Clustering Credit Card Dataset

Data Mining

Dr.Magda Madbouly

Team Members

Name	ID	Role
Eyad Tamer	2103116	Report Summary
Nureen Ehab Mahmoud	20221465124	Data preprocessing
Zainab Mohamed Abdallah	20221310251	Exploratory Data Analysis
Noha Nael	2103130	k- Medoids
Basmala Akram	2103159	Hierarchical clustering
Tasbih Abdelhakim	20201594406	Evaluation

Table of content

INTRODUCTION

- On this task we chose a dataset which is present on Kaggle, our aim was to focus on the following points:
- data PREPROCESSING, gaining a better understanding of the data structure and characteristics, implementing k-medoids and finally evaluating the results of each clustering technique.
- This dataset summarizes the usage behavior of about 9000 active credit card holders during the last 6 months.

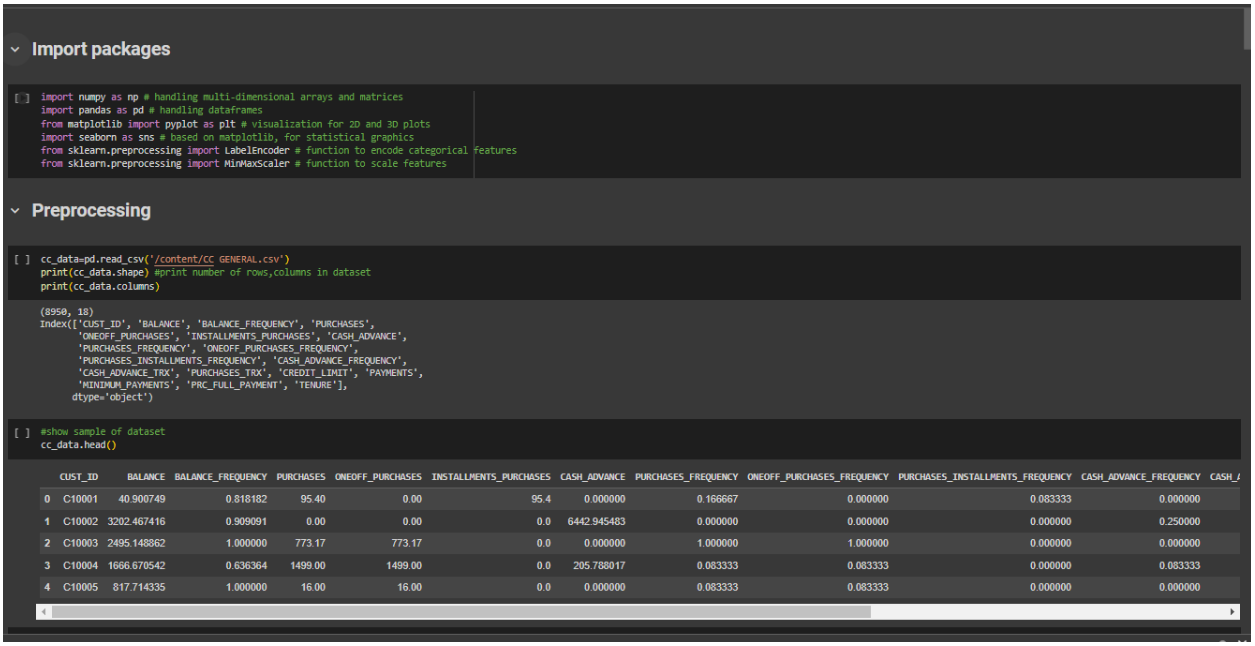
The link for the dataset USED:

https://www.kaggle.com/datasets/arjunbhasin2013/ccdata?resource=download

Data preprocessing & EDA

1. **First Step:**

Here we just imported the packages used, after that we loaded the csv data using read_csv function and then we used the .head() to print the first 5 rows and columns of the dataset and the results are shown below



Import packages

```
import numpy as np # handling multi-dimensional arrays and matrices
import pandas as pd # handling dataframes
from matplotlib import pyplot as plt # visualization for 2D and 3D plots
import seaborn as sns # based on matplotlib, for statistical graphics
from sklearn.preprocessing import LabelEncoder # function to encode categorical features
from sklearn.preprocessing import MinMaxScaler # function to scale features
```

Preprocessing

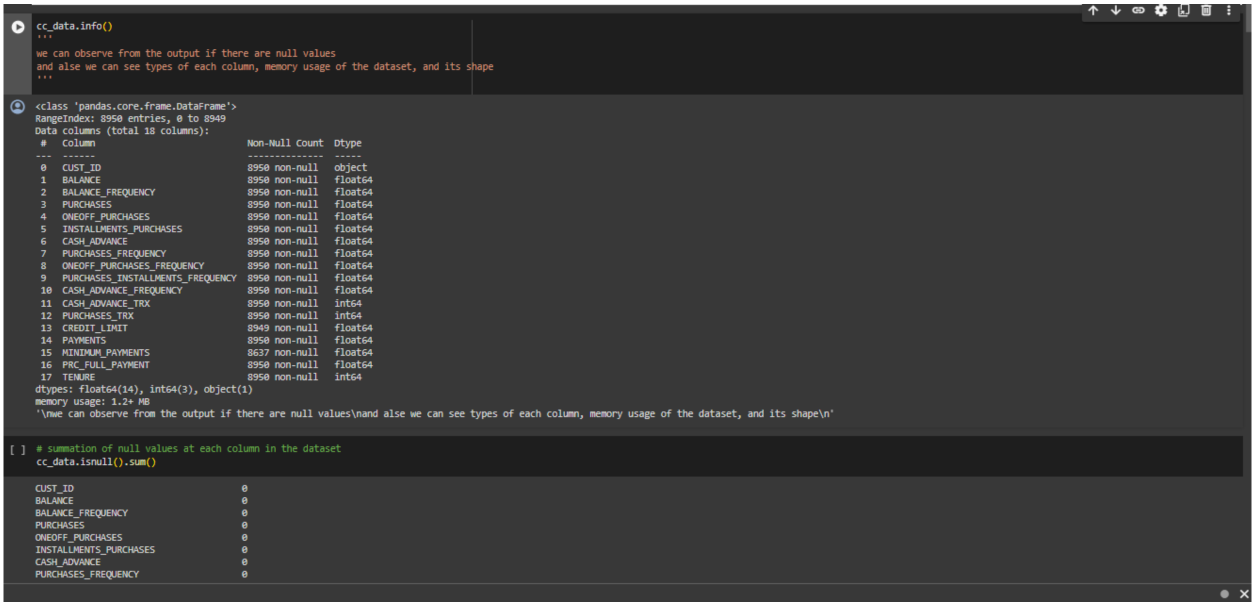
```
cc_data=pd.read_csv('/content/CC GENERAL.csv')
print(cc_data.shape) #print number of rows,columns in dataset
print(cc_data.columns)
```

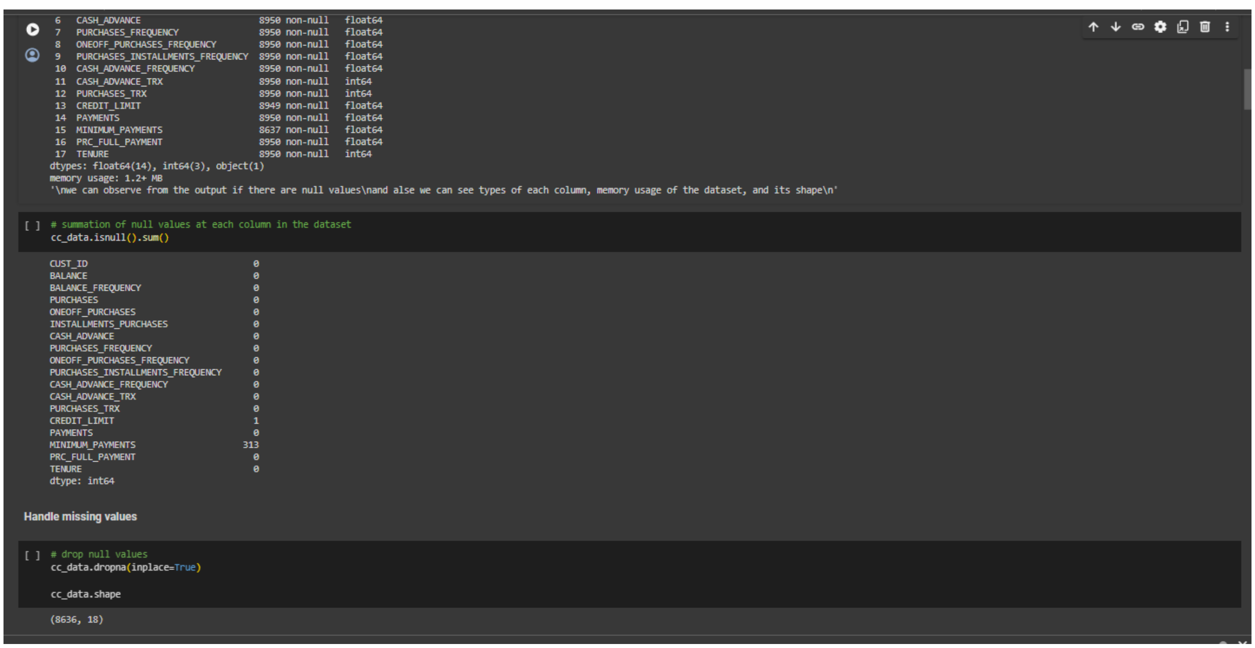
```
(8950, 18)
Index(['CUST_ID', 'BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES',
       'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'CREDIT_LIMIT', 'PAYMENTS',
       'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT', 'TENURE'],
      dtype='object')
```

```
#show sample of dataset
cc_data.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	0.000000	0.083333	0.000000
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	0.000000	0.000000	0.250000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	1.000000	0.000000	0.000000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	0.083333	0.000000	0.083333
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	0.083333	0.000000	0.000000

Here we can see that from the output there are null values, so to handle the null values we used the function dropna in order to get rid of the null values





2. **<u>Next step</u>** was checking for duplicates, we used the function ".duplicated()" in order to check and afterwards we used the .describe() function to check if we do have any outliers and here are the results

