Unsupervised Machine Learning for Customer Market Segmentation

Guided Project completed by Suhaimi William Chan

Instructor: Ryan Ahmed

Course Objectives

In this course, we are going to focus on the following learning objectives:

1. *Understand how to leverage the power of machine learning to transform marketing departments and perform customer segmentation*
2. *Apply Python libraries to import and visualize dataset images.*
3. *Understand the theory and intuition behind k-means clustering machine learning algorithm*
4. *Learn how to obtain the optimal number of clusters using the elbow method*
5. *Apply Scikit-Learn library to find the optimal number of clusters using elbow method*
6. *Apply k-means in Scikit-Learn to perform customer segmentation*
7. *Understand the theory and intuition behind Principal Component Analysis (PCA) algorithm*
8. *Apply Principal Component Analysis (PCA) technique to perform dimensionality reduction and data visualization*
9. *Compile and fit unsupervised machine learning models such as PCA and K-Means to training data*

Project Structure

The hands on project on **Unsupervised Machine Learning for Customer Segmentation** is divided into following tasks:

Task 1: Understand the problem statement and business case

Task 2: Import libraries and datasets

Task 3: Visualize and explore datasets

Task 4: Understand the theory and intuition behind k-means clustering machine learning algorithm

Task 5: Learn how to obtain the optimal number of clusters using the elbow method

Task 6: Use Scikit-Learn library to find the optimal number of clusters using elbow method

Task 7: Apply k-means using Scikit-Learn to perform customer segmentation

Task 8: Apply Principal Component Analysis (PCA) technique to perform dimensionality reduction and data visualization

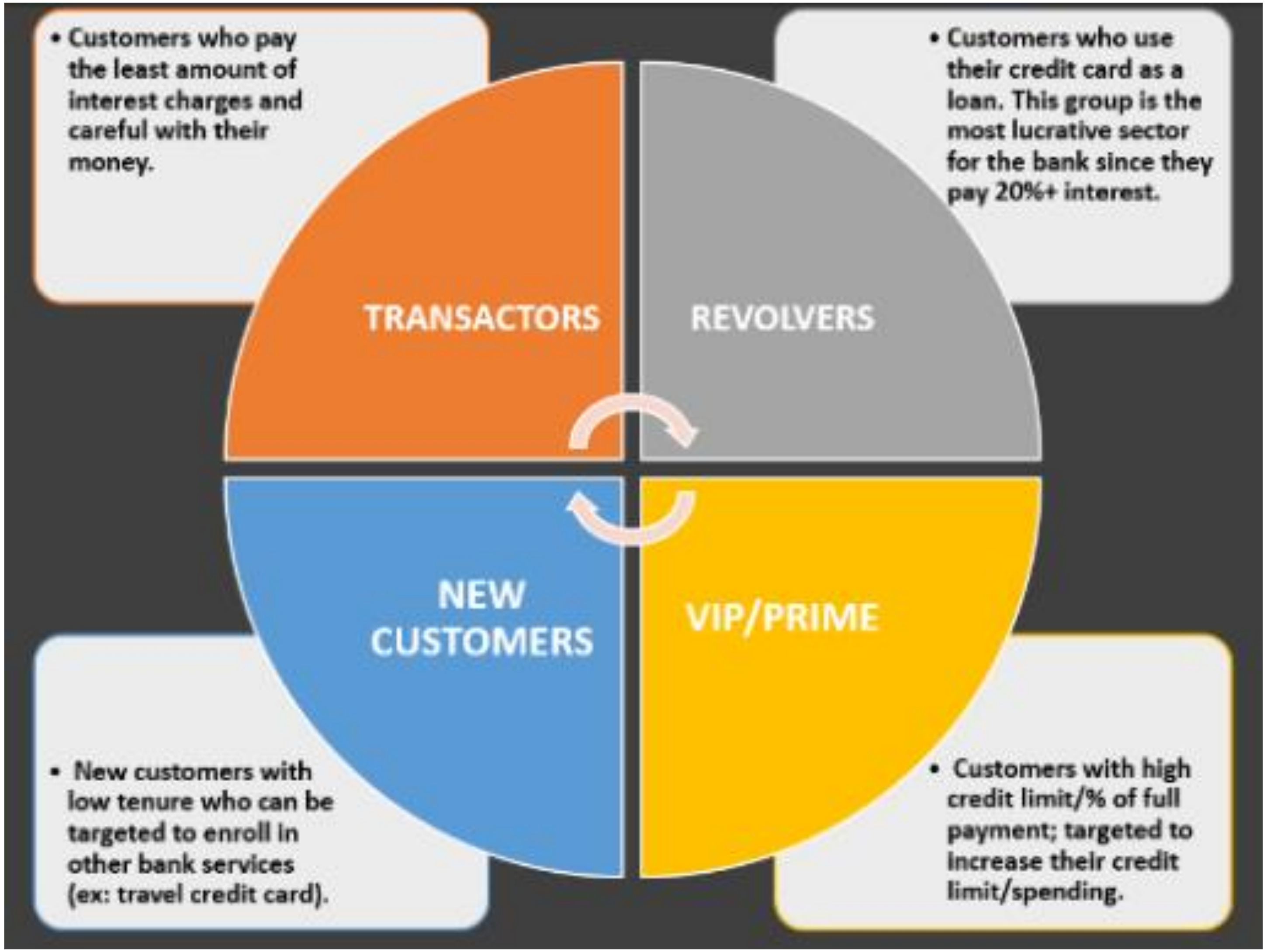
File Edit View Insert Cell Kernel Widgets Help

Customize and control Google Chro

TASK #1: UNDERSTAND THE PROBLEM STATEMENT AND BUSINESS CASE

- In this project, you have been hired as a data scientist at a bank and you have been provided with extensive data on the bank's customers for the past 6 months.
- Data includes transactions frequency, amount, tenure..etc.
- The bank marketing team would like to leverage AI/ML to launch a targeted marketing ad campaign that is tailored to specific group of customers.
- In order for this campaign to be successful, the bank has to divide its customers into at least 3 distinctive groups.
- This process is known as "marketing segmentation" and it crucial for maximizing marketing campaign conversion rate.

- Data Source: https://www.kaggle.com/arjunbhasin2013/ccdata
- Photo Credit: https://www.needpix.com/photo/1011172/marketing-customer-polaroid-center-presentation-online-board-target-economy


• Customers who pay the least amount of interest charges and careful with their money.
• Customers who use their credit card as a loan. This group is the most lucrative sector for the bank since they pay 20%+ interest.
TRANSACTORS
REVOLVERS
NEW CUSTOMERS
VIP/PRIME
• New customers with low tenure who can be targeted to enroll in other bank services (ex: travel credit card).
• Customers with high credit limit/% of full payment; targeted to increase their credit limit/spending.

TASK #2: IMPORT LIBRARIES AND DATASETS

In [ ]:
```python
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.decomposition import PCA
from jupyterthemes import jtplot
jtplot.style(theme='monokai', context='notebook', ticks=True, grid=False)
# setting the style of the notebook to be monokai theme
# this line of code is important to ensure that we are able to see the x and y axes clearly
# If you don't run this code line, you will notice that the xlabel and ylabel on any plot is black on black
```

In [ ]:
```python
# You have to include the full link to the csv file containing your dataset
creditcard_df = pd.read_csv('Marketing_data.csv')

# CUSTID: Identification of Credit Card holder
# BALANCE: Balance amount left in customer's account to make purchases
# BALANCE_FREQUENCY: How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, (
# PURCHASES: Amount of purchases made from account
# ONEOFFPURCHASES: Maximum purchase amount done in one-go
```

```
# CUSTID: Identification of Credit Card holder
# BALANCE: Balance amount left in customer's account to make purchases
# BALANCE_FREQUENCY: How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, (
# PURCHASES: Amount of purchases made from account
# ONEOFFPURCHASES: Maximum purchase amount done in one-go
# INSTALLMENTS_PURCHASES: Amount of purchase done in installment
# CASH_ADVANCE: Cash in advance given by the user
# PURCHASES_FREQUENCY: How frequently the Purchases are being made, score between 0 and 1 (1 = frequently pu
# ONEOFF_PURCHASES_FREQUENCY: How frequently Purchases are happening in one-go (1 = frequently purchased, 0
# PURCHASES_INSTALLMENTS_FREQUENCY: How frequently purchases in installments are being done (1 = frequently
# 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 [3]:

```
creditcard_df
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CAS
0	C10001	40.900749	0.818182	95.40	0.00	95.40	0.000
1	C10002	3202.467416	0.909091	0.00	0.00	0.00	6442
2	C10003	2495.148862	1.000000	773.17	773.17	0.00	0.000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.00	205.7
4	C10005	817.714335	1.000000	16.00	16.00	0.00	0.000
...	...	...	...	...	...	...	...
8945	C19186	28.493517	1.000000	291.12	0.00	291.12	0.000
8946	C19187	19.183215	1.000000	300.00	0.00	300.00	0.000

```
8950 rows × 18 columns
```

```
In [4]:
creditcard_df.info()
# Let's apply info and get additional insights on our dataframe
# 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
```

MINI CHALLENGE #1:

- What is the average, minimum and maximum "BALANCE" amount?

In [8]:
```
print('Average, min, max =', creditcard_df['BALANCE'].mean(),
      creditcard_df['BALANCE'].min(), creditcard_df['BALANCE'].max())
```

```
Average, min, max = 1564.4748276781038 0.0 19043.13856
```

In [ ]:
```
# Let's apply describe() and get more statistical insights on our dataframe
# 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_FREQUENCY are gene
# Average credit limit ~ 4500
# Percent of full payment is 15%
# Average tenure is 11 years
```

```
In [7]: creditcard_df.describe()
        # Let's apply describe() and get more statistical insights on our dataframe
        # 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_FREQUENCY are gene
        # Average credit limit ~ 4500
        # Percent of full payment is 15%
        # Average tenure is 11 years
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVA
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760

MINI CHALLENGE #2:

- Obtain the features (row) of the customer who made the maximim "ONEOFF_PURCHASES"
- Obtain the features of the customer who made the maximum cash advance transaction? how many cash advance transactions did that customer make? how often did he/she pay their bill?

In [8]:
```
creditcard_df[creditcard_df['ONEOFF_PURCHASES'] == 40761.25]
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH
550	C10574	11547.52001	1.0	49039.57	40761.25	8278.32	558.16

In [10]:
```
creditcard_df['CASH_ADVANCE'].max()
```

```
47137.211760000006
```

In [12]:
```
creditcard_df[creditcard_df['CASH_ADVANCE'] == 47137.211760000006]
```

TALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS
	1.0	123	21	19600.0	39048.59762

In [12]:
```
creditcard_df[creditcard_df['CASH_ADVANCE'] == 47137.211760000006]
```

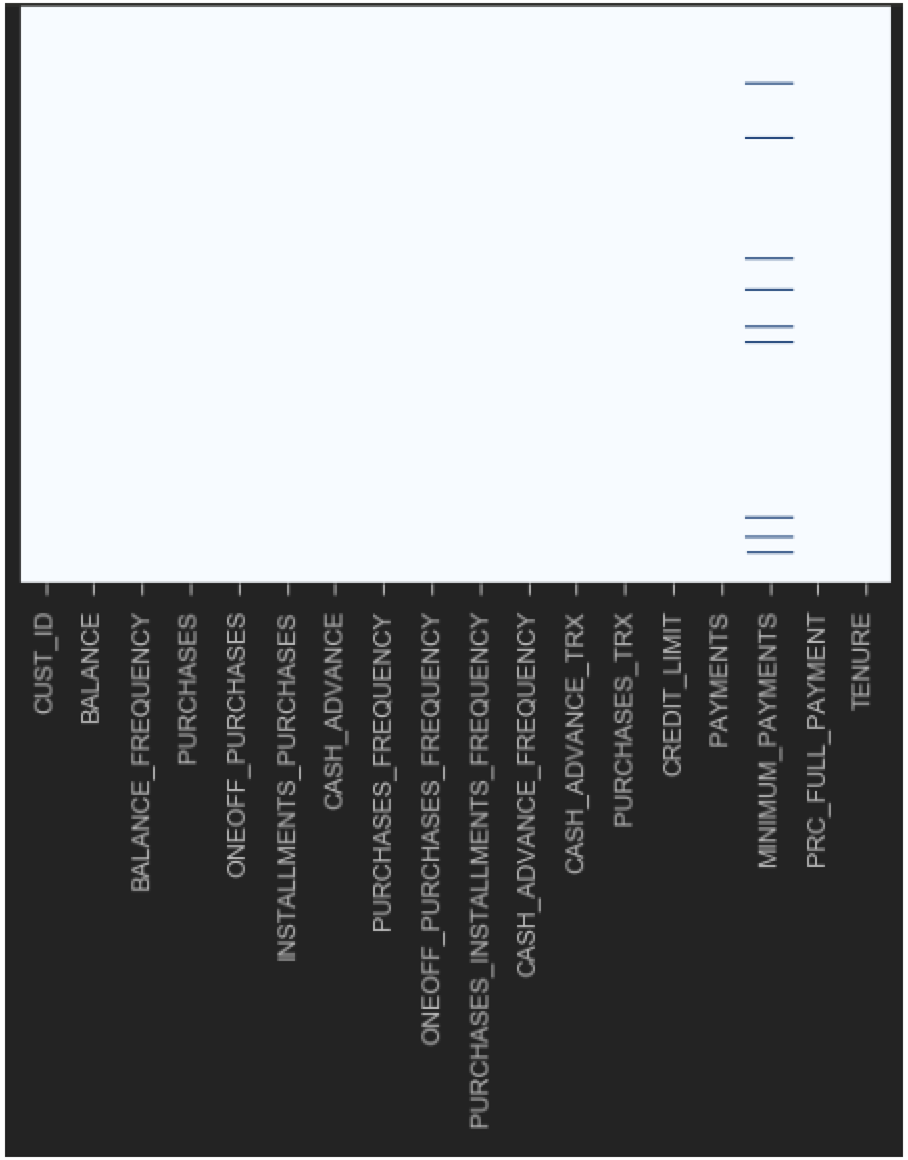
Y	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
	123	21	19600.0	39048.59762	5394.173671	0.0	12

