THE PROBLEM STATEMENT AND BUSINESS CASE

Inputs for the Market Segmentation Project:

1. Given: Extensive data on bank's customers for the last 6 months. Data includes transactions on frequency, amount, tenure etc.
2. The marketing team would like to leverage AI/ML to launch a targeted marketing ad campaign that is tailored to specific group of customers.
3. In order for this campaign to be successful, the bank has to divide its customers into at least 3 distinctive groups.
4. This "marketing segmentation" is crucial for maximizing campaign conversion rate.
5. The customers are categorized as follows: i. Transactors: Customers who pay the least amount of interest charges and are careful with their money. ii. Revolvers: Customers who use their credit card as a loan. This group is the most lucrative sector for the bank since they pay 20% + profit. iii. VIP/Prime: Customers with high credit limit/ percentage of full payment, targeted to increase their credit limit/spending. New Customers: New customers with low tenure who can be targeted to enroll in other bank services (ex: travel credit card)
6. Data Source: https://www.kaggle.com/arjunbhasin2013/ccdata

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.decomposition import PCA
from jupyterthemes import jtplot
jtplot.style(theme='monokai', context='notebook', ticks=True, grid=False)
# this line of code is important to ensure that we are able to see the x and y axes
```

In [2]:
```
creditcard_df = pd.read_csv('Marketing_data.csv')
```

Different fields in data are given below:

CUSTID: Identification of Credit Card holder
BALANCE: Balance amount left in customer's account to make purchases
BALANCE_FREQUENCY: How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated)
PURCHASES: Amount of purchases made from account
ONEOFFPURCHASES: Maximum purchase amount done in one-go
INSTALLMENTS_PURCHASES: Amount of purchase done in installment
CASH_ADVANCE: Cash in advance given by the user
PURCHASES_FREQUENCY: How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
ONEOFF_PURCHASES_FREQUENCY: How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)

PURCHASES_INSTALLMENTS_FREQUENCY: How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)
CASH_ADVANCE_FREQUENCY: How frequently the cash in advance being paid
CASH_ADVANCE_TRX: Number of Transactions made with "Cash in Advance"
PURCHASES_TRX: Number of purchase transactions made
CREDIT_LIMIT: Limit of Credit Card for user
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
```

Out[3]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLM
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()
# info gives additional insights on the dataframe
# There are 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
```

Computing the average, minimum and maximum "BALANCE" amount.

In [5]:
```python
mean_balance = np.mean(creditcard_df.iloc[:,1])
print("The average balance amount is:", mean_balance)
min_balance = np.min(creditcard_df.iloc[:,1])
print("The minimum balance is: ", min_balance)
max_balance = np.max(creditcard_df.iloc[:,1])
print("The maximum balance is:", max_balance)
```

```
The average balance amount is: 1564.4748276781038
The minimum balance is:  0.0
The maximum balance is: 19043.13856
```

In [6]:
```python
# describe() gives statistical insights on the dataframe
creditcard_df.describe()
```

Out[6]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PI
count	8950.000000	8950.000000	8950.000000	8950.000000	8!
mean	1564.474828	0.877271	1003.204834	592.437371	4
std	2081.531879	0.236904	2136.634782	1659.887917	9
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	4
max	19043.138560	1.000000	49039.570000	40761.250000	22!

Some insights that can be seen from this data are:

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 generally low
Average credit limit is 4500
Percent of full payment is 15\%, i.e. only 15\% of credit card users pay in full.
Average tenure is 11 years, these customers have been using the credit card for a long period of time.

Let us solve for:

1. The features (i.e the row) of the customer who made the maximim "ONEOFF_PURCHASES"
2. 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 [7]:
```
creditcard_df[creditcard_df['ONEOFF_PURCHASES']== 40761.25]
```

Out[7]:

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

In [8]:
```
creditcard_df['CASH_ADVANCE'].max()
```

Out[8]: 47137.21176

In [9]:
```
creditcard_df[creditcard_df['CASH_ADVANCE'] == 47137.211760000006]
```

Out[9]:

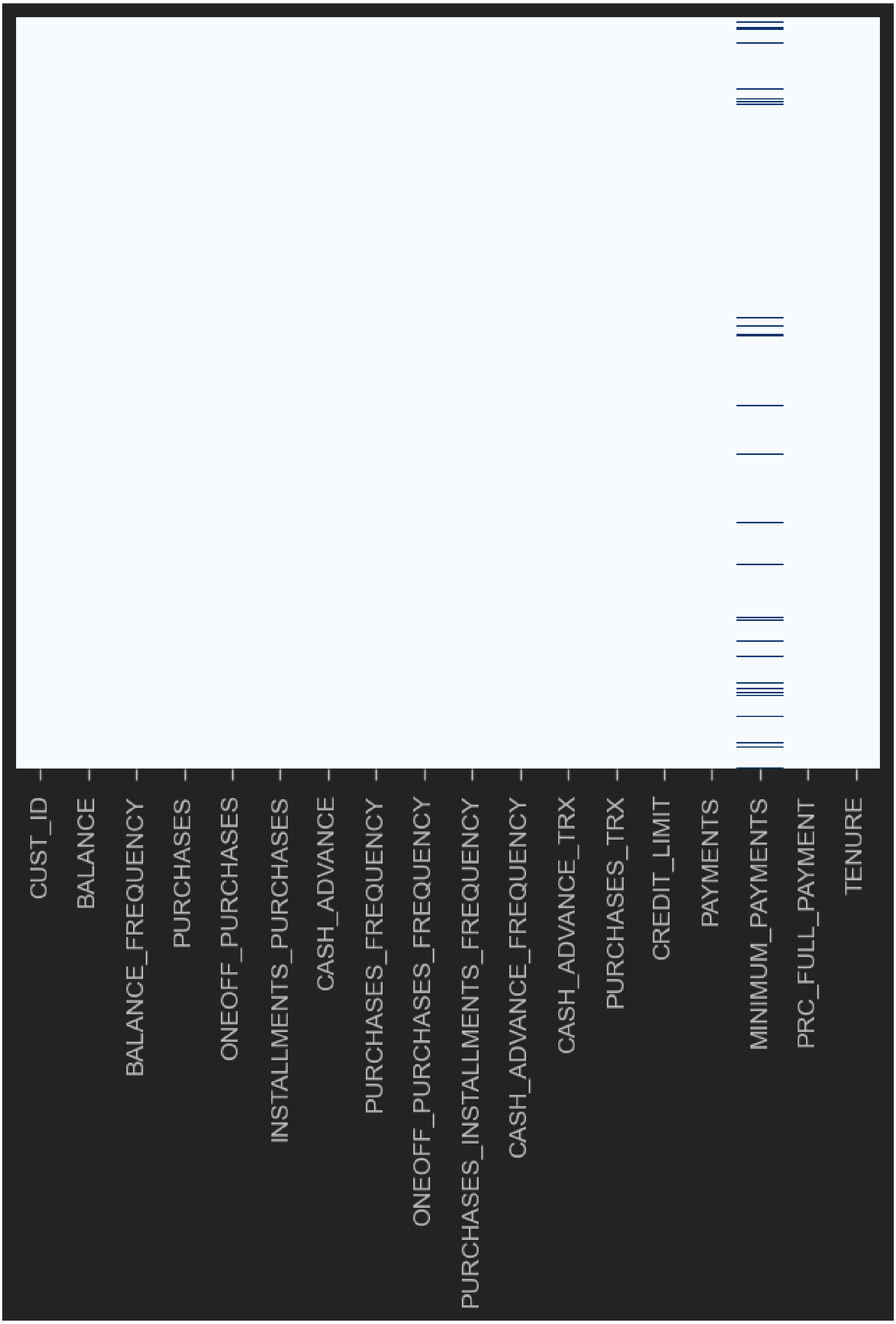
CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_P

The customer C12226 made 123 cash advance transactions. The percentage of full payment is 0, so the customer never paid their bill. Customers who use credit card for cash advance transactions are very important for the company as they pay more interest. These are customers of interest to the company.

Visualize And Explore The Dataset

In [10]:
```
# To check for missing data, a heatmap is used. Missing elements in the data are re
# Missing elements are there in the column minimum payments, see the .info() above.
sns.heatmap(creditcard_df.isnull(), yticklabels = False, cbar = False, cmap="Blues'
```

Out[10]: <AxesSubplot:>



In [11]:
```
# list all the missing values - to check if there are very few missing values in ot
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]:
```
# Fill up the missing elements with mean of the 'MINIMUM_PAYMENT'
creditcard_df.loc[(creditcard_df['MINIMUM_PAYMENTS'].isnull() == True), 'MINIMUM_PA
# Note that mean automatically excludes missing values.
```

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                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