```python
# Check for duplicates
duplicates = cc_data[cc_data.duplicated()]
duplicates
```

CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVAN

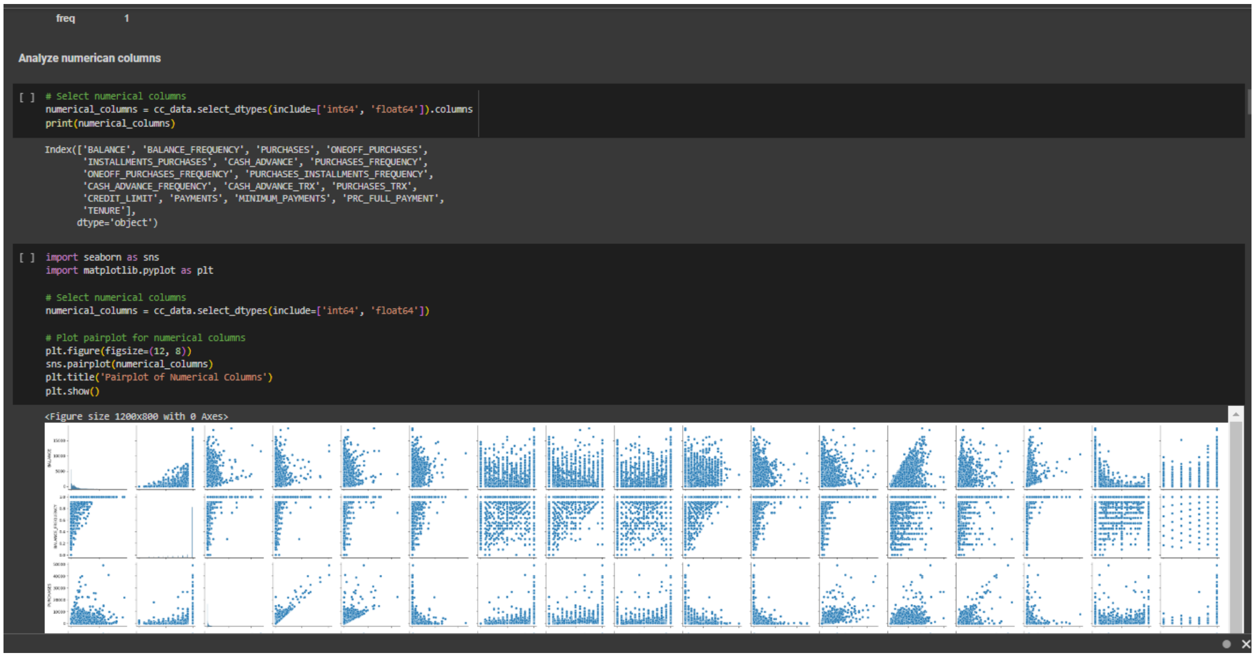
```python
# describe function Summarize Statistics for Numerical data to recognize the nature of our dataset, and to see if there are outliers
cc_data.describe()
```

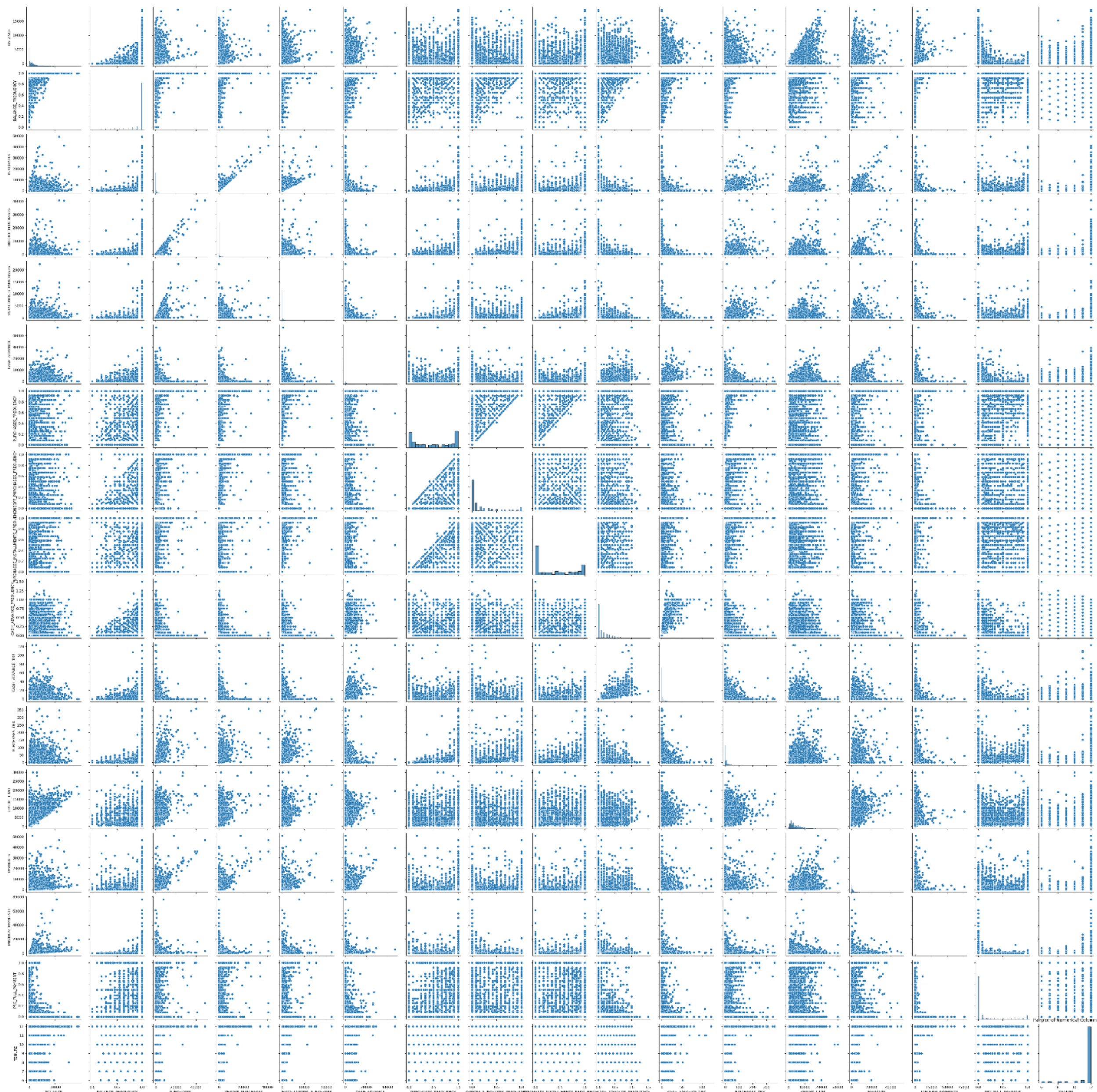
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_AD
count	8636.000000	8636.000000	8636.000000	8636.000000	8636.000000	8636.000000	8636.000000	8636.000000	8636.000000	8636.000000	8
mean	1601.224893	0.895035	1025.433874	604.901438	420.843533	994.175523	0.496000	0.205909	0.368820	0.137604	[illegible]
std	2095.571300	0.207697	2167.107984	1684.307803	917.245182	2121.458303	0.401273	0.300054	0.398093	0.201791	[illegible]
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	[illegible]
25%	148.095189	0.909091	43.367500	0.000000	0.000000	0.000000	0.083333	0.000000	0.000000	0.000000	[illegible]
50%	916.855459	1.000000	375.405000	44.995000	94.785000	0.000000	0.500000	0.083333	0.166667	0.000000	[illegible]
75%	2105.195853	1.000000	1145.980000	599.100000	484.147500	1132.385490	0.916667	0.333333	0.750000	0.250000	[illegible]
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000	1.000000	1.000000	1.500000	[illegible]