TASK #3: VISUALIZE AND EXPLORE DATASET

In [ ]:

In [13]:
```python
# Let's see if we have any missing data, luckily we don't have many!
sns.heatmap(creditcard_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2628cdcb508>
```


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

```
In [14]: creditcard_df.isnull().sum()

CUST_ID                               0
BALANCE                               0
BALANCE_FREQUENCY                     0
PURCHASES                             0
ONEOFF_PURCHASES                      0
INSTALLMENTS_PURCHASES                0
CASH_ADVANCE                          0
PURCHASES_FREQUENCY                   0
ONEOFF_PURCHASES_FREQUENCY            0
PURCHASES_INSTALLMENTS_FREQUENCY      0
CASH_ADVANCE_FREQUENCY                0
CASH_ADVANCE_TRX                      0
PURCHASES_TRX                         0
CREDIT_LIMIT                          1
PAYMENTS                              0
MINIMUM_PAYMENTS                    313
PRC_FULL_PAYMENT                      0
TENURE                                0
dtype: int64
```

```
In [16]: # Fill up the missing elements with mean of the 'MINIMUM_PAYMENT'
creditcard_df.loc[(creditcard_df['MINIMUM_PAYMENTS'].isnull() == True),
                  'MINIMUM_PAYMENTS'] = creditcard_df['MINIMUM_PAYMENTS'].mean()
```

```
In [17]: creditcard_df.isnull().sum()
```

```
CUST_ID                             0
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
CREDIT_LIMIT                        1
PAYMENTS                            0
```

```
MINI CHALLENGE #3:
 - Fill out missing elements in the "CREDIT_LIMIT" column
 - Double check and make sure that no missing elements are present
```

```
In [18]: creditcard_df.loc[(creditcard_df['CREDIT_LIMIT'].isnull() == True),
                           'CREDIT_LIMIT'] = creditcard_df['CREDIT_LIMIT'].mean()
```

In [19]:

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

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

In [20]:

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

```
0
```

MINI CHALLENGE #4:
- Drop Customer ID column 'CUST_ID' and make sure that the column has been removed from the dataframe

In [21]:
```python
creditcard_df.drop('CUST_ID', axis = 1, inplace = True)
```

In [22]:
```python
creditcard_df
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANC
0	40.900749	0.818182	95.40	0.00	95.40	0.000000
1	3202.467416	0.909091	0.00	0.00	0.00	6442.945483
2	2495.148862	1.000000	773.17	773.17	0.00	0.000000
3	1666.670542	0.636364	1499.00	1499.00	0.00	205.788017
4	817.714335	1.000000	16.00	16.00	0.00	0.000000
...	...	...	...	...	...	...
8945	28.493517	1.000000	291.12	0.00	291.12	0.000000
8946	19.183215	1.000000	300.00	0.00	300.00	0.000000
8947	23.398673	0.833333	144.40	0.00	144.40	0.000000

8947	23.398673	0.833333	144.40	0.00	144.40	0.000000
8948	13.457564	0.833333	0.00	0.00	0.00	36.558778
8949	372.708075	0.666667	1093.25	1093.25	0.00	127.040008

8950 rows × 17 columns

In [23]:

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

```
17
```

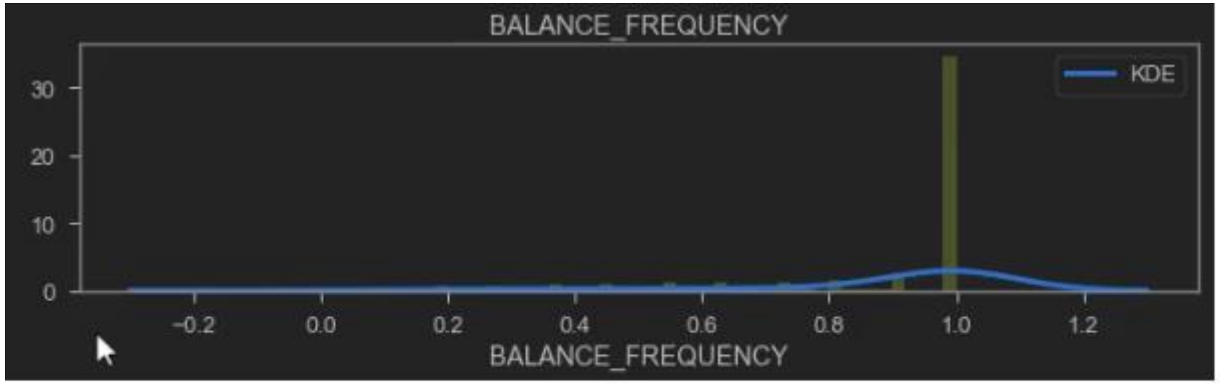
In [24]:

```
creditcard_df.columns
```

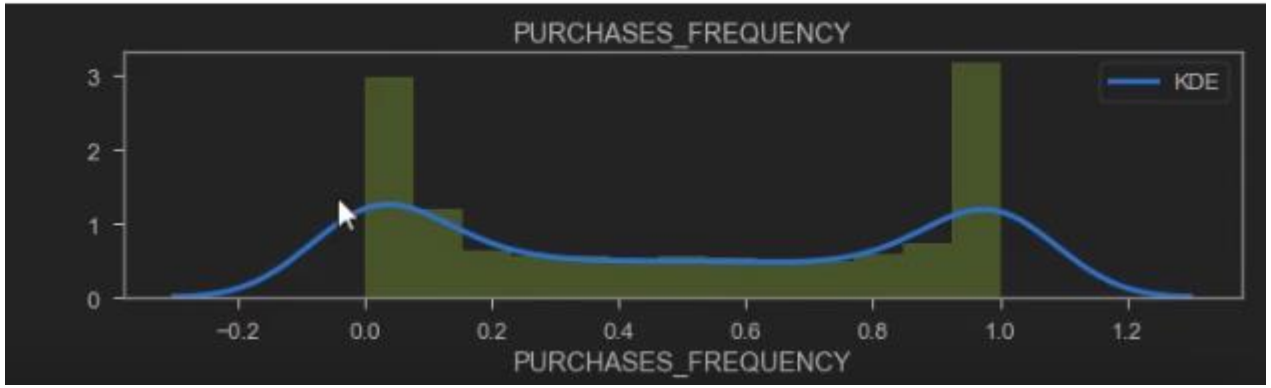
```
Index(['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES',
       'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FREQUENCY',
       'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY',
       'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX',
       'CREDIT_LIMIT',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

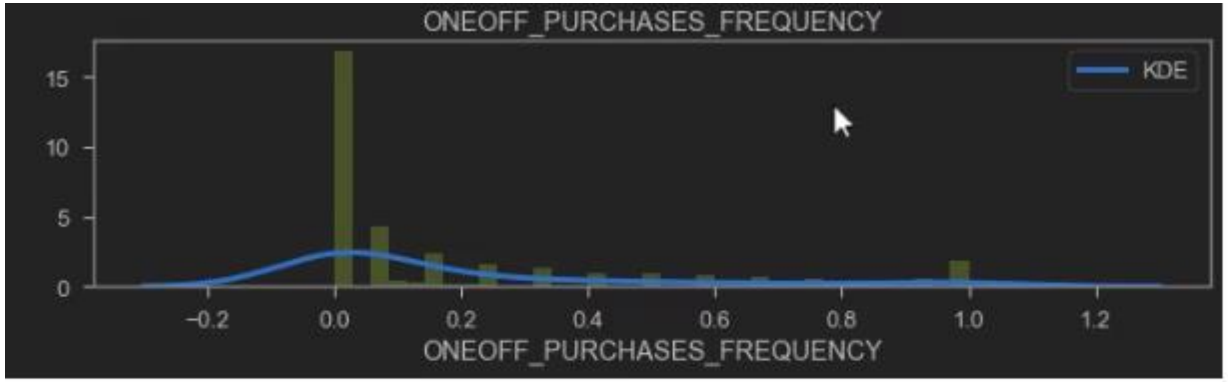
```
In [ ]: # distplot combines the matplotlib.hist function with seaborn kdeplot()
        # KDE Plot represents the Kernel Density Estimate
        # KDE is used for visualizing the Probability Density of a continuous variable.
        # KDE demonstrates the probability density at different values 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't do one off puchas
        # 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
```

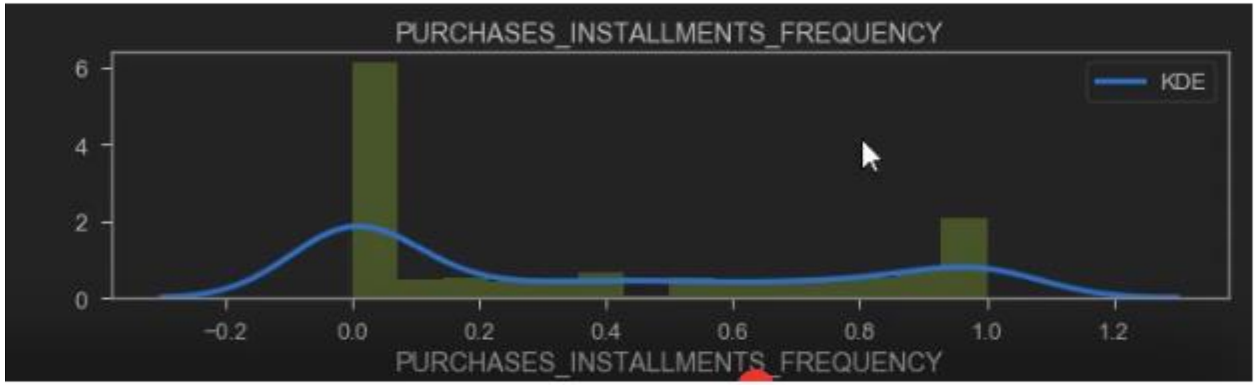
```
plt.figure(figsize=(10,50))
for i in range(len(creditcard_df.columns)):
    plt.subplot(17, 1, i+1)
    sns.distplot(creditcard_df[creditcard_df.columns[i]],
                 kde_kws={"color": "b", "lw": 3, "label": "KDE"}, hist_kws={"color": "g"})
    plt.title(creditcard_df.columns[i])

plt.tight_layout()
```

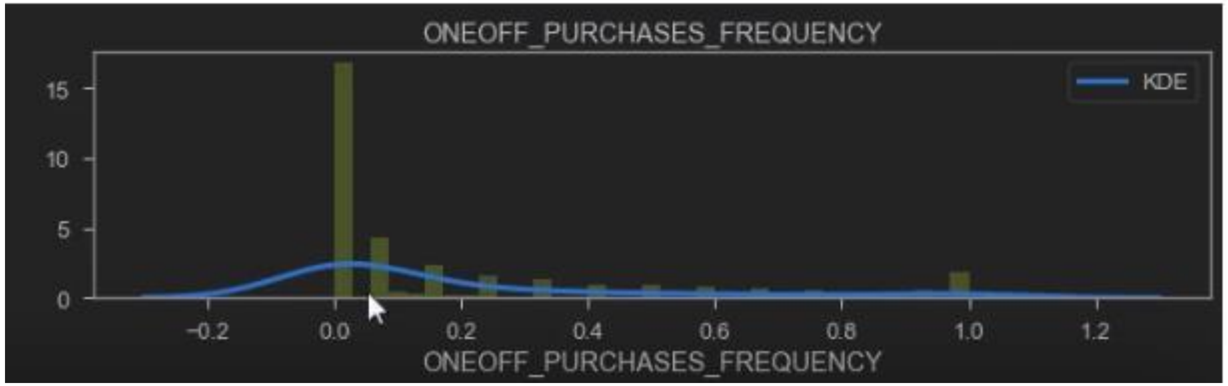

BALANCE_FREQUENCY
KDE
30
20
10
0
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
BALANCE_FREQUENCY