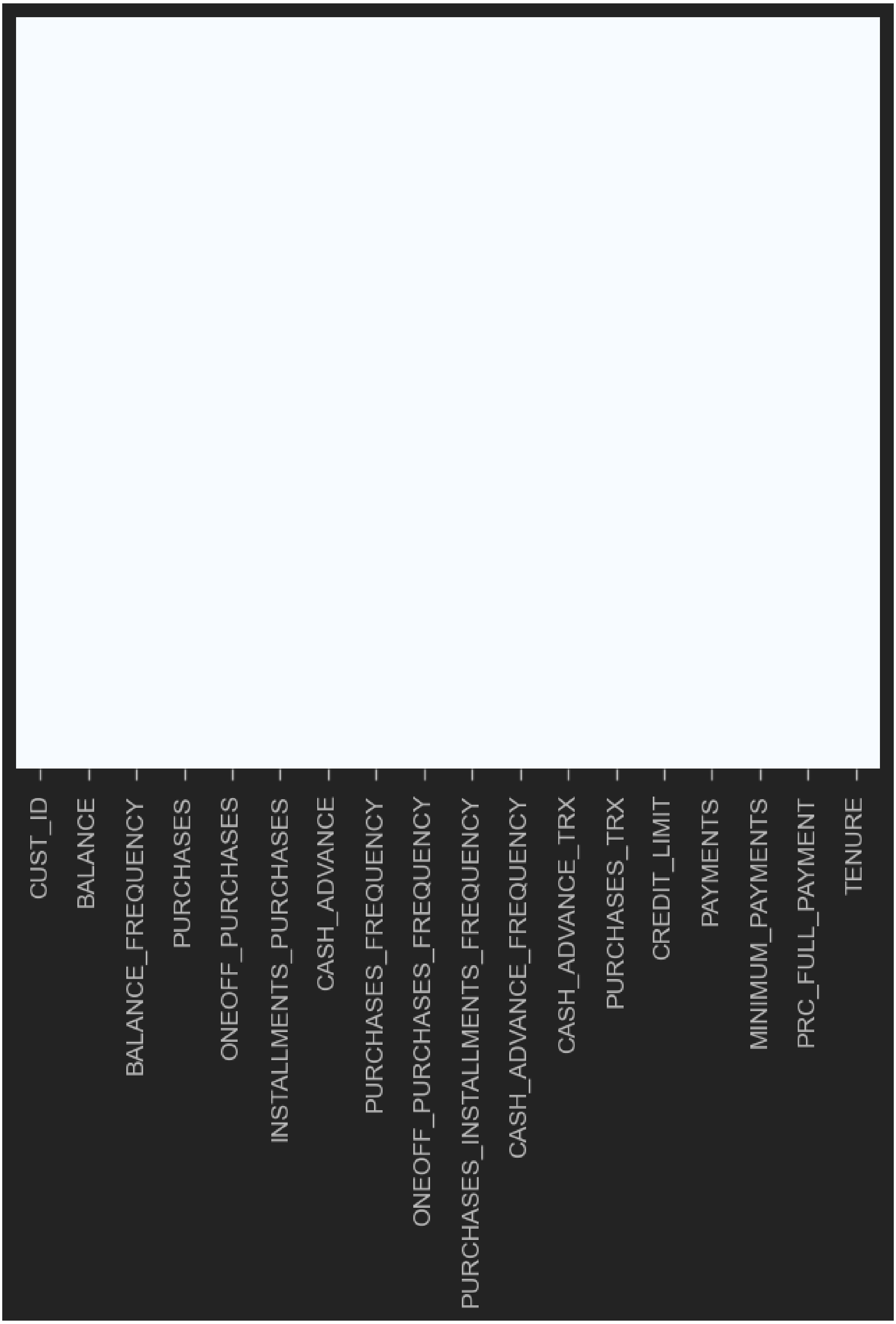
Next, we

1. Fill out missing elements in the "CREDIT_LIMIT" column, and
2. Double check to make sure that no missing elements are present

In [14]:
```
creditcard_df.loc[(creditcard_df['CREDIT_LIMIT'].isnull() == True), 'CREDIT_LIMIT']
```

In [15]:
```
# visual check using heatmap:
sns.heatmap(creditcard_df.isnull(), yticklabels = False, cbar = False, cmap="Blues"
```

Out[15]: <AxesSubplot:>



In [16]:
```
# check for duplicated entries in the data
creditcard_df.duplicated().sum()
```

Out[16]:
```
0
```

Let us Drop Customer ID column 'CUST_ID' as it is not required in this analysis.

In [17]:
```
creditcard_df.drop('CUST_ID', axis = 1, inplace = True) # inplace = True changes th
```

In [18]:
```
creditcard_df
```

Out[18]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PUR
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	
...	...	...	...	...	
8945	28.493517	1.000000	291.12	0.00	
8946	19.183215	1.000000	300.00	0.00	
8947	23.398673	0.833333	144.40	0.00	
8948	13.457564	0.833333	0.00	0.00	
8949	372.708075	0.666667	1093.25	1093.25	

8950 rows × 17 columns

In [19]:
```
n = len(creditcard_df.columns)
n
```

Out[19]:
```
17
```

In [20]:
```
creditcard_df.columns
```

Out[20]:
```
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')
```

Now, let us plot the columns of this dataframe.

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.

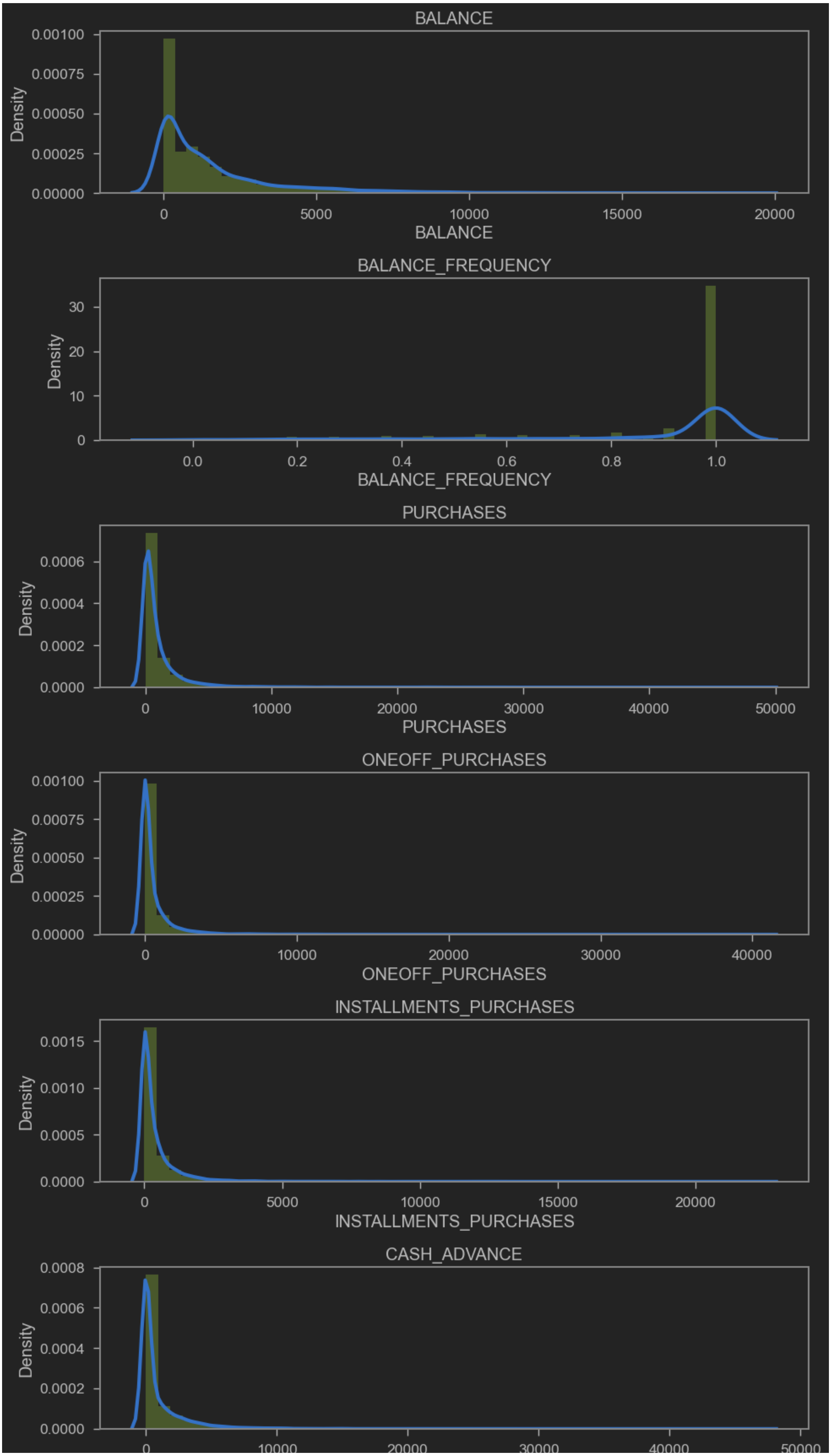
Mean of balance is $1500
 'Balance_Frequency' for most customers is updated frequently. It is approximately 1.
 For 'PURCHASES_FREQUENCY', there are two distinct group of customers.
 For 'ONEOFF_PURCHASES_FREQUENCY' and 'PURCHASES_INSTALLMENT_FREQUENCY' most users don't do one off puchases or installment purchases frequently.
 Very small number of customers pay their balance in full 'PRC_FULL_PAYMENT', approximately 0
 Credit limit average is around $4500. Most customers have 11 years tenure.