```python
# Summary Statistics for Categorical data
cc_data.describe(exclude=[np.number])
```

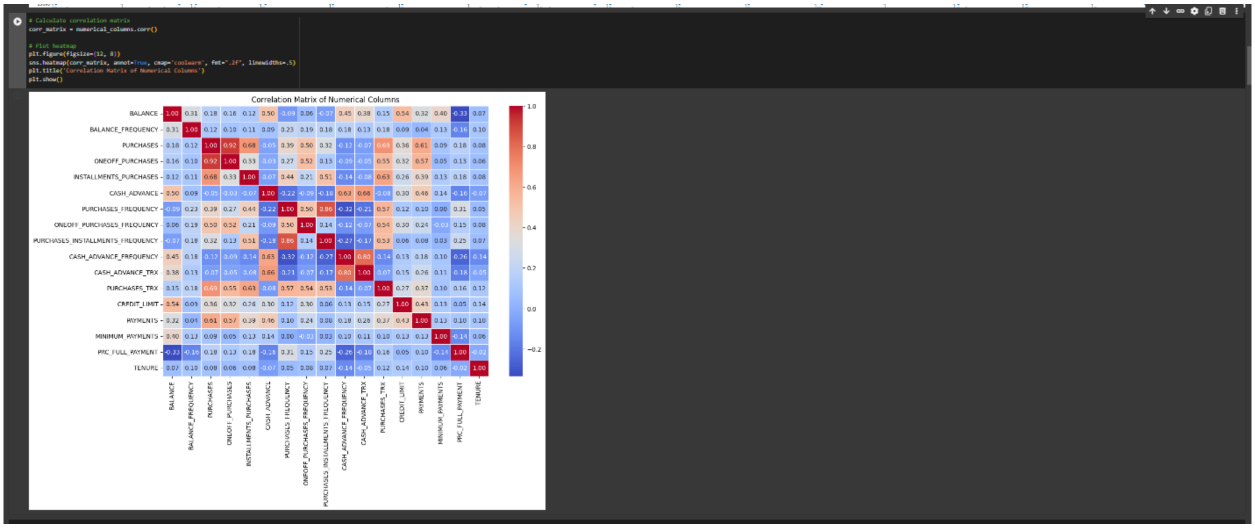
	CUST_ID
count	8636
unique	8636
top	C10001
freq	1

3. For the first cell we printed the numerical data by using “select_dtypes” method which is available in pandas, so we used data types int64 and float64 which represents numerical data.
4. Afterwards, we used plt.figure, sns.pairplot and plt.title and plt.show in order to visualize relationships between numerical columns in the dataframe

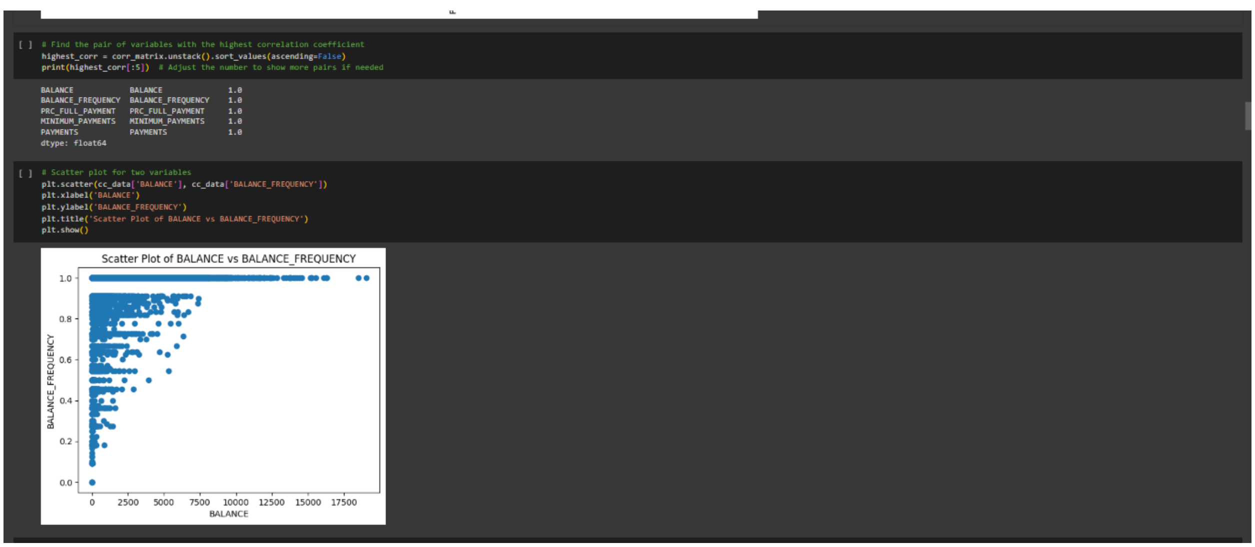




5. Then we calculated the correlation matrix for numerical columns and then visualize it as a heatmap to understand the relationships between different numerical features

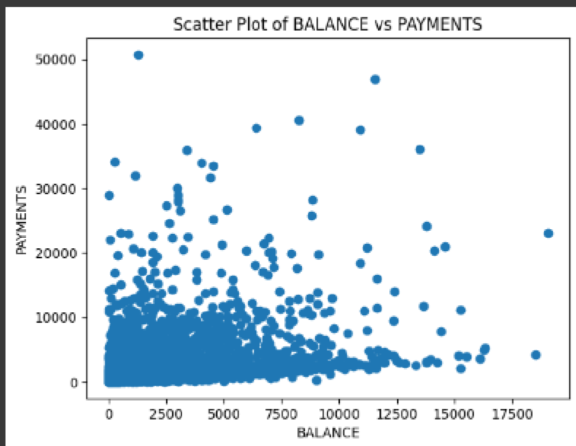


6. Now we used the following code to find the variables with the highest correlation from the correlation matrix which was calculated previously.
7. Then, we used the scatterplot to visualize the relationship between "balance" and "balance frequency" which helps understand the patterns between the 2 variables.



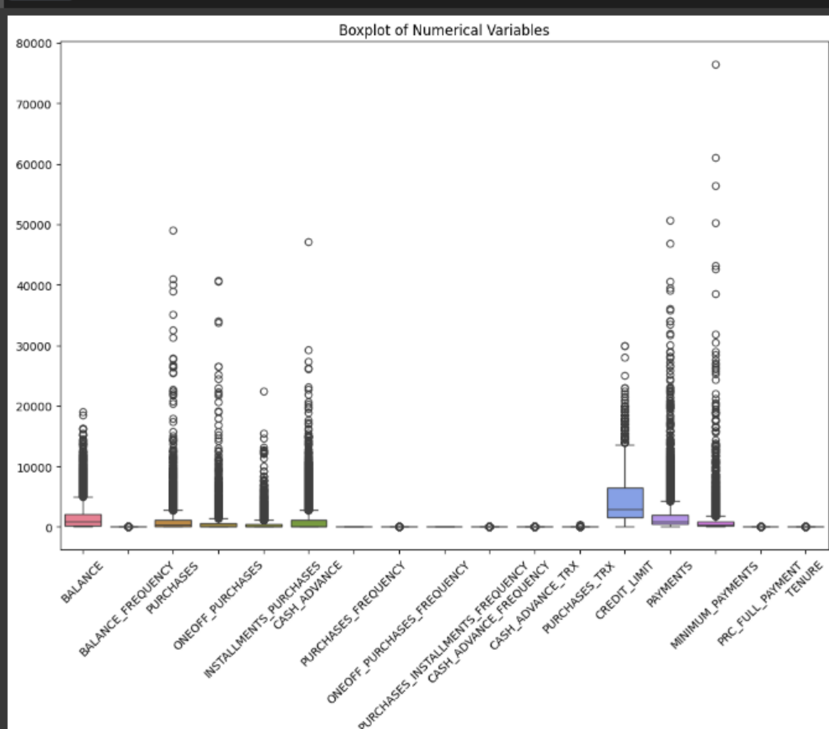
8. SAMA THING WAS DONE BUT THIS TIME THE RELATIONSHIP BETWEEN "BALANCE" AND "PAYMENTS

```python
# Scatter plot for two variables
plt.scatter(cc_data['BALANCE'], cc_data['PAYMENTS'])
plt.xlabel('BALANCE')
plt.ylabel('PAYMENTS')
plt.title('Scatter Plot of BALANCE vs PAYMENTS')
plt.show()
```



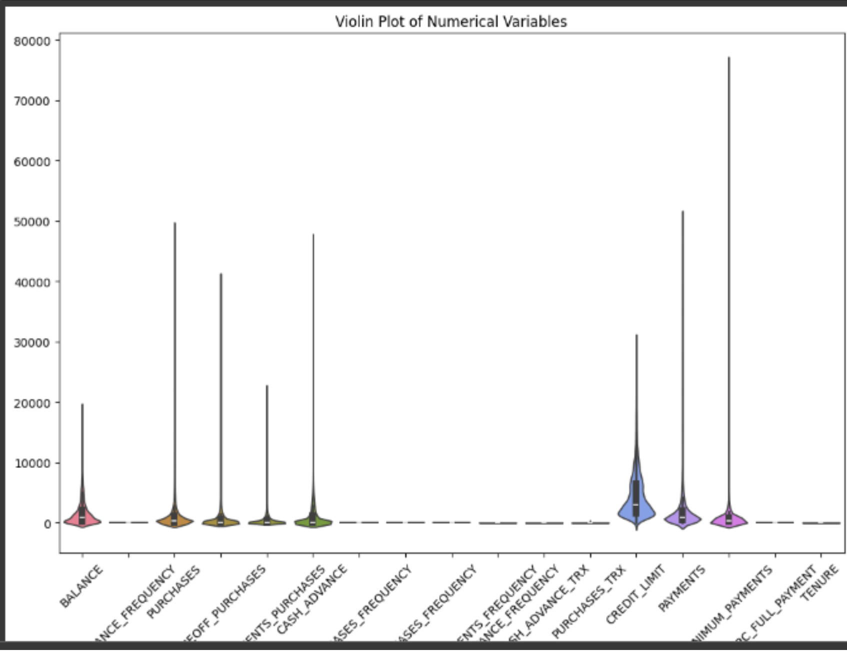
9. now we used a boxplot in order to visualize the distribution of multiple numerical values and also it presents the outliers on the graph

```python
plt.figure(figsize=(12, 8))
sns.boxplot(data=numerical_columns)
plt.title('Boxplot of Numerical Variables')
plt.xticks(rotation=45)
plt.show()
```



10. Same thing was done but this time we used the violon plot

```
# Violin plot for multiple numerical variables
plt.figure(figsize=(12, 8))
sns.violinplot(data=numerical_columns)
plt.title('Violin Plot of Numerical Variables')
plt.xticks(rotation=45)
plt.show()
```



11. NOW WE HAVE SELECTED THE CATEGORICAL COLUMN THAT WILL BE ENCODED

```
# Select categorical columns
categorical_columns = cc_data.select_dtypes(include=['object'])
categorical_columns
#we have only one categorical column that we're going to encode
```

	CUST_ID
0	C10001
1	C10002
2	C10003
4	C10005
5	C10006
...	...
8943	C19184
8945	C19186
8947	C19188
8948	C19189
8949	C19190

8636 rows × 1 columns

12. AFTERWARDS WE USED LABELENCODER() TO ENCODE THE SPECIFIED COLUMN TO NUMERICAL AND THEN WE PRINTED THE FIRST 5 ROWS AND COLUMNS USING THE HEAD FUNCTION

