CUSTOMER MARKETING CLUSTRING

Machine Learning Project

MAY 25, 2023

Presented By: Biplab Kumar Bhunia, B.Sc. Data Science (3rd Year), Netaji Subhash Engineering Collage

ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to my faculty **(Mr. Ashoke Kumar Gupta)** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him/her time to time shall carry us a long way in the journey of life on which we are about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

PROJECT OBJECTIVE

Customer marketing clustering is a technique used in marketing to segment customers into different groups based on their shared characteristics or behaviours. This segmentation can help businesses to better understand, their customers and target them with more relevant and personalized marketing messages. Clustering algorithms are used to group customers based on their similarities, which can include demographic information such as age, gender, income, and location, as well as behavioral data such as purchase history, website activity, and social media engagement. By analyzing these data points, businesses can identify patterns and group customers who share similar characteristics or behaviours. Once customers are segmented into clusters, businesses can develop targeted marketing campaigns that speak to the specific needs and interests of each group. For example, a business might create a campaign aimed at customers who have previously purchased highend products, while another campaign might be targeted at customers who have only made a few purchases in the past. customer marketing clustering can help businesses to better.

PURPOSE OF CUSTOMER MARKETING CLUSTERING

The purpose of customer marketing clustering is to segment customers into distinct groups based on shared characteristics or behaviours, in order to improve marketing efforts and better target specific customer groups with more relevant and personalized messaging. By grouping customers based on similarities, businesses can better understand their customers and tailor marketing campaigns to meet their specific needs and preferences. This can result in increased customer engagement, improved customer satisfaction, and higher sales and profitability. Some common goals of customer marketing clustering include:

1. **Improving customer segmentation**: Clustering allows businesses to segment customers more accurately and effectively than traditional demographic-based segmentation methods

2. . **Increasing customer engagement**: Targeted marketing messages can be more effective at capturing customers' attention and encouraging them to engage with a brand

3. . **Enhancing customer satisfaction**: By tailoring marketing efforts to specific customer groups, businesses can demonstrate an understanding of their customers' needs and preferences, which can lead to increased customer satisfaction.

4. **Maximizing ROI:** Targeted marketing campaigns can result in higher conversion rates and lower marketing costs, resulting in a higher return on investment (ROI) for businesses. Overall, the purpose of customer marketing clustering is to improve the effectiveness and efficiency of marketing efforts by better understanding and targeting specific customer groups.

BENEFITS OF CUSTOMER MARKETING CLUSTERING

Customer marketing clustering offers several benefits to businesses looking to improve their marketing efforts and better understand their customers.

Here are some of the key benefits:

1. Improved customer targeting: By grouping customers based on shared characteristics or behaviours, businesses can target specific customer segments with more relevant and personalized marketing messages. This can lead to higher engagement and conversion rates.

2. Increased customer loyalty: Targeted marketing messages that address the unique needs and preferences of specific customer segments can help build customer loyalty and improve overall customer satisfaction.

3. More effective use of resources: By focusing marketing efforts on specific customer segments, businesses can optimize their marketing spend and resources. This can result in a higher return on investment (ROI) and increased profitability.

4. Enhanced customer insights: Customer marketing clustering can provide businesses with valuable insights into customer behaviours and preferences, which can inform future marketing strategies and product development.

5. Improved competitiveness: By better understanding and targeting specific customer segments, businesses can gain a competitive advantage in the market and improve their overall position. Overall, customer marketing clustering can help businesses improve the effectiveness and efficiency of their marketing efforts, build stronger customer relationships, and achieve better business outcomes

SUMMARY OF THE PROJECT

- In this project, you have been hired as a consultant to a bank in New York City. The bank has extensive data on their customers for the past 6 months. The marketing team at the bank wants to launch a targeted ad marketing campaign by dividing their customers into at least 3 distinctive groups.

- Marketing is crucial for the growth and sustainability of any business. Marketers can help build the company's brand, engage customers, grow revenue, and increase sales.

SYSTEM USED

HARDWARE –

Device name - Heaven

Processor - 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz 3.11 GHz

RAM – 8.00 GB (7.79 GB usable)

SOFTWARE-

Edition – Windows 11 Home Single Language

OS build - 22621.1702

LIBRARIES PACKAGES USED

- **__future__** module is a built-in module in Python that is used to inherit new features that will be available in the new Python versions. This module includes all the latest functions which were not present in the previous version in Python, And we can use this by importing the future module.

- **jupyter Widgets** are interactive browser controls for Jupyter notebooks. The ipywidgets package provides a basic, lightweight set of core form controls I that use the framework of interactive controls. These included controls include a text area, text box, select and multiselect controls, checkbox, sliders, tab panels, grid layout, etc.

- **IPython** module provides a rich toolkit to help one make the most of using Python interactively. It allows the object to create a rich display of Html, Images, Latex, Sound and Video.

- **pandas** is an open-source library that is made mainly for working with relational or labelled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and tome series.

- **NumPy** is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python.
-
- **The datetime module** supplies classes for manipulating dates and times.
  While date and time arithmetic is supported, the focus of the implementation is on efficient attribute extraction for output formatting and manipulation.

- **The matplotlib.ticker.Multiple L**ocator class is used for setting a tick for every integer multiple of a base within the view interval.
- **Seaborn i**s an open-source Python library built on top of matplotlib. It is used for data visualization and exploratory data analysis. Seaborn works easily with dataframes and the Pandas library. The graphs created can also be customized easily.
- **The plotly Python library** is an interactive, open source plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3-dimensional use-cases.

Built on top of the Plotly JavaScript library (plotly js), plotly enables Python users to create beautiful interactive web-based visualizations that can be displayed in Jupyter notebooks, saved to standalone HTML files.

CUSTOMER MARKETING CLUSTARING

TASK #1: UNDERSTAND THE PROBLEM STATEMENT AND BUSINESS CASE

TASK #2: IMPORT LIBRARIES AND DATASETS

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
```

In [2]:

```
# You have to include the full link to the csv file containing your dataset
creditcard_df = pd.read_csv(r'C:\Users\kumar\Desktop\Jupyter Notebook\project ..all\Cust
# 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 = f
# 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
# ONEOFF_PURCHASES_FREQUENCY: How frequently Purchases are happening in one-go (1 = freq
# PURCHASES_INSTALLMENTS_FREQUENCY: How frequently purchases in installments are being d
# 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
# 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
```

In this section, we will provide data visualizations that summarizes or extracts relevant characteristics of features in our dataset. Let's look at each column in detail, get a better understanding of the dataset, and group them together when appropriate.

Data Description and Exploratory Visualisations

In [3]:

```
creditcard_df
```

Out[3]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
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

In [4]:

```
creditcard_df.info()
# 18 features with 8950 points
```

```
<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 [5]:

```
creditcard_df.describe()
# Mean balance is $1564
# Balance frequency is frequently updated on average ~0.9
# Purchases average is $1000
# one off purchase average is ~$600
# Average purchases frequency is around 0.5
# average ONEOFF_PURCHASES_FREQUENCY, PURCHASES_INSTALLMENTS_FREQUENCY, and CASH_ADVANCE
# Average credit limit ~ 4500
# Percent of full payment is 15%
# Average tenure is 11 years
```

