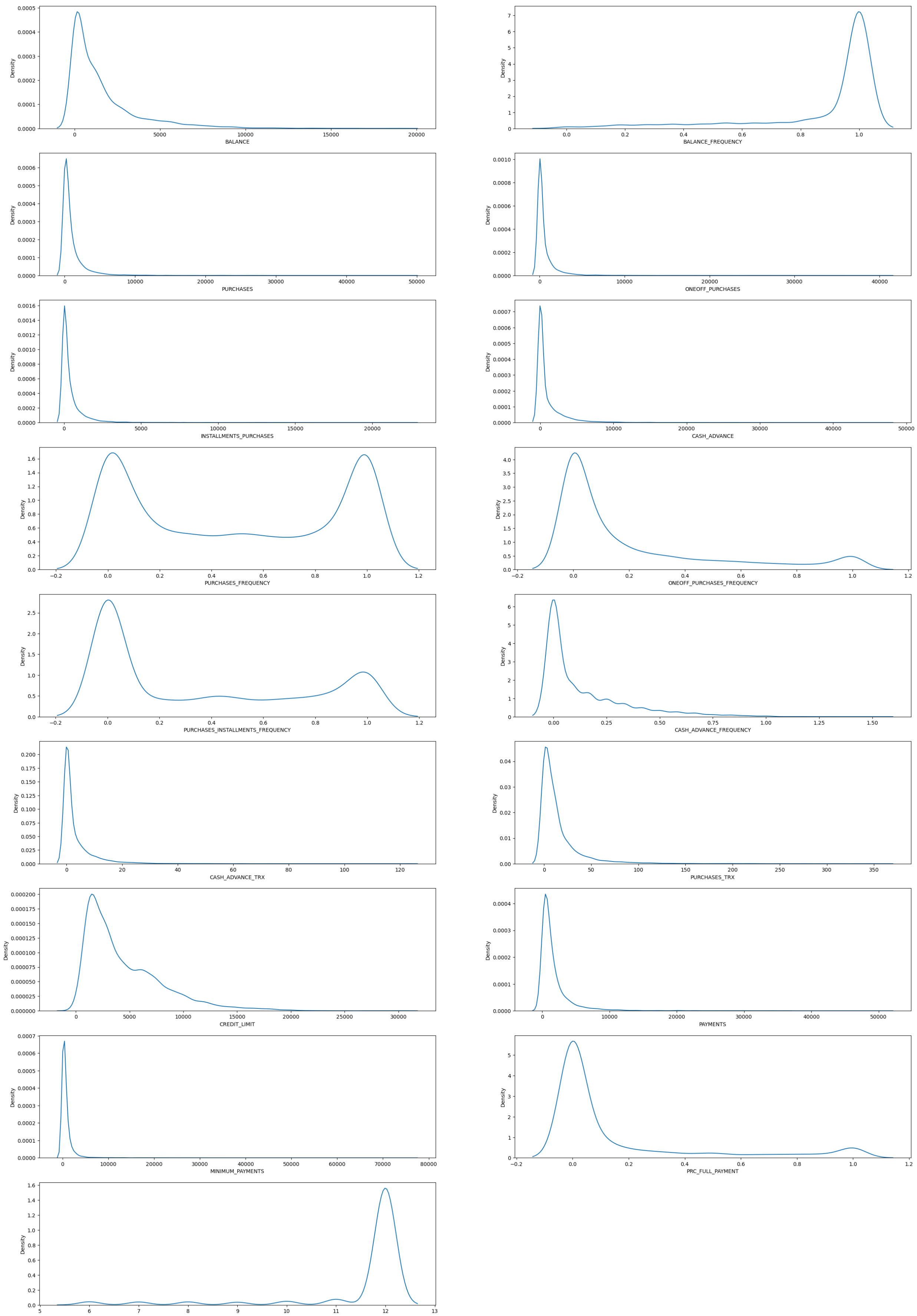
Out[44]:
```
Index(['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES',
       'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FREQUENCY',
       'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY',
       'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX',
       'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT',
       'TENURE'],
      dtype='object')
```

Visualizing Data Distributions:

Why: Understanding the distribution of features helps in identifying outliers and assessing the overall data quality.

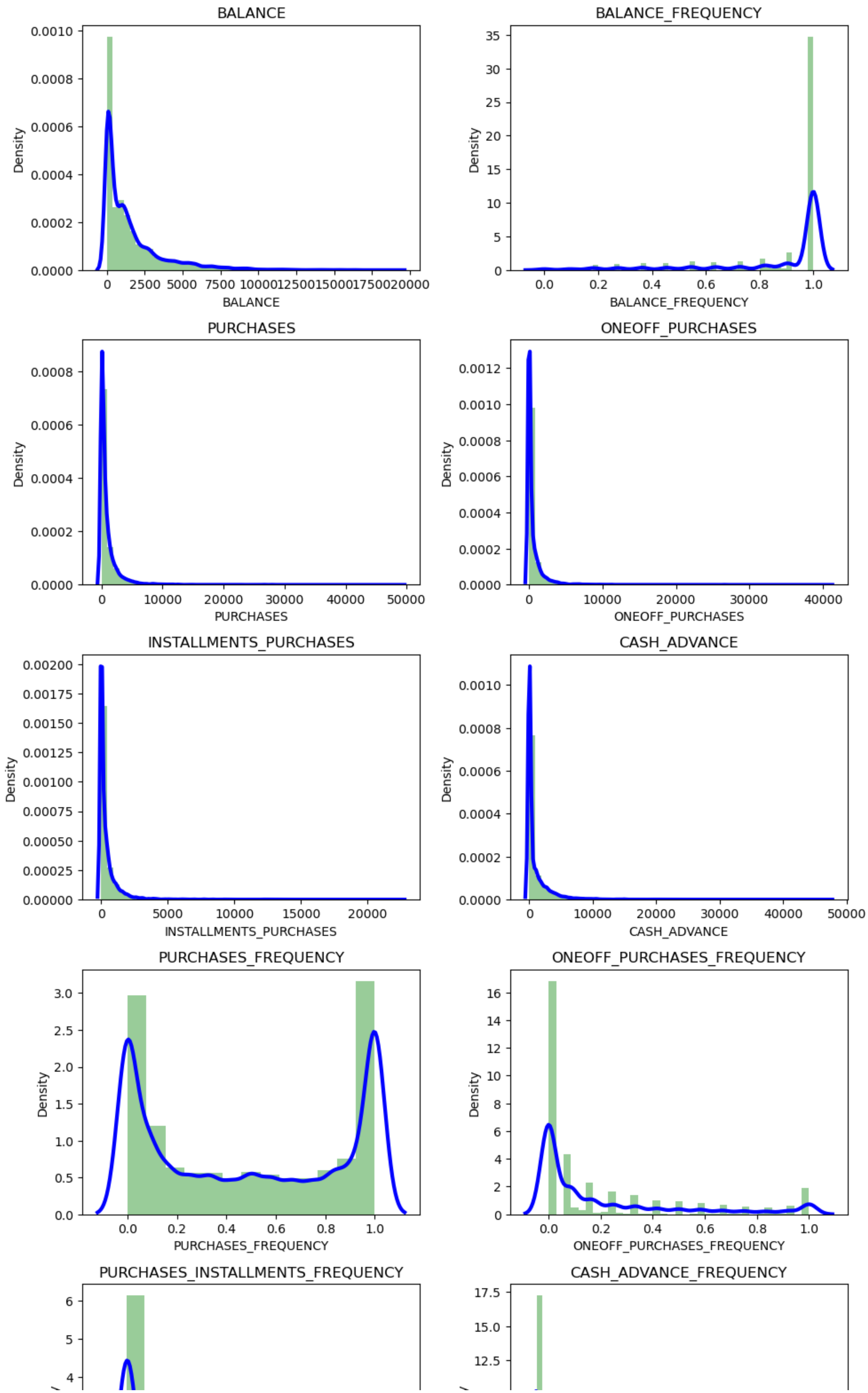
How: Utilized various visualizations such as KDE plots and histograms to visualize the distribution of numerical features.