```
# use Label Encoding to encode "CUST_ID" column (categorical) to numerical
LE = LabelEncoder()
cc_data["CUST_ID"] = LE.fit_transform(cc_data["CUST_ID"])

cc_data.head()
```

	CUST_ID	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_
0	0	40.900749	0.818182	95.40	0.00	95.40	0.000000	0.166667	0.000000	0.083333	0.00	0	
1	1	3202.467416	0.909091	0.00	0.00	0.00	6442.945483	0.000000	0.000000	0.000000	0.25	4	
2	2	2495.148862	1.000000	773.17	773.17	0.00	0.000000	1.000000	1.000000	0.000000	0.00	0	
4	3	817.714335	1.000000	16.00	16.00	0.00	0.000000	0.083333	0.083333	0.000000	0.00	0	
5	4	1809.828751	1.000000	1333.28	0.00	1333.28	0.000000	0.666667	0.000000	0.583333	0.00	0	

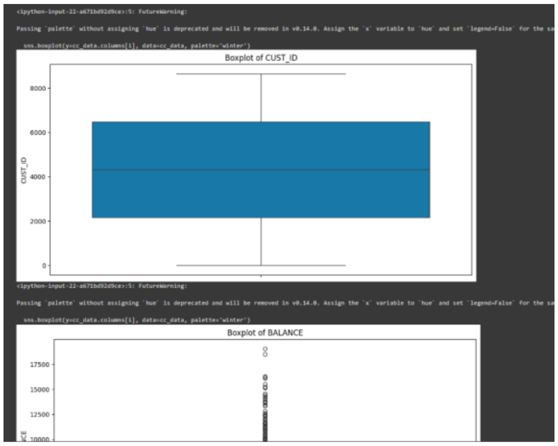
```
[ ] # Calculate additional statistics after encoding
```

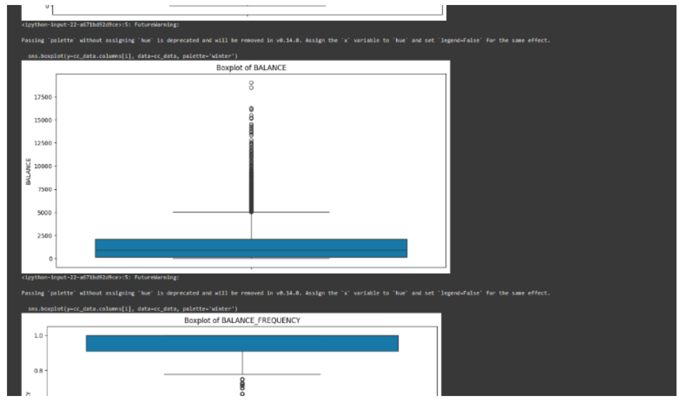
13. These statistics can provide insights into the central tendency, spread, skewness, and shape of the distributions of the variables in the DataFrame after encoding categorical variables.

```
# Calculate additional statistics after encoding
print("\nAdditional Statistics:")
print("Median:")
print(cc_data.median())
print("\nVariance:")
print(cc_data.var())
print("\nSkewness:")
print(cc_data.skew())
print("\nKurtosis:")
print(cc_data.kurtosis())
```

```
Additional Statistics:
Median:
CUST_ID                               4317.500000
BALANCE                                916.855459
BALANCE_FREQUENCY                        1.000000
PURCHASES                              375.405000
ONEOFF_PURCHASES                        44.995000
INSTALLMENTS_PURCHASES                  94.785000
CASH_ADVANCE                             0.000000
PURCHASES_FREQUENCY                      0.500000
ONEOFF_PURCHASES_FREQUENCY               0.083333
PURCHASES_INSTALLMENTS_FREQUENCY         0.166667
CASH_ADVANCE_FREQUENCY                   0.000000
CASH_ADVANCE_TRX                         0.000000
PURCHASES_TRX                            7.000000
CREDIT_LIMIT                          3000.000000
PAYMENTS                               896.675701
MINIMUM_PAYMENTS                       312.452292
PRC_FULL_PAYMENT                         0.000000
TENURE                                  12.000000
dtype: float64

Variance:
CUST_ID                               6.215761e+06
BALANCE                               4.391419e+06
BALANCE_FREQUENCY                     4.313799e-02
PURCHASES                             4.696357e+06
```

14. The overall purpose of this code is to generate a series of boxplots, one for each column in the dataset (excluding the last one), to visually inspect the distribution of data and identify potential outliers. We observe the outliers, but we didn't define them precisely.





15. We use IQR method to define outliers with accurate numbers instead of depending on our observation. Outliers are identified using IQR by checking which data points fall below the lower bound or above the upper bound and then remove these points.

This code displays the difference between the outliers before and after removal. These iterations provide a summary of outlier counts for each column before and after their removal, allowing for comparison and assessment of the impact of outlier removal on the dataset.

```python
[ ] # Display outliers before and after removal
    for iteration, outliers_dict in outliers_before.items():
        print(f"Iteration {iteration + 1}: Outliers before removal")
        for column, outliers in outliers_dict.items():
            print(f"Column '{column}' has {len(outliers)} outliers before removal.")
        print("---------------------------------------------")

    for iteration, outliers_dict in outliers_after.items():
        print(f"Iteration {iteration + 1}: Outliers after removal")
        for column, outliers in outliers_dict.items():
            print(f"Column '{column}' has {len(outliers)} outliers after removal.")
        print("---------------------------------------------")
```

Here is the output :

```
Iteration 1: Outliers before removal
Column 'CUST_ID' has 0 outliers before removal.
Column 'BALANCE' has 68 outliers before removal.
Column 'BALANCE_FREQUENCY' has 744 outliers before removal.
Column 'PURCHASES' has 232 outliers before removal.
Column 'ONEOFF_PURCHASES' has 371 outliers before removal.
Column 'INSTALLMENTS_PURCHASES' has 236 outliers before removal.
Column 'CASH_ADVANCE' has 302 outliers before removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers before removal.
Column 'CASH_ADVANCE_FREQUENCY' has 1 outliers before removal.
Column 'CASH_ADVANCE_TRX' has 239 outliers before removal.
Column 'PURCHASES_TRX' has 196 outliers before removal.
Column 'CREDIT_LIMIT' has 3 outliers before removal.
Column 'PAYMENTS' has 266 outliers before removal.
Column 'MINIMUM_PAYMENTS' has 294 outliers before removal.
Column 'PRC_FULL_PAYMENT' has 642 outliers before removal.
Column 'TENURE' has 1290 outliers before removal.
---------------------------------------------
Iteration 2: Outliers before removal
Column 'CUST_ID' has 0 outliers before removal.
Column 'BALANCE' has 11 outliers before removal.
Column 'BALANCE_FREQUENCY' has 834 outliers before removal.
Column 'PURCHASES' has 0 outliers before removal.
Column 'ONEOFF_PURCHASES' has 62 outliers before removal.
Column 'INSTALLMENTS_PURCHASES' has 35 outliers before removal.
Column 'CASH_ADVANCE' has 56 outliers before removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers before removal.
Column 'CASH_ADVANCE_FREQUENCY' has 0 outliers before removal.
Column 'CASH_ADVANCE_TRX' has 0 outliers before removal.
Column 'PURCHASES_TRX' has 7 outliers before removal.
Column 'CREDIT_LIMIT' has 0 outliers before removal.
Column 'PAYMENTS' has 21 outliers before removal.
Column 'MINIMUM_PAYMENTS' has 0 outliers before removal.
Column 'PRC_FULL_PAYMENT' has 250 outliers before removal.
Column 'TENURE' has 0 outliers before removal.
---------------------------------------------
Iteration 3: Outliers before removal
Column 'CUST_ID' has 0 outliers before removal.
Column 'BALANCE' has 0 outliers before removal.
Column 'BALANCE_FREQUENCY' has 0 outliers before removal.
Column 'PURCHASES' has 1 outliers before removal.
Column 'ONEOFF_PURCHASES' has 36 outliers before removal.
Column 'INSTALLMENTS_PURCHASES' has 35 outliers before removal.
Column 'CASH_ADVANCE' has 0 outliers before removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers before removal.
Column 'CASH_ADVANCE_FREQUENCY' has 0 outliers before removal.
Column 'CASH_ADVANCE_TRX' has 0 outliers before removal.
Column 'PURCHASES_TRX' has 7 outliers before removal.
Column 'CREDIT_LIMIT' has 0 outliers before removal.
Column 'PAYMENTS' has 8 outliers before removal.
```