Out[5]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALL
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	
std	2081.531879	0.236904	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	
max	19043.138560	1.000000	49039.570000	40761.250000	

In [6]:

```
creditcard_df[creditcard_df['ONEOFF_PURCHASES']==creditcard_df['ONEOFF_PURCHASES'].max()
```

Out[6]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	IN
550	C10574	11547.52001	1.0	49039.57	40761.25	

In [7]:

```
# Let's see who made one off purchase of $40761!
creditcard_df[creditcard_df['ONEOFF_PURCHASES'] == 40761.25]
```

Out[7]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	IN
550	C10574	11547.52001	1.0	49039.57	40761.25	

In [8]:

```
creditcard_df[['BALANCE','BALANCE_FREQUENCY','PURCHASES','ONEOFF_PURCHASES']][creditcard
```

Out[8]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
550	11547.52001	1.0	49039.57	40761.25

In [9]:

```
creditcard_df['CASH_ADVANCE'].max()
```

Out[9]:

```
47137.21176
```

In [10]:

```
# Let's see who made cash advance of $47137!
# This customer made 123 cash advance transactions!!
# Never paid credit card in full

creditcard_df[creditcard_df['CASH_ADVANCE'] == 47137.211760]
```

Out[10]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
2159	C12226	10905.05381	1.0	431.93	133.5

TASK #3: VISUALIZE AND EXPLORE DATASET

In [11]:

```
#check is there any null values
creditcard_df.isnull().sum()
```

Out[11]:

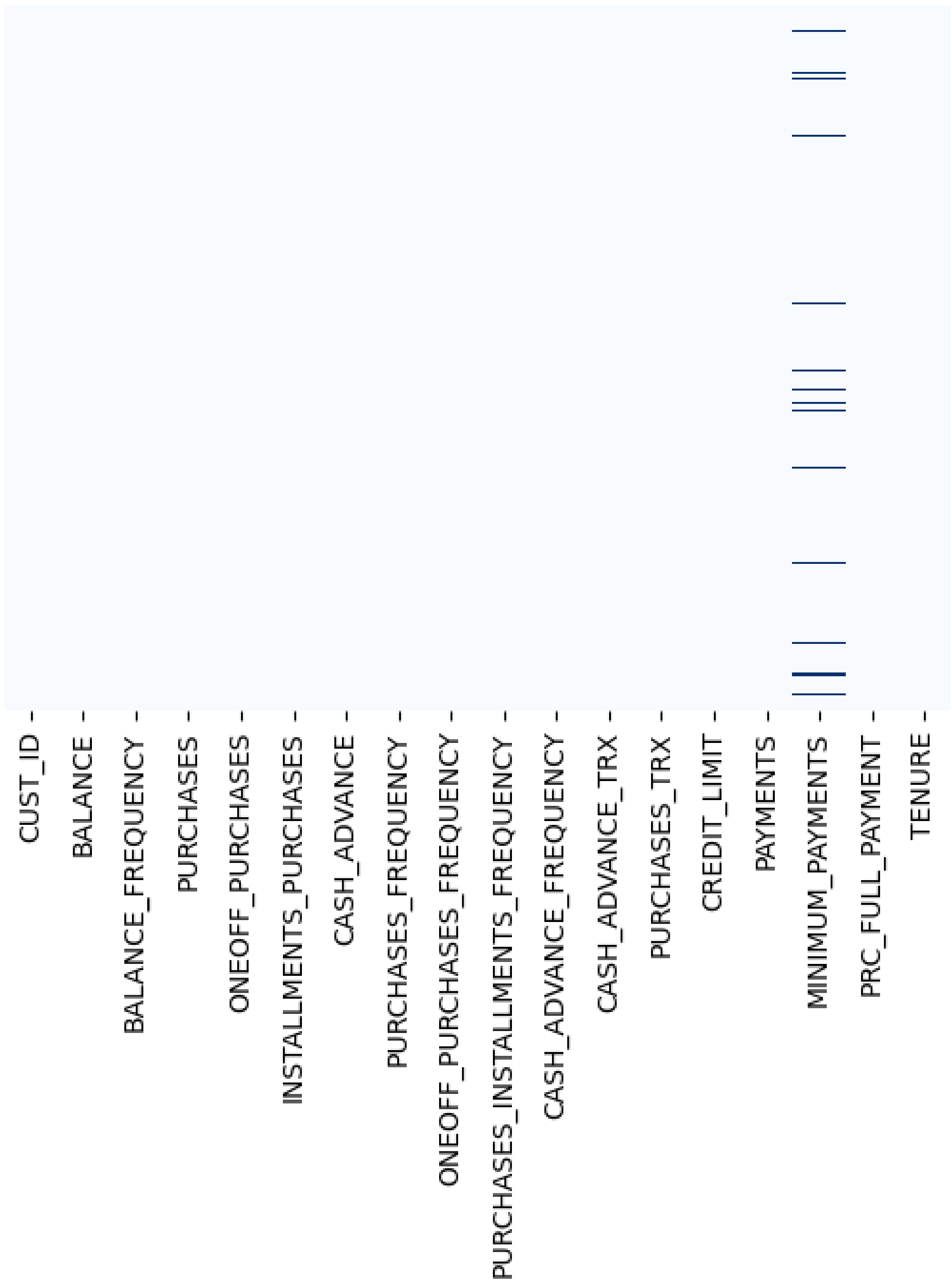
```
CUST_ID                               0
BALANCE                               0
BALANCE_FREQUENCY                     0
PURCHASES                             0
ONEOFF_PURCHASES                      0
INSTALLMENTS_PURCHASES                0
CASH_ADVANCE                          0
PURCHASES_FREQUENCY                   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
```

In [12]:

```
# Let's see if we have any missing data, luckily we don't!
sns.heatmap(creditcard_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

Out[12]:

```
<AxesSubplot:>
```



In [13]:

```
creditcard_df.isnull().sum()
```

Out[13]:

```
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
```

In [14]:

```
# Fill up the missing elements with mean of the 'MINIMUM_PAYMENT'
creditcard_df.loc[(creditcard_df['MINIMUM_PAYMENTS'].isnull() == True), 'MINIMUM_PAYMENT
```

In [15]:

```
# Fill up the missing elements with mean of the 'CREDIT_LIMIT'
creditcard_df.loc[(creditcard_df['CREDIT_LIMIT'].isnull() == True), 'CREDIT_LIMIT'] = cr
```

In [16]:

```
sns.heatmap(creditcard_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

Out[16]:

```
<AxesSubplot:>
```

In [17]:

```
creditcard_df.isnull().sum()
```

Out[17]:

```
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
```

In [18]:

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

Out[18]:

```
0
```

In [19]:

```
creditcard_df=creditcard_df.dropna()
```

In [20]:

```
# Let's drop Customer ID since it has no meaning here
creditcard_df.drop("CUST_ID", axis = 1, inplace= True)
```

In [21]:

```
creditcard_df.head()
```

Out[21]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENT
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	

In [22]:

```
n = len(creditcard_df.columns)
n
```

Out[22]:

```
17
```

In [23]:

```
creditcard_df.columns
```

Out[23]:

```
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_PAYMEN
T',
       'TENURE'],
      dtype='object')
```