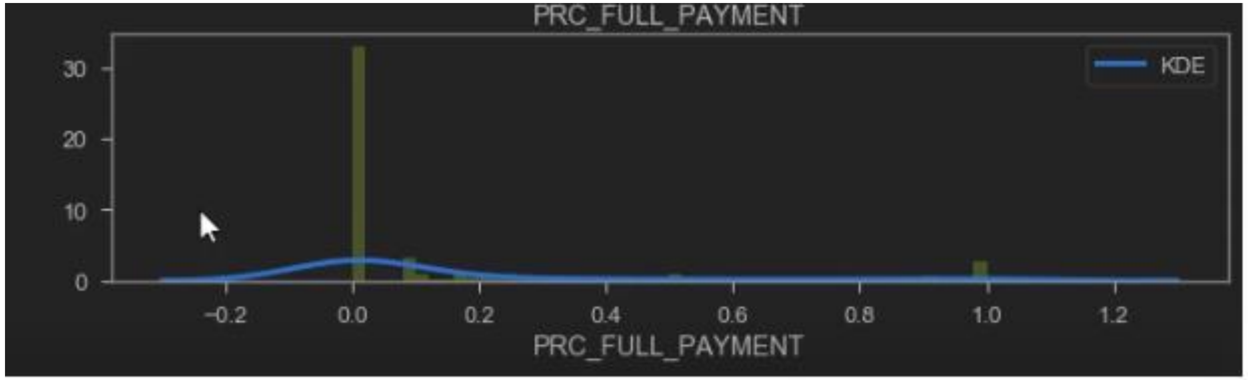

PURCHASES
KDE
0.0006
0.0004
0.0002
0.0000
0
10000
20000
30000
40000
50000
PURCHASES

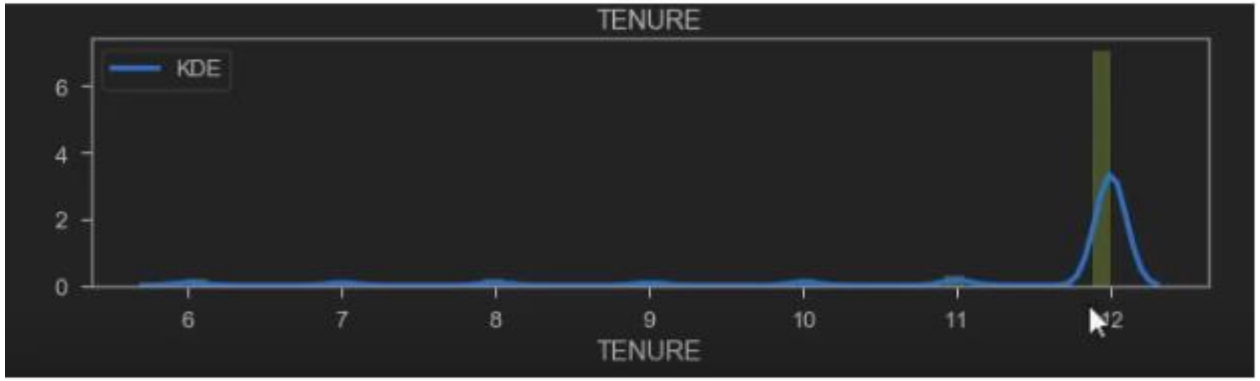

PURCHASES_FREQUENCY
KDE
3
2
1
0
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
PURCHASES_FREQUENCY


ONEOFF_PURCHASES_FREQUENCY
KDE
15
10
5
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
ONEOFF_PURCHASES_FREQUENCY


PURCHASES_INSTALLMENTS_FREQUENCY
KDE
6
4
2
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
PURCHASES_INSTALLMENTS_FREQUENCY


ONEOFF_PURCHASES_FREQUENCY
KDE
15
10
5
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
ONEOFF_PURCHASES_FREQUENCY


PRC_FULL_PAYMENT
KDE
30
20
10
0
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
PRC_FULL_PAYMENT

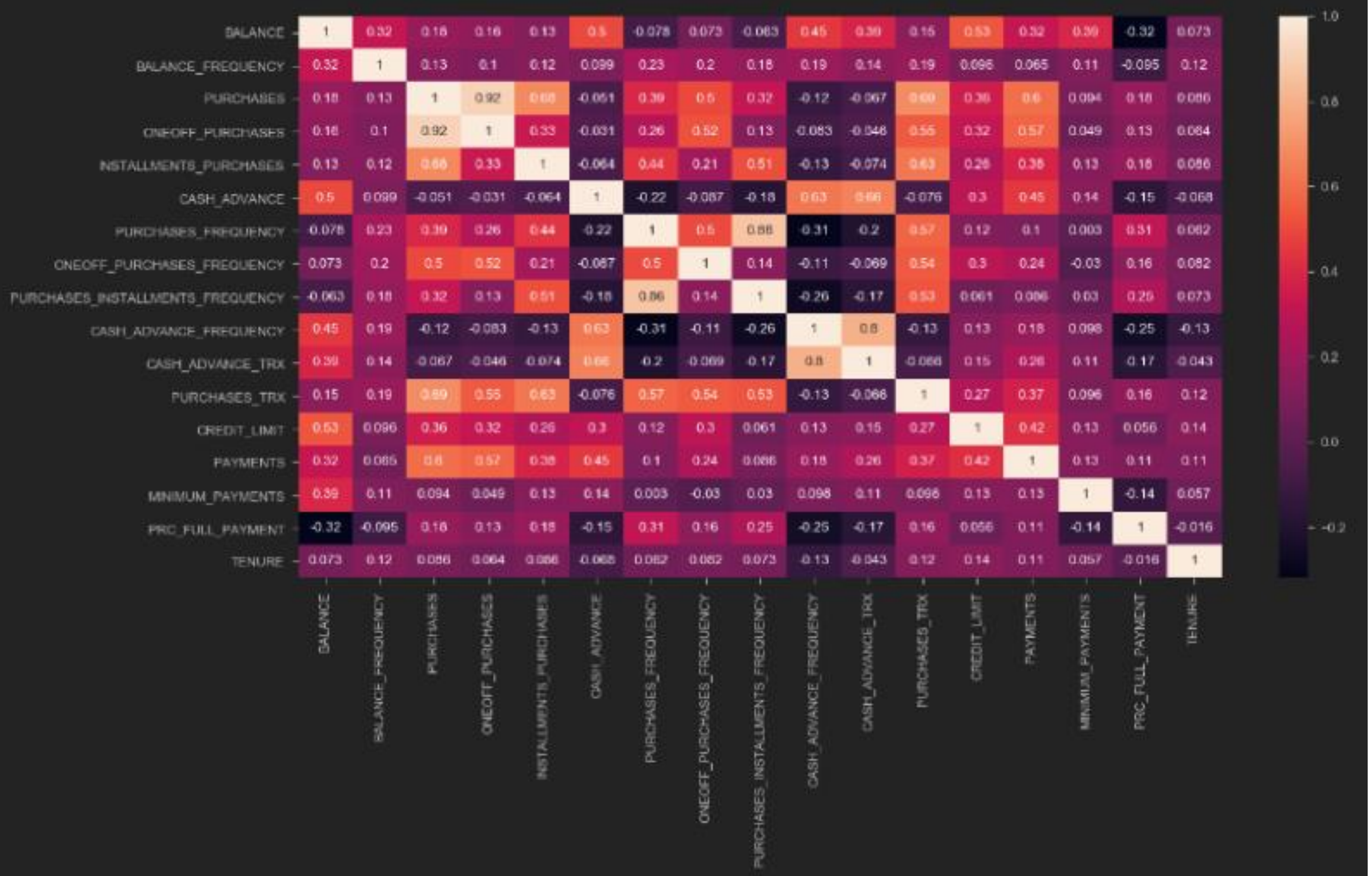

TENURE
KDE
6
4
2
0
6
7
8
9
10
11
12
TENURE

MINI CHALLENGE #5:

- Obtain the correlation matrix between features

In [37]:
```
correlations = creditcard_df.corr()
f, ax = plt.subplots(figsize = (20,10))
sns.heatmap(correlations, annot = True)
```

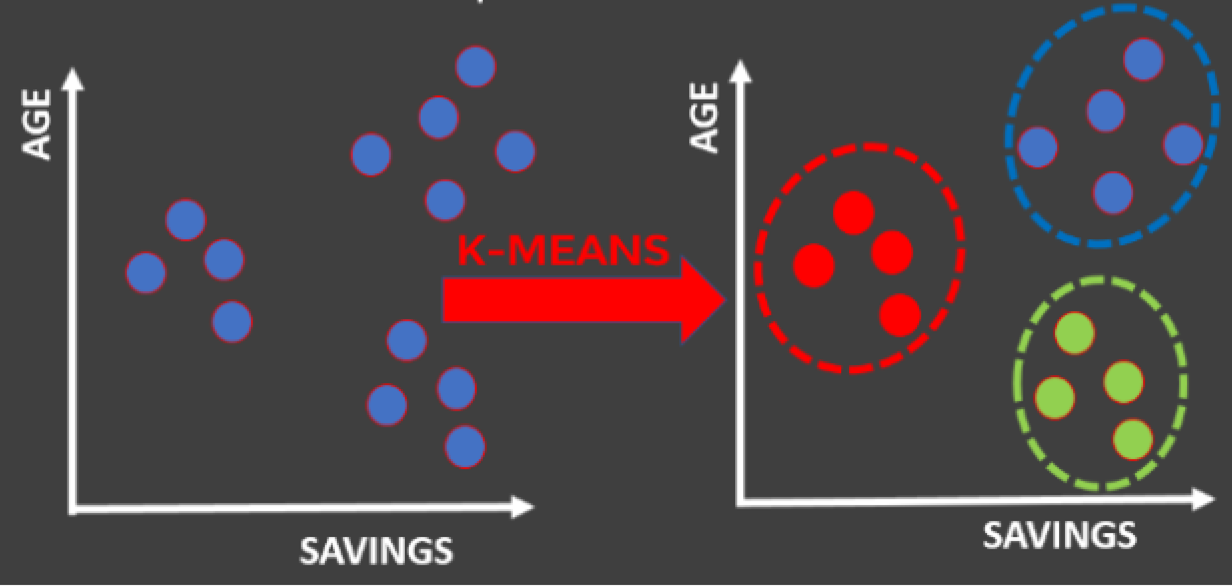
<matplotlib.axes._subplots.AxesSubplot at 0x2628e630c48>



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

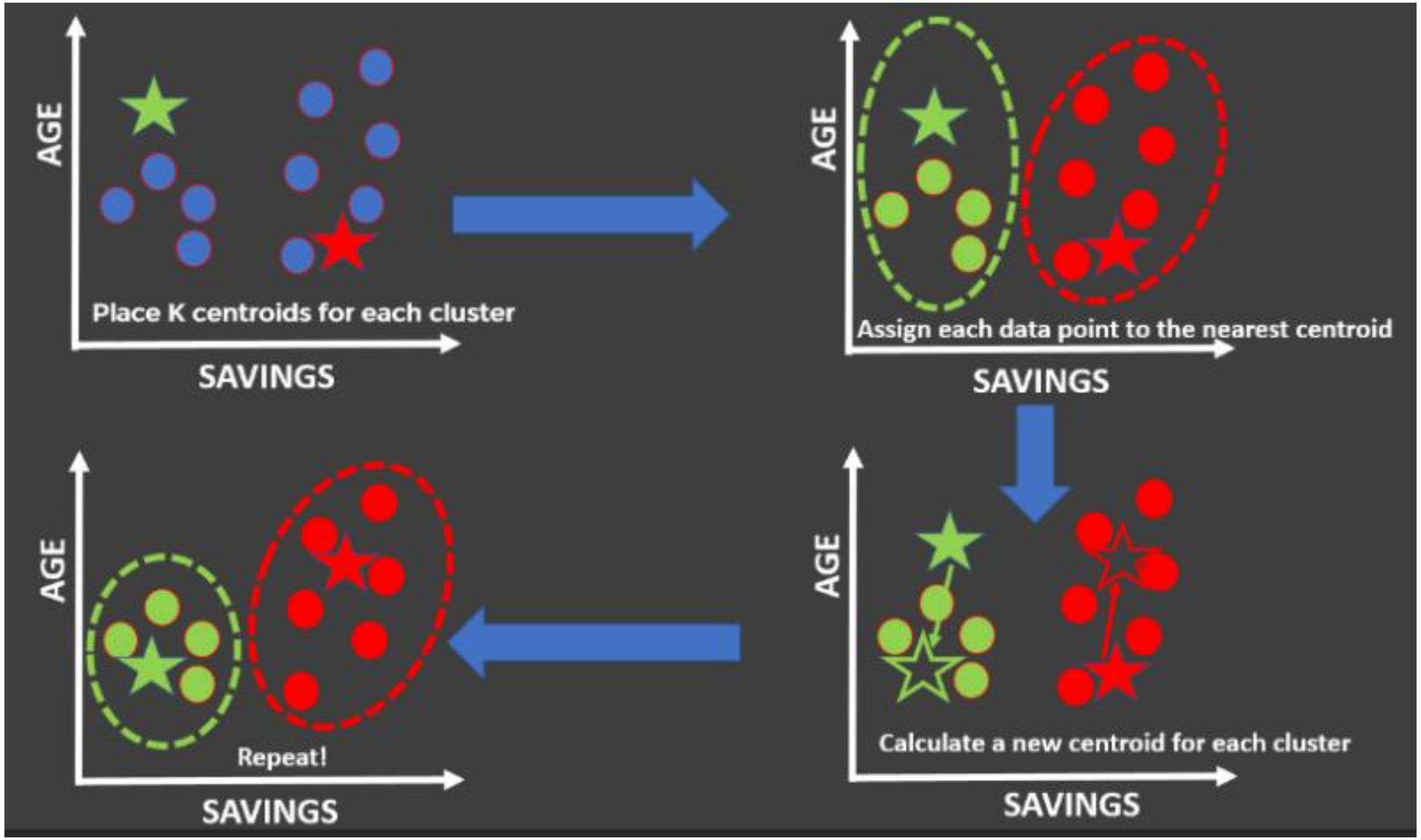
K-MEANS INTUITION

- K-means is an unsupervised learning algorithm (clustering).
- K-means works by grouping some data points together (clustering) in an unsupervised fashion.
- The algorithm groups observations with similar attribute values together by measuring the Euclidian distance between points.



K-MEANS ALGORITHM STEPS

1. Choose number of clusters “K”
2. Select random K points that are going to be the centroids for each cluster
3. Assign each data point to the nearest centroid, doing so will enable us to create “K” number of clusters
4. Calculate a new centroid for each cluster
5. Reassign each data point to the new closest centroid
6. Go to step 4 and repeat.


AGE
Place K centroids for each cluster
SAVINGS
AGE
Assign each data point to the nearest centroid
SAVINGS
AGE
Calculate a new centroid for each cluster
SAVINGS
AGE
Repeat!
SAVINGS

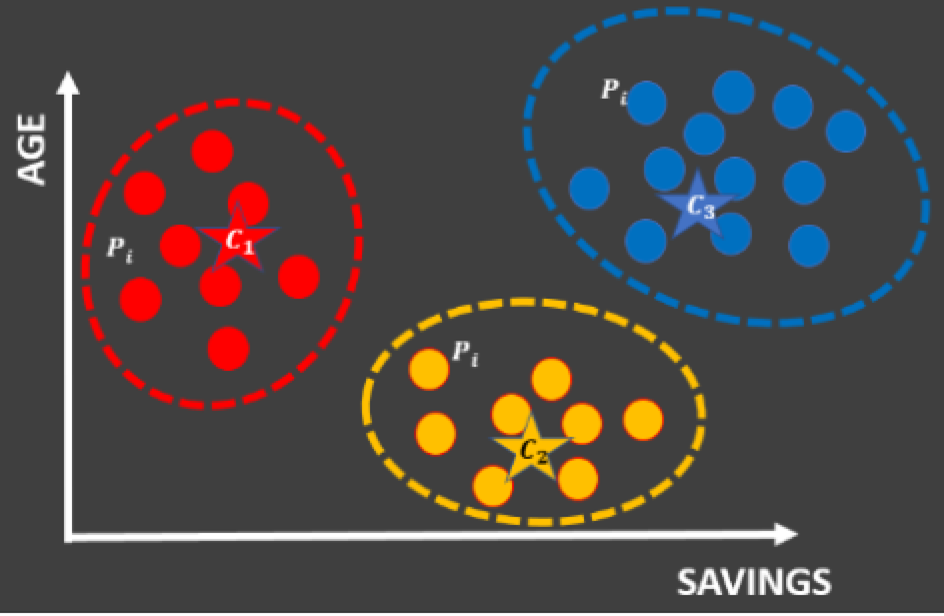
MINI CHALLENGE #6:

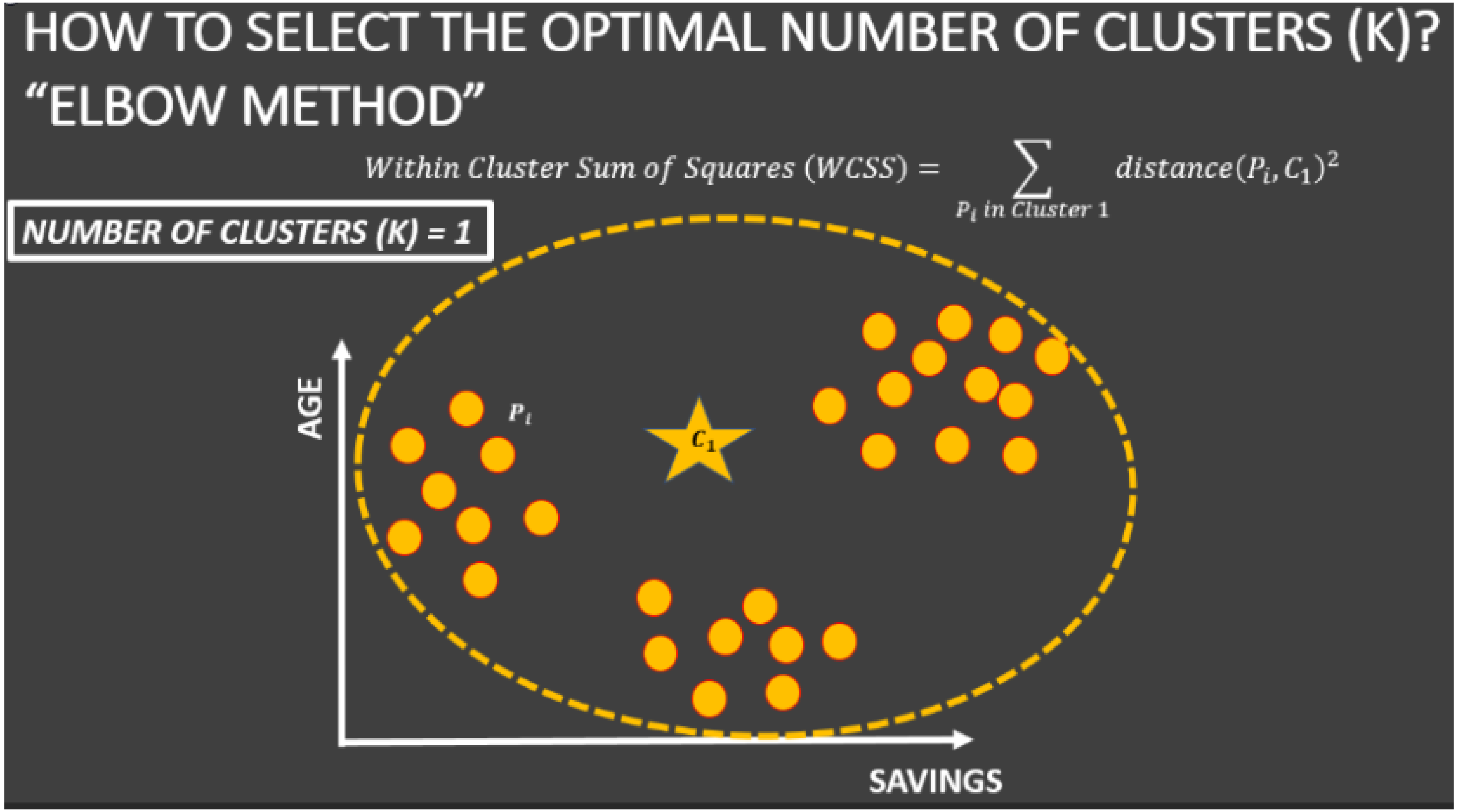
- Which of the following conditions could terminate the K-means clustering algorithm? (choose 2)
    - K-means terminates after a fixed number of iterations is reached (correct)
    - K-means terminates when the number of clusters does not increase between iterations (wrong)
    - K-means terminates when the centroid locations do not change between iterations (correct)

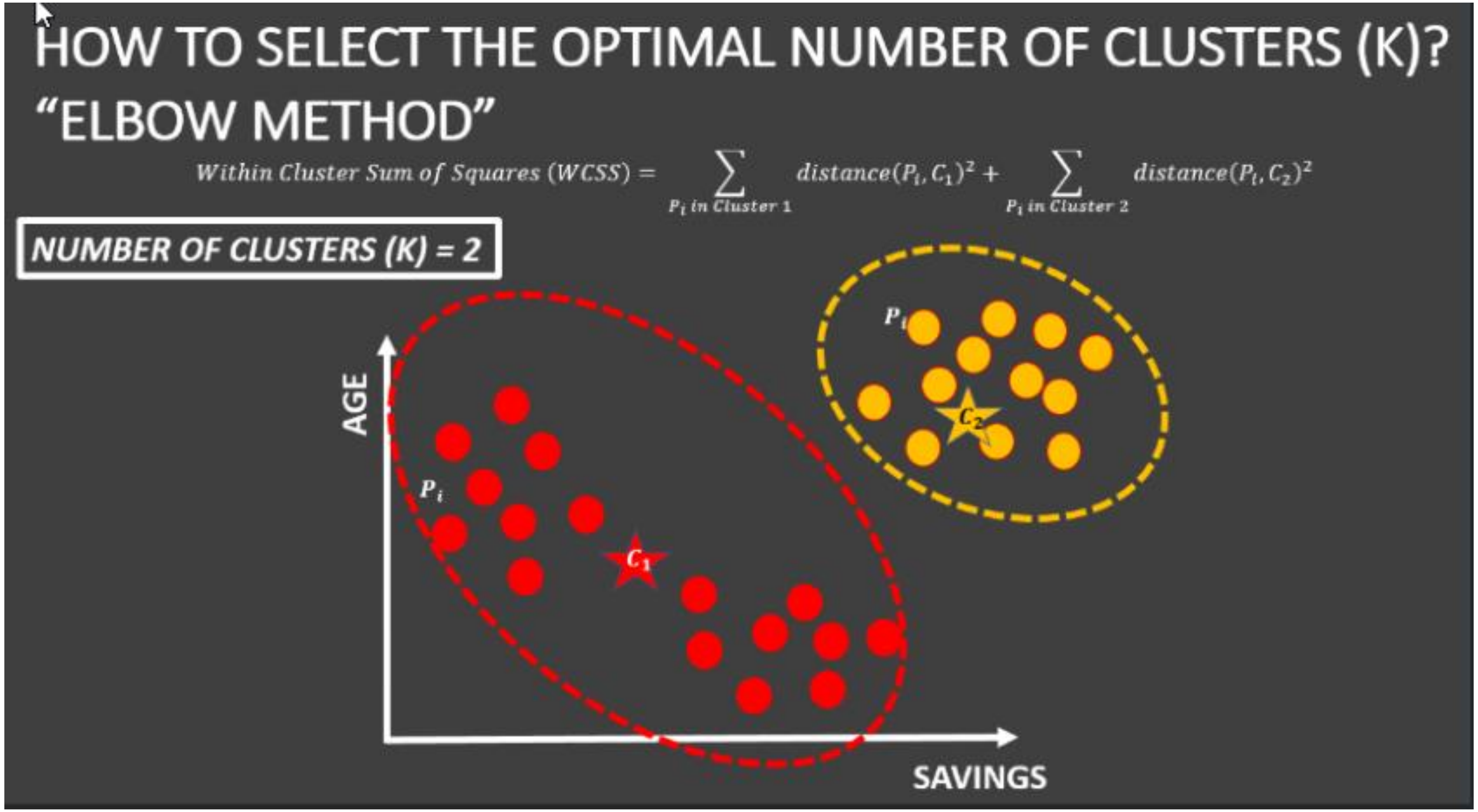
TASK #5: LEARN HOW TO OBTAIN THE OPTIMAL NUMBER OF CLUSTERS (ELBOW METHOD)

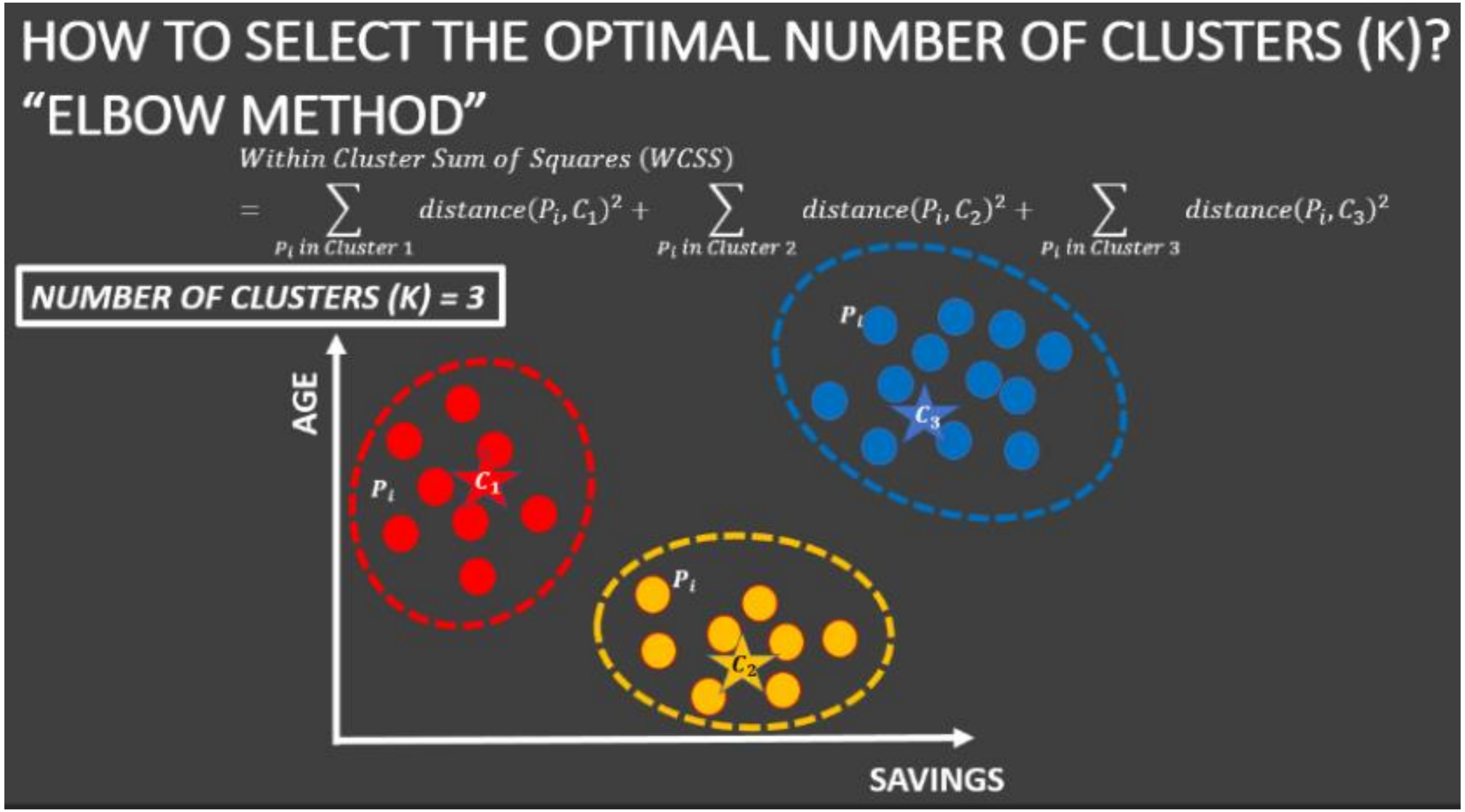
HOW TO SELECT THE OPTIMAL NUMBER OF CLUSTERS (K)? "ELBOW METHOD"

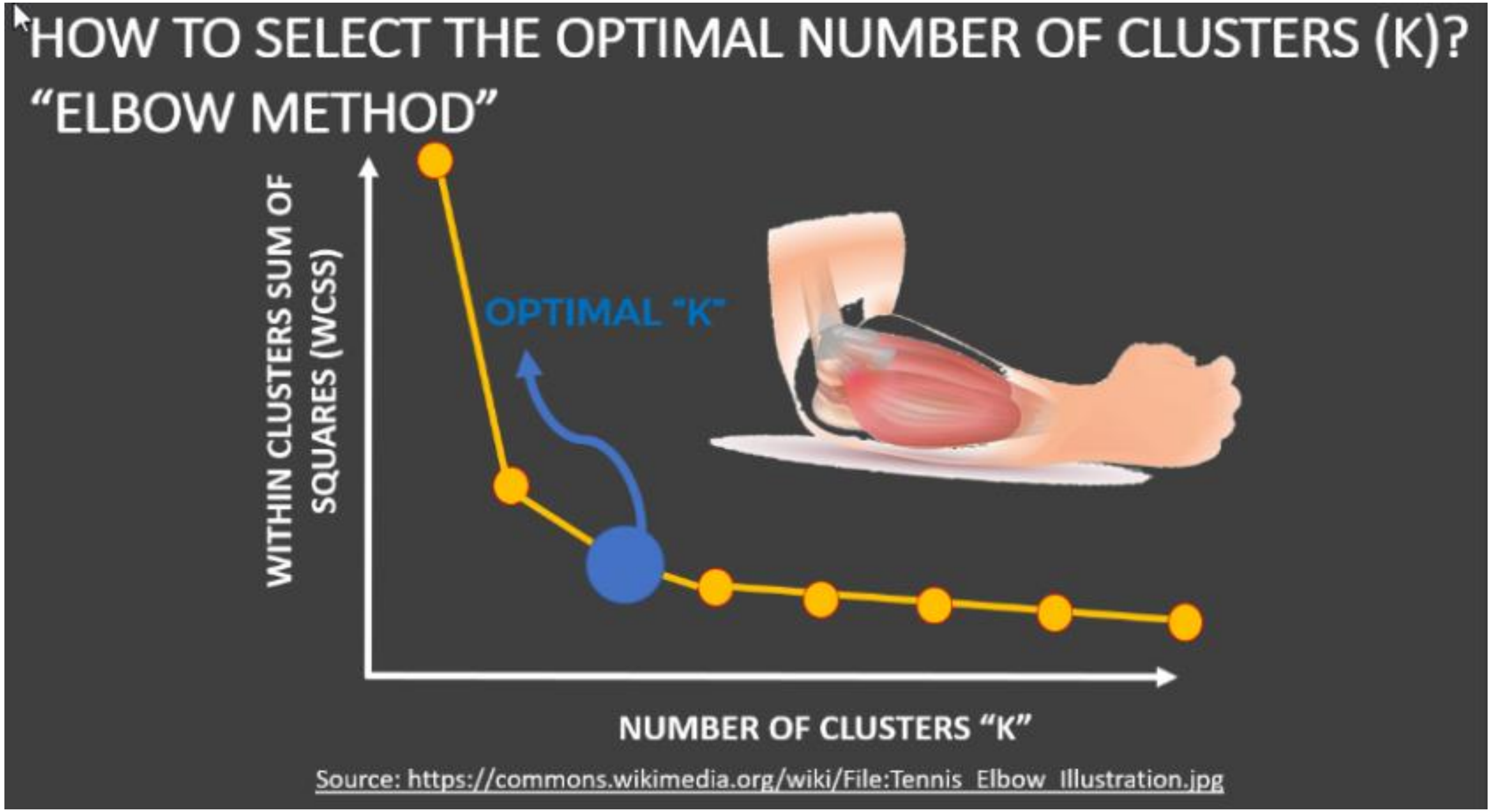
$$\textit{Within Cluster Sum of Squares } (WCSS)$$
$$= \sum_{P_i \text{ in Cluster } 1} distance(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster } 2} distance(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster } 3} distance(P_i, C_3)^2$$




HOW TO SELECT THE OPTIMAL NUMBER OF CLUSTERS (K)?
"ELBOW METHOD"
$Within\ Cluster\ Sum\ of\ Squares\ (WCSS) = \sum_{P_i\ in\ Cluster\ 1} distance(P_i, C_1)^2$
NUMBER OF CLUSTERS (K) = 1
AGE
P_i
C_1
SAVINGS


HOW TO SELECT THE OPTIMAL NUMBER OF CLUSTERS (K)?
"ELBOW METHOD"
$Within\ Cluster\ Sum\ of\ Squares\ (WCSS) = \sum_{P_i\ in\ Cluster\ 1} distance(P_i, C_1)^2 + \sum_{P_i\ in\ Cluster\ 2} distance(P_i, C_2)^2$
NUMBER OF CLUSTERS (K) = 2
AGE
P_i
C_1
P_i
C_2
SAVINGS


HOW TO SELECT THE OPTIMAL NUMBER OF CLUSTERS (K)?
"ELBOW METHOD"
Within Cluster Sum of Squares (WCSS)
$= \sum_{P_i \text{ in Cluster 1}} distance(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster 2}} distance(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster 3}} distance(P_i, C_3)^2$
NUMBER OF CLUSTERS (K) = 3
AGE
SAVINGS
P_i
C_1
P_i
C_2
P_i
C_3


HOW TO SELECT THE OPTIMAL NUMBER OF CLUSTERS (K)?
"ELBOW METHOD"
WITHIN CLUSTERS SUM OF SQUARES (WCSS)
OPTIMAL "K"
NUMBER OF CLUSTERS "K"
Source: https://commons.wikimedia.org/wiki/File:Tennis_Elbow_Illustration.jpg

TASK #6: 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://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/

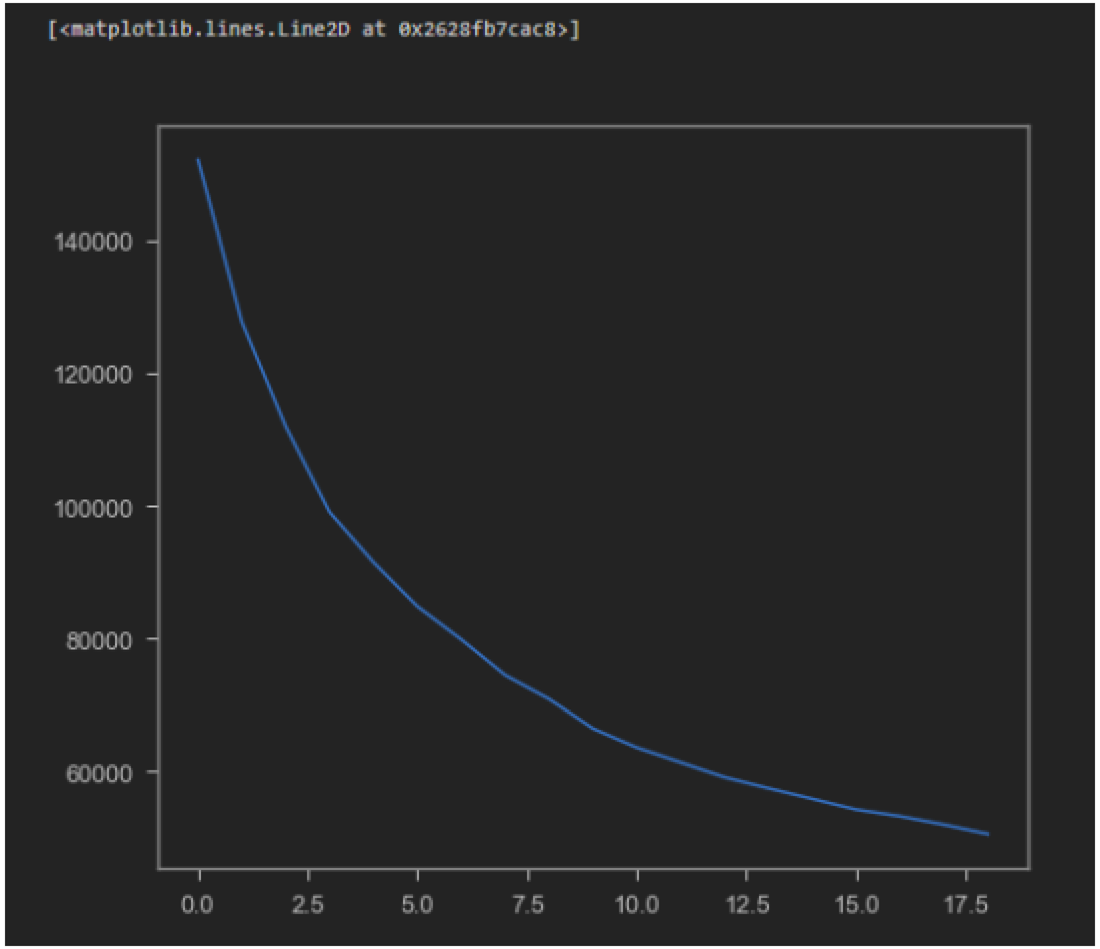
In [38]:
```
# Let's scale the data first
scaler = StandardScaler()
creditcard_df_scaled = scaler.fit_transform(creditcard_df)
```

In [39]:
```
creditcard_df_scaled.shape
```

```
(8950, 17)
```

```
In [40]: creditcard_df_scaled

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



        # 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 or 8.
```

```
In [42]:
# Index(['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES',
#        'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FREQUENCY',
#        'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY',
#        'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX',
#        'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT',
#        'TENURE'], dtype='object')
scores_1 = []
range_values = range(1,20)
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-')
# 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 or 8.

[<matplotlib.lines.Line2D at 0x2628fb7cac8>]
```


[<matplotlib.lines.Line2D at 0x2628fb7cac8>]
140000
120000
100000
80000
60000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

MINI CHALLENGE #7:

- Let's assume that our data only consists of the first 7 columns of "creditcard_df_scaled", what is the optimal number of clusters would be in this case? modify the code and rerun the cells.