```
--------------------------------------------
Iteration 4: Outliers before removal
Column 'CUST_ID' has 0 outliers before removal.
Column 'BALANCE' has 0 outliers before removal.
Column 'BALANCE_FREQUENCY' has 0 outliers before removal.
Column 'PURCHASES' has 0 outliers before removal.
Column 'ONEOFF_PURCHASES' has 19 outliers before removal.
Column 'INSTALLMENTS_PURCHASES' has 12 outliers before removal.
Column 'CASH_ADVANCE' has 0 outliers before removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers before removal.
Column 'CASH_ADVANCE_FREQUENCY' has 0 outliers before removal.
Column 'CASH_ADVANCE_TRX' has 0 outliers before removal.
Column 'PURCHASES_TRX' has 0 outliers before removal.
Column 'CREDIT_LIMIT' has 0 outliers before removal.
Column 'PAYMENTS' has 5 outliers before removal.
Column 'MINIMUM_PAYMENTS' has 0 outliers before removal.
Column 'PRC_FULL_PAYMENT' has 0 outliers before removal.
Column 'TENURE' has 0 outliers before removal.
--------------------------------------------
Iteration 5: Outliers before removal
Column 'CUST_ID' has 0 outliers before removal.
Column 'BALANCE' has 0 outliers before removal.
Column 'BALANCE_FREQUENCY' has 0 outliers before removal.
Column 'PURCHASES' has 0 outliers before removal.
Column 'ONEOFF_PURCHASES' has 11 outliers before removal.
Column 'INSTALLMENTS_PURCHASES' has 6 outliers before removal.
Column 'CASH_ADVANCE' has 0 outliers before removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers before removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers before removal.
Column 'CASH_ADVANCE_FREQUENCY' has 0 outliers before removal.
Column 'CASH_ADVANCE_TRX' has 0 outliers before removal.
Column 'PURCHASES_TRX' has 0 outliers before removal.
Column 'CREDIT_LIMIT' has 0 outliers before removal.
Column 'PAYMENTS' has 2 outliers before removal.
Column 'MINIMUM_PAYMENTS' has 0 outliers before removal.
Column 'PRC_FULL_PAYMENT' has 0 outliers before removal.
Column 'TENURE' has 0 outliers before removal.
--------------------------------------------
Iteration 1: Outliers after removal
Column 'CUST_ID' has 0 outliers after removal.
Column 'BALANCE' has 11 outliers after removal.
Column 'BALANCE_FREQUENCY' has 834 outliers after removal.
Column 'PURCHASES' has 0 outliers after removal.
Column 'ONEOFF_PURCHASES' has 62 outliers after removal.
Column 'INSTALLMENTS_PURCHASES' has 35 outliers after removal.
Column 'CASH_ADVANCE' has 56 outliers after removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_TRX' has 0 outliers after removal.
Column 'PURCHASES_TRX' has 7 outliers after removal.
Column 'CREDIT_LIMIT' has 0 outliers after removal.
Column 'PAYMENTS' has 21 outliers after removal.
Column 'MINIMUM_PAYMENTS' has 0 outliers after removal.
Column 'PRC_FULL_PAYMENT' has 250 outliers after removal.
Column 'TENURE' has 0 outliers after removal.
--------------------------------------------
Iteration 2: Outliers after removal
Column 'CUST_ID' has 0 outliers after removal.
Column 'BALANCE' has 0 outliers after removal.
Column 'BALANCE_FREQUENCY' has 0 outliers after removal.
Column 'PURCHASES' has 1 outliers after removal.
Column 'ONEOFF_PURCHASES' has 36 outliers after removal.
Column 'INSTALLMENTS_PURCHASES' has 35 outliers after removal.
Column 'CASH_ADVANCE' has 0 outliers after removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_TRX' has 0 outliers after removal.
Column 'PURCHASES_TRX' has 7 outliers after removal.
Column 'CREDIT_LIMIT' has 0 outliers after removal.
Column 'PAYMENTS' has 8 outliers after removal.
Column 'MINIMUM_PAYMENTS' has 0 outliers after removal.
Column 'PRC_FULL_PAYMENT' has 0 outliers after removal.
Column 'TENURE' has 0 outliers after removal.
--------------------------------------------
Iteration 3: Outliers after removal
Column 'CUST_ID' has 0 outliers after removal.
Column 'BALANCE' has 0 outliers after removal.
Column 'BALANCE_FREQUENCY' has 0 outliers after removal.
Column 'PURCHASES' has 0 outliers after removal.
Column 'ONEOFF_PURCHASES' has 19 outliers after removal.
Column 'INSTALLMENTS_PURCHASES' has 12 outliers after removal.
Column 'CASH_ADVANCE' has 0 outliers after removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_TRX' has 0 outliers after removal.
Column 'PURCHASES_TRX' has 0 outliers after removal.
Column 'CREDIT_LIMIT' has 0 outliers after removal.
Column 'PAYMENTS' has 5 outliers after removal.
Column 'MINIMUM_PAYMENTS' has 0 outliers after removal.
Column 'PRC_FULL_PAYMENT' has 0 outliers after removal.
Column 'TENURE' has 0 outliers after removal.
--------------------------------------------
Iteration 4: Outliers after removal
Column 'CUST_ID' has 0 outliers after removal.
Column 'BALANCE' has 0 outliers after removal.
Column 'BALANCE_FREQUENCY' has 0 outliers after removal.
Column 'PURCHASES' has 0 outliers after removal.
Column 'ONEOFF_PURCHASES' has 11 outliers after removal.
Column 'INSTALLMENTS_PURCHASES' has 6 outliers after removal.
Column 'CASH_ADVANCE' has 0 outliers after removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_TRX' has 0 outliers after removal.
Column 'PURCHASES_TRX' has 0 outliers after removal.
Column 'CREDIT_LIMIT' has 0 outliers after removal.
Column 'PAYMENTS' has 2 outliers after removal.
Column 'MINIMUM_PAYMENTS' has 0 outliers after removal.
Column 'PRC_FULL_PAYMENT' has 0 outliers after removal.
Column 'TENURE' has 0 outliers after removal.
--------------------------------------------
Iteration 5: Outliers after removal
Column 'CUST_ID' has 0 outliers after removal.
Column 'BALANCE' has 0 outliers after removal.
Column 'BALANCE_FREQUENCY' has 0 outliers after removal.
Column 'PURCHASES' has 0 outliers after removal.
Column 'ONEOFF_PURCHASES' has 2 outliers after removal.
Column 'INSTALLMENTS_PURCHASES' has 2 outliers after removal.
Column 'CASH_ADVANCE' has 0 outliers after removal.
Column 'PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'ONEOFF_PURCHASES_FREQUENCY' has 0 outliers after removal.
Column 'PURCHASES_INSTALLMENTS_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_FREQUENCY' has 0 outliers after removal.
Column 'CASH_ADVANCE_TRX' has 0 outliers after removal.
Column 'PURCHASES_TRX' has 0 outliers after removal.
Column 'CREDIT_LIMIT' has 0 outliers after removal.
Column 'PAYMENTS' has 0 outliers after removal.
Column 'MINIMUM_PAYMENTS' has 0 outliers after removal.
Column 'PRC_FULL_PAYMENT' has 0 outliers after removal.
Column 'TENURE' has 0 outliers after removal.
--------------------------------------------
```

16. This code calculates lower and upper bounds for each column in the DataFrame cc_data based on the interquartile range (IQR) method for outlier detection. The we print the lower and the upper bounds for 'Payments'.

```
[ ] cleaned_data.shape

    (3057, 18)
```

```
[ ] # Accessing outliers for columns and storing lower and upper bounds
    lower_upper_bounds = {}  # Dictionary to store lower and upper bounds for columns
    for column in cc_data:
        lower_upper_bounds[column] = (cc_data[column].quantile(0.25) - 1.5 * (cc_data[column].quantile(0.75) - cc_data[column].quantile(0.25)),
                                      cc_data[column].quantile(0.75) + 1.5 * (cc_data[column].quantile(0.75) - cc_data[column].quantile(0.25)))
```

```
[ ] # Printing lower and upper bounds for 'PAYMENTS'
    print("Lower bound for PAYMENTS:", lower_upper_bounds['PAYMENTS'][0])
    print("Upper bound for PAYMENTS:", lower_upper_bounds['PAYMENTS'][1])

    Lower bound for PAYMENTS: -1880.315042625
    Upper bound for PAYMENTS: 4250.016370375
```

displaying the present outliers in 'payments' after removal and before removal, there were 5 iterations made.

```
# Printing outliers found for 'PAYMENTS' before removal
print("\nOutliers found for 'PAYMENTS' before removal:")
for iteration, outliers_dict in outliers_before.items():
    print(f"Iteration {iteration + 1}: Outliers before removal")
    for column, outliers in outliers_dict.items():
        if column == 'PAYMENTS':  # Check if the column is 'PAYMENTS'
            print(outliers)  # Print outliers for 'PAYMENTS'
    print("--------------------------------------------")
```

```
Outliers found for 'PAYMENTS' before removal:
Iteration 1: Outliers before removal
      CUST_ID      BALANCE  BALANCE_FREQUENCY  PURCHASES  ONEOFF_PURCHASES  \
23         22  3800.151377           0.818182    4248.35           3454.56
34         33  3517.101616           0.727273     547.28              0.00
39         38  1411.602230           0.454545     963.24            963.24
50         47  4931.331857           1.000000     901.42            646.07
71         64  2990.422186           0.909091    4523.27           1664.09
...       ...          ...                ...        ...               ...
8052     7788  3342.878215           0.818182    1368.70           1300.00
8237     7963  2144.040539           1.000000       0.00              0.00
8315     8039  2648.244646           1.000000       0.00              0.00
8737     8436  2533.618119           0.909091    5633.83           2985.92
8857     8548  2330.222764           1.000000    1320.00              0.00

      INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
23                    793.79   7974.415626             1.000000
34                    547.28      0.000000             1.000000
39                      0.00   6173.682877             0.083333
50                    255.35   8530.648614             0.625000
71                   2859.18  27296.485760             0.666667
...                      ...           ...                  ...
8052                   68.70   1594.327632             0.083333
8237                    0.00  14127.466640             0.000000
8315                    0.00  10458.978150             0.000000
8737                 2647.91   2451.807788             0.916667
8857                 1320.00  14926.790590             0.428571

      ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
23                      0.083333                          0.916667
34                      0.000000                          1.000000
```

```
8446     8163  2373.686499               1.0    1310.47             0.00

      INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
1294                    0.00    342.874785             0.583333
3690                  299.20    947.496827             0.833333
5356                    0.00   1438.951061             0.166667
7458                 1520.60   1279.568823             0.916667
8446                 1310.47   3667.381175             1.000000

      ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
1294                    0.583333                              0.00
3690                    0.166667                              0.75
5356                    0.083333                              0.00
7458                    0.166667                              0.75
8446                    0.000000                              1.00

      CASH_ADVANCE_FREQUENCY  CASH_ADVANCE_TRX  PURCHASES_TRX  CREDIT_LIMIT  \
1294                0.083333                 1              7        7500.0
3690                0.083333                 1             15        8000.0
5356                0.583333                15              1        4100.0
7458                0.250000                 6             15        9000.0
8446                0.250000                 5             31        3000.0

         PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT  TENURE
1294  4613.453210       1148.353805               0.0      12
3690  4618.241000        199.905433               0.0      12
5356  4591.237633        723.343619               0.0      12
7458  4619.793197        853.911515               0.0      12
8446  4613.536289        936.940012               0.0      12
---------------------------------------------
Iteration 5: Outliers before removal
      CUST_ID      BALANCE  BALANCE_FREQUENCY  PURCHASES  ONEOFF_PURCHASES  \
582       543  7088.358802                1.0       0.00              0.00
4146     3998  2015.554911                1.0    1446.48            493.88

      INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
582                      0.0   3118.050456             0.000000
4146                   952.6      0.000000             0.666667

      ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
582                         0.00                          0.000000
4146                        0.25                          0.666667

      CASH_ADVANCE_FREQUENCY  CASH_ADVANCE_TRX  PURCHASES_TRX  CREDIT_LIMIT  \
582                 0.083333                 2              0        7500.0
4146                0.000000                 0             27       13000.0

         PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT  TENURE
582   4539.526102       2183.319473               0.0      12
4146  4537.716739        563.638570               0.0      12
---------------------------------------------
```

17. After running this code, scaled-data will contain the scaled features from cleaned-data, with each feature scaled to the range specified by the MinMaxScaler. This scaling is often performed to ensure that all features have the same scale, which can be important for certain machine learning algorithms.

Scaling the numerical features

```
# initialize the MinMaxScaler
MS = MinMaxScaler()
# fit the scaler on the features and transform them
scaled_features = MS.fit_transform(cleaned_data)
scaled_features

# create a new DataFrame with the scaled features
scaled_data = pd.DataFrame(scaled_features, columns=cleaned_data.columns)
```