In [24]:

```
# distplot combines the matplotlib.hist function with seaborn kdeplot()
# KDE Plot represents the Kernel Density Estimate
# KDE is used for visualizing the Probability Density of a continuous variable.
# KDE demonstrates the probability density at different values in a continuous variable.

# Mean of balance is $1500
# 'Balance_Frequency' for most customers is updated frequently ~1
# For 'PURCHASES_FREQUENCY', there are two distinct group of customers
# For 'ONEOFF_PURCHASES_FREQUENCY' and 'PURCHASES_INSTALLMENT_FREQUENCY' most users don'
# Very small number of customers pay their balance in full 'PRC_FULL_PAYMENT'~0
# Credit limit average is around $4500
# Most customers are ~11 years tenure

plt.figure(figsize=(10,50))
for i in range(len(creditcard_df.columns)):
  plt.subplot(17, 1, i+1)
  sns.distplot(creditcard_df[creditcard_df.columns[i]], kde_kws={"color": "b", "lw": 3,
  plt.title(creditcard_df.columns[i])

plt.tight_layout()
```

```
D:\Jupyter Notebook\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes
-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\Jupyter Notebook\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes
-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\Jupyter Notebook\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes
-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\Jupyter Notebook\lib\site-packages\seaborn\distributions.py:2619: F
```

sns.pairplot(creditcard_df)

Correlation between 'PURCHASES' and ONEOFF_PURCHASES & INSTALMENT_PURCHASES

Trend between 'PURCHASES' and 'CREDIT_LIMIT' & 'PAYMENTS'

In [25]:

```
correlations = creditcard_df.corr()
```

In [26]:

```
correlations
```

Out[26]:

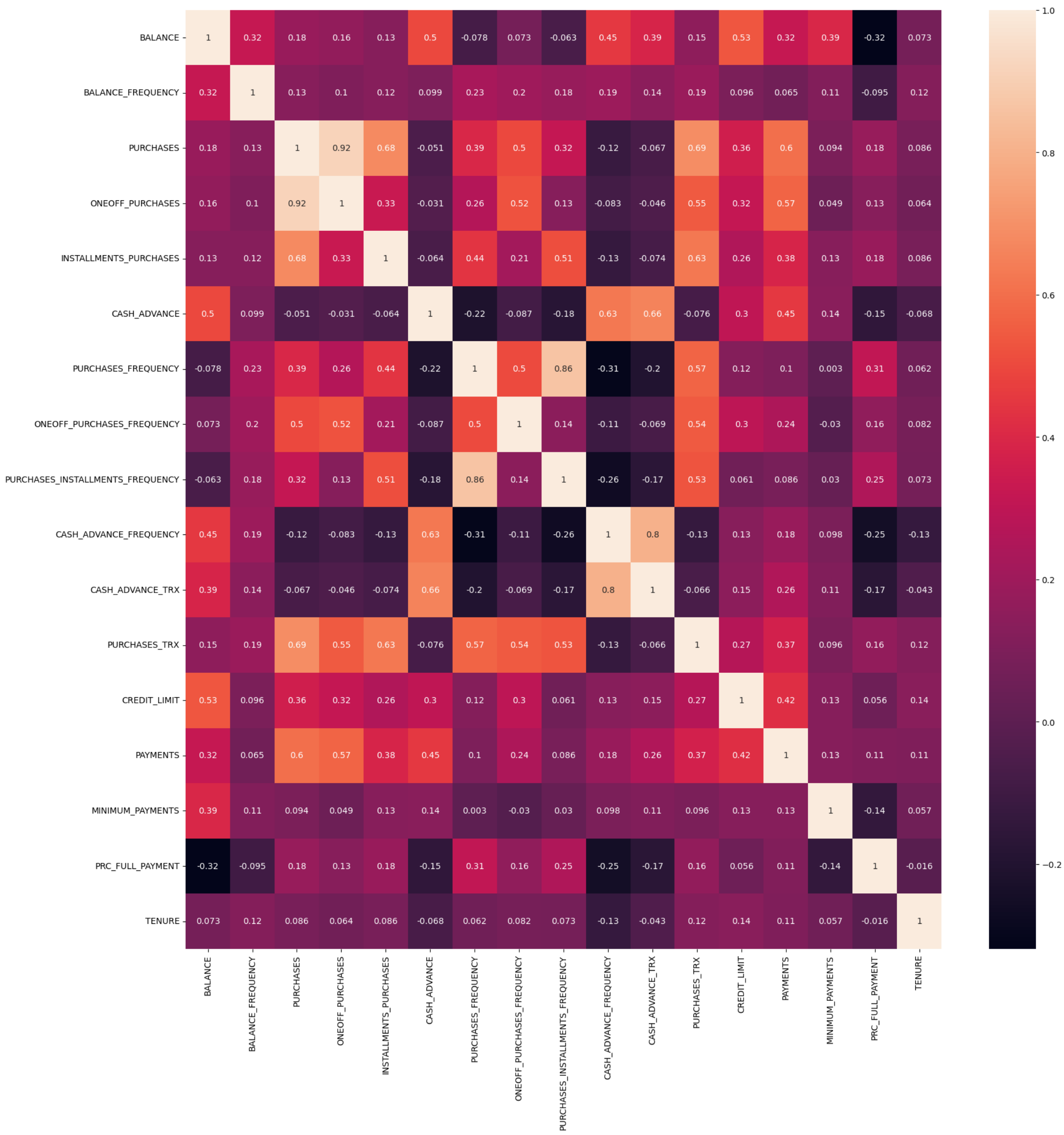
	BALANCE	BALANCE_FREQUENCY	PURCHASES
BALANCE	1.000000	0.322412	0.181261
BALANCE_FREQUENCY	0.322412	1.000000	0.133674
PURCHASES	0.181261	0.133674	1.000000
ONEOFF_PURCHASES	0.164350	0.104323	0.916845
INSTALLMENTS_PURCHASES	0.126469	0.124292	0.679896
CASH_ADVANCE	0.496692	0.099388	-0.051474
PURCHASES_FREQUENCY	-0.077944	0.229715	0.393017
ONEOFF_PURCHASES_FREQUENCY	0.073166	0.202415	0.498430
PURCHASES_INSTALLMENTS_FREQUENCY	-0.063186	0.176079	0.315567
CASH_ADVANCE_FREQUENCY	0.449218	0.191873	-0.120143
CASH_ADVANCE_TRX	0.385152	0.141555	-0.067175
PURCHASES_TRX	0.154338	0.189626	0.689561
CREDIT_LIMIT	0.531267	0.095795	0.356959
PAYMENTS	0.322802	0.065008	0.603264
MINIMUM_PAYMENTS	0.394282	0.114249	0.093515
PRC_FULL_PAYMENT	-0.318959	-0.095082	0.180379
TENURE	0.072692	0.119776	0.086288

In [27]:

```
f, ax = plt.subplots(figsize = (20, 20))
sns.heatmap(correlations, annot = True)

# 'PURCHASES' have high correlation between one-off purchases, 'installment purchases, p
# Strong Positive Correlation between 'PURCHASES_FREQUENCY' and 'PURCHASES_INSTALLMENT_F
```