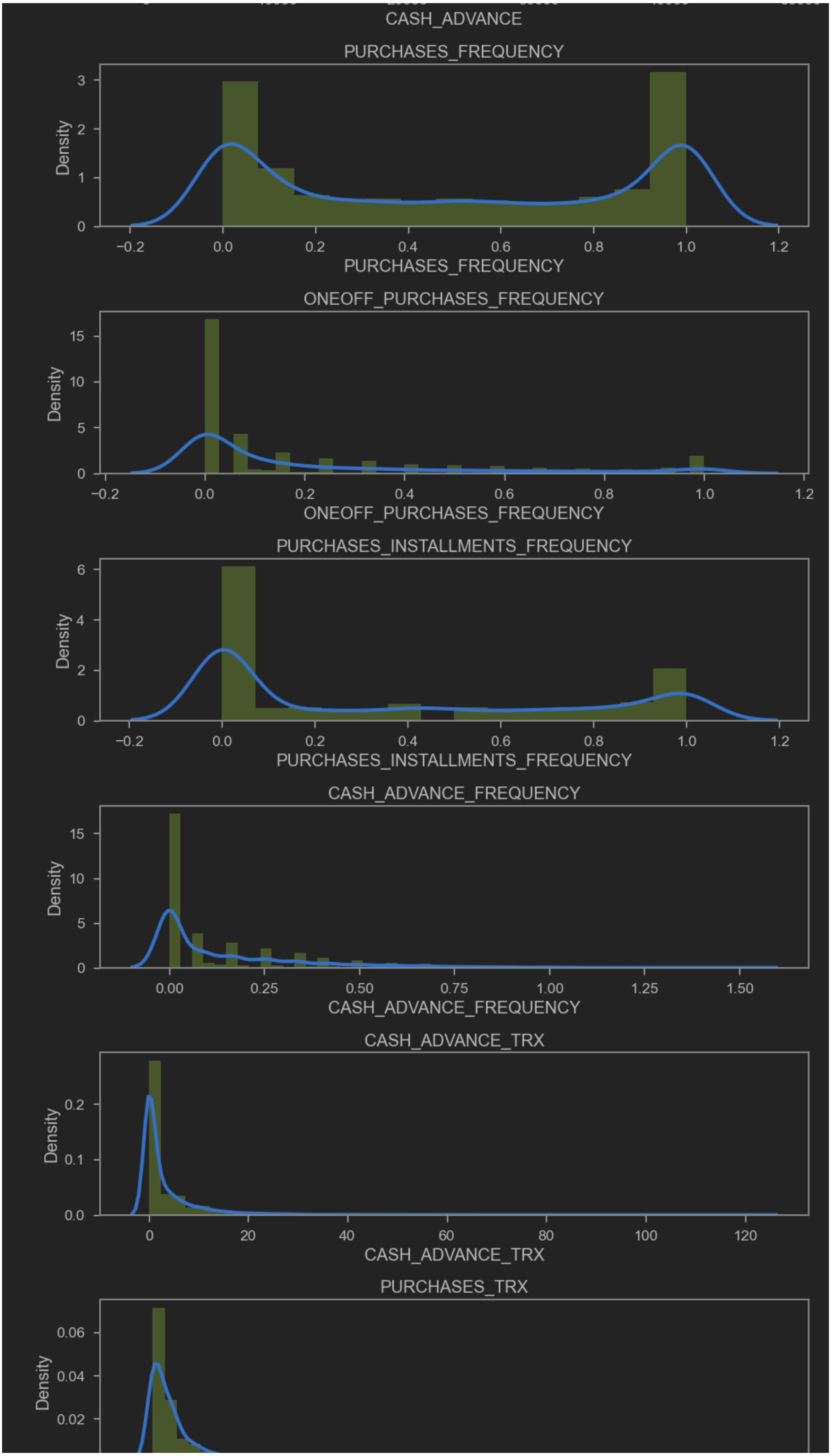
In [21]:
```
plt.figure(figsize=(10,50))
for i in range(len(creditcard_df.columns)):
  plt.subplot(17, 1, i+1)
  sns.distplot(creditcard_df[creditcard_df.columns[i]], kde_kws={'color': 'b', 'lw'
  plt.title(creditcard_df.columns[i])

plt.tight_layout()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
```

```
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

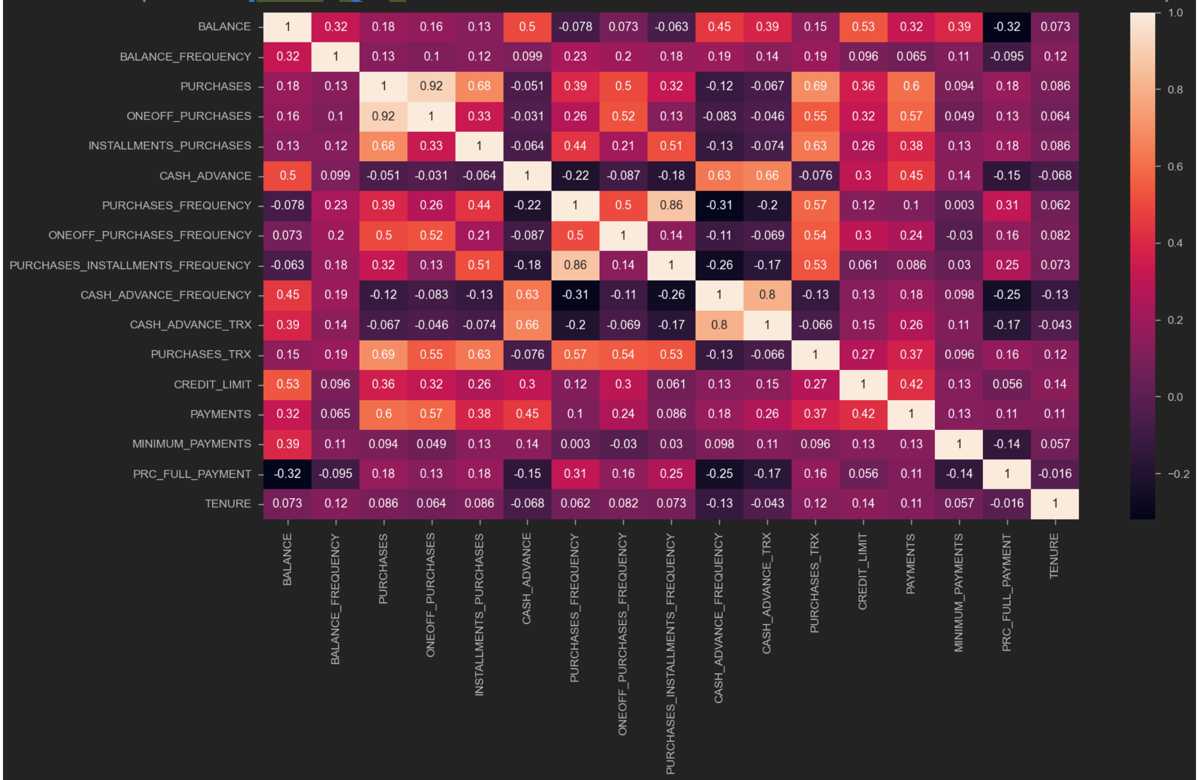

BALANCE
Density
0.00100
0.00075
0.00050
0.00025
0.00000
0
5000
10000
15000
20000
BALANCE
BALANCE_FREQUENCY
Density
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
BALANCE_FREQUENCY
PURCHASES
Density
0.0006
0.0004
0.0002
0.0000
0
10000
20000
30000
40000
50000
PURCHASES
ONEOFF_PURCHASES
Density
0.00100
0.00075
0.00050
0.00025
0.00000
0
10000
20000
30000
40000
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
Density
0.0015
0.0010
0.0005
0.0000
0
5000
10000
15000
20000
INSTALLMENTS_PURCHASES
CASH_ADVANCE
Density
0.0008
0.0006
0.0004
0.0002
0.0000
0
10000
20000
30000
40000
50000


CASH_ADVANCE
PURCHASES_FREQUENCY
Density
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
Density
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
Density
6
4
2
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
Density
15
10
5
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
Density
0.2
0.1
0.0
0
20
40
60
80
100
120
CASH_ADVANCE_TRX
PURCHASES_TRX
Density
0.06
0.04
0.02

The correlation matrix between features is given below.

In [22]:
```python
correlations = creditcard_df.corr()
f, ax = plt.subplots(figsize = (20,10))
sns.heatmap(correlations, annot = True)
```

Out[22]: <AxesSubplot:>



THE THEORY AND INTUITON OF K-MEANS CLUSTERING:

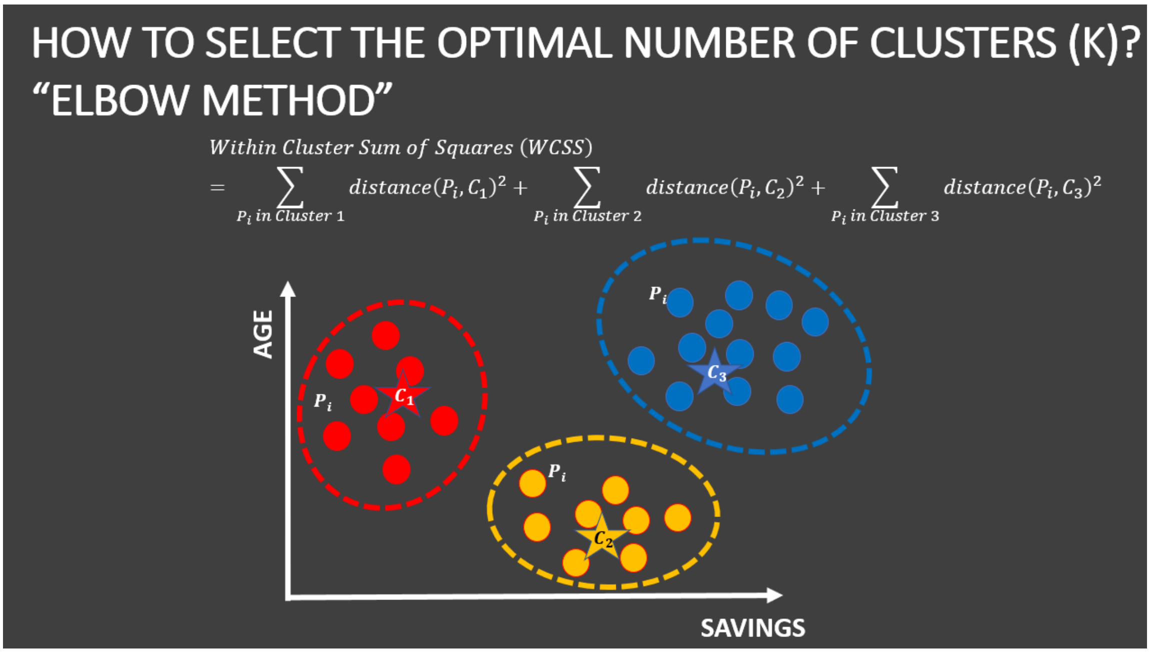