```
# display the updated DataFrame with scaled features
scaled_data.head()
```

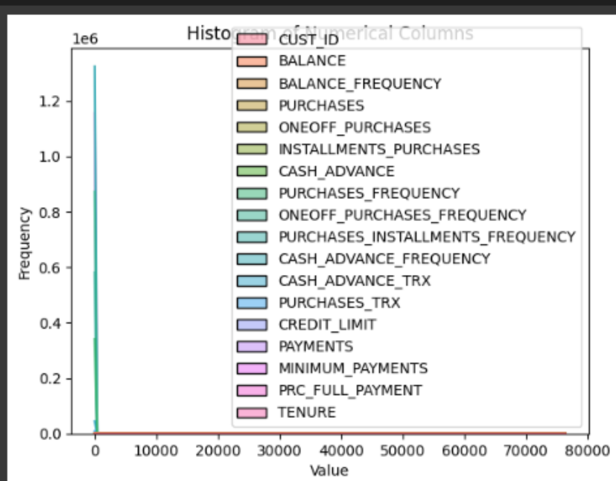
	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT
0	0.000000	0.283609	0.0	0.246452	0.406797	0.000000	0.0	1.000000	1.000000	0.000000	0.0	0.0	0.184615	0.411765
1	0.000118	0.092743	0.0	0.005100	0.008418	0.000000	0.0	0.083333	0.083333	0.000000	0.0	0.0	0.015385	0.058824
2	0.000237	0.205630	0.0	0.424990	0.000000	0.920303	0.0	0.666667	0.000000	0.583333	0.0	0.0	0.123077	0.092437
3	0.000473	0.207203	0.0	0.139041	0.000000	0.301089	0.0	1.000000	0.000000	1.000000	0.0	0.0	0.184615	0.120448
4	0.000592	0.115183	0.0	0.274605	0.348037	0.138051	0.0	0.333333	0.083333	0.250000	0.0	0.0	0.076923	0.383754

More visualization after encoding and scaling

18. With this modification, the code should generate a histogram of numerical columns from the DataFrame cc_data, with kernel density estimation (KDE) overlaid, along with appropriate labels and a title.

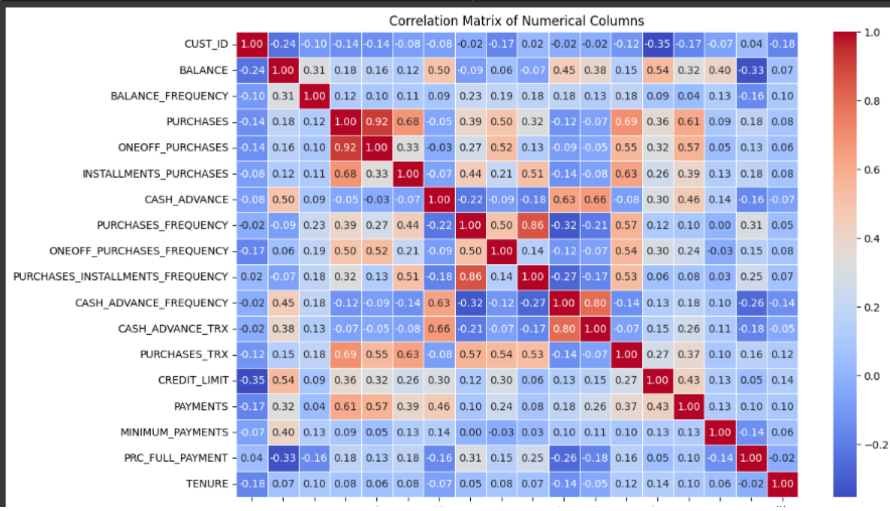
More visualization after encoding and scaling

```
import seaborn as sns

# Histogram of numerical columns
sns.histplot(cc_data, kde=True)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Numerical Columns')
plt.show()
```



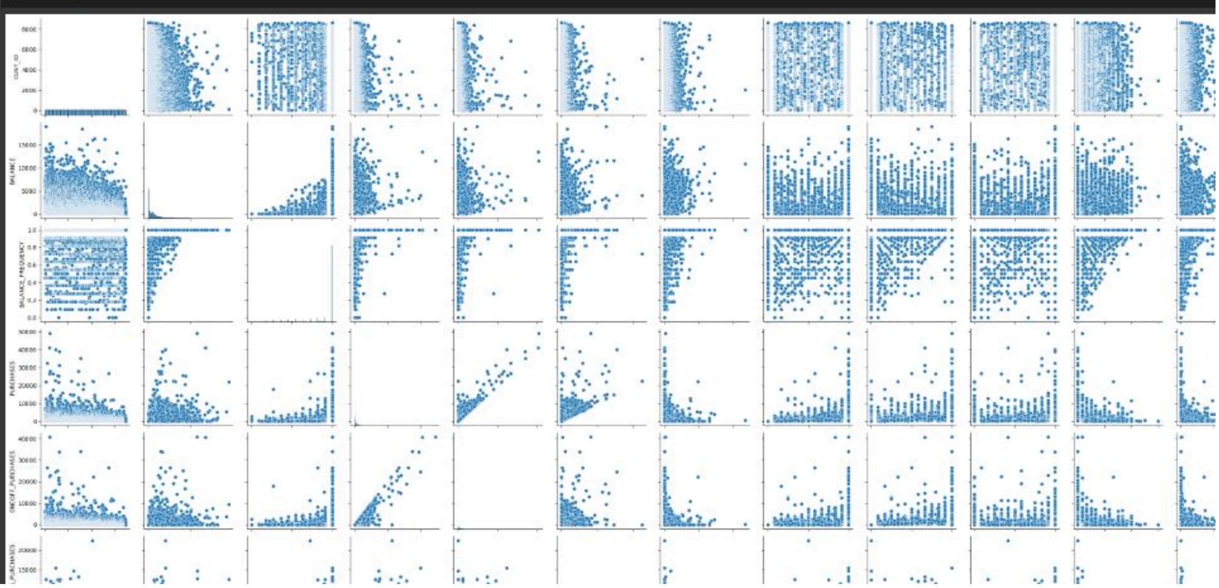
19. This code calculates the correlation matrix for numerical columns in the dataframe cc_data, then visualize it as a heatmap to understand the relationship between different numerical features

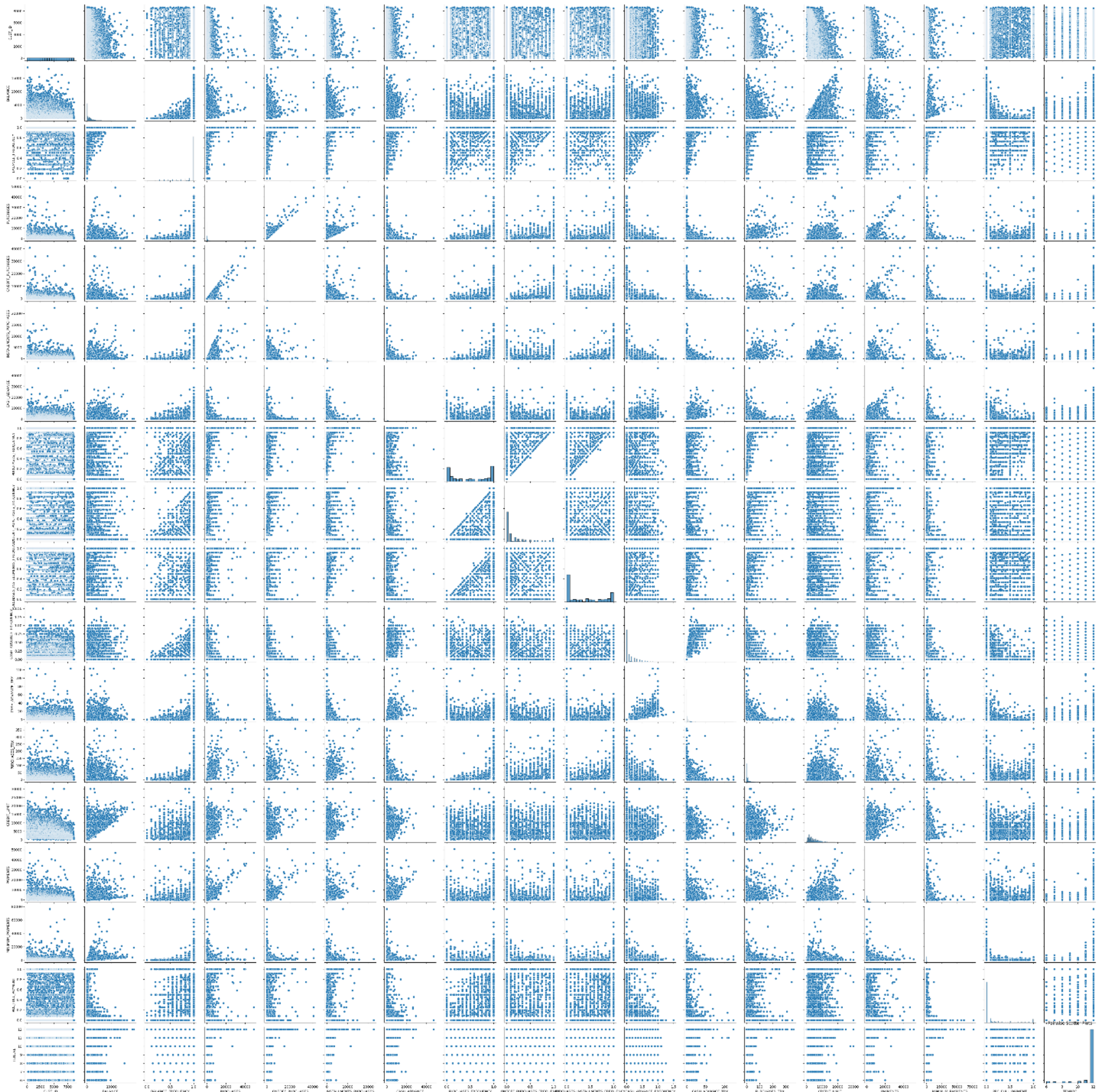
```
# Calculate correlation matrix for whole data after encoding and scaling
corr_matrix = cc_data.corr()

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix of Numerical Columns')
plt.show()
```



20. THIS IS TO DISPLAY THE PAIR PLOTS:

```
# Pairwise scatter plots
sns.pairplot(cc_data)
plt.title('Pairwise Scatter Plots')
plt.show()
```





K-Medoids

Now moving to the k-medoids, this loop allows us to assess the quality of clustering for different numbers of clusters, helping us to determine the optimal number of clusters for our dataset based on the silhouette score. The output allows us to assess the clustering quality for different numbers of clusters. Typically, we would choose the number of clusters that maximizes the silhouette score, indicating the best separation of data points into distinct clusters. In this case, we would look for the highest silhouette score among the values displayed.