In [43]:

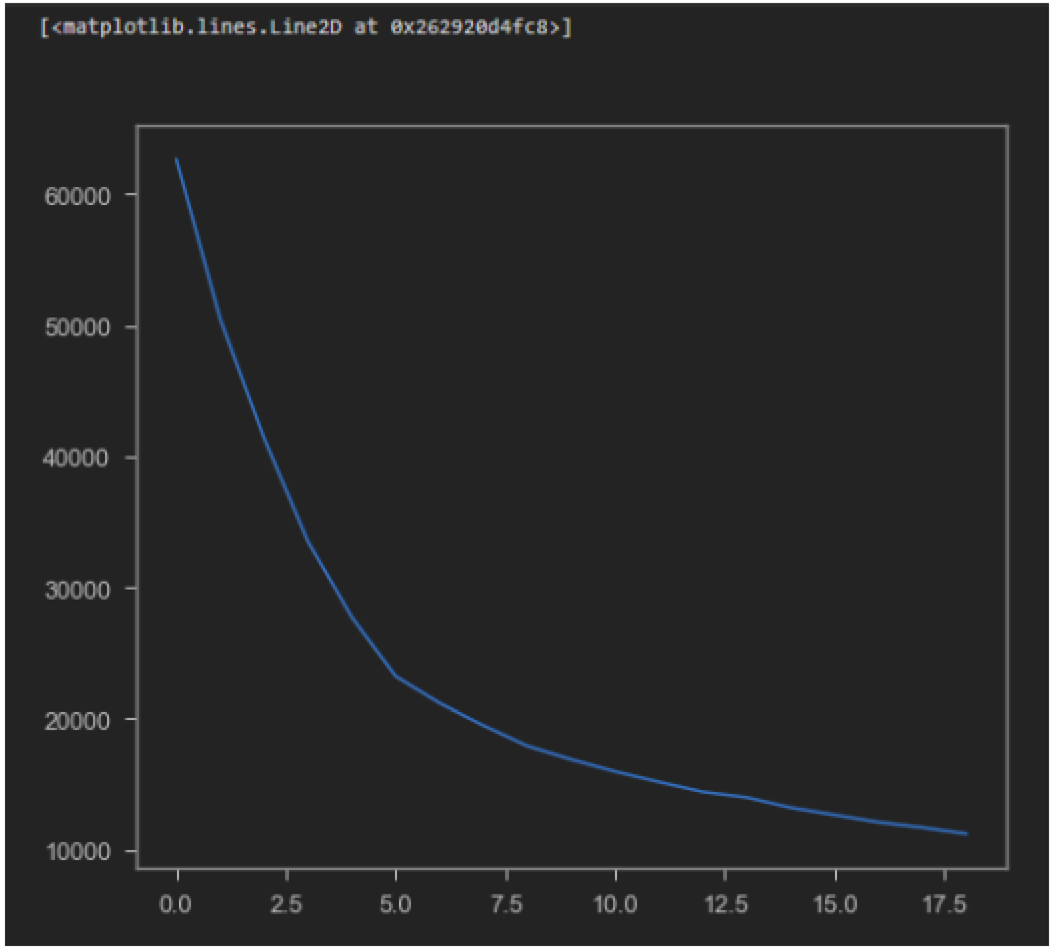
```
creditcard_df_scaled[:, :7].shape
```

```
(8950, 7)
```

In [44]:

```
scores_1 = []
range_values = range(1,20)
for i in range_values:
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(creditcard_df_scaled[:, :7])
    scores_1.append(kmeans.inertia_)

plt.plot(scores_1, 'bx-')
```

```
[<matplotlib.lines.Line2D at 0x262920d4fc8>]
```


[<matplotlib.lines.Line2D at 0x262920d4fc8>]
60000
50000
40000
30000
20000
10000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

TASK #7: APPLY K-MEANS METHOD

In [45]:

```
kmeans = KMeans(7)
kmeans.fit(creditcard_df_scaled)
labels = kmeans.labels_ # Labels (cluster) associated to each data point
```

In [46]:

```
kmeans.cluster_centers_.shape
```

```
(7, 17)
```

In [47]:

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

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
0	1.666473	0.392099	-0.205327	-0.149913	-0.210162	1.990753
1	-0.701872	-2.134325	-0.306924	-0.230292	-0.302515	-0.323078
2	-0.367555	0.330562	-0.039840	-0.234950	0.337266	-0.368099
3	-0.335506	-0.348076	-0.284525	-0.208973	-0.288475	0.065539
4	0.007813	0.402983	-0.343915	-0.225214	-0.399316	-0.104212
5	0.126801	0.429730	0.939029	0.895888	0.574411	-0.309125
6	1.430238	0.419467	6.915048	6.083034	5.172266	0.038778

```
In [48]:
# 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
# Second customers cluster (revolvers) who use credit card as a loan (most lucrative sector): highest balan
# Third customer cluster (VIP/Prime): high credit limit $16K and highest percentage of full payment, target
# Fourth customer cluster (low tenure): these are customers with low tenure (7 years), low balance
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
0	5033.096672	0.970155	564.520447	343.613277	221.020895	5153.573564
1	103.587241	0.371669	347.456361	210.199629	137.506773	301.361116
2	799.439665	0.955578	918.085545	202.468866	716.053491	206.951098
3	866.148306	0.794815	395.311749	245.585564	150.203132	1116.308792
4	1580.736068	0.972734	268.425678	218.628807	49.971063	760.334088
5	1828.399674	0.979070	3009.455142	2079.427650	930.500678	330.620880
6	4541.393882	0.976638	15777.311395	10689.027791	5088.283605	1060.190695

```
In [49]: labels.shape # Labels associated to each data point

(8950,)

In [50]: labels.max()

6

In [51]: labels.min()

0

In [52]: y_kmeans = kmeans.fit_predict(creditcard_df_scaled)
         y_kmeans

array([2, 6, 5, ..., 3, 3, 3])
```

```python
In [53]: # concatenate the clusters labels to our original dataframe
creditcard_df_cluster = pd.concat([creditcard_df, pd.DataFrame({'cluster':labels})], axis = 1)
creditcard_df_cluster.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
0	40.900749	0.818182	95.40	0.00	95.4	0.000000
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017
4	817.714335	1.000000	16.00	16.00	0.0	0.000000

PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CA
0.307019	0.145697	0.203860	0.5
0.270757	0.074792	0.188993	0.0
0.883916	0.095855	0.830697	0.0
0.410589	0.121144	0.272729	0.1
0.165154	0.102107	0.065464	0.1
0.928932	0.761065	0.576983	0.0
0.928101	0.763090	0.781501	0.0

CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_P
0.517064	16.365772	8.709172	8160.463697	4149.870001	2152.885952
0.030637	0.677338	4.355518	3865.955724	1149.108580	263.988609
0.039554	0.775550	19.032274	3482.112568	1091.026929	827.537625
0.196000	3.233704	5.125596	2468.226470	602.104087	376.247870
0.152124	2.964097	3.161211	3399.161094	1012.580763	827.406808
0.053020	1.059937	44.458991	7047.418985	2847.821449	730.293998
0.085271	2.988372	130.197674	12493.023256	15581.496801	3383.304083

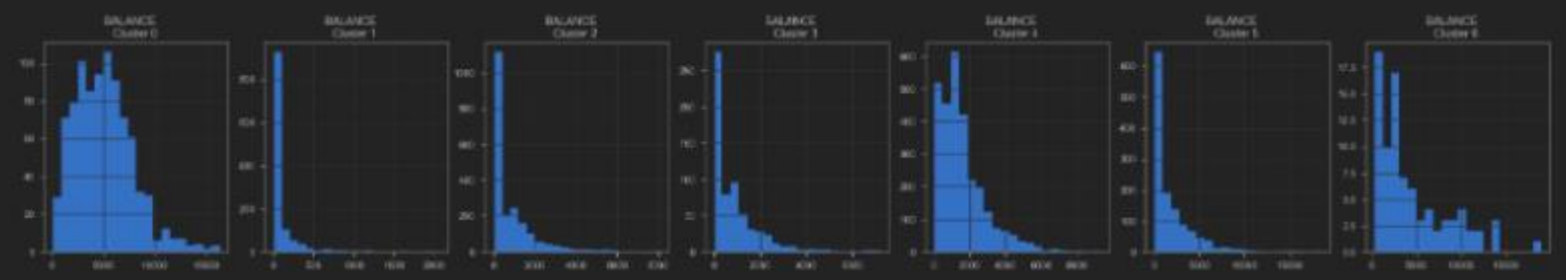
MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
2152.885952	0.039307	11.610738
263.988609	0.240000	11.786015
827.537625	0.243576	11.854768
376.247870	0.157487	7.243243
827.406808	0.021266	11.881732
730.293998	0.287217	11.929022
3383.304083	0.394721	11.965116

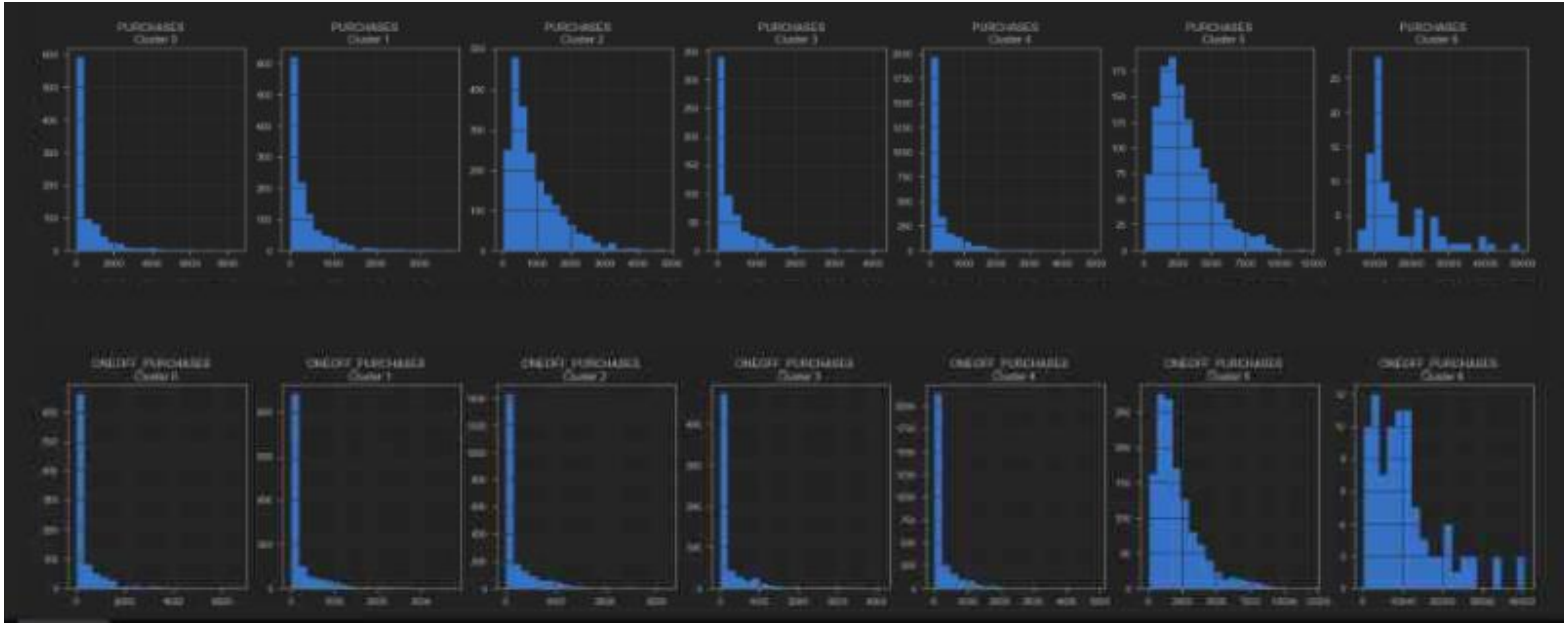
```python
In [53]: # concatenate the clusters labels to our original dataframe
creditcard_df_cluster = pd.concat([creditcard_df, pd.DataFrame({'cluster':labels})], axis = 1)
creditcard_df_cluster.head()
```