1. K-means is an unsupervised learning algorithm(clustering). This means there is no ground truth to compare to.
2. K-means works by grouping some data points together (clustering) in an unsupervised manner
3. The algoithm groups observations with similar attribute values together by measuring the Euclidean distance between points.

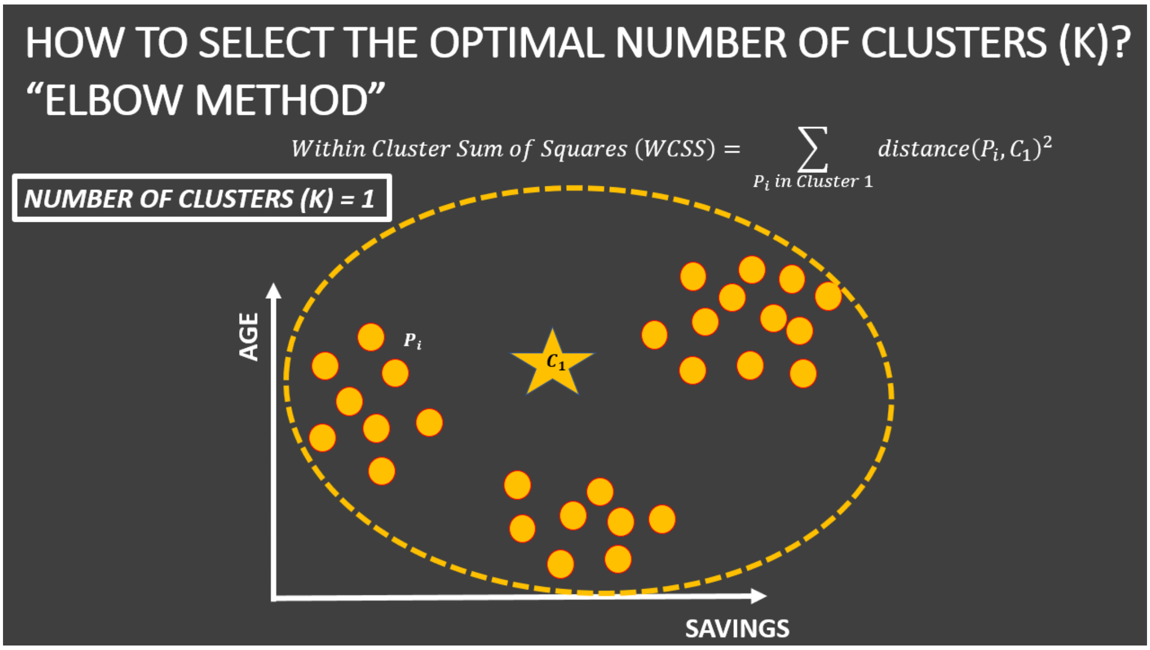
K-Means Algorithm:

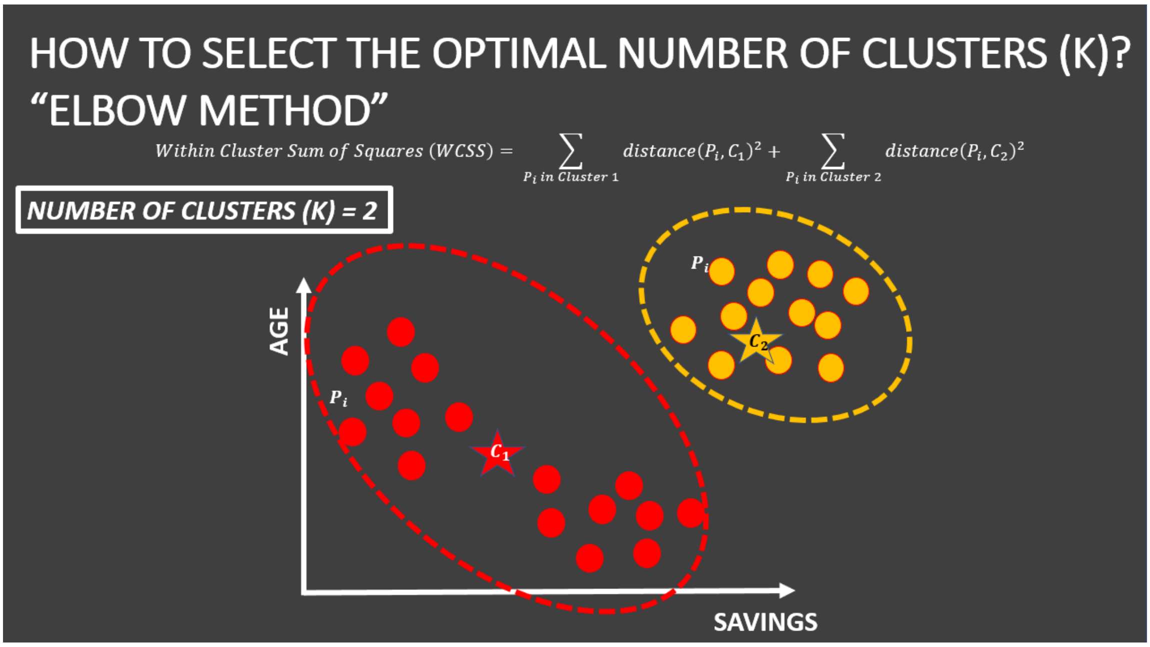
1. Choose the number of clusters "K".
2. Select a random K points that are going to be the centroids for each cluster.
3. Assign each data point to the nearest centroid. This creates "K" clusters.
4. Calculate a new centroid for each cluster
5. Reassign each data point to the new closest centroid.
6. Go to step 4 and calculate new centroids again and repeat.

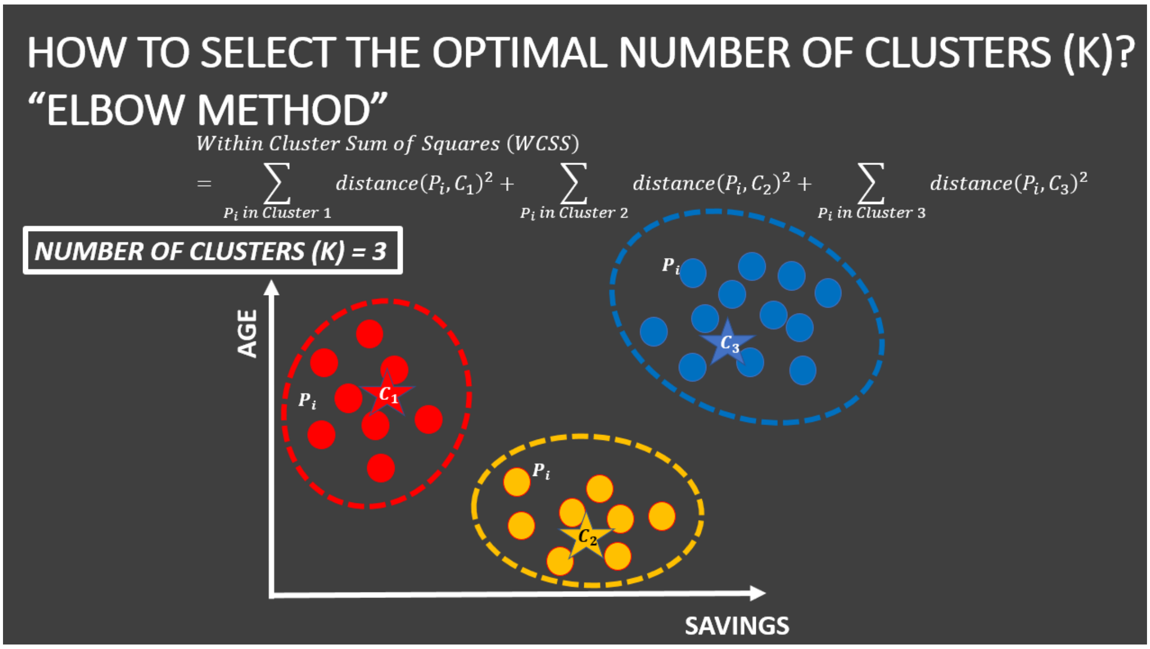
The following conditions could terminate the K-means clustering algorithm.

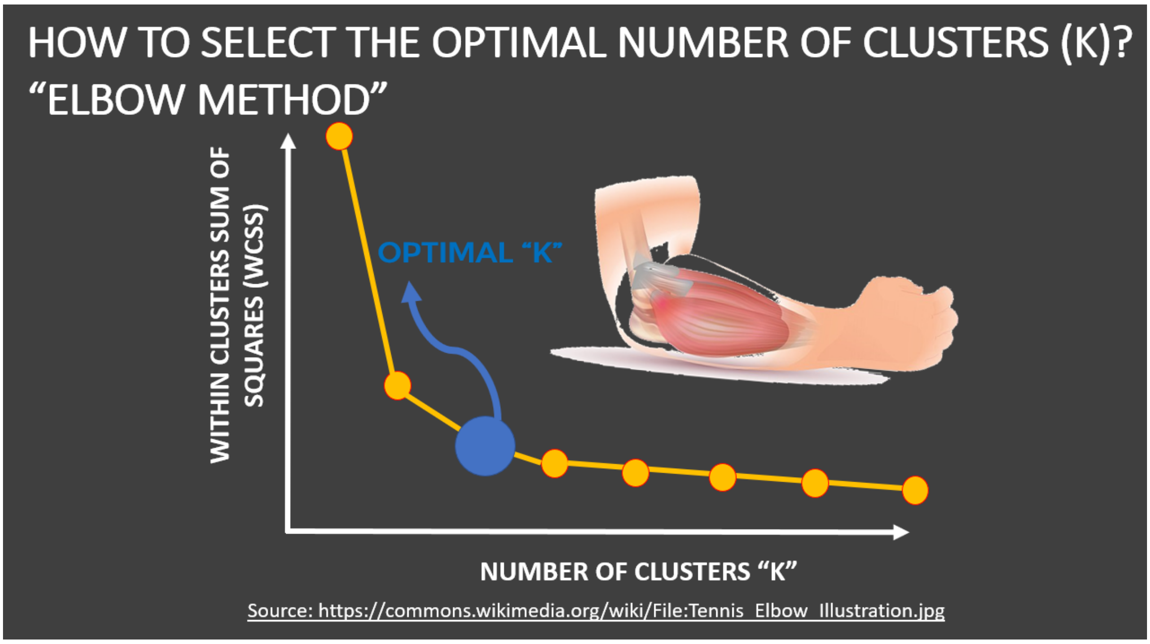
```
- K-means terminates after a fixed number of iterations is reached
- K-means terminates when the centroid locations do not change
between       iterations.
```


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


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
SAVINGS
P_i
C_1


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
SAVINGS
P_i
C_1
C_2





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 [23]:
```
# Scale the data
scaler = StandardScaler()
creditcard_df_scaled = scaler.fit_transform(creditcard_df)
```

In [24]:
```
creditcard_df_scaled.shape
```

Out[24]: (8950, 17)

In [25]:
```
creditcard_df_scaled
```

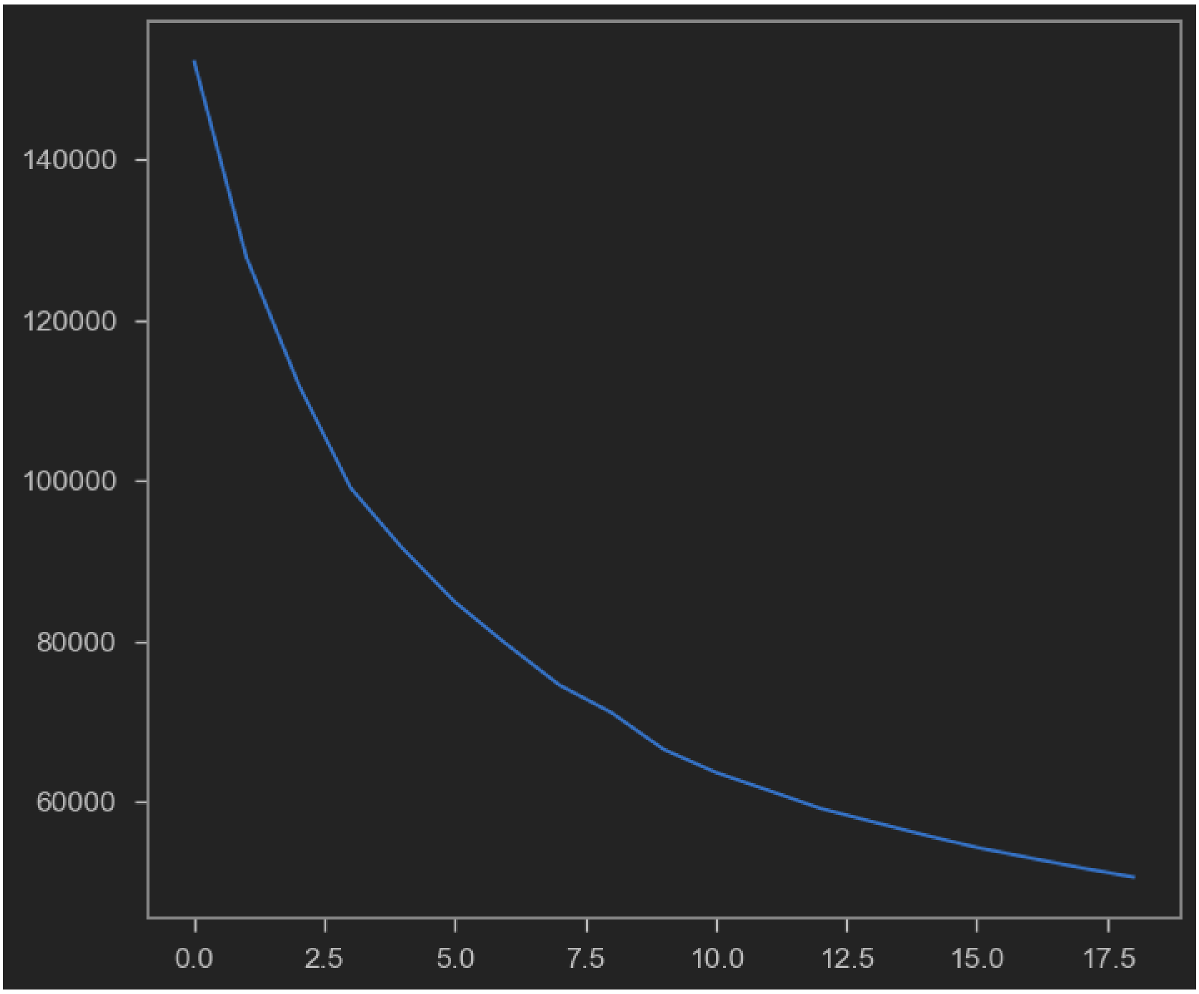
Out[25]:
```
array([[-0.73198937, -0.24943448, -0.42489974, ..., -0.31096755,