```
[ ] pip install scikit-learn-extra

    Collecting scikit-learn-extra
      Downloading scikit_learn_extra-0.3.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.0 MB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 10.0 MB/s eta 0:00:00
    Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.25.2)
    Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.11.4)
    Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.2.2)
    Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.4.2)
    Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (3.5.0)
    Installing collected packages: scikit-learn-extra
    Successfully installed scikit-learn-extra-0.3.0

[ ] from sklearn.metrics import silhouette_score
    from sklearn_extra.cluster import KMedoids
    # Compute silhouette score for different values of k to get the optimal number of k
    for n_clusters in range(2, 9):  # Trying k from 2 to 9
        kmedoids = KMedoids(n_clusters=n_clusters, random_state=0)
        labels = kmedoids.fit_predict(scaled_data)
        silhouette_avg = silhouette_score(scaled_data, labels)
        print(f"For n_clusters = {n_clusters}, the average silhouette_score is : {silhouette_avg}")

    For n_clusters = 2, the average silhouette_score is : 0.32297533940101164
    For n_clusters = 3, the average silhouette_score is : 0.17173806475782621
    For n_clusters = 4, the average silhouette_score is : 0.2046578100490175
    For n_clusters = 5, the average silhouette_score is : 0.15597909497594725
    For n_clusters = 6, the average silhouette_score is : 0.13979017809426852
    For n_clusters = 7, the average silhouette_score is : 0.13600439280738721
    For n_clusters = 8, the average silhouette_score is : 0.13003006663796307
```

The following code performs KMedoids clustering on the provided data, prints the coordinates of the cluster centers, and then prints each data point along with its assigned cluster label. It helps visualize which data points belong to which cluster.Here's a breakdown of what each part does:

- **data = np.array(scaled_data):** Converts the scaled data into a NumPy array.
- **k = 2:** Specifies the number of clusters to form.
- **kmedoids = KMedoids(n_clusters=k).fit(data):** Creates a KMedoids clustering model with k clusters and fits it to the data. This assigns each data point to one of the k clusters and identifies the cluster centers (medoids).
- *labels = kmedoids.labels_:* Retrieves the cluster labels assigned to each data point by the KMedoids algorithm.
- **clusters = kmedoids.cluster_centers_:** Retrieves the cluster centers (medoids).
- The nested loop iterates over each cluster (i) and each data point (j). For each data point, if it belongs to the current cluster (labels[j] == i), it prints the data point, indicating which cluster it belongs to.

```python
data=np.array(scaled_data)
k=2
kmedoids = KMedoids(n_clusters=k).fit(data)

labels= kmedoids.labels_
clusters=kmedoids.cluster_centers_

print('clusters',clusters)

for i in range(k):
  for j in range(len(data)):
    if (labels[j]==i):
      y=data[j]
      print('cluster ',i,':',y)
```

```
Streaming output truncated to the last 5000 lines.
cluster  1 : [0.03028869 0.27514531 0.         0.         0.         0.
 0.11089862 0.         0.         0.         0.09090869 0.1
 0.         0.15966387 0.19325319 0.2000655  0.         0.        ]
cluster  1 : [0.03064363 0.62208542 0.         0.         0.         0.
 0.10025496 0.         0.         0.         0.18181848 0.15
 0.         0.32773109 0.30400319 0.66549857 0.         0.        ]
cluster  1 : [0.03088027 0.65861937 0.         0.14806515 0.21014611 0.0449356
 0.49203972 0.416667   0.25       0.166667   0.18181848 0.3
 0.15384615 0.32773109 0.3384121  0.68989168 0.         0.        ]
cluster  1 : [0.03099858 0.40997494 0.         0.56706617 0.93600543 0.
 0.         0.166667   0.166667   0.         0.         0.
 0.03076923 0.27170868 0.18972386 0.25018198 0.         0.        ]
cluster  1 : [0.0311169  0.45395821 0.         0.         0.         0.
 0.06819983 0.         0.         0.         0.63636304 0.45
 0.         0.21568627 0.23517781 0.40495245 0.         0.        ]
cluster  1 : [0.03147184 0.17278529 0.         0.0784075  0.12942024 0.
 0.54598987 0.166667   0.166667   0.         0.27272717 0.15
 0.03076923 0.32773109 0.50657714 0.12712229 0.         0.        ]
cluster  1 : [0.03159016 0.63087431 0.         0.         0.         0.
 0.64497198 0.         0.         0.         0.09090869 0.1
 0.         0.66386555 0.39940555 0.44528799 0.         0.        ]
cluster  1 : [0.0319451  0.41179413 0.         0.21507395 0.35500334 0.
 0.32942228 0.25       0.25       0.         0.18181848 0.4
 0.18461538 0.21568627 0.25352449 0.4622829  0.         0.        ]
```


```
Streaming output truncated to the last 5000 lines.
cluster  1 : [0.03028869 0.27514531 0.         0.         0.         0.
 0.11089862 0.         0.         0.         0.09090869 0.1
 0.         0.15966387 0.19325319 0.2000655  0.         0.        ]
cluster  1 : [0.03064363 0.62208542 0.         0.         0.         0.
 0.10025496 0.         0.         0.         0.18181848 0.15
 0.         0.32773109 0.30400319 0.66549857 0.         0.        ]
cluster  1 : [0.03088027 0.65861937 0.         0.14806515 0.21014611 0.0449356
 0.49203972 0.416667   0.25       0.166667   0.18181848 0.3
 0.15384615 0.32773109 0.3384121  0.68989168 0.         0.        ]
cluster  1 : [0.03099858 0.40997494 0.         0.56706617 0.93600543 0.
 0.         0.166667   0.166667   0.         0.         0.
 0.03076923 0.27170868 0.18972386 0.25018198 0.         0.        ]
cluster  1 : [0.0311169  0.45395821 0.         0.         0.         0.
 0.06819983 0.         0.         0.         0.63636304 0.45
 0.         0.21568627 0.23517781 0.40495245 0.         0.        ]
cluster  1 : [0.03147184 0.17278529 0.         0.0784075  0.12942024 0.
 0.54598987 0.166667   0.166667   0.         0.27272717 0.15
 0.03076923 0.32773109 0.50657714 0.12712229 0.         0.        ]
cluster  1 : [0.03159016 0.63087431 0.         0.         0.         0.
 0.64497198 0.         0.         0.         0.09090869 0.1
 0.         0.66386555 0.39940555 0.44528799 0.         0.        ]
cluster  1 : [0.0319451  0.41179413 0.         0.21507395 0.35500334 0.
 0.32942228 0.25       0.25       0.         0.18181848 0.4
 0.18461538 0.21568627 0.25352449 0.4622829  0.         0.        ]
```


```
cluster  1 : [0.03324657 0.10518277 0.         0.         0.         0.
 0.07878458 0.         0.         0.         0.36363587 0.25
 0.         0.04761905 0.15904457 0.11104303 0.         0.        ]
cluster  1 : [0.03336488 0.21279198 0.         0.         0.         0.
 0.66365299 0.         0.         0.         0.18181848 0.15
 0.         0.29971989 0.25637924 0.2059449  0.         0.        ]
cluster  1 : [0.03395646 0.53817166 0.         0.0047654  0.         0.01031931
 0.42490801 0.083333   0.         0.083333   0.45454565 0.55
 0.01538462 0.31092437 0.28739188 0.38595964 0.         0.        ]
cluster  1 : [0.03419309 0.7642624  0.         0.         0.         0.
 0.4776768  0.         0.         0.         0.36363587 0.35
 0.         0.41176471 0.33333846 0.82064571 0.         0.        ]
cluster  1 : [0.03431141 0.25378937 0.         0.14190998 0.23423812 0.
 0.         0.166667   0.166667   0.         0.         0.
 0.06153846 0.43977591 0.23186683 0.19028715 0.         0.        ]
cluster  1 : [0.03442972 0.21220965 0.         0.         0.         0.
 0.63824948 0.         0.         0.         0.72727283 0.45
 0.         0.24369748 0.1040787  0.13164921 0.         0.        ]
cluster  1 : [3.49029815e-02 9.64882607e-02 0.00000000e+00 1.59377789e-05
 2.63070666e-05 0.00000000e+00 3.32487287e-01 8.33330000e-02
 8.33330000e-02 0.00000000e+00 3.63635868e-01 3.50000000e-01
 1.53846154e-02 4.76190476e-02 2.41175259e-01 1.20239456e-01
 0.00000000e+00 0.00000000e+00]
```

Hierarchical Clustering

In hierarchical clustering, the "ward" method is a linkage criterion used to measure the distance between clusters during the clustering process. Specifically, it minimizes the variance when forming clusters. Here's what it does:

- **Initialization:** Initially, each data point is considered as a separate cluster.
- **Distance Calculation:** The distance between each pair of clusters is calculated. This distance can be measured in various ways, such as Euclidean distance, Manhattan distance, etc. In your case, the Euclidean distance is used (metric='euclidean').
- **Cluster Merge:** At each step, the pair of clusters with the smallest distance according to the chosen linkage criterion is merged into a single cluster. The "ward" method merges clusters to minimize the variance within the newly formed cluster.
- **Update Distance Matrix:** After merging, the distance matrix is updated to reflect the distances between the newly formed cluster and the remaining clusters.
- **Repeat:** Steps 2-4 are repeated until all data points belong to a single cluster or until a specified number of clusters is reached.

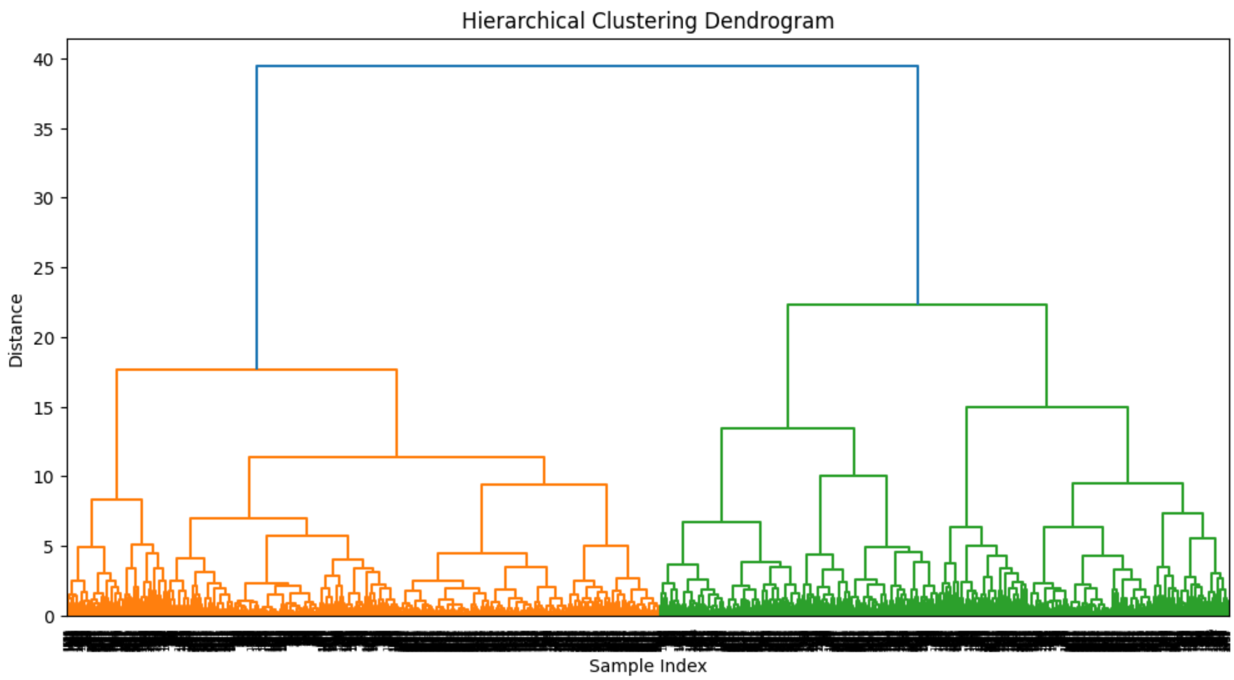
The output of hierarchical clustering with the "ward" method is a dendrogram, which visually represents the clustering process and shows how clusters are merged at each step.

We will import the following functions and performs hierarchical clustering on the scaled data using the Ward method Euclidean distance metric, and then visualizes the resulting dendrogram.

```
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

np.set_printoptions(threshold=np.inf)
```

```
Z = linkage(scaled_data, method='ward',metric='euclidean')

plt.figure(figsize=(12, 6))
dendrogram(Z)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

Here is the output :



Now after running, the variable labels will contain the cluster assignments for each data point in the scaled dataset, based on the agglomerative hierarchical clustering algorithm with 2 clusters, Euclidean distance metric, and Ward linkage criterion.

```
[ ]  cluster = AgglomerativeClustering(n_clusters=2,metric='euclidean',linkage='ward')
     labels=cluster.fit_predict(scaled_data)
```

```
[ ]  print(labels)
```

```
[0 1 0 0 0 0 1 0 0 1 0 1 0 0 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 1 1 1 1 0
 1 0 1 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 1 0 0 0 1
 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 1 1 0 1 1 1 0 1 0 1 1 0 1
 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 1
 1 1 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 1 1 0 1 1 0 0 0 1 0 1 0 1 0
 0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 0 1 0 0 1 1 0
 1 0 0 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 1 0 1 0 0 0 0 1 1 1 0 1 1 1 1 0 0 1
 1 1 0 1 1 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0 0 0 1 1 1 1 0 0 1 1 1 1 1 0 0 0 1
 0 0 1 0 0 0 1 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 0
 0 1 1 1 0 0 1 1 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 1 1 1 0 0 1 1 0 1 0 1 0 1
 0 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1
 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 1 0 1 1 0
 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 1 0 1 1 0 1 0
 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 1
 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0 0 1 0 1
 1 1 0 0 0 1 0 1 1 0 1 0 1 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 1 1 0 1 1 0 1
 0 0 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1
 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0
 0 1 1 0 0 1 0 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 1 1
 0 0 1 1 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1 1 1 0
 0 0 1 0 0 0 1 0 1 1 0 0 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0
 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0
 1 1 0 1 1 0 1 0 0 1 1 0 0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 1 1
 1 0 0 0 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1
 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 0
 1 0 1 0 0 0 1 1 0 1 1 1 1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0 1 1 0 1
 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1
 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1
```

Evaluation

The **silhouette coefficient** is a measure of cluster cohesion and separation. Cohesion measures how closely related the data points in a cluster are, while separation measures the degree of dissimilarity between clusters. A good clustering algorithm should produce clusters with high cohesion and separation.

The silhouette score measures how well each data point fits its assigned cluster compared to other clusters. It ranges from -1 to 1, where a score of 1 indicates that the point is well-matched to its cluster and poorly matched to other clusters.

The silhouette score is calculated as follows:

For each data point i:

- a(i): the average distance from i to all other points in the same cluster
- b(i): the average distance from i to all points in the nearest cluster
- s(i) = (b(i) — a(i)) / max(a(i), b(i))

The silhouette score is the average of s(i) over all data points. A score closer to 1 indicates better clustering.

The Davies-Bouldin Index is a validation metric that is used to evaluate clustering models. It is calculated as the average similarity measure of each cluster with the cluster most like it. In this context, similarity is defined as the ratio between inter-cluster and intra-cluster distances. As such, this index ranks well-separated clusters with less dispersion as having a better score.

The Davies-Bouldin Index is calculated as follows:

- $\Delta(xk\ )$ is the intercluster distance within the cluster xk .
- $\delta(xi\ ,\ xj)$ is the inter cluster distance between the clusters xi and xj .

The Calinski and Harabasz Index (also known as Variance ratio criterion). This index computes the ratio of the sum of between-cluster dispersion and within-cluster dispersion for all clusters. Higher values indicate better clustering.

- **Between-cluster dispersion:** This refers to the variation between different clusters. In other words, it measures how much the cluster centers differ from each other. If the between-cluster dispersion is high, it means that the clusters are well-separated from each other.
- **Within-cluster dispersion:** This measures the variation within each cluster. It assesses how tightly the data points are clustered around their respective cluster centers. Lower

within-cluster dispersion indicates that the points within each cluster are closer to each other.

Calinski-Harabasz Index computation: The index is calculated as the ratio of between-cluster dispersion to within-cluster dispersion. Mathematically, it's represented as:

CH = $(B/W) \times ((N-k)/(k-1))$ Where:

- B is the between-cluster dispersion.
- W is the within-cluster dispersion.
- N is the total number of data points.
- k is the number of clusters.

Comparing clustering techniques:

1. **Silhouette Score:**
   - K-medoids: 0.322975339401011644
   - Hierarchical Clustering: 0.27627621841002925
   - Higher silhouette score indicates better-defined clusters where data points are closer to other points in their own cluster than to points in other clusters. K-medoids outperforms Hierarchical Clustering in this aspect.

2. **Davies-Bouldin Index:**
   - K-medoids: 1.3243641946424665
   - Hierarchical Clustering: 1.4285021170311918
   - Lower Davies-Bouldin index suggests better separation between clusters. Again, K-medoids has a lower index, indicating better defined clusters.

3. **Calinski-Harabasz Index:**
   - K-medoids: 1568.9592250868711
   - Hierarchical Clustering: 1320.4645654005237
   - A higher Calinski-Harabasz index signifies dense and well-separated clusters. K-medoids has a higher index, suggesting better clustering quality.

Due to understanding the internal metrics, K-medoids clustering outperforms Hierarchical Clustering in all three-evaluation metrics provided. Based on these metrics, K-medoids is the better clustering algorithm for this dataset

This code evaluates the clustering quality achieved by the K-medoids algorithm using three different metrics: silhouette score, Davies-Bouldin index, and Calinski-Harabasz index. The output will display the computed evaluation metrics for the K-medoids clustering algorithm, providing insights into the quality of the clustering results. Each metric gives a different perspective on the clustering performance, helping to assess the clustering quality from multiple angles. The output for the second cell will display the computed evaluation metrics for hierarchical clustering, allowing for comparison with the K-medoids clustering results. Each metric gives insights into the quality of the clustering results, helping to assess the clustering performance of hierarchical clustering.

```
calinski_harabasz = calinski_harabasz_score(scaled_data, labels)

print("K-medoids:")
print("Silhouette Score:", silhouette)
print("Davies-Bouldin Index:", davies_bouldin)
print("Calinski-Harabasz Index:", calinski_harabasz)
```

```
K-medoids:
Silhouette Score: 0.32297533940101164
Davies-Bouldin Index: 1.3243641946424665
Calinski-Harabasz Index: 1568.9592250868711
```

```
# Evaluate Hierarchical Clustering
hierarchical_labels = cluster.labels_

silhouette_hierarchical = silhouette_score(scaled_data, hierarchical_labels)
davies_bouldin_hierarchical = davies_bouldin_score(scaled_data, hierarchical_labels)
calinski_harabasz_hierarchical = calinski_harabasz_score(scaled_data, hierarchical_labels)

print("Hierarchical Clustering:")
print("Silhouette Score:", silhouette_hierarchical)
print("Davies-Bouldin Index:", davies_bouldin_hierarchical)
print("Calinski-Harabasz Index:", calinski_harabasz_hierarchical)
```

```
Hierarchical Clustering:
Silhouette Score: 0.27627621841002925
Davies-Bouldin Index: 1.4285021170311918
Calinski-Harabasz Index: 1320.4645654005237
```

So for visualizing the k-medoids clusters, we created a scatter plot of the data points and we used the first 2 columns of the data and they were plotted against each other, after this we added the x,y label and the legend and we used .show() in order to display the output. Same method was used for visualizing the hierarchical clustering.

```
# Visualize K-medoids clusters
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', s=50, alpha=0.5)
plt.scatter(clusters[:, 0], clusters[:, 1], c='red', s=200, marker='X', label='Cluster Centers')
plt.title('K-medoids Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()

# Visualize Hierarchical Clustering
plt.scatter(scaled_data.iloc[:, 0], scaled_data.iloc[:, 1], c=hierarchical_labels, cmap='viridis', s=50, alpha=0.5)
plt.title('Hierarchical Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Here is the output :