Out[27]:

```
<AxesSubplot:>
```



TASK #4: UNDERSTAND THE THEORY AND INTUITON BEHIND K-MEANS

TASK #5: FIND THE OPTIMAL NUMBER OF CLUSTERS USING ELBOW METHOD

- The elbow method is a heuristic method of interpretation and validation of consistency within cluster analysis designed to help find the appropriate number of clusters in a dataset.
- If the line chart looks like an arm, then the "elbow" on the arm is the value of k that is the best.
- Source:
  - https://en.wikipedia.org/wiki/Elbow_method_(clustering) (https://en.wikipedia.org/wiki/Elbow_method_(clustering))
  - https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/ (https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/)

Let's scale the data first

scaler = StandardScaler() creditcard_df_scaled = scaler.fit_transform(creditcard_df)

In [28]:

```
# Let's scale the data first
scaler = StandardScaler()
creditcard_df_scaled = scaler.fit_transform(creditcard_df)
```

In [29]:

```
creditcard_df_scaled.shape
```

Out[29]:

```
(8950, 17)
```

In [30]:

```
creditcard_df_scaled
```

Out[30]:

```
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]])
```

In [31]:

```
pip install -U threadpoolctl
```

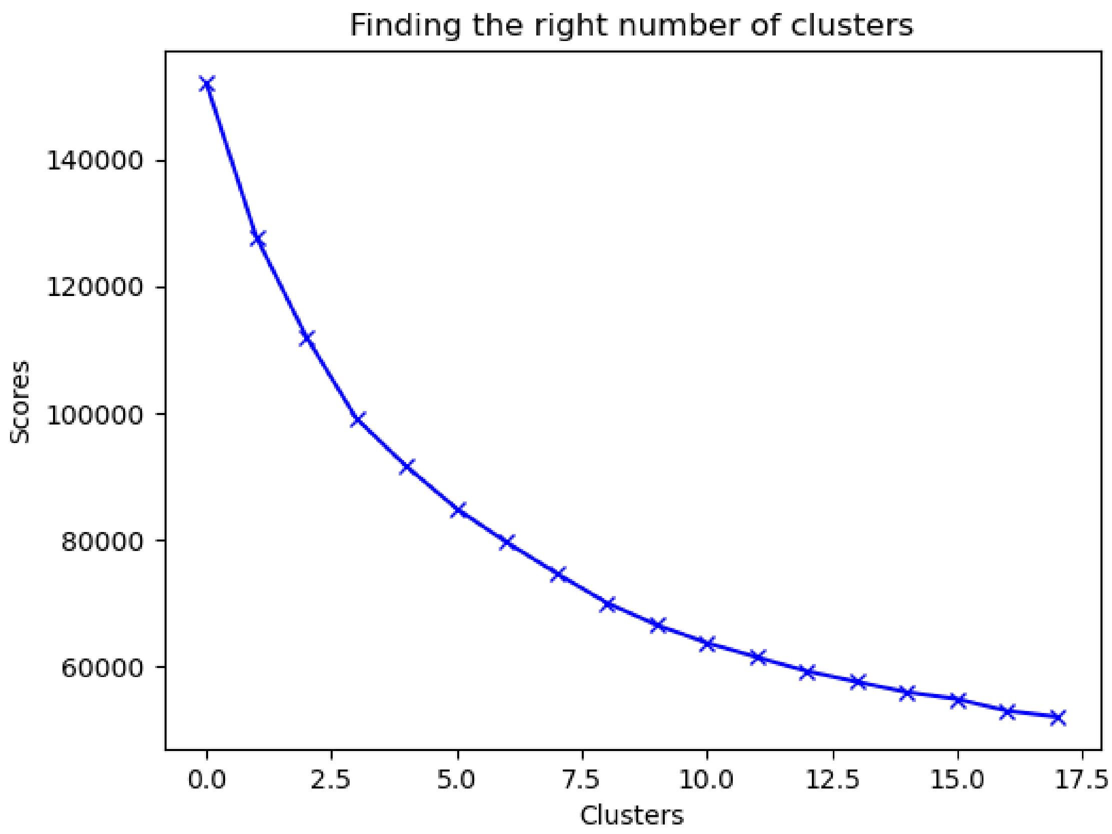
```
Requirement already satisfied: threadpoolctl in d:\jupyter notebook\lib\s
ite-packages (3.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [32]:

```
scores_1 = []

range_values = range(1, 19)

for i in range_values:
  kmeans = KMeans(n_clusters = i)
  kmeans.fit(creditcard_df_scaled)
  scores_1.append(kmeans.inertia_)

plt.plot(scores_1, 'bx-')
plt.title('Finding the right number of clusters')
plt.xlabel('Clusters')
plt.ylabel('Scores')
plt.show()

# From this we can observe that, 4th cluster seems to be forming the elbow of the curve.
# However, the values does not reduce linearly until 8th cluster.
# Let's choose the number of clusters to be 7.
```

```
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
```

```
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



TASK #6: APPLY K-MEANS METHOD

In [33]:

```
from sklearn.metrics import silhouette_score,silhouette_samples
```

In [34]:

```
kmeans = KMeans(3)
kmeans.fit(creditcard_df_scaled)
labels = kmeans.labels_
```

```
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

In [35]:

```
labels
```

Out[35]:

```
array([1, 0, 1, ..., 1, 1, 1])
```

In [36]:

```
kmeans.cluster_centers_.shape
```

Out[36]:

```
(3, 17)
```

In [37]:

```
cluster_centers = pd.DataFrame(data = kmeans.cluster_centers_, columns = [creditcard_df.
cluster_centers
```

Out[37]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	1.164422	0.340369	-0.289665	-0.207007	
1	-0.366668	-0.179335	-0.234786	-0.206464	
2	0.304769	0.439915	1.509242	1.266620	

In [38]:

```
# 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 = [creditcard_df.columns]
cluster_centers

# First Customers cluster (Transactors): Those are customers who pay least
#amount of intrerest charges and careful with their money,
#Cluster with lowest balance ($104) and cash advance ($303), Percentage of full payment
# Second customers cluster (revolvers) who use credit card as a loan (most lucrative sec
#highest balance ($5000) and cash advance (~$5000),
#low purchase frequency, high cash advance frequency (0.5),
#high cash advance transactions (16) and low percentage of full payment (3%)
# Third customer cluster (VIP/Prime): high credit limit $16K
#and highest percentage of full payment,
#target for increase credit limit and increase spending habits
# Fourth customer cluster (low tenure): these are customers with low tenure (7 years),
#low balance
```

Out[38]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENT
0	3988.121471	0.957901	384.331701	248.847985	
1	801.285856	0.834788	501.581185	249.748741	
2	2198.825426	0.981483	4227.724677	2694.767179	

In [39]:

```
labels.shape # Labels associated to each data point
```

Out[39]:

```
(8950,)
```

In [40]:

```
labels.max()
```

Out[40]:

```
2
```

In [41]:

```
labels.min()
```

Out[41]:

```
0
```

In [42]:

```
y_kmeans = kmeans.fit_predict(creditcard_df_scaled)
y_kmeans
```

```
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

Out[42]:

```
array([0, 1, 0, ..., 0, 0, 0])
```

In [43]:

```
# concatenate the clusters labels to our original dataframe
creditcard_df_cluster = pd.concat([creditcard_df, pd.DataFrame({'cluster':labels})], axi
creditcard_df_cluster.describe()
```

Out[43]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALL
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	
std	2081.531879	0.236904	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	
max	19043.138560	1.000000	49039.570000	40761.250000	

In [44]:

```
creditcard_df_cluster[creditcard_df_cluster['cluster']==0].count()
```

Out[44]:

```
BALANCE                             1590
BALANCE_FREQUENCY                   1590
PURCHASES                           1590
ONEOFF_PURCHASES                    1590
INSTALLMENTS_PURCHASES              1590
CASH_ADVANCE                        1590
PURCHASES_FREQUENCY                 1590
ONEOFF_PURCHASES_FREQUENCY          1590
PURCHASES_INSTALLMENTS_FREQUENCY    1590
CASH_ADVANCE_FREQUENCY              1590
CASH_ADVANCE_TRX                    1590
PURCHASES_TRX                       1590
CREDIT_LIMIT                        1590
PAYMENTS                            1590
MINIMUM_PAYMENTS                    1590
PRC_FULL_PAYMENT                    1590
TENURE                              1590
cluster                             1590
dtype: int64
```

In [45]:

```
creditcard_df_cluster.describe()
```

Out[45]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALL
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	
std	2081.531879	0.236904	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	
max	19043.138560	1.000000	49039.570000	40761.250000	

In [46]:

```
creditcard_df_cluster[creditcard_df_cluster['cluster']==0].describe()
```

Out[46]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLI
count	1590.000000	1590.000000	1590.000000	1590.000000	
mean	3990.839154	0.957822	384.756151	249.255119	
std	2685.948448	0.116038	739.703841	575.620913	
min	4.382924	0.181818	0.000000	0.000000	
25%	1873.857963	1.000000	0.000000	0.000000	
50%	3465.047301	1.000000	0.000000	0.000000	
75%	5567.984401	1.000000	462.750000	226.732500	
max	16304.889250	1.000000	7194.530000	6678.260000	

In [47]:

```
creditcard_df_cluster[creditcard_df_cluster['cluster']==1].describe()
```

Out[47]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLM
count	6107.000000	6107.000000	6107.000000	6107.000000	
mean	802.449062	0.834923	502.076995	250.290431	
std	960.680914	0.268183	593.244338	477.382971	
min	0.000000	0.000000	0.000000	0.000000	
25%	57.412479	0.727273	55.600000	0.000000	
50%	427.905890	1.000000	302.000000	0.000000	
75%	1246.300289	1.000000	735.390000	301.295000	
max	6937.806466	1.000000	5080.850000	4900.000000	

In [48]:

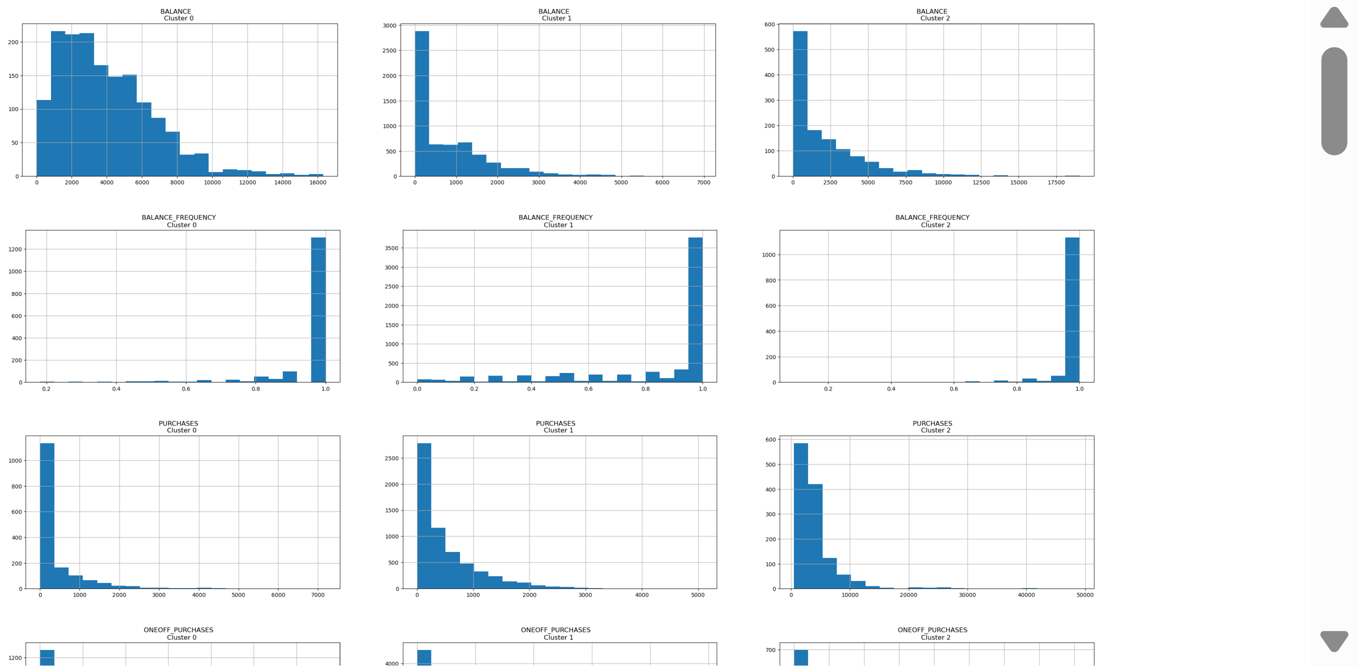
```
creditcard_df_cluster[creditcard_df_cluster['cluster']==2].describe()
```

Out[48]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALL
count	1253.000000	1253.000000	1253.000000	1253.00000	
mean	2199.568262	0.981453	4230.436369	2695.51091	
std	2568.989774	0.073846	4252.781315	3607.59115	
min	12.423203	0.090909	498.170000	0.00000	
25%	357.627180	1.000000	2122.980000	912.70000	
50%	1193.708983	1.000000	3094.970000	1774.91000	
75%	3145.204423	1.000000	4767.110000	3162.94000	
max	19043.138560	1.000000	49039.570000	40761.25000	

In [49]:

```
# Plot the histogram of various clusters
for i in creditcard_df.columns:
    plt.figure(figsize = (35, 5))
    for j in range(3):
        plt.subplot(1,3,j+1)
        cluster = creditcard_df_cluster[creditcard_df_cluster['cluster'] == j]
        cluster[i].hist(bins = 20)
        plt.title('{}    \nCluster {} '.format(i,j))

    plt.show()
```



TASK 7: APPLY PRINCIPAL COMPONENT ANALYSIS AND VISUALIZE THE RESULTS

In [50]:

```
# Obtain the principal components
pca = PCA(n_components=2)
principal_comp = pca.fit_transform(creditcard_df_scaled)
principal_comp
```