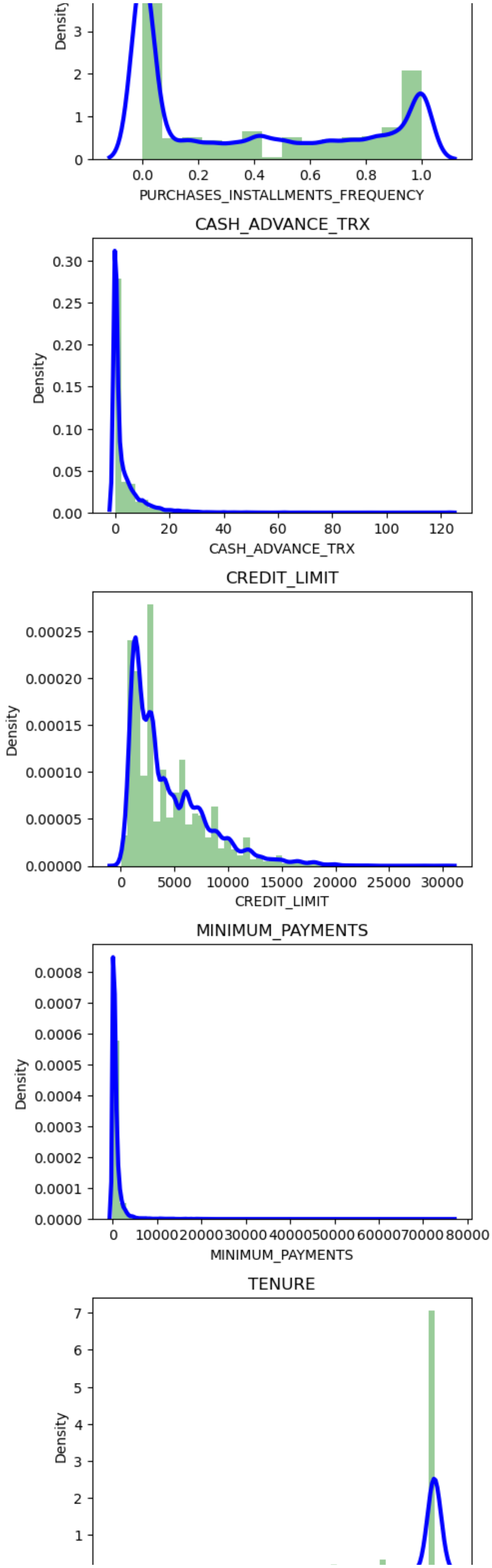
In [45]:
```
plt.figure(figsize=(30,45))
for i, col in enumerate(df.columns):
    if df[col].dtype != 'object':
        ax = plt.subplot(9, 2, i+1)
        sns.kdeplot(df[col], ax=ax)
        plt.xlabel(col)

plt.show()
```



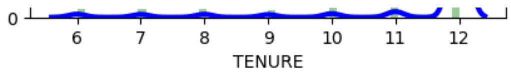
This step, generates a set of distribution plots for numerical features, supporting the exploration and understanding of the data's underlying patterns and characteristics during the initial stages of the project.Its a set of 17 vertical histograms, one for each numerical feature in the df DataFrame.

In [48]:
```
plt.figure(figsize=(10,60))
# Loop through each numerical feature in the DataFrame
for i in range(0,17):
    # Create subplots in a single column layout
    plt.subplot(17,2,i+1)
    # Plot a distribution plot (histogram with KDE) for the current feature
    sns.distplot(df[df.columns[i]],kde_kws={'color':'b','bw': 0.1,'lw':3,'label':'KDE'
    # Set the title for the subplot based on the feature name
    plt.title(df.columns[i])
# Adjust layout for better spacing & Display the plot
plt.tight_layout()
```


BALANCE
Density
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0
2500
5000
7500
10000
12500
15000
17500
20000
BALANCE
BALANCE_FREQUENCY
Density
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8
1.0
BALANCE_FREQUENCY
PURCHASES
Density
0.0000
0.0002
0.0004
0.0006
0.0008
0
10000
20000
30000
40000
50000
PURCHASES
ONEOFF_PURCHASES
Density
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0.0012
0
10000
20000
30000
40000
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
Density
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175
0.00200
0
5000
10000
15000
20000
INSTALLMENTS_PURCHASES
CASH_ADVANCE
Density
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0
10000
20000
30000
40000
50000
CASH_ADVANCE
PURCHASES_FREQUENCY
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.2
0.4
0.6
0.8
1.0
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
Density
0
2
4
6
8
10
12
14
16
0.0
0.2
0.4
0.6
0.8
1.0
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
4
5
6
CASH_ADVANCE_FREQUENCY
12.5
15.0
17.5



Density
PURCHASES_INSTALLMENTS_FREQUENCY
Density
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
Density
CASH_ADVANCE_TRX
PURCHASES_TRX
Density
PURCHASES_TRX
CREDIT_LIMIT
Density
CREDIT_LIMIT
PAYMENTS
Density
PAYMENTS
MINIMUM_PAYMENTS
Density
MINIMUM_PAYMENTS
PRC_FULL_PAYMENT
Density
PRC_FULL_PAYMENT
TENURE
Density