        -0.52555097,  0.36067954],
       [ 0.78696085,  0.13432467, -0.46955188, ...,  0.08931021,
         0.2342269 ,  0.36067954],
       [ 0.44713513,  0.51808382, -0.10766823, ..., -0.10166318,
        -0.52555097,  0.36067954],
       ...,
       [-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 [26]:
```
scores_1 = []
range_values = range(1,20)
for i in range_values:
    kmeans = KMeans(n_clusters = i) # n_clusters is the no. of clusters to form and
    kmeans.fit(creditcard_df_scaled) # this returns a fitted estimator
    scores_1.append(kmeans.inertia_)

plt.plot(scores_1, 'bx-')

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

Out[26]: [<matplotlib.lines.Line2D at 0x29a1359b460>]



Let's assume that our data only consists of the first 7 columns of "creditcard_df_scaled", Calculate the optimal number of clusters in this case.

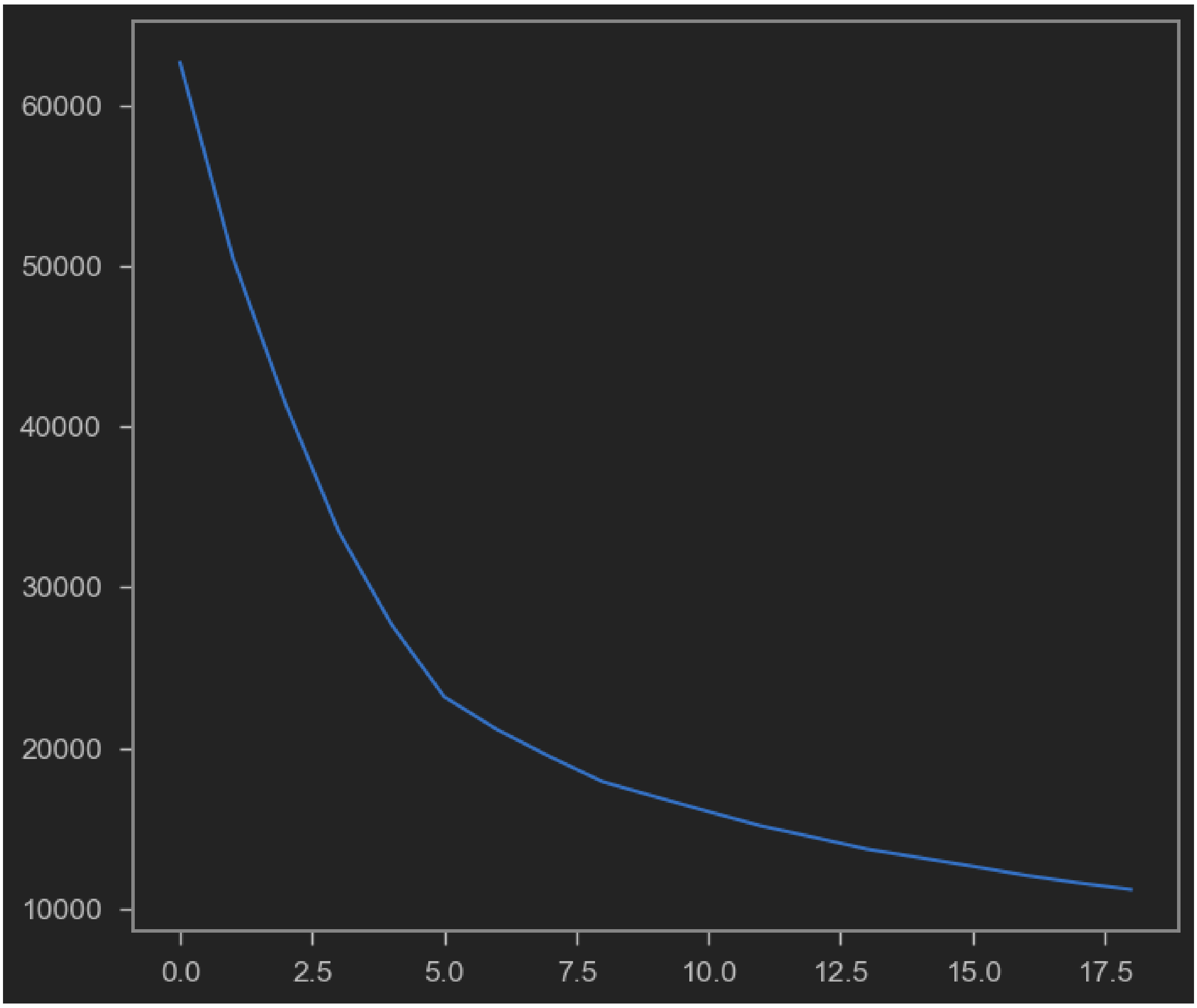
In [27]:
```
creditcard_df_scaled[7:]
```

Out[27]:
```
array([[ 0.12452002,  0.51808382, -0.26538766, ..., -0.14253532,
        -0.52555097,  0.36067954],
       [-0.26402625,  0.51808382, -0.06632989, ..., -0.23696766,
        -0.52555097,  0.36067954],
       [-0.67850402, -1.40071194,  0.13030337, ..., -0.32779151,
        -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 [28]:
```
scores_1 = []
range_values = range(1,20)
for i in range_values:
    kmeans = KMeans(n_clusters = i) # n_clusters is the no. of clusters to form and
    kmeans.fit(creditcard_df_scaled[:,:7]) # this returns a fitted estimator
    scores_1.append(kmeans.inertia_)

plt.plot(scores_1, 'bx-')
# from the figure below, we can change the number of clusters to 5 or 4 and see how
# i.e. check the error SS and see if it reduces.
```

Out[28]:
```
[<matplotlib.lines.Line2D at 0x29a15cd7940>]
```



APPLY THE K-MEANS METHOD

In [29]:
```
kmeans = KMeans(7) # change the number of clusters to 4 or 5 and redo.
kmeans.fit(creditcard_df_scaled)
labels = kmeans.labels_
```

In [30]:
```
kmeans.cluster_centers_.shape
```

Out[30]:
```
(7, 17)
```

In [31]:
```
cluster_centers = pd.DataFrame(data = kmeans.cluster_centers_, columns = [creditcar
cluster_centers
```

Out[31]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASE
0	0.128727	0.429707	0.936945	0.893701	0.57350
1	1.668800	0.396392	-0.206192	-0.150467	-0.21118
2	-0.368430	0.330056	-0.039725	-0.235423	0.33840
3	-0.701828	-2.136168	-0.307232	-0.230688	-0.30251
4	0.005970	0.402619	-0.343708	-0.225103	-0.39903
5	-0.336070	-0.346074	-0.284289	-0.209289	-0.28733
6	1.430238	0.419467	6.915048	6.083034	5.17226

In [32]:
```
# To understand what these numbers mean, we need to perform an inverse transformati
cluster_centers = scaler.inverse_transform(cluster_centers)
cluster_centers = pd.DataFrame(data = cluster_centers, columns = [creditcard_df.col
cluster_centers
```

Out[32]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCH
0	1832.409401	0.979064	3005.002862	2075.798208	929.67
1	5037.940211	0.971172	562.671846	342.692852	220.09
2	797.619923	0.955458	918.330731	201.683338	717.08
3	103.679857	0.371232	346.799789	209.543058	137.50
4	1576.900592	0.972647	268.867401	218.813018	50.22
5	864.973648	0.795289	395.817971	245.060808	151.23
6	4541.393882	0.976638	15777.311395	10689.027791	5088.28

First Customers cluster (Transactors): Those are customers who pay least amount of interest charges and careful with their money, Cluster with lowest balance ($\$104$) and cash advance ($\$303$), Percentage of full payment = 23\%

Second customers cluster (Revolvers): who use credit card as a loan (most lucrative sector): 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

In [33]:
```python
labels.shape # Labels associated with each data point
```
Out[33]:
```
(8950,)
```