Out[50]:

```
array([[-1.68222049, -1.07645313],
       [-1.13829445,  2.50648021],
       [ 0.96967681, -0.38353889],
       ...,
       [-0.92620383, -1.81078999],
       [-2.33654899, -0.65796168],
       [-0.5564211 , -0.40045945]])
```

In [51]:

```
# Create a dataframe with the two components
pca_df = pd.DataFrame(data = principal_comp, columns =['pca1','pca2'])
pca_df.head()
```

Out[51]:

	pca1	pca2
0	-1.682220	-1.076453
1	-1.138294	2.506480
2	0.969677	-0.383539
3	-0.873626	0.043169
4	-1.599435	-0.688586

In [52]:

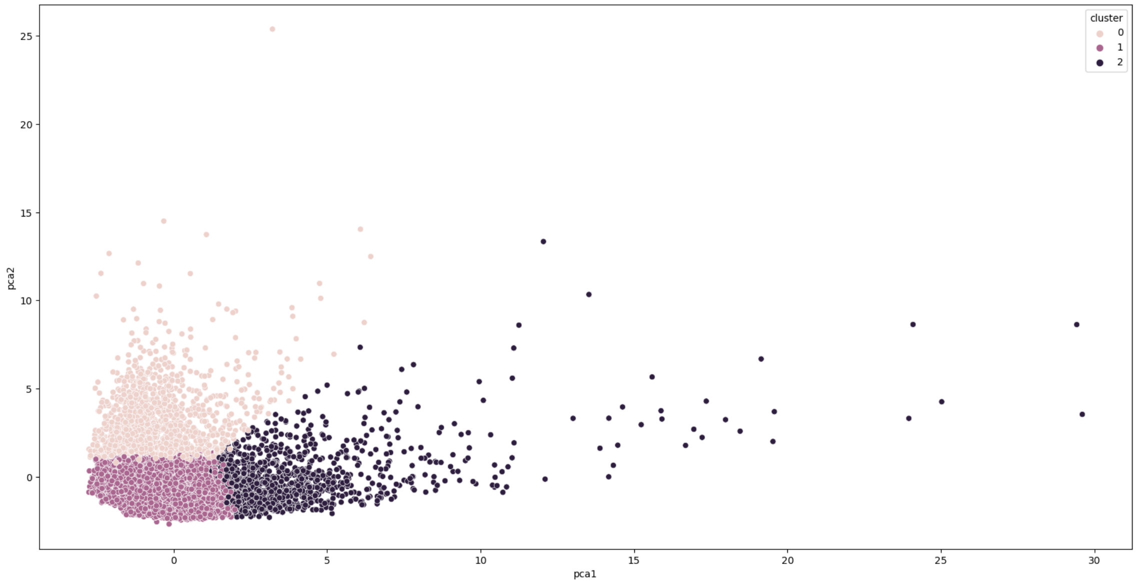
```
# Concatenate the clusters labels to the dataframe
pca_df = pd.concat([pca_df,pd.DataFrame({'cluster':labels})], axis = 1)
pca_df.head()
```

Out[52]:

	pca1	pca2	cluster
0	-1.682220	-1.076453	1
1	-1.138294	2.506480	0
2	0.969677	-0.383539	1
3	-0.873626	0.043169	1
4	-1.599435	-0.688586	1

In [53]:

```
plt.figure(figsize=(20,10))
ax = sns.scatterplot(x="pca1", y="pca2", hue = "cluster", data = pca_df)
plt.show()
```



In [54]:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

range_n_clusters = [2, 3, 4, 5, 6,7,8,9,10]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(creditcard_df_scaled) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(creditcard_df_scaled)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(creditcard_df_scaled, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(creditcard_df_scaled, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples
```

```
    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(creditcard_df_scaled[:, 0], creditcard_df_scaled[:, 1], marker='.', s=30
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
```

```
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'a
uto' in 1.4. Set the value of `n_init` explicitly to suppress the warn
ing
  warnings.warn(

For n_clusters = 2 The average silhouette_score is : 0.209849815806675
25

D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'a
uto' in 1.4. Set the value of `n_init` explicitly to suppress the warn
ing
  warnings.warn(

For n_clusters = 3 The average silhouette_score is : 0.251115475809893
96

D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'a
```

TASK #8: UNDERSTAND THE THEORY AND INTUITION BEHIND AUTOENCODERS

In [56]:

```
conda install -c conda-forge pinocchio
```

```
Note: you may need to restart the kernel to use updated packages.

usage: conda-script.py [-h] [-V] command ...
conda-script.py: error: unrecognized arguments: pinocchio
```

TASK #9: APPLY AUTOENCODERS (PERFORM DIMENSIONALITY REDUCTION USING AUTOENCODERS)

In [57]:

```
from tensorflow.keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchN
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.initializers import glorot_uniform
from keras.optimizers import SGD

encoding_dim = 7

input_df = Input(shape=(17,))


# Glorot normal initializer (Xavier normal initializer) draws samples from a truncated n

x = Dense(encoding_dim, activation='relu')(input_df)
x = Dense(500, activation='relu', kernel_initializer = 'glorot_uniform')(x)
x = Dense(500, activation='relu', kernel_initializer = 'glorot_uniform')(x)
x = Dense(2000, activation='relu', kernel_initializer = 'glorot_uniform')(x)

encoded = Dense(10, activation='relu', kernel_initializer = 'glorot_uniform')(x)

x = Dense(2000, activation='relu', kernel_initializer = 'glorot_uniform')(encoded)
x = Dense(500, activation='relu', kernel_initializer = 'glorot_uniform')(x)

decoded = Dense(17, kernel_initializer = 'glorot_uniform')(x)

# autoencoder
autoencoder = Model(input_df, decoded)

#encoder - used for our dimention reduction
encoder = Model(input_df, encoded)

autoencoder.compile(optimizer= 'adam', loss='mean_squared_error')
```

In [58]:

```
creditcard_df_scaled.shape
```

Out[58]:

```
(8950, 17)
```

In [60]:

```
autoencoder.fit(creditcard_df_scaled, creditcard_df_scaled, batch_size = 128, epochs = 2
```