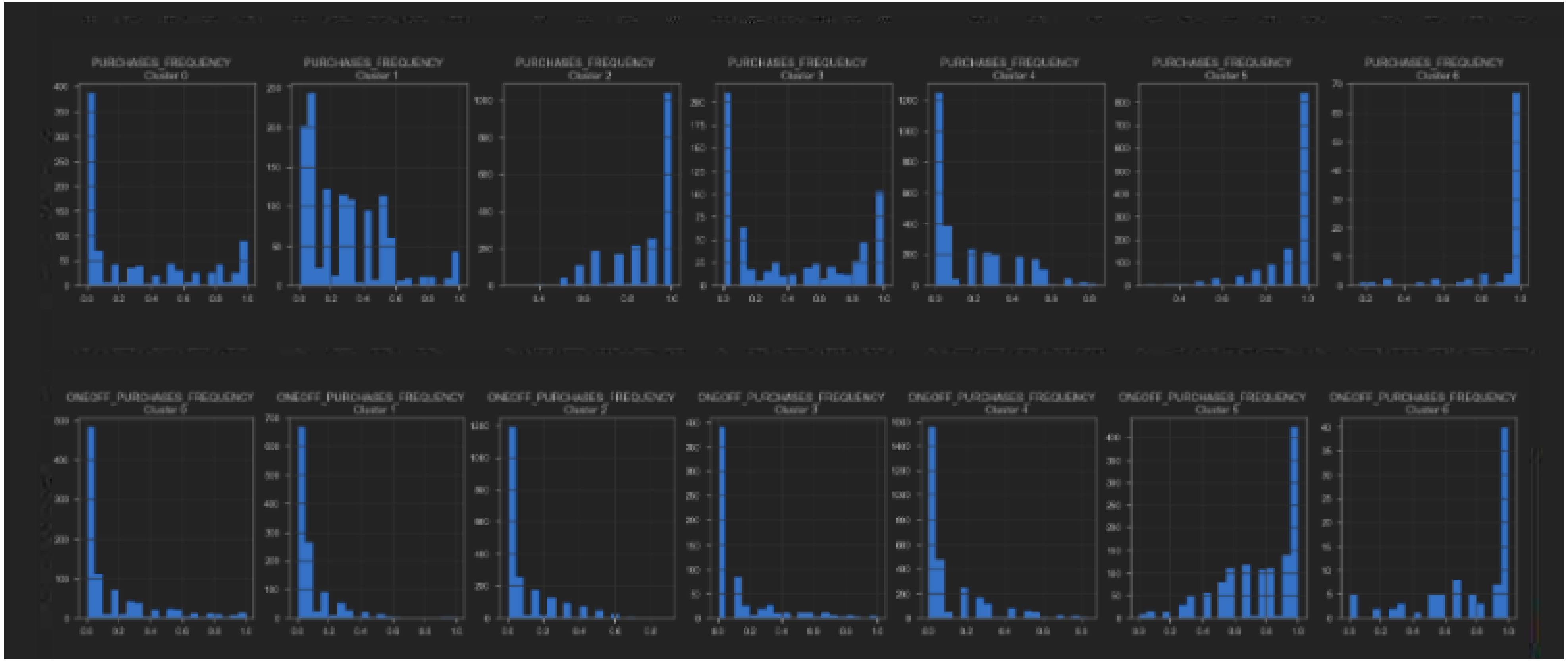
I_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE	cluster
	2	1000.0	201.802084	139.509787	0.000000	12	4
	0	7000.0	4103.032597	1072.340217	0.222222	12	0
	12	7500.0	622.066742	627.284787	0.000000	12	5
	1	7500.0	0.000000	864.206542	0.000000	12	4
	1	1200.0	678.334763	244.791237	0.000000	12	4

In [54]:

```
# Plot the histogram of various clusters
for i in creditcard_df.columns:
  plt.figure(figsize = (35, 5))
  for j in range(7):
    plt.subplot(1,7,j+1)
    cluster = creditcard_df_cluster[creditcard_df_cluster['cluster'] == j]
    cluster[i].hist(bins = 20)
    plt.title('{}    \nCluster {} '.format(i,j))

plt.show()
```






PURCHASES_FREQUENCY
Cluster 0
PURCHASES_FREQUENCY
Cluster 1
PURCHASES_FREQUENCY
Cluster 2
PURCHASES_FREQUENCY
Cluster 3
PURCHASES_FREQUENCY
Cluster 4
PURCHASES_FREQUENCY
Cluster 5
PURCHASES_FREQUENCY
Cluster 6
ONEOFF_PURCHASES_FREQUENCY
Cluster 0
ONEOFF_PURCHASES_FREQUENCY
Cluster 1
ONEOFF_PURCHASES_FREQUENCY
Cluster 2
ONEOFF_PURCHASES_FREQUENCY
Cluster 3
ONEOFF_PURCHASES_FREQUENCY
Cluster 4
ONEOFF_PURCHASES_FREQUENCY
Cluster 5
ONEOFF_PURCHASES_FREQUENCY
Cluster 6

```
MINI CHALLENGE #8:
 - Repeat the same procedure with 8 or 5 or 4 clusters instead of 7
```

```
In [56]: kmeans = KMeans(5)
         kmeans.fit(creditcard_df_scaled)
         labels = kmeans.labels_ # Labels (cluster) associated to each data point
```

```
In [57]: kmeans.cluster_centers_.shape
```

```
(5, 17)
```

```
In [58]: cluster_centers = pd.DataFrame(data = kmeans.cluster_centers_, columns = [creditcard_df.columns])
         cluster_centers
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
0	-0.242188	-0.335038	-0.339548	-0.221313	-0.396128	-0.168494
1	0.157739	0.429926	0.895701	0.835877	0.582143	-0.298657
2	1.430238	0.419467	6.915048	6.083034	5.172266	0.038778
3	-0.443442	0.103191	-0.099301	-0.257555	0.238336	-0.381977
4	1.481094	0.382972	-0.235548	-0.172446	-0.240230	1.760672

```python
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
# Second customers cluster (revolvers) who use credit card as a loan (most lucrative sector): highest balan
# Third customer cluster (VIP/Prime): high credit limit $16K and highest percentage of full payment, target
# Fourth customer cluster (low tenure): these are customers with low tenure (7 years), low balance
```

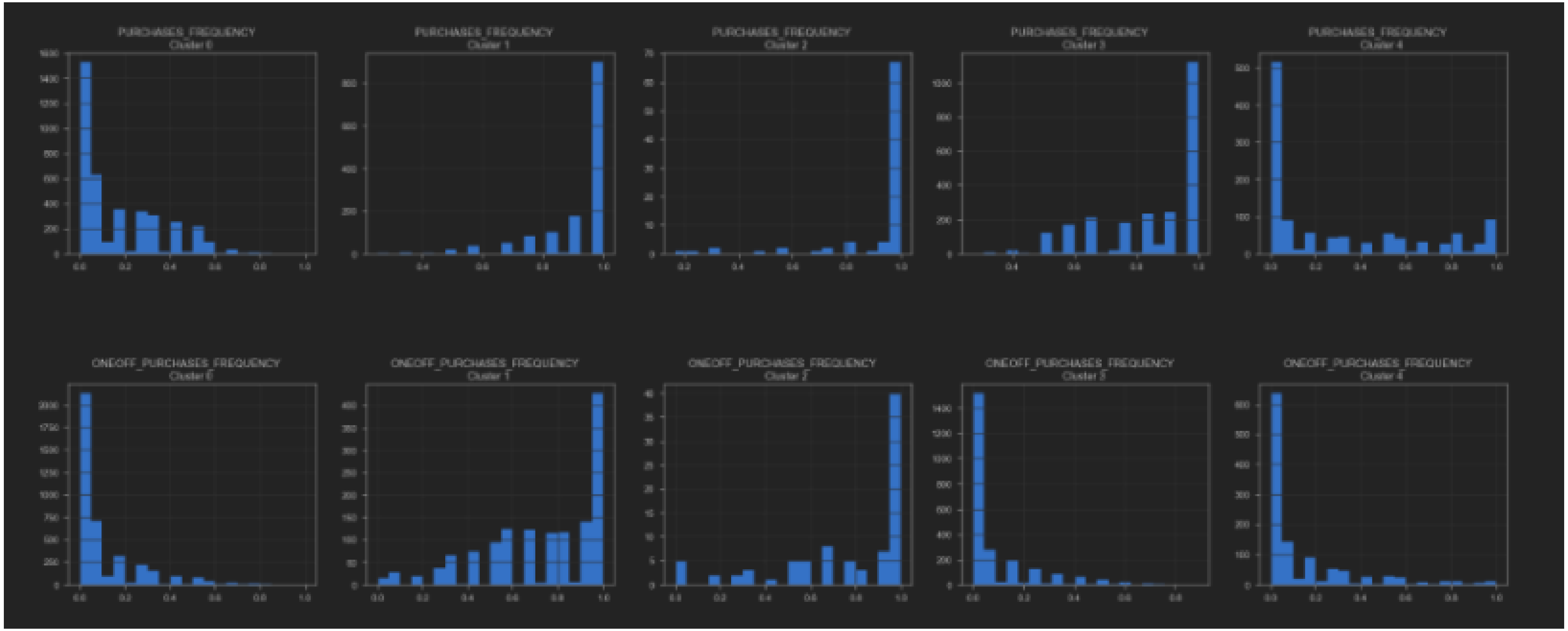
CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
2.243003	2.941476	3350.334466	1005.991021	602.097184	0.069941	11.456234
1.153127	43.511143	6960.026796	2776.202091	834.876950	0.264311	11.888569
2.988372	130.197674	12493.023256	15581.496801	3383.304083	0.394721	11.965116
0.708299	16.563071	3198.301375	929.237052	674.645925	0.273139	11.453942
14.734334	7.462489	7585.882211	3633.650163	2021.387424	0.036197	11.374228

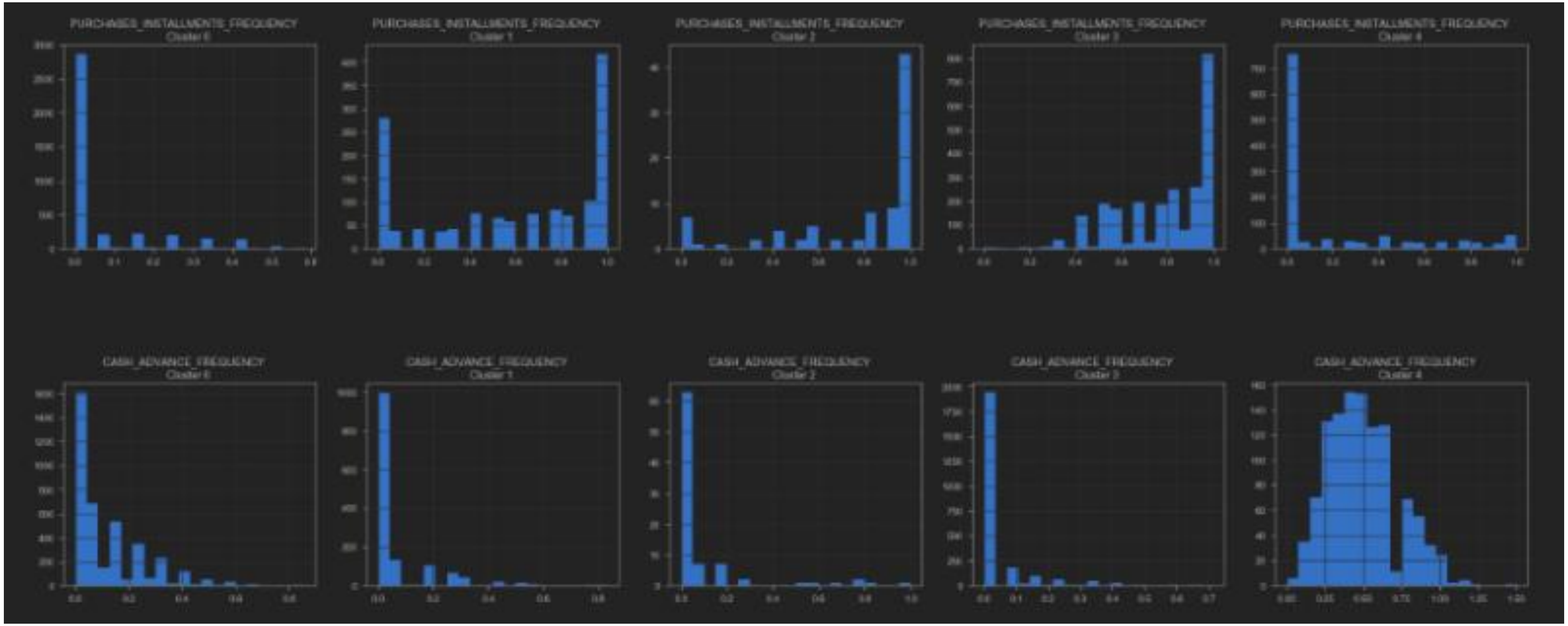
```
In [60]: labels.shape # Labels associated to each data point

(8950,)

In [61]: labels.max()

4

In [62]: labels.min()

0

In [63]: y_kmeans = kmeans.fit_predict(creditcard_df_scaled)
         y_kmeans

array([4, 2, 1, ..., 3, 4, 4])
```

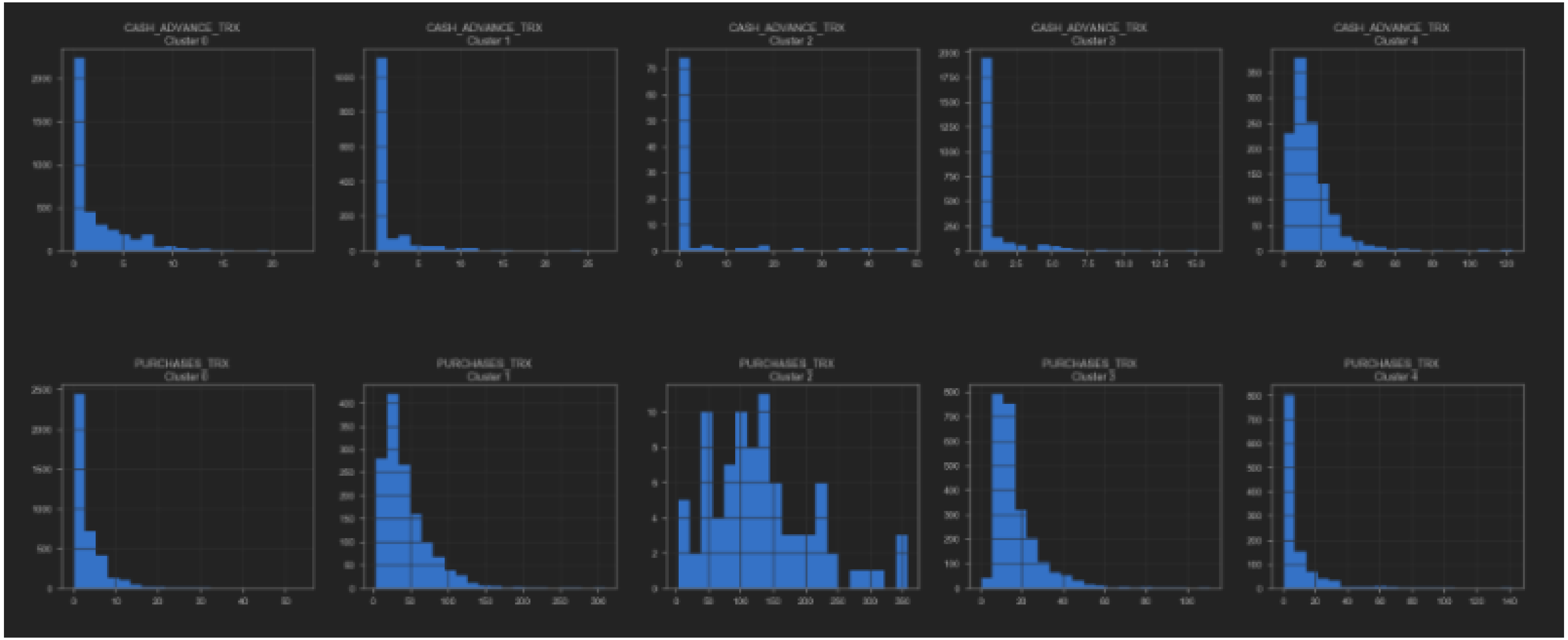
```python
In [64]: # concatenate the clusters labels to our original dataframe
creditcard_df_cluster = pd.concat([creditcard_df, pd.DataFrame({'cluster':labels})], axis = 1)
creditcard_df_cluster.head()
```

I_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE	cluster
	2	1000.0	201.802084	139.509787	0.000000	12	0
	0	7000.0	4103.032597	1072.340217	0.222222	12	4
	12	7500.0	622.066742	627.284787	0.000000	12	1
	1	7500.0	0.000000	864.206542	0.000000	12	0
	1	1200.0	678.334763	244.791237	0.000000	12	0

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

plt.show()
```


PURCHASES_FREQUENCY
Cluster 0
PURCHASES_FREQUENCY
Cluster 1
PURCHASES_FREQUENCY
Cluster 2
PURCHASES_FREQUENCY
Cluster 3
PURCHASES_FREQUENCY
Cluster 4
ONEOFF_PURCHASES_FREQUENCY
Cluster 0
ONEOFF_PURCHASES_FREQUENCY
Cluster 1
ONEOFF_PURCHASES_FREQUENCY
Cluster 2
ONEOFF_PURCHASES_FREQUENCY
Cluster 3
ONEOFF_PURCHASES_FREQUENCY
Cluster 4


PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY


CASH_ADVANCE_TRX
Cluster 0
CASH_ADVANCE_TRX
Cluster 1
CASH_ADVANCE_TRX
Cluster 2
CASH_ADVANCE_TRX
Cluster 3
CASH_ADVANCE_TRX
Cluster 4
PURCHASES_TRX
Cluster 0
PURCHASES_TRX
Cluster 1
PURCHASES_TRX
Cluster 2
PURCHASES_TRX
Cluster 3
PURCHASES_TRX
Cluster 4

TASK 8: APPLY PRINCIPAL COMPONENT ANALYSIS AND VISUALIZE THE RESULTS

PRINCIPAL COMPONENT ANALYSIS (PCA)

- PCA is an unsupervised machine learning algorithm.
- PCA performs dimensionality reductions while attempting at keeping the original information unchanged.
- PCA works by trying to find a new set of features called components.
- Components are composites of the uncorrelated given input features.

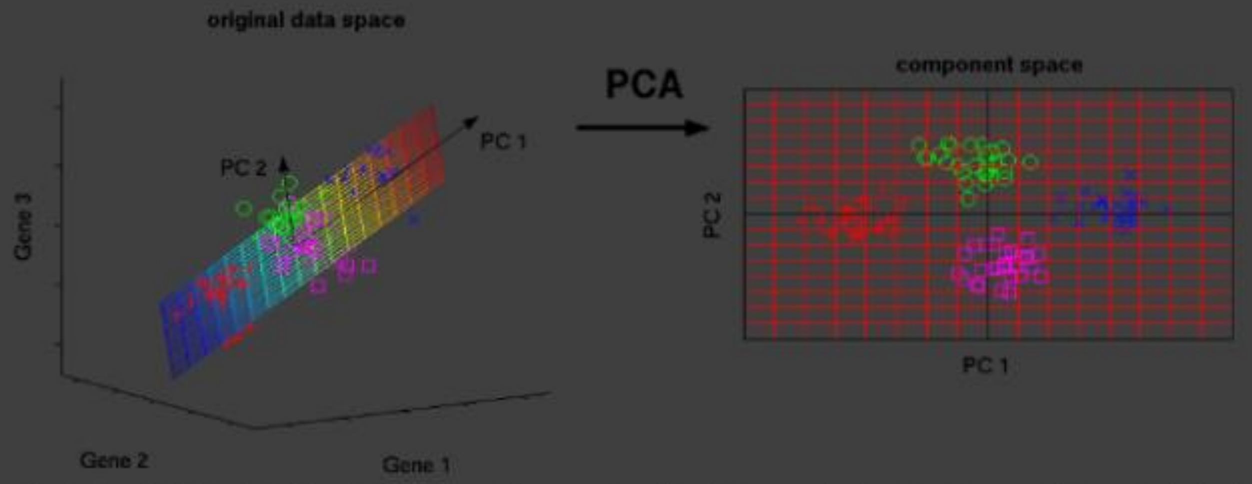


Photo Credit: http://phdthesis-bioinformatics-maxplanckinstitute-molecularplantphys.matthias-scholz.de/

```
In [66]:
# Obtain the principal components
pca = PCA(n_components=2)
principal_comp = pca.fit_transform(creditcard_df_scaled)
principal_comp
```

```
array([[-1.68221882, -1.07645695],
       [-1.1383011 ,  2.5065022 ],
       [ 0.96968621, -0.38352992],
       ...,
       [-0.92619882, -1.81080592],
       [-2.33654972, -0.65797375],
       [-0.55642724, -0.40044493]])
```

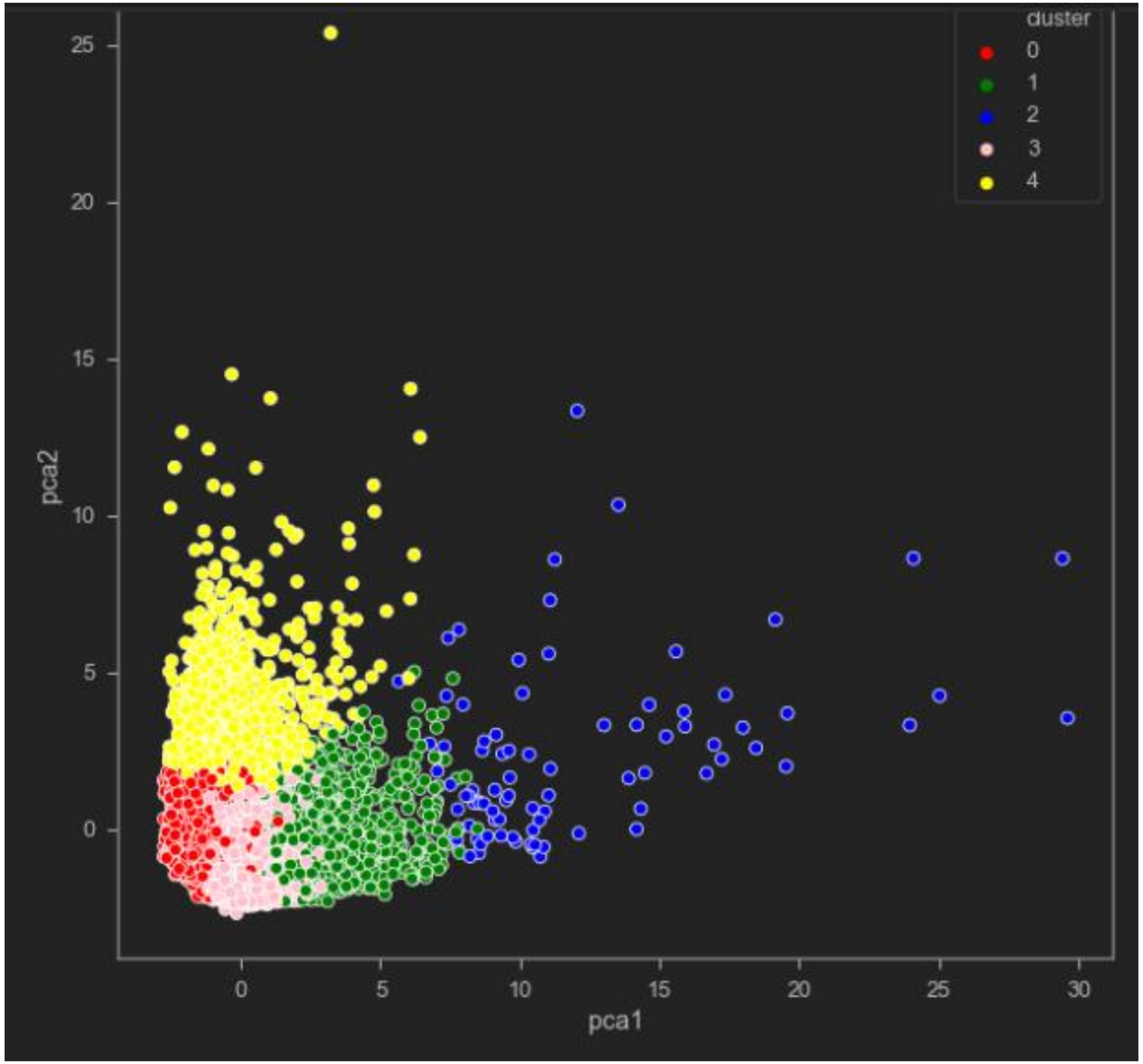
```
In [67]:
# Create a dataframe with the two components
pca_df = pd.DataFrame(data = principal_comp, columns =['pca1','pca2'])
pca_df.head()
```

	pca1	pca2
0	-1.682219	-1.076457
1	-1.138301	2.506502
2	0.969686	-0.383530
3	-0.873628	0.043165
4	-1.599431	-0.688591

```
In [68]:
# Concatenate the clusters labels to the dataframe
pca_df = pd.concat([pca_df,pd.DataFrame({'cluster':labels})], axis = 1)
pca_df.head()
```

	pca1	pca2	cluster
0	-1.682219	-1.076457	0
1	-1.138301	2.506502	4
2	0.969686	-0.383530	1
3	-0.873628	0.043165	0
4	-1.599431	-0.688591	0

```
In [69]:
plt.figure(figsize=(10,10))
ax = sns.scatterplot(x="pca1", y="pca2", hue = "cluster",
                     data = pca_df, palette =['red','green','blue','pink','yellow'])
plt.show()
```


cluster
0
1
2
3
4
pca2
pca1
25
20
15
10
5
0
0
5
10
15
20
25
30