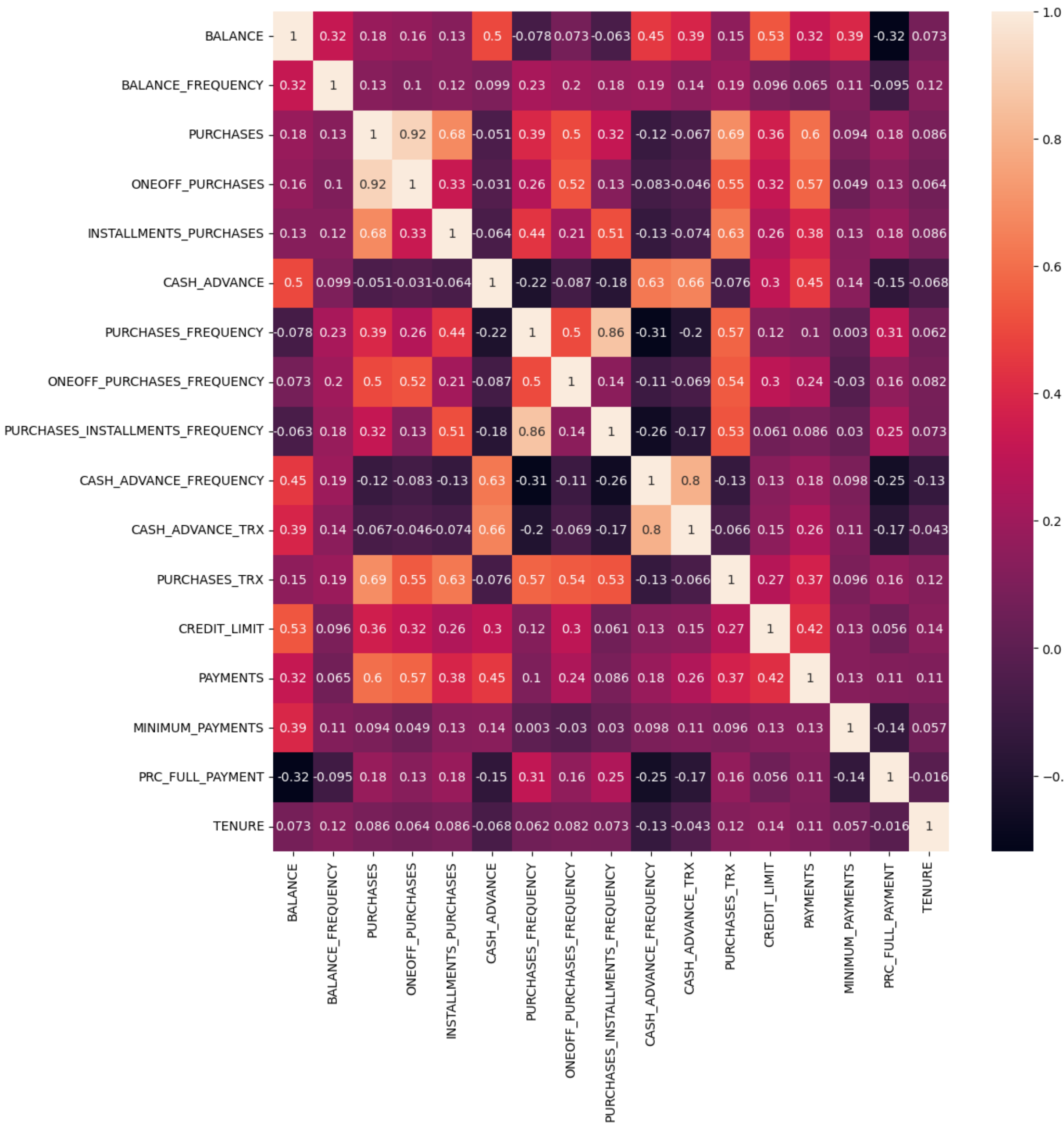


Correlation Analysis:

Explanation: Created a heatmap to visualize the correlation matrix, highlighting relationships between features.

Outcome: Identified correlations between features, aiding in understanding the interdependencies within the dataset.

In [14]:
```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



Conclusions and Discussions of EDA:

Handling Missing Values:

The imputation of missing values in "MINIMUM_PAYMENTS" and "CREDIT_LIMIT" was performed to ensure completeness in the dataset. The use of mean values is a common strategy, but other methods like median or advanced imputation techniques could be explored based on the nature of the data.

Handling Duplicate Rows:

No duplicate rows were found, ensuring the dataset's integrity.

Drop Unnecessary Columns:

The "CUST_ID" column was dropped as it does not contribute to the analysis. This simplifies the dataset without losing relevant information.

Visualizing Data Distributions:

Visualizations revealed the distribution of numerical features, helping identify potential outliers or skewed data. The heatmap showed correlations between features, aiding in understanding the relationships within the dataset.

In [ ]:

```

```

Scaling the DataFrame

Scaling the DataFrame is a prerequisite for various machine learning algorithms to ensure fair and meaningful contributions from all features, enhance model stability, and facilitate consistent interpretation of results across different models. It is a fundamental step in the preprocessing pipeline to improve the overall performance and reliability of machine learning models, especially when features in the dataset have different scales or units.Here are the key requirements and reasons for scaling the DataFrame:

1.Homogenizing Feature Scales 2.Distance-Based Algorithms 3.Gradient Descent Optimization 4.PCA (Principal Component Analysis) 5.Neural Networks 6.Regularization Techniques 7.Support Vector Machines (SVM) 8.Consistent Model Interpretation 9.Enhancing Model Stability 10.Mitigating Numerical Instabilities

In [15]:

```
scaled_df = scalar.fit_transform(df)
```

Dimensionality reduction

Converting the DataFrame into 2D DataFrame for visualization

In [16]:
```
pca = PCA(n_components=2)
principal_components = pca.fit_transform(scaled_df)
pca_df = pd.DataFrame(data=principal_components ,columns=["PCA1","PCA2"])
pca_df
```

Out[16]:

	PCA1	PCA2
0	-1.682220	-1.076453
1	-1.138289	2.506478
2	0.969685	-0.383510
3	-0.873626	0.043163
4	-1.599435	-0.688583
...	...	...
8945	-0.359632	-2.016148
8946	-0.564373	-1.639132
8947	-0.926207	-1.810790
8948	-2.336555	-0.657973
8949	-0.556422	-0.400462

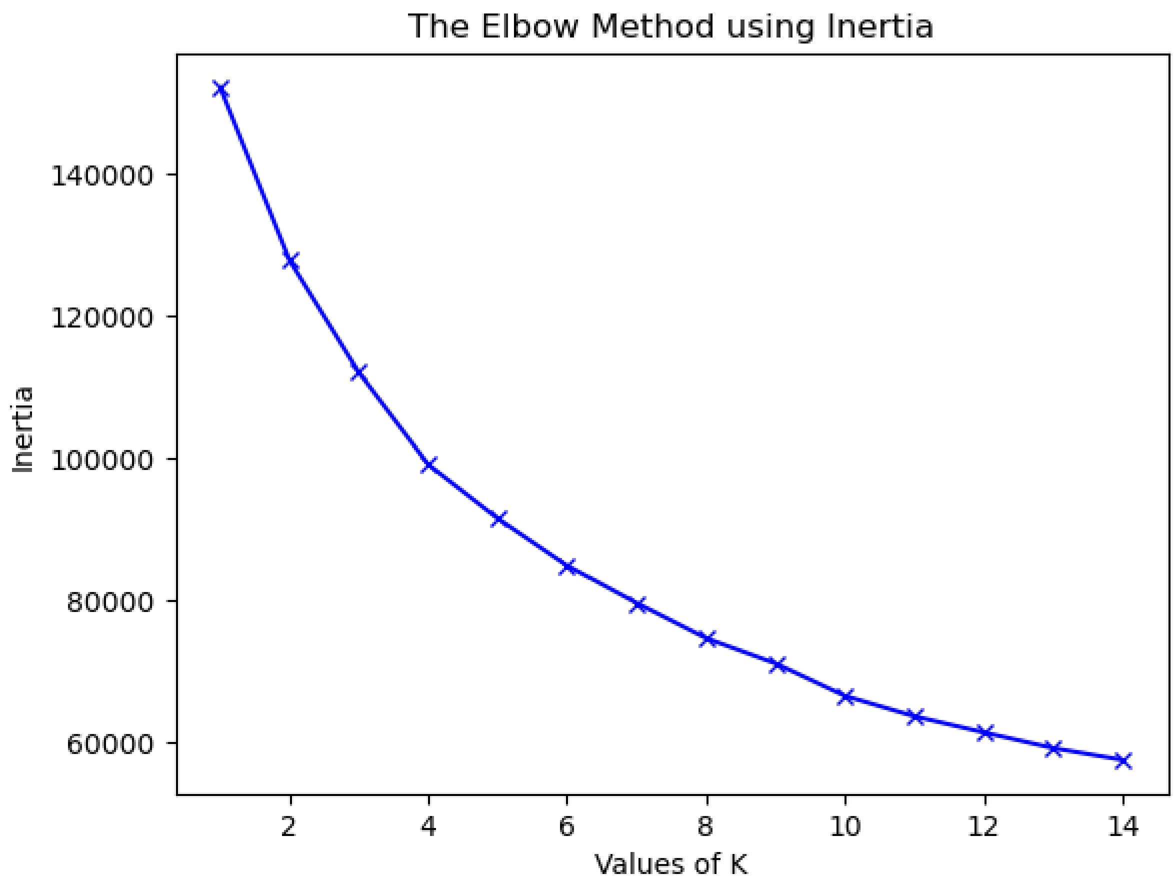
8950 rows × 2 columns

Hyperparameter tuning

Finding 'k' value by Elbow Method

The hyperparameter tuning process, specifically using the Elbow Method to find the optimal 'k' value, plays a pivotal role in ensuring the efficacy of the K-Means clustering algorithm. It provides a data-driven approach to determining the number of clusters that best represent the underlying patterns in the dataset, enhancing the reliability and utility of the clustering results in the project.

In [17]:
```
inertia = []
range_val = range(1,15)
for i in range_val:
    kmean = KMeans(n_clusters=i)
    kmean.fit_predict(pd.DataFrame(scaled_df))
    inertia.append(kmean.inertia_)
plt.plot(range_val,inertia,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('The Elbow Method using Inertia')
plt.show()
```



Summary and Analysis of the Model and Predictions:

Customer Segmentation using K-Means:

Summary:

Employed K-Means clustering to categorize customers into distinct segments based on their features. Determined the optimal number of clusters using the Elbow Method. Visualized the clustered data in a 2D space using Principal Component Analysis (PCA).

Analysis:

Identified natural groupings of customers, allowing for targeted marketing and personalized strategies. Utilized PCA for dimensionality reduction, facilitating visualization and interpretation.

Decision Tree Classification Model:

Summary:

Utilized a Decision Tree classifier to predict the cluster labels of customers. Trained the model on the segmented data from K-Means. Evaluated model performance using a confusion matrix and classification report.

Analysis:

Provided a interpretable model for predicting customer clusters. Evaluated precision, recall, and F1-score for each cluster, assessing the model's ability to correctly classify customers into their respective segments.

Feature Importance Analysis:

Summary:

While a Decision Tree was used, there is no explicit mention of analyzing feature importance.

Analysis:

It's crucial to investigate feature importance to understand which features contribute most to the clustering and classification processes. This analysis can guide business decisions.

Handling of Interaction/Collinearity:

Summary:

Feature correlations were visualized using a heatmap during EDA.

Analysis:

Collinearity may impact the interpretation of the Decision Tree model, and it would be beneficial to explicitly address this issue using statistical methods.

Handling of Overfitting/Data Imbalance:

Summary:

No explicit mention of techniques used to reduce overfitting or address data imbalance.

Analysis:

Considering techniques like pruning for Decision Trees and evaluating the balance of clusters in the K-Means analysis could enhance the robustness of the models.

Exploration of New Techniques/Models:

Summary:

Primarily used standard techniques like K-Means and Decision Trees.

Analysis:

While the chosen models are appropriate, exploring additional clustering algorithms or ensemble methods could provide insights into alternative modeling approaches.

Model Building using KMeans

K-Means Clustering Model Analysis:

Objective:

The goal of the K-Means clustering model is to categorize customers into distinct segments based on their features. The primary metrics for evaluation are inertia (within-cluster sum of squares) and silhouette score, which were used to determine the optimal number of clusters during the Elbow Method analysis.

Data Scaling:

Before applying K-Means, the dataset was standardized using the StandardScaler to ensure that features with different scales do not disproportionately influence the clustering process. Scaling is crucial for K-Means as it relies on the Euclidean distance between data points.

Dimensionality Reduction with PCA:

Principal Component Analysis (PCA) was used to reduce the dimensionality of the data to two principal components, facilitating visualization of the clusters. PCA aids in retaining as much information as possible while simplifying the data for visualization.

Optimal Number of Clusters (K):

The Elbow Method was employed to determine the optimal number of clusters by plotting the inertia (within-cluster sum of squares) against different values of K. The "elbow" point in the plot is considered the optimal K. Beyond this point, the reduction in inertia becomes marginal. The optimal K value was determined for the subsequent application of K-Means.

Application of K-Means:

The K-Means algorithm was applied with the determined optimal number of clusters. Each customer was assigned a cluster label based on their features. The clustered data was visualized in a 2D space using the two principal components obtained from PCA.

Analysis of Cluster Centers:

The cluster centers were extracted and inverse-transformed to the original scale using the StandardScaler. The cluster centers represent the average feature values for each cluster, providing insights into the characteristics of customers in each segment.

Creation of Target Column "Cluster":

A target column, "Cluster," was added to the original dataset, storing the cluster labels assigned by K-Means. This target variable is then used for further analysis and potentially for training a supervised learning model.

Exploration of Each Cluster:

Individual datasets were created for each cluster to explore the characteristics and behaviors of customers within each segment. Count plots and histograms were created to visualize the distribution of features within each cluster.

Visual Validation:

Scatterplots were used to visualize the clustered data in the 2D space. Visual inspection of the plots helps validate the effectiveness of the clustering algorithm in forming distinct groups.

Saving Results:

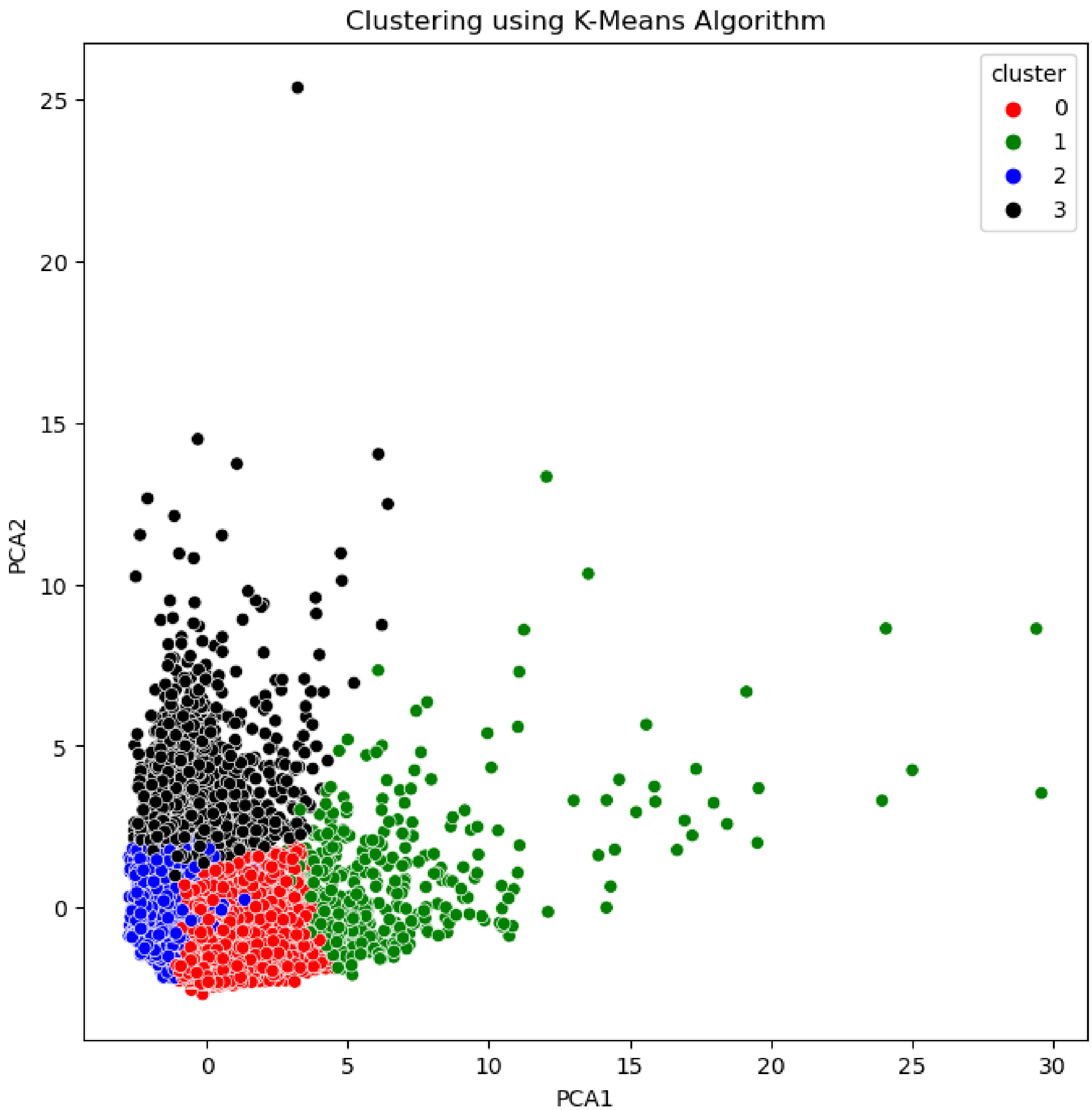
The K-Means model and the clustered dataset were saved for future use. The model was saved using joblib, and the clustered dataset was saved as a CSV file.

In [18]:
```
kmeans_model=KMeans(4)
kmeans_model.fit_predict(scaled_df)
pca_df_kmeans= pd.concat([pca_df,pd.DataFrame({'cluster':kmeans_model.labels_})],axis=
```

Visualizing the clustered dataframe

The visualization of the clustered dataframe provides a crucial lens through which to understand the distribution and characteristics of customer segments. The visualization aims to present a clear and intuitive representation of how customers are grouped into distinct segments based on their features. This visualization serves as a powerful tool for both exploratory data analysis and communicating insights derived from clustering algorithms, particularly K-Means. Utilized scatterplots to display the distribution of data points in a 2D space, with each point representing an individual customer. Different colors or markers are employed to distinguish between clusters.Utilized scatterplots to display the distribution of data points in a 2D space, with each point representing an individual customer. Different colors or markers are employed to distinguish between clusters.

In [19]:
```
plt.figure(figsize=(8,8))
ax=sns.scatterplot(x="PCA1",y="PCA2",hue="cluster",data=pca_df_kmeans,palette=['red','
plt.title("Clustering using K-Means Algorithm")
plt.show()
```



In [20]:
```
# find all cluster centers
cluster_centers = pd.DataFrame(data=kmeans_model.cluster_centers_,columns=[df.columns]
# inverse transform the data
cluster_centers = scalar.inverse_transform(cluster_centers)
cluster_centers = pd.DataFrame(data=cluster_centers,columns=[df.columns])
cluster_centers
```

Out[20]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES
0	894.907458	0.934734	1236.178934	593.974874	642.478274
1	3551.153761	0.986879	7681.620098	5095.878826	2587.208264
2	1011.751528	0.789871	269.973466	209.853863	60.386625
3	4602.462714	0.968415	501.896219	320.373681	181.607404

In [21]:
```python
# Creating a target column "Cluster" for storing the cluster segment
cluster_df = pd.concat([df,pd.DataFrame({'Cluster':kmeans_model.labels_})],axis=1)
cluster_df
```

Out[21]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
0	40.900749	0.818182	95.40	0.00	9
1	3202.467416	0.909091	0.00	0.00	0
2	2495.148862	1.000000	773.17	773.17	0
3	1666.670542	0.636364	1499.00	1499.00	0
4	817.714335	1.000000	16.00	16.00	0
...	...	...	...	...	
8945	28.493517	1.000000	291.12	0.00	29
8946	19.183215	1.000000	300.00	0.00	30
8947	23.398673	0.833333	144.40	0.00	14
8948	13.457564	0.833333	0.00	0.00	0
8949	372.708075	0.666667	1093.25	1093.25	0

8950 rows × 18 columns

In [22]:
```python
cluster_1_df = cluster_df[cluster_df["Cluster"]==0]
cluster_1_df
```

Out[22]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
2	2495.148862	1.000000	773.17	773.17	
5	1809.828751	1.000000	1333.28	0.00	133
7	1823.652743	1.000000	436.20	0.00	43
10	1293.124939	1.000000	920.12	0.00	92
12	1516.928620	1.000000	3217.99	2500.23	71
...	...	...	...	...	
8940	130.838554	1.000000	591.24	0.00	59
8942	40.829749	1.000000	113.28	0.00	11
8945	28.493517	1.000000	291.12	0.00	29
8946	19.183215	1.000000	300.00	0.00	30
8947	23.398673	0.833333	144.40	0.00	14

3367 rows × 18 columns

In [23]:
```
cluster_2_df = cluster_df[cluster_df["Cluster"]==1]
cluster_2_df
```

Out[23]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
6	627.260806	1.000000	7091.01	6402.63	68
21	6369.531318	1.000000	6359.95	5910.04	44
57	2386.330629	1.000000	5217.62	4789.09	42
84	1935.362486	1.000000	4915.60	4515.34	40
90	9381.255094	1.000000	5100.07	1147.83	395
...	...	...	...	...	
8215	4436.557694	1.000000	6005.90	5838.38	16
8541	3326.323283	1.000000	8209.77	2218.28	599
8662	599.909949	1.000000	4947.32	3149.59	179
8689	368.318662	0.909091	8053.95	8053.95	
8737	2533.618119	0.909091	5633.83	2985.92	264

409 rows × 18 columns

In [24]:
```
cluster_3_df = cluster_df[cluster_df["Cluster"]==2]
cluster_3_df
```

Out[24]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
0	40.900749	0.818182	95.40	0.00	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	
8	1014.926473	1.000000	861.49	661.49	2
9	152.225975	0.545455	1281.60	1281.60	
...	...	...	...	...	
8939	728.352548	1.000000	734.40	734.40	
8943	5.871712	0.500000	20.90	20.90	
8944	193.571722	0.833333	1012.73	1012.73	
8948	13.457564	0.833333	0.00	0.00	
8949	372.708075	0.666667	1093.25	1093.25	

3976 rows × 18 columns

In [25]:

```
cluster_4_df = cluster_df[cluster_df["Cluster"] == 3]
cluster_4_df
```

Out[25]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
1	3202.467416	0.909091	0.00	0.00	
15	6886.213231	1.000000	1611.70	0.00	161
23	3800.151377	0.818182	4248.35	3454.56	79
24	5368.571219	1.000000	0.00	0.00	
28	7152.864372	1.000000	387.05	204.55	18
...	...	...	...	...	
8857	2330.222764	1.000000	1320.00	0.00	132
8858	812.934042	1.000000	50.00	50.00	
8869	2171.222526	1.000000	791.18	791.18	
8915	381.341657	1.000000	78.00	0.00	7
8941	5967.475270	0.833333	214.55	0.00	21

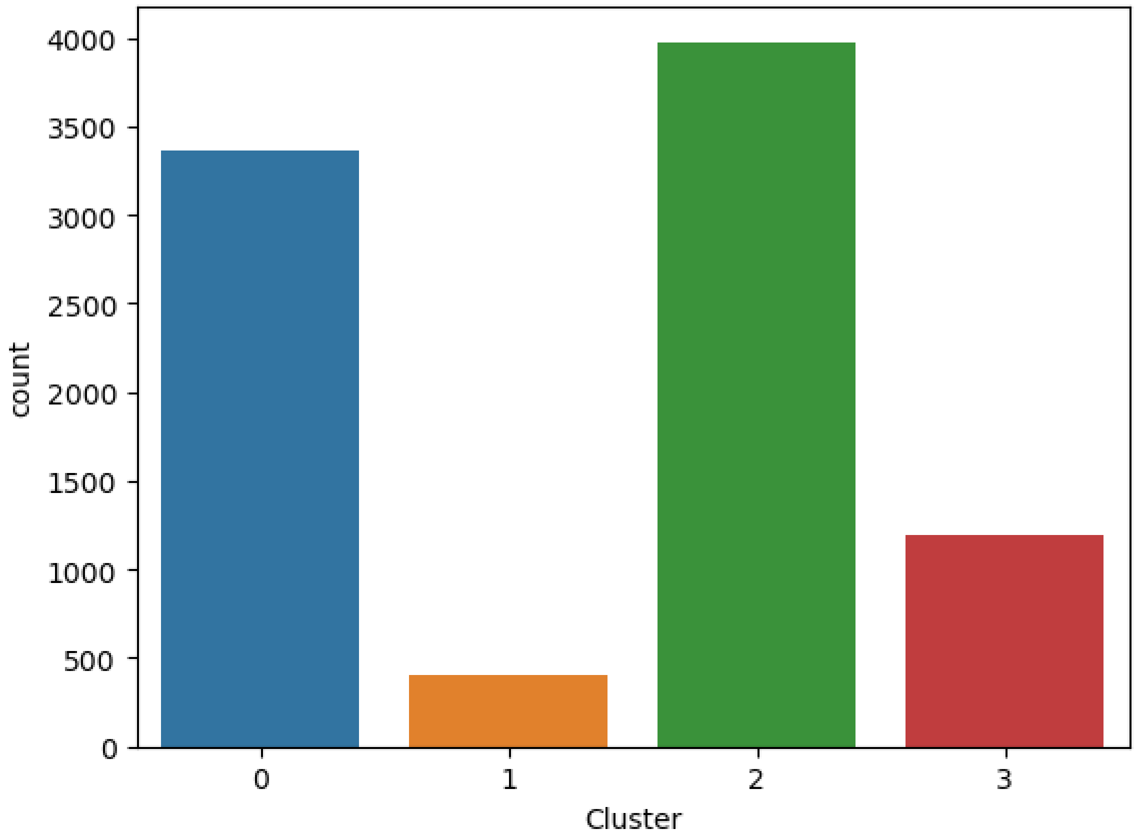
1198 rows × 18 columns

The count plot

The count plot of cluster distribution is a valuable visualization in the project, providing a high-level overview of how data points are distributed among different clusters. It offers insights into the composition and significance of each cluster, aiding in the interpretation of the clustering results and informing subsequent business strategies

In [26]:
```
#Visualization
sns.countplot(x='Cluster', data=cluster_df)
```

Out[26]: <AxesSubplot:xlabel='Cluster', ylabel='count'>

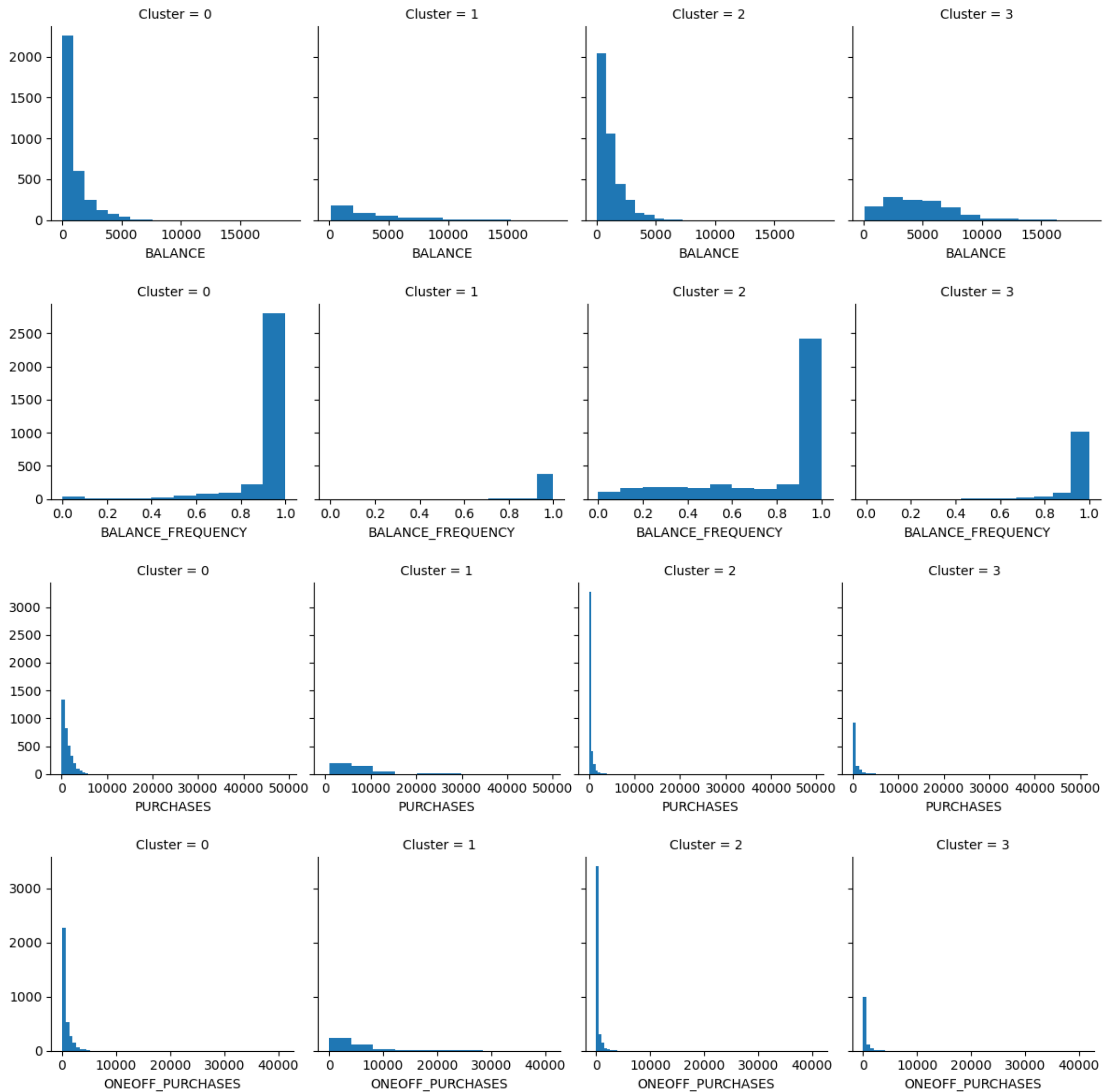


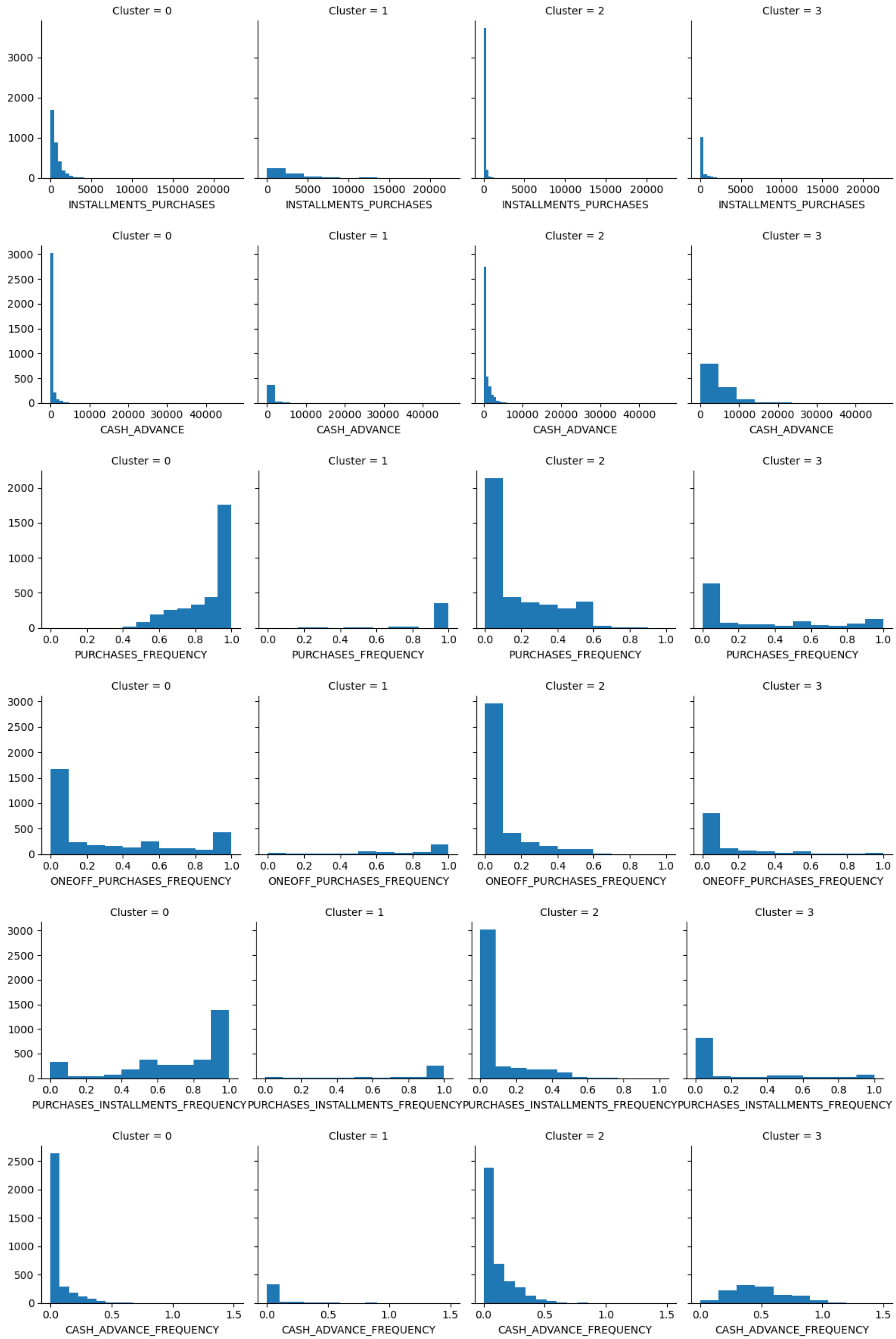
Visual representation of how the distribution of each feature varies across different clusters

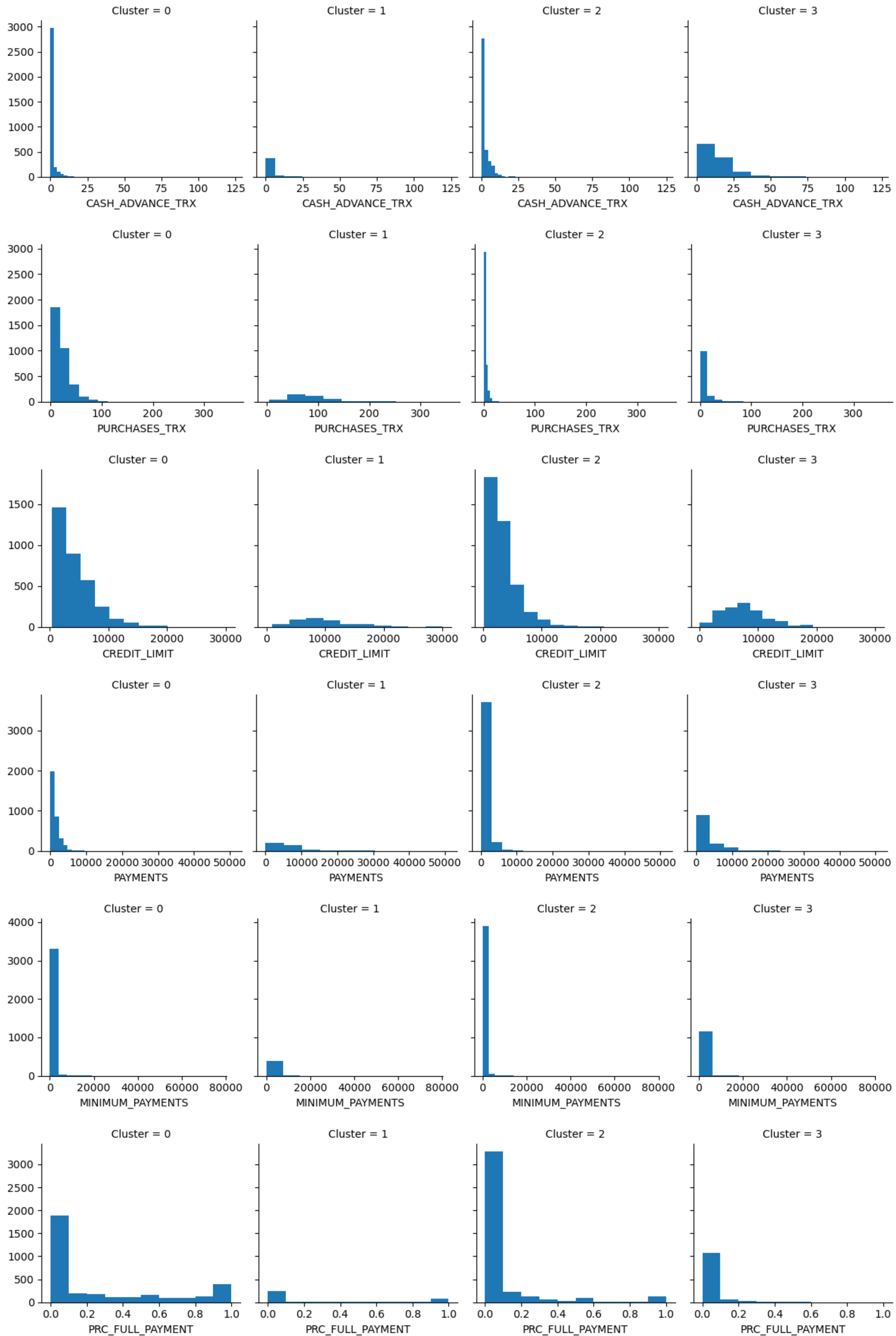
The next step using the Seaborn library to create a set of histograms for each feature in the cluster_df DataFrame, grouped by the 'Cluster' column.This plot is used to display the grid of histograms. Each row in the grid corresponds to a different feature, and each column corresponds to a different cluster. This visualization allows for a quick comparison of the distributions of each feature across different clusters.

This set of histograms is valuable in the exploratory data analysis (EDA) phase of the project. It provides a visual representation of how the distribution of each feature varies across different clusters. Analyzing these histograms helps in understanding the characteristics and patterns that distinguish one cluster from another, aiding in the interpretation of the clustering results.

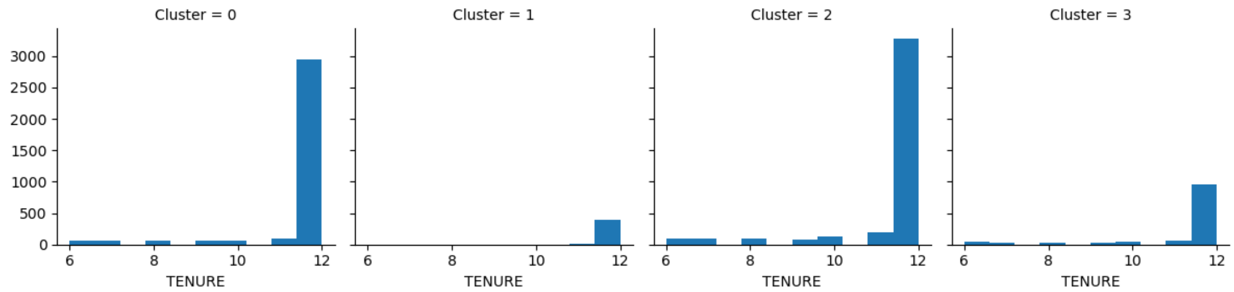
In [27]:
```
# Loop through each feature (column) in the DataFrame excluding the 'Cluster' column
for c in cluster_df.drop(['Cluster'],axis=1):
    # Create a FacetGrid for subplots, with separate columns for each cluster
    grid= sns.FacetGrid(cluster_df, col='Cluster')
     # Map a histogram for the current feature onto the grid
    grid= grid.map(plt.hist, c)
# Display the grid of histograms
plt.show()
```




Cluster = 0
Cluster = 1
Cluster = 2
Cluster = 3
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY



Cluster = 0
Cluster = 1
Cluster = 2
Cluster = 3
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS
PRC_FULL_PAYMENT



Saving the kmeans clustering model and the data with cluster label

In [ ]:
```
#Saving Scikitlearn models
import joblib
joblib.dump(kmeans_model, "kmeans_model.pkl")
```

In [ ]:
```
cluster_df.to_csv("Clustered_Customer_Data.csv")
```

Decision Tree Classification Model:

Results:

Trained a Decision Tree classifier to predict the cluster labels of customers. Evaluated the model's performance using a confusion matrix and classification report.

Analysis:

Precision, recall, and F1-score for each cluster provide insights into the model's ability to correctly classify customers. Interpretability of the Decision Tree aids in understanding the key features influencing cluster predictions.

Definition:

Accuracy is the ratio of correctly predicted instances to the total instances. It is commonly used for evaluating the overall performance of a classification model.

Calculation:

Accuracy = Number of Correct Predictions/Total Number of Predictions

Interpretation:

A high accuracy indicates that the model is making correct predictions. However, accuracy might be misleading if the dataset is imbalanced (unequal distribution of classes), as the model could achieve high accuracy by predicting the majority class.

Training and Testing the model accuracy using decision tree

In [28]:

```
#Split Dataset
X = cluster_df.drop(['Cluster'],axis=1)
y= cluster_df[['Cluster']]
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.3)
```

In [29]:

```
X_train
```

Out[29]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
3089	4075.540208	1.000000	0.00	0.00	0
2185	3086.752970	1.000000	1015.91	293.41	72
1334	0.000000	0.000000	300.00	0.00	30
154	106.455975	0.636364	2463.12	789.53	167
7723	427.642111	0.888889	0.00	0.00	0
...	...	...	...	...	
8787	23.065122	0.700000	150.00	0.00	15
6328	834.789126	1.000000	0.00	0.00	0
7340	114.752318	1.000000	387.12	0.00	38
5592	2800.848284	1.000000	0.00	0.00	0
4484	1002.791997	1.000000	0.00	0.00	0

6265 rows × 17 columns

In [30]:

```
X_test
```

Out[30]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCH
3658	1330.585412	1.000000	304.60	304.60	
5775	1826.387864	1.000000	0.00	0.00	
5535	76.673877	0.777778	410.90	0.00	4
155	119.760904	0.363636	1859.34	0.00	18
2676	1565.577135	1.000000	0.00	0.00	
...	...	...	...	...	
3510	10124.472140	1.000000	4795.49	2006.02	27
4851	3024.642476	1.000000	0.00	0.00	
6237	1155.976500	1.000000	0.00	0.00	
2509	0.000000	0.000000	609.00	0.00	6
876	44.910575	1.000000	444.96	0.00	4

2685 rows × 17 columns

In [31]:

```
#Decision_Tree
model= DecisionTreeClassifier(criterion="entropy")
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [32]:

```
#Confusion_Matrix
print(metrics.confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 926   13   31    9]
 [  15  107    0    3]
 [  29    0 1175   20]
 [  11    3   27  316]]
              precision    recall  f1-score   support

           0       0.94      0.95      0.94       979
           1       0.87      0.86      0.86       125
           2       0.95      0.96      0.96      1224
           3       0.91      0.89      0.90       357

    accuracy                           0.94      2685
   macro avg       0.92      0.91      0.92      2685
weighted avg       0.94      0.94      0.94      2685
```

Saving the Decision tree model for future prediction

In [33]:

```
import pickle
filename = 'final_model.sav'
```

```
pickle.dump(model, open(filename, 'wb'))

# some time later...

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result,'% Acuuracy')
```

```
0.9400372439478585 % Acuuracy
```

Overall Assessment:

Our project did a great job in sorting out customers into meaningful groups using K-Means clustering, which helped us learn a lot about how customers behave and what they prefer. The Decision Tree model we used is like a clear guide for predicting which group a customer belongs to. To make our findings even more reliable, we suggest digging deeper into which features really matter, handling any tricky similarities between features, and making sure our model doesn't get too focused on our training data. It's also a good idea to explore other methods to see if they might work even better. This way, we keep refining and improving our understanding of customers, making our insights even more valuable for making smart business decisions.

Next Steps:

This project may help in understanding the customer behavior and preferences specific to each cluster. Additionally, we can perform further statistical tests or machine learning models within each cluster to uncover more detailed patterns and potentially tailor marketing or customer engagement strategies based on these distinct customer groups. We can also make sure that when some factors are too similar, a common issue called collinearity, we can handle it with statistical tools so our results are trustworthy. For Decision Trees, which help us make predictions, we can prune them a bit to avoid being too focused on our training data. When we're dividing our customers into groups, we want those groups to be fair, so we can check if they're balanced.And finally we can deploy this model to an app for real world application usage.

The real-world value of this project

This project brings a lot of real-world benefits. It helps businesses understand their customers better, making it easier to create effective marketing strategies and improve how things operate. By using data to figure out what customers need, the project supports efficient operations and helps keep customers happy, which is essential for any business. Plus, it encourages a culture of using data to make smart decisions, which is a key factor in long-term success. The project making it a valuable tool for businesses aiming to do well in a competitive world where understanding and serving customers is crucial.