```
Epoch 1/25
70/70 [==============================] - 3s 44ms/step - loss: 0.0528
Epoch 2/25
70/70 [==============================] - 4s 55ms/step - loss: 0.0514
Epoch 3/25
70/70 [==============================] - 5s 71ms/step - loss: 0.0480
Epoch 4/25
70/70 [==============================] - 4s 55ms/step - loss: 0.0487
Epoch 5/25
70/70 [==============================] - 3s 47ms/step - loss: 0.0487
Epoch 6/25
70/70 [==============================] - 4s 50ms/step - loss: 0.0492
Epoch 7/25
70/70 [==============================] - 4s 56ms/step - loss: 0.0454
Epoch 8/25
70/70 [==============================] - 3s 45ms/step - loss: 0.0459
Epoch 9/25
70/70 [==============================] - 3s 44ms/step - loss: 0.0500
Epoch 10/25
70/70 [==============================] - 3s 44ms/step - loss: 0.0427
Epoch 11/25
70/70 [==============================] - 3s 46ms/step - loss: 0.0460
Epoch 12/25
70/70 [==============================] - 3s 46ms/step - loss: 0.0414
Epoch 13/25
70/70 [==============================] - 3s 47ms/step - loss: 0.0394
Epoch 14/25
70/70 [==============================] - 4s 58ms/step - loss: 0.0378
Epoch 15/25
70/70 [==============================] - 3s 49ms/step - loss: 0.0358
Epoch 16/25
70/70 [==============================] - 4s 56ms/step - loss: 0.0373
Epoch 17/25
70/70 [==============================] - 3s 46ms/step - loss: 0.0333
Epoch 18/25
70/70 [==============================] - 3s 45ms/step - loss: 0.0333
Epoch 19/25
70/70 [==============================] - 3s 47ms/step - loss: 0.0339
Epoch 20/25
70/70 [==============================] - 3s 45ms/step - loss: 0.0327
Epoch 21/25
70/70 [==============================] - 3s 49ms/step - loss: 0.0294
Epoch 22/25
70/70 [==============================] - 4s 58ms/step - loss: 0.0297
Epoch 23/25
70/70 [==============================] - 4s 55ms/step - loss: 0.0328
Epoch 24/25
70/70 [==============================] - 3s 45ms/step - loss: 0.0340
Epoch 25/25
70/70 [==============================] - 4s 54ms/step - loss: 0.0348
```

Out[60]:

```
<keras.callbacks.History at 0x254096c1100>
```

In [61]:

```
autoencoder.save_weights('autoencoder.h5')
```

In [62]:

```
pred = encoder.predict(creditcard_df_scaled)
```

```
280/280 [==============================] - 1s 4ms/step
```

In [63]:

```
pred.shape
```

Out[63]:

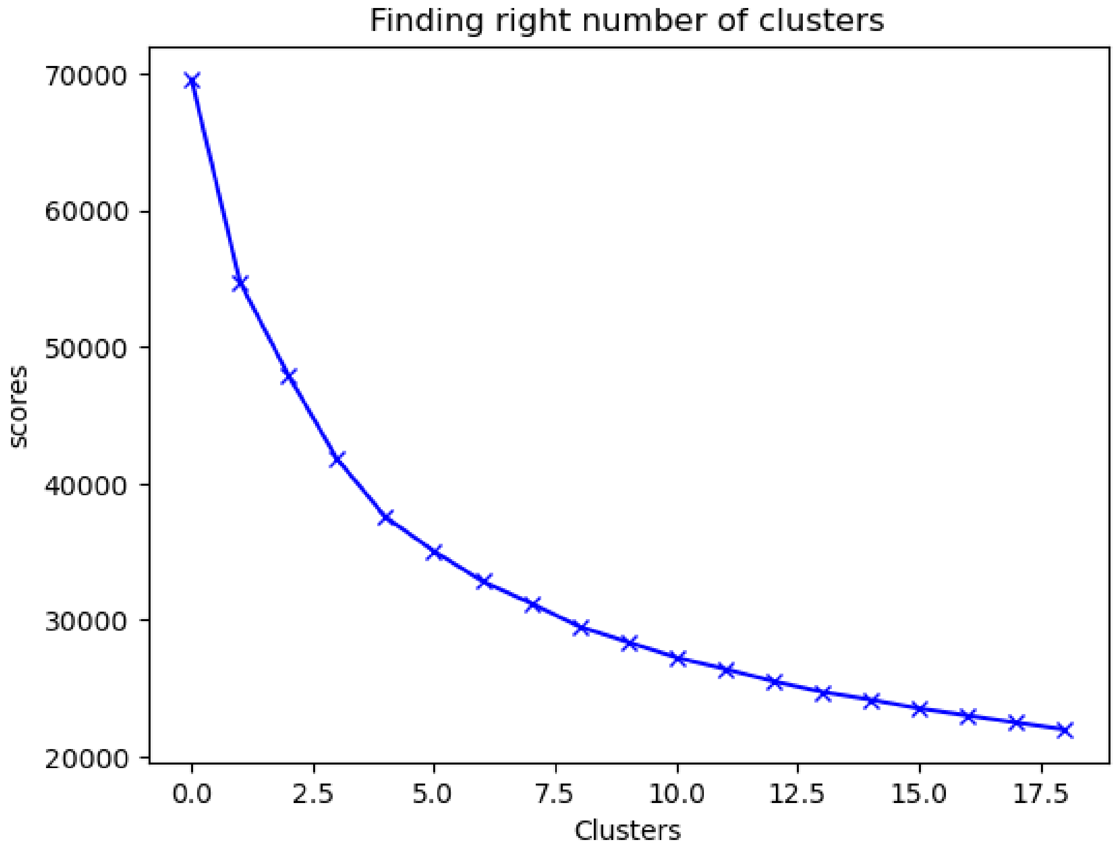
```
(8950, 10)
```

In [64]:

```
scores_2 = []

range_values = range(1, 20)

for i in range_values:
  kmeans = KMeans(n_clusters= i)
  kmeans.fit(pred)
  scores_2.append(kmeans.inertia_)

plt.plot(scores_2, 'bx-')
plt.title('Finding right number of clusters')
plt.xlabel('Clusters')
plt.ylabel('scores')
plt.show()
```

```
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
```

```
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



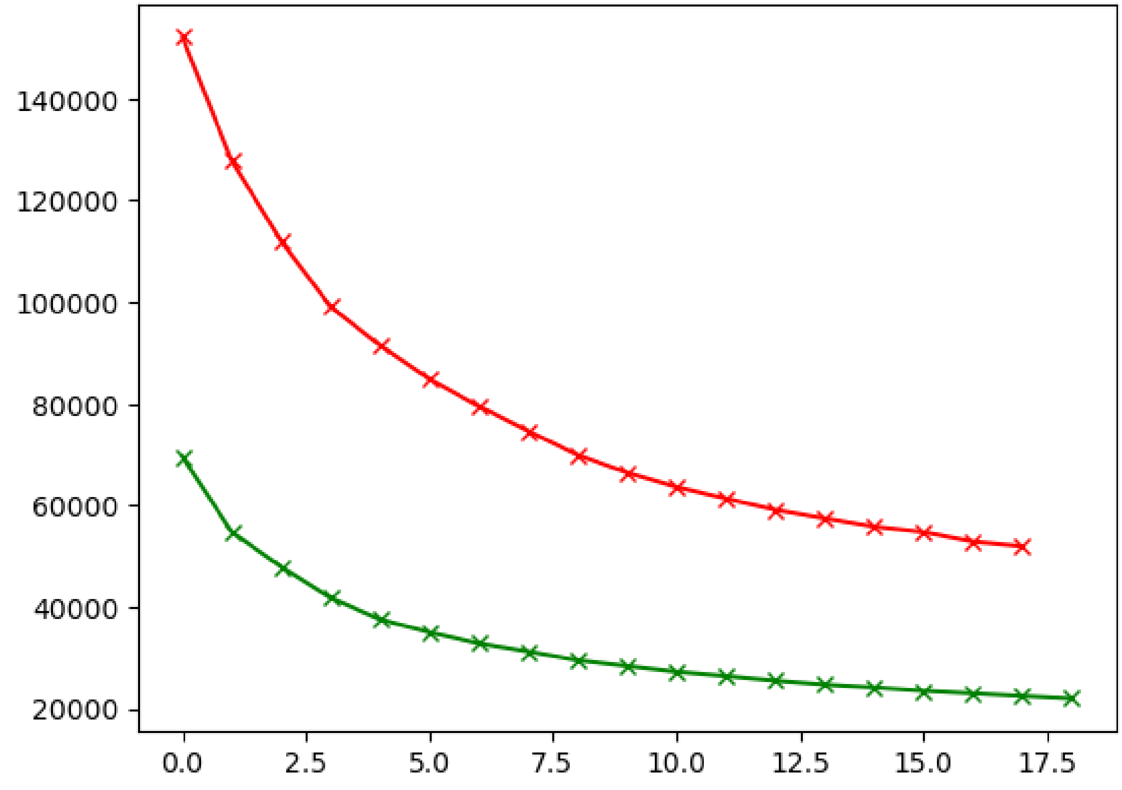
In [65]:

```
plt.plot(scores_1, 'bx-', color = 'r')
plt.plot(scores_2, 'bx-', color = 'g')
```

```
C:\Users\kumar\AppData\Local\Temp\ipykernel_13256\3067751309.py:1: UserWa
rning: color is redundantly defined by the 'color' keyword argument and t
he fmt string "bx-" (-> color='b'). The keyword argument will take preced
ence.
  plt.plot(scores_1, 'bx-', color = 'r')
C:\Users\kumar\AppData\Local\Temp\ipykernel_13256\3067751309.py:2: UserWa
rning: color is redundantly defined by the 'color' keyword argument and t
he fmt string "bx-" (-> color='b'). The keyword argument will take preced
ence.
  plt.plot(scores_2, 'bx-', color = 'g')
```