In [34]:
```python
labels.max()
```
Out[34]:
```
6
```

In [35]:
```python
labels.min()
```
Out[35]:
```
0
```

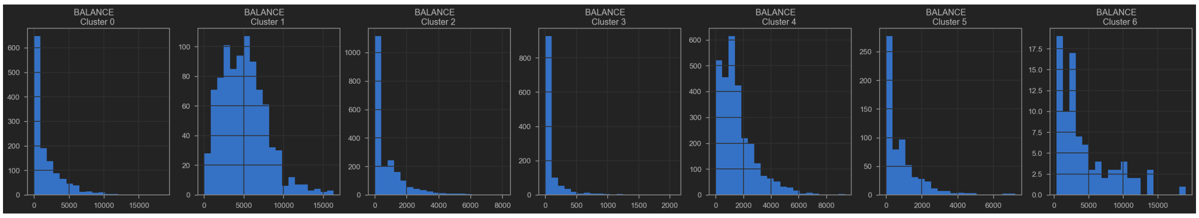
In [36]:
```python
y_kmeans = kmeans.fit_predict(creditcard_df_scaled)
y_kmeans # every data point in the dataframe is now associated with a label.
```
Out[36]:
```
array([0, 2, 4, ..., 5, 0, 0])
```

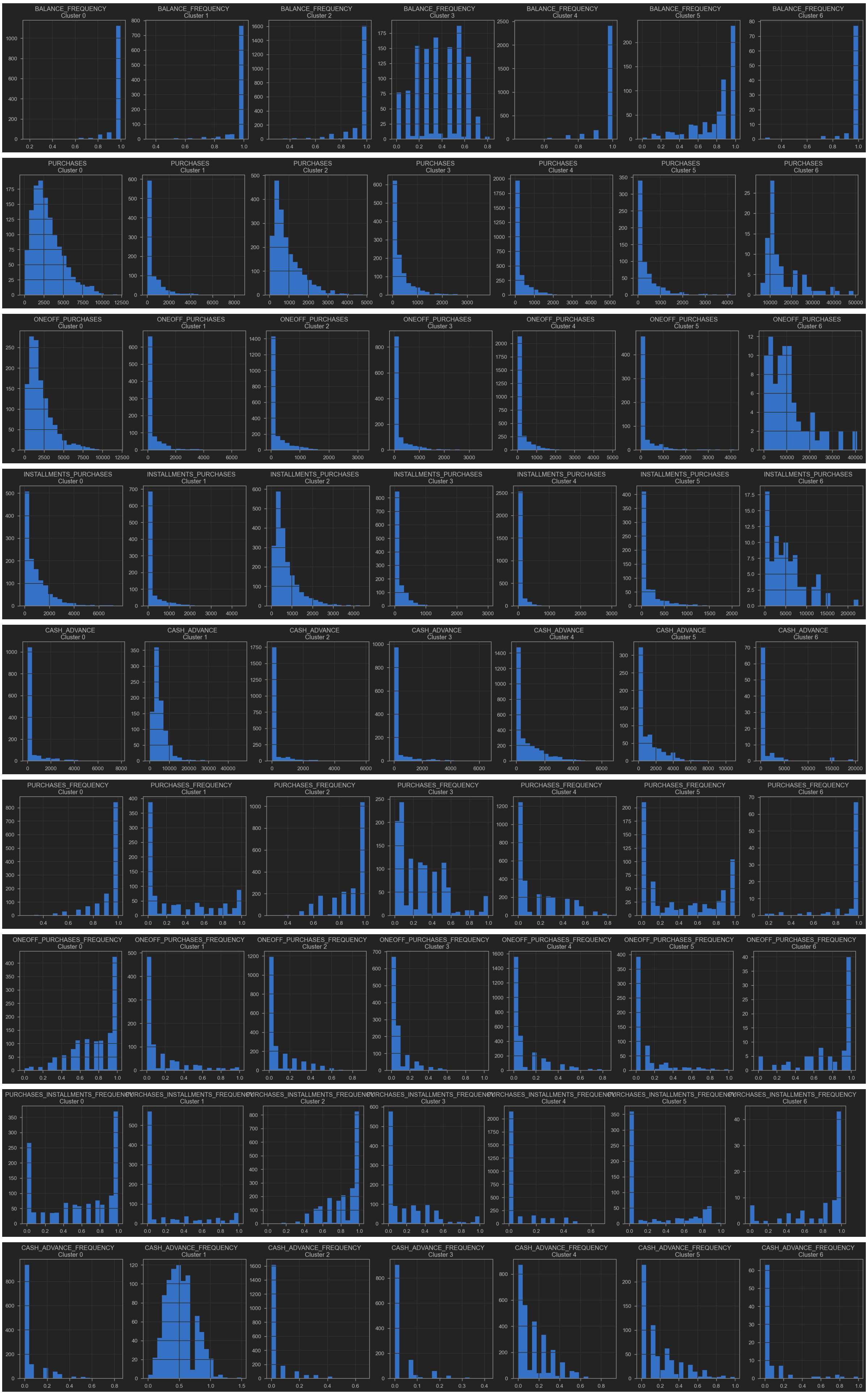
In [37]:
```python
# concatenate the clusters labels to our original dataframe, i.e. we are labelling
creditcard_df_cluster = pd.concat([creditcard_df, pd.DataFrame({'cluster':labels})]
creditcard_df_cluster.head()
```
Out[37]:

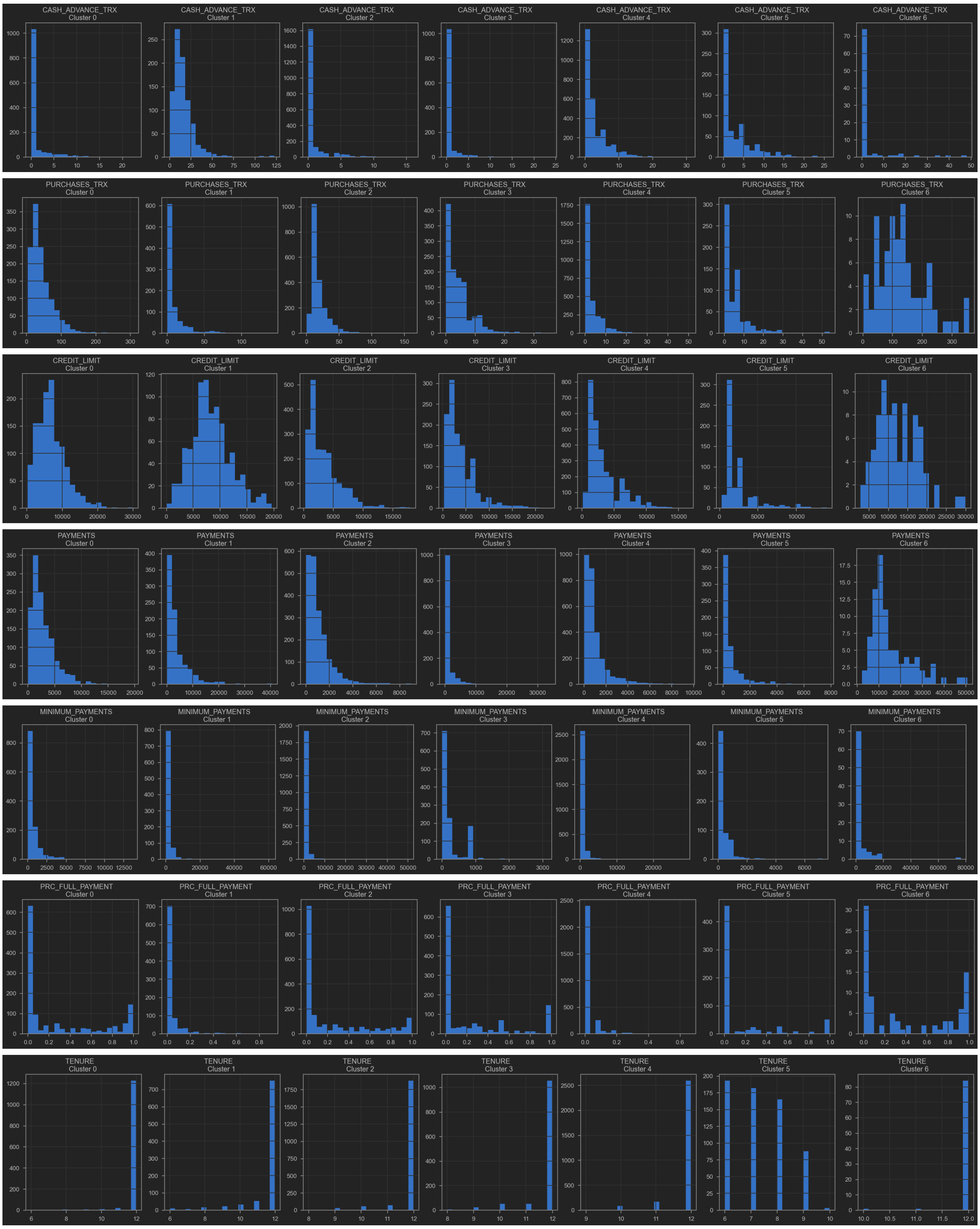
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	

In [38]:
```python
# 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)  # why is there cluster[i] here?
    plt.title('{}    \nCluster {} '.format(i,j))

  plt.show()
```




BALANCE_FREQUENCY Cluster 0
BALANCE_FREQUENCY Cluster 1
BALANCE_FREQUENCY Cluster 2
BALANCE_FREQUENCY Cluster 3
BALANCE_FREQUENCY Cluster 4
BALANCE_FREQUENCY Cluster 5
BALANCE_FREQUENCY Cluster 6
PURCHASES Cluster 0
PURCHASES Cluster 1
PURCHASES Cluster 2
PURCHASES Cluster 3
PURCHASES Cluster 4
PURCHASES Cluster 5
PURCHASES Cluster 6
ONEOFF_PURCHASES Cluster 0
ONEOFF_PURCHASES Cluster 1
ONEOFF_PURCHASES Cluster 2
ONEOFF_PURCHASES Cluster 3
ONEOFF_PURCHASES Cluster 4
ONEOFF_PURCHASES Cluster 5
ONEOFF_PURCHASES Cluster 6
INSTALLMENTS_PURCHASES Cluster 0
INSTALLMENTS_PURCHASES Cluster 1
INSTALLMENTS_PURCHASES Cluster 2
INSTALLMENTS_PURCHASES Cluster 3
INSTALLMENTS_PURCHASES Cluster 4
INSTALLMENTS_PURCHASES Cluster 5
INSTALLMENTS_PURCHASES Cluster 6
CASH_ADVANCE Cluster 0
CASH_ADVANCE Cluster 1
CASH_ADVANCE Cluster 2
CASH_ADVANCE Cluster 3
CASH_ADVANCE Cluster 4
CASH_ADVANCE Cluster 5
CASH_ADVANCE Cluster 6
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
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 0
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 1
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 2
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 3
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 4
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 5
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 6
CASH_ADVANCE_FREQUENCY Cluster 0
CASH_ADVANCE_FREQUENCY Cluster 1
CASH_ADVANCE_FREQUENCY Cluster 2
CASH_ADVANCE_FREQUENCY Cluster 3
CASH_ADVANCE_FREQUENCY Cluster 4
CASH_ADVANCE_FREQUENCY Cluster 5
CASH_ADVANCE_FREQUENCY Cluster 6



APPLY PRINCIPAL COMPONENT ANALYSIS AND VISUALIZE THE RESULTS

1. PCA is an unsupervised machine learning algorithm.
2. PCA performs dimensionality reducions while attempting at keeping the original information unchanged.
3. PCA works by trying to fidn a new set of features called components.
4. Components are composites of the uncorrelated given input features.

In [39]:
```
# Obtain the principal components
pca = PCA(n_components = 2)
principal_comp = pca.fit_transform(creditcard_df_scaled)
principal_comp
```

Out[39]:
```
array([[-1.6822215 , -1.07645371],
       [-1.13831646,  2.50643918],
       [ 0.96967197, -0.3835442 ],
       ...,
       [-0.92619322, -1.8107684 ],
       [-2.33654193, -0.65794895],
       [-0.5564267 , -0.40047336]])
```

In [40]:
```
# Create a dataframe with the two components
pca_df = pd.DataFrame(data = principal_comp, columns =['pca1','pca2'])
pca_df.head()
```

Out[40]:

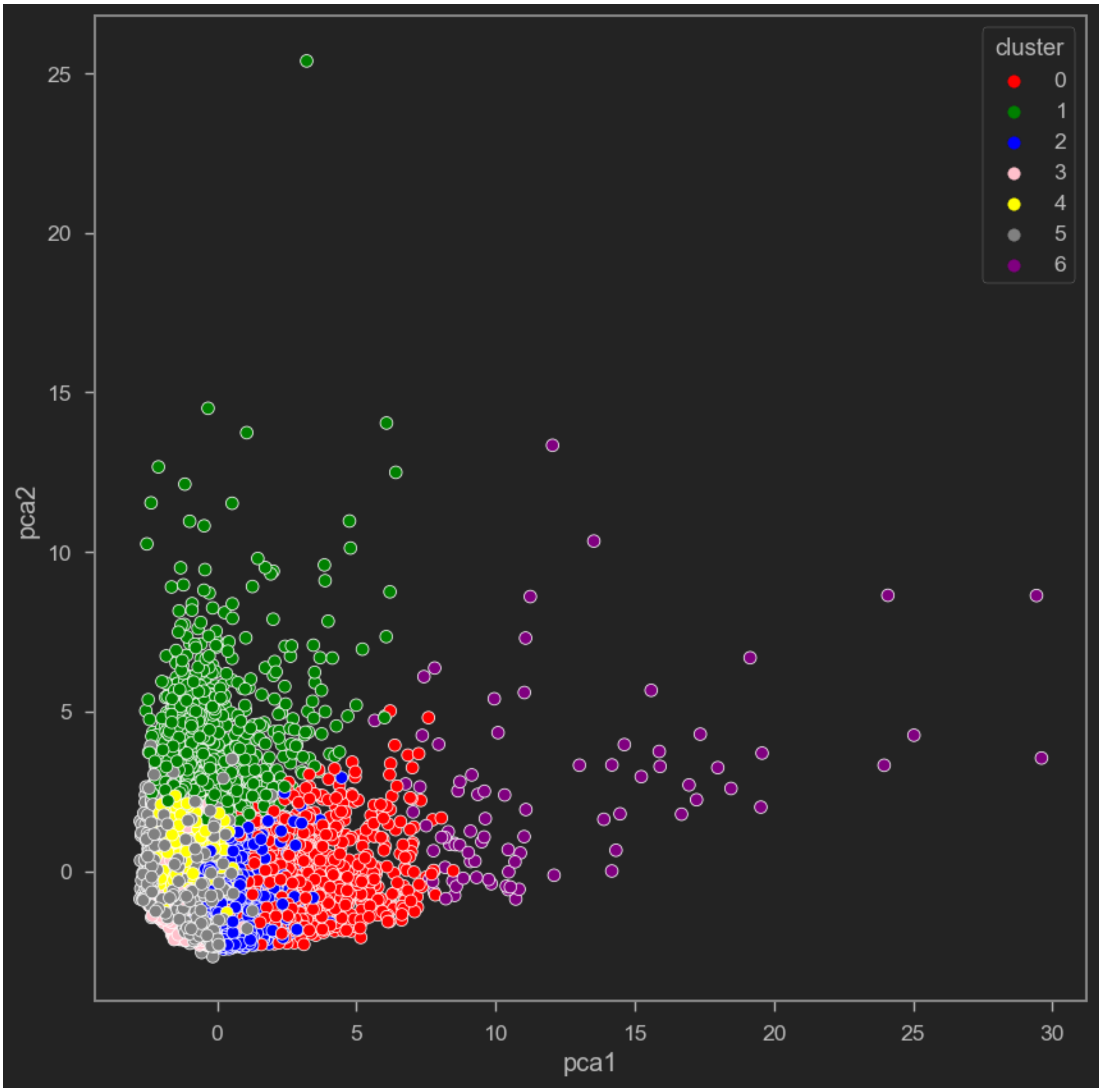
	pca1	pca2
0	-1.682222	-1.076454
1	-1.138316	2.506439
2	0.969672	-0.383544
3	-0.873625	0.043172
4	-1.599434	-0.688583

In [41]:
```
# Concatenate the clusters labels to the dataframe
pca_df = pd.concat([pca_df,pd.DataFrame({'cluster':labels})], axis = 1)
pca_df.head()
```

Out[41]:

	pca1	pca2	cluster
0	-1.682222	-1.076454	4
1	-1.138316	2.506439	1
2	0.969672	-0.383544	0
3	-0.873625	0.043172	4
4	-1.599434	-0.688583	4

In [42]:
```
plt.figure(figsize=(10,10))
ax = sns.scatterplot(x="pca1", y="pca2", hue = "cluster", data = pca_df, palette =[
plt.show()
```



As there are 17 dimensions for each point in a cluster, it is not possible to physically represent the points. Therefore, the points are converted to principle components and plotted to represent them as a cluster of points. Also, this gives a visual representation of how out clustering method worked. We can see that the points are divided into 7 clusters, and that the points in each cluster are near each other. Showing that we can group the customers based on the customers credit card usage patterns.