Out[65]:

```
[<matplotlib.lines.Line2D at 0x254099f4130>]
```



In [66]:

```
kmeans = KMeans(4)
kmeans.fit(pred)
labels = kmeans.labels_
y_kmeans = kmeans.fit_predict(creditcard_df_scaled)
```

```
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Jupyter Notebook\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
ureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

In [67]:

```
df_cluster_dr = pd.concat([creditcard_df, pd.DataFrame({'cluster':labels})], axis = 1)
df_cluster_dr.head()
```

Out[67]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENT
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	

In [68]:

```
pca = PCA(n_components=2)
prin_comp = pca.fit_transform(pred)
pca_df = pd.DataFrame(data = prin_comp, columns =['pca1','pca2'])
pca_df.head()
```

Out[68]:

	pca1	pca2
0	-1.664853	-0.173598
1	-0.348903	1.432988
2	-1.068302	-0.506685
3	-0.478761	0.112044
4	-1.721791	-0.110719

In [69]:

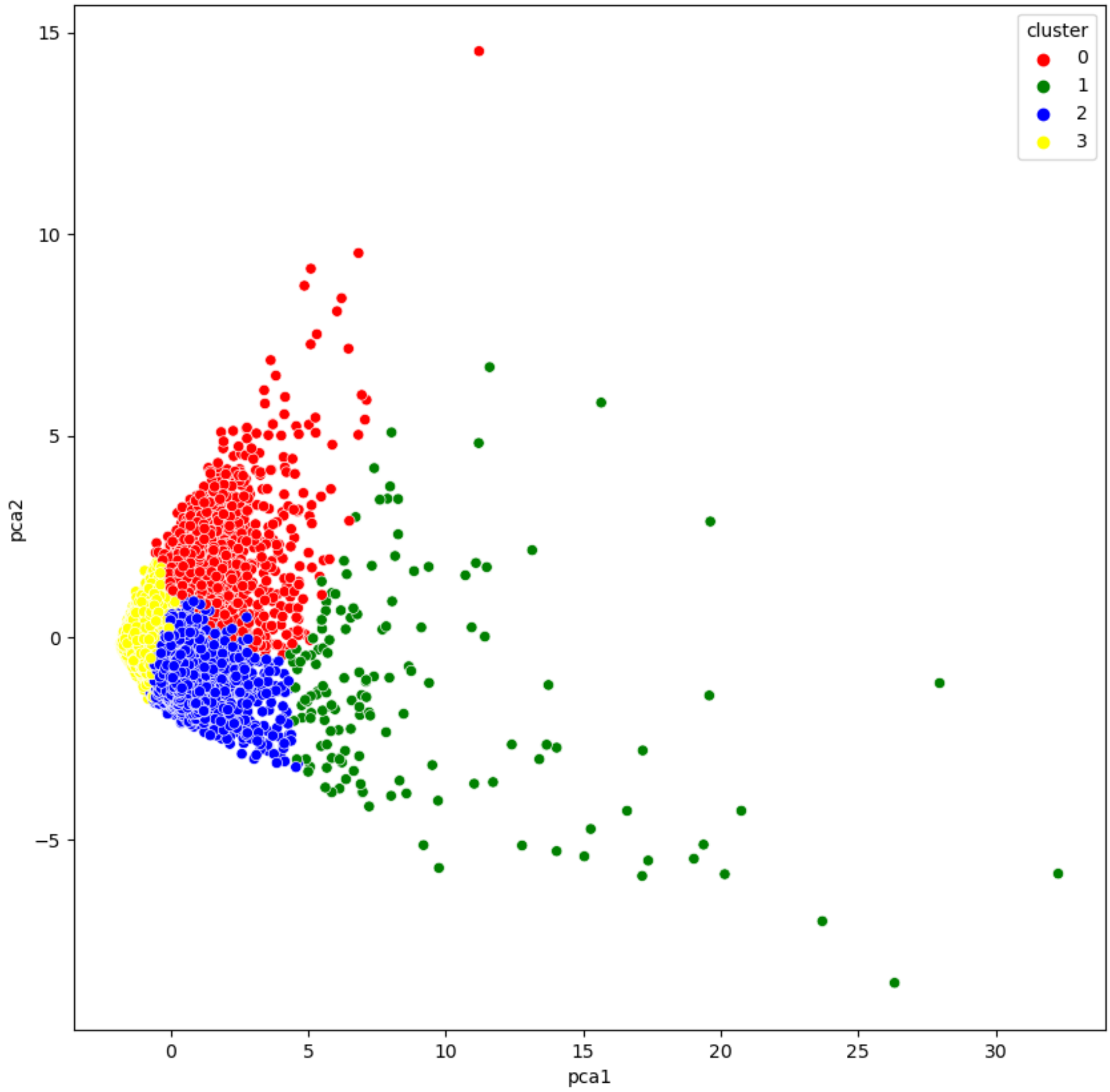
```
pca_df = pd.concat([pca_df,pd.DataFrame({'cluster':labels})], axis = 1)
pca_df.head()
```

Out[69]:

	pca1	pca2	cluster
0	-1.664853	-0.173598	3
1	-0.348903	1.432988	3
2	-1.068302	-0.506685	3
3	-0.478761	0.112044	3
4	-1.721791	-0.110719	3

In [70]:

```
plt.figure(figsize=(10,10))
ax = sns.scatterplot(x="pca1", y="pca2", hue = "cluster", data = pca_df, palette =['red'
plt.show()
```



Thank You

In [ ]: