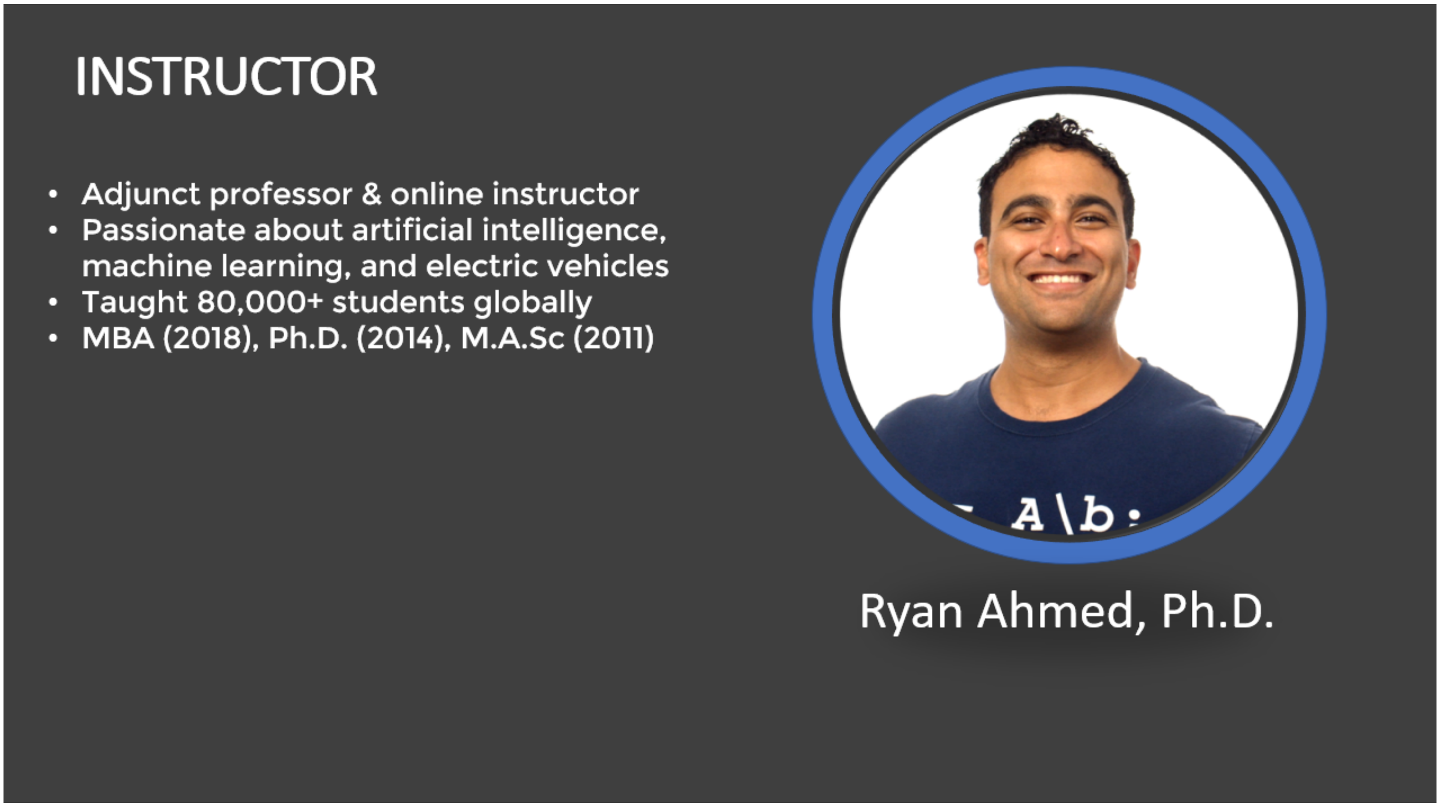
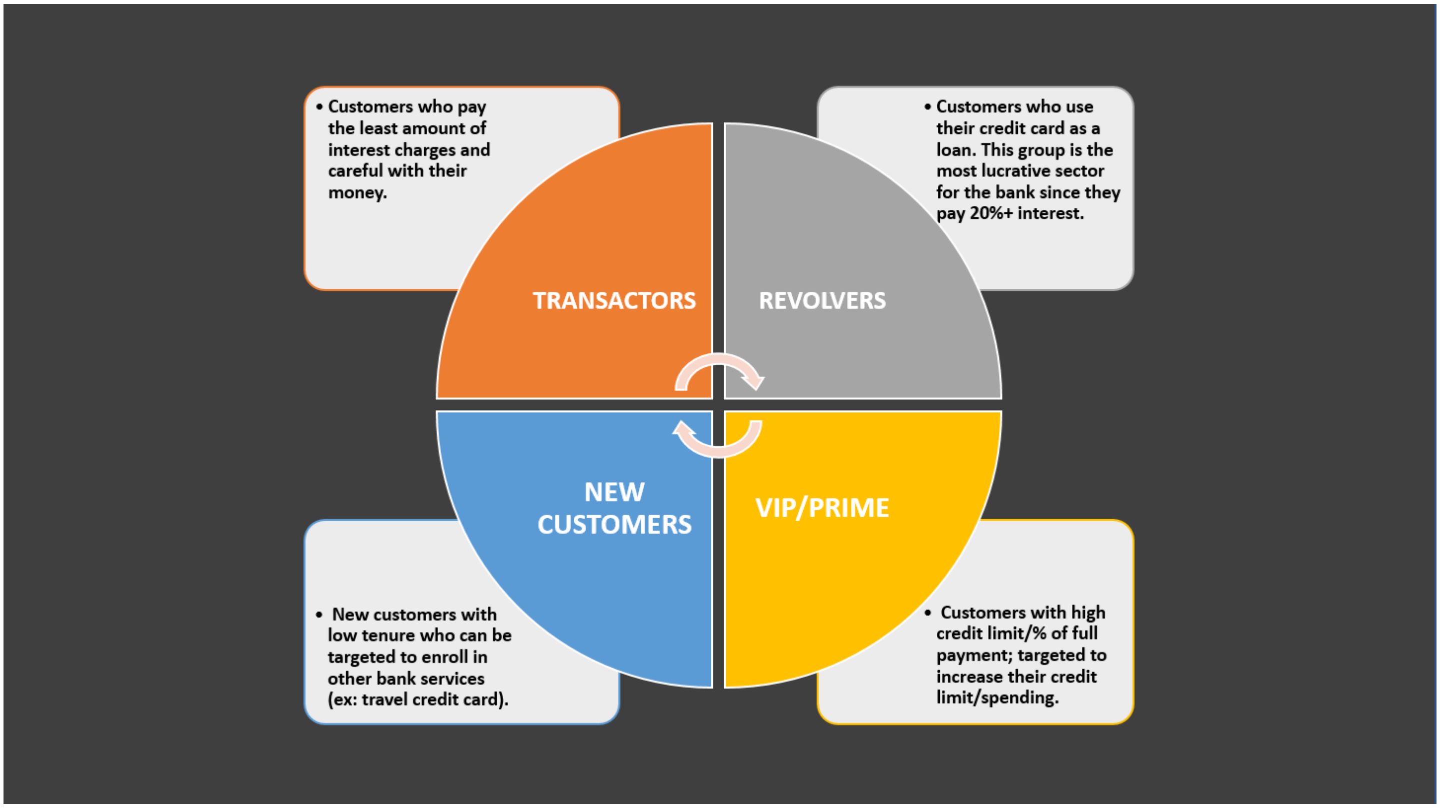
TASK #1: UNDERSTAND THE PROBLEM STATEMENT AND BUSINESS CASE


INSTRUCTOR
• Adjunct professor & online instructor
• Passionate about artificial intelligence, machine learning, and electric vehicles
• Taught 80,000+ students globally
• MBA (2018), Ph.D. (2014), M.A.Sc (2011)
A\b:
Ryan Ahmed, Ph.D.



Data Source: [https://www.kaggle.com/arjunbhasin2013/ccdata (https://www.kaggle.com/arjunbhasin2013/ccdata)](https://www.kaggle.com/arjunbhasin2013/ccdata)

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.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 and it will be hard to see them.
```

In [2]:
```
# 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, 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 = fre
quently purchased, 0 = not frequently purchased)
# ONEOFF_PURCHASES_FREQUENCY: How frequently Purchases are happening in one-go (1 = frequently pur
chased, 0 = not frequently purchased)
# PURCHASES_INSTALLMENTS_FREQUENCY: How frequently purchases in installments are being done (1 = f
requently 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
```

	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
8947	C19188	23.398673	0.833333	144.40	0.00	144.40	0.000
8948	C19189	13.457564	0.833333	0.00	0.00	0.00	36.55
8949	C19190	372.708075	0.666667	1093.25	1093.25	0.00	127.0

8950 rows × 18 columns

In [4]:
```
creditcard_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   CUST_ID                           8950 non-null   object 
 1   BALANCE                           8950 non-null   float64
 2   BALANCE_FREQUENCY                 8950 non-null   float64
 3   PURCHASES                         8950 non-null   float64
 4   ONEOFF_PURCHASES                  8950 non-null   float64
 5   INSTALLMENTS_PURCHASES            8950 non-null   float64
 6   CASH_ADVANCE                      8950 non-null   float64
 7   PURCHASES_FREQUENCY               8950 non-null   float64
 8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 11  CASH_ADVANCE_TRX                  8950 non-null   int64  
 12  PURCHASES_TRX                     8950 non-null   int64  
 13  CREDIT_LIMIT                      8949 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8637 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64  
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

In [ ]:
```
# Let's apply info and get additional insights on our dataframe
# 18 features with 8950 points
```

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 [9]:
```
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 generally low
# Average credit limit ~ 4500
# Percent of full payment is 15%
# Average tenure is 11 years
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVAI
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 [10]:
```
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 [11]:
```
creditcard_df['CASH_ADVANCE'].max()
```

```
47137.211760000006
```

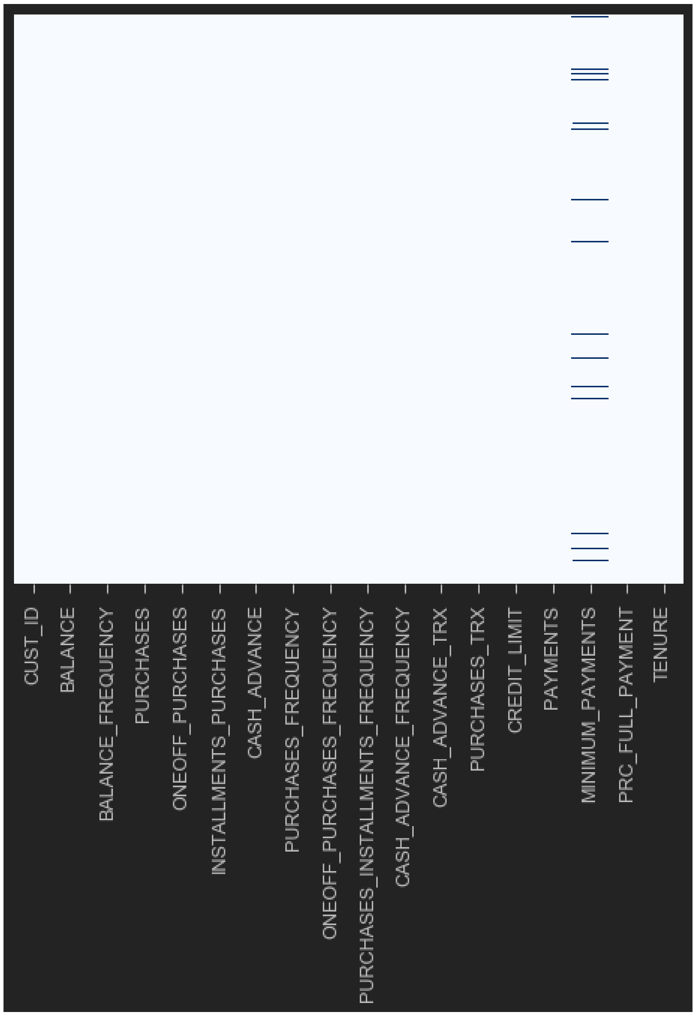
In [13]:
```
creditcard_df[creditcard_df['CASH_ADVANCE'] == 47137.211760000006]
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CAS
2159	C12226	10905.05381	1.0	431.93	133.5	298.43	4713

TASK #3: VISUALIZE AND EXPLORE DATASET

In [14]:
```
# 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 0x24fb467a288>
```



In [15]:
```
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 [17]:
```
# Fill up the missing elements with mean of the 'MINIMUM_PAYMENT'
creditcard_df.loc[(creditcard_df['MINIMUM_PAYMENTS'].isnull() == True), 'MINIMUM_PAYMENTS'] = cred
itcard_df['MINIMUM_PAYMENTS'].mean()
```

In [18]:
```
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                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

MINI CHALLENGE #3:

- Fill out missing elements in the "CREDIT_LIMIT" column
- Double check and make sure that no missing elements are present

In [19]:
```
creditcard_df.loc[(creditcard_df['CREDIT_LIMIT'].isnull() == True), 'CREDIT_LIMIT'] = creditcard_d
f['CREDIT_LIMIT'].mean()
```

In [20]:
```
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 [21]:
```
sns.heatmap(creditcard_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x24fb3047908>
```

In [22]:
```
# 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 [23]:
```
creditcard_df.drop('CUST_ID', axis = 1, inplace = True)
```

In [26]:
```
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
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 [27]:
```
n = len(creditcard_df.columns)
n
```

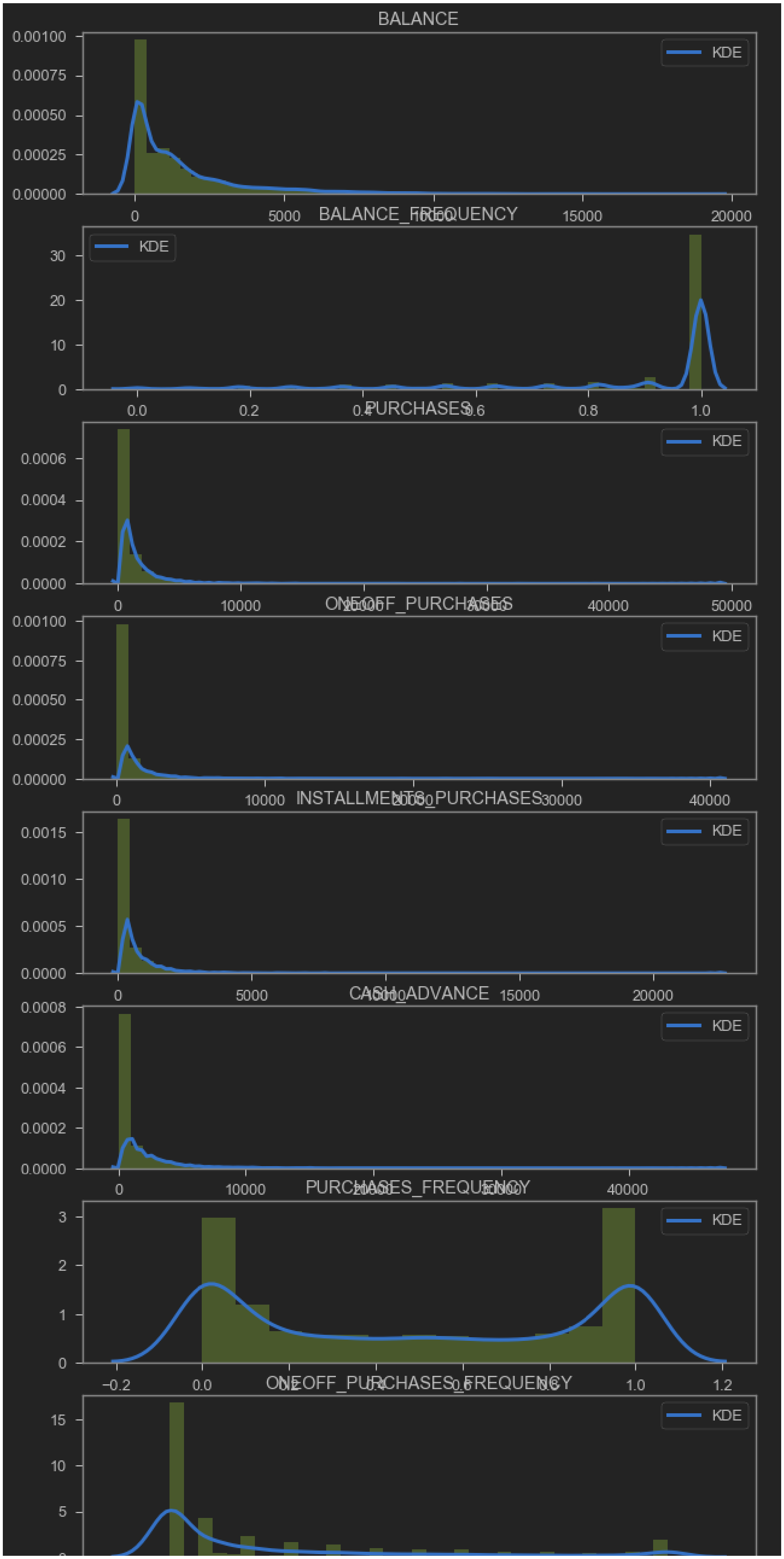
```
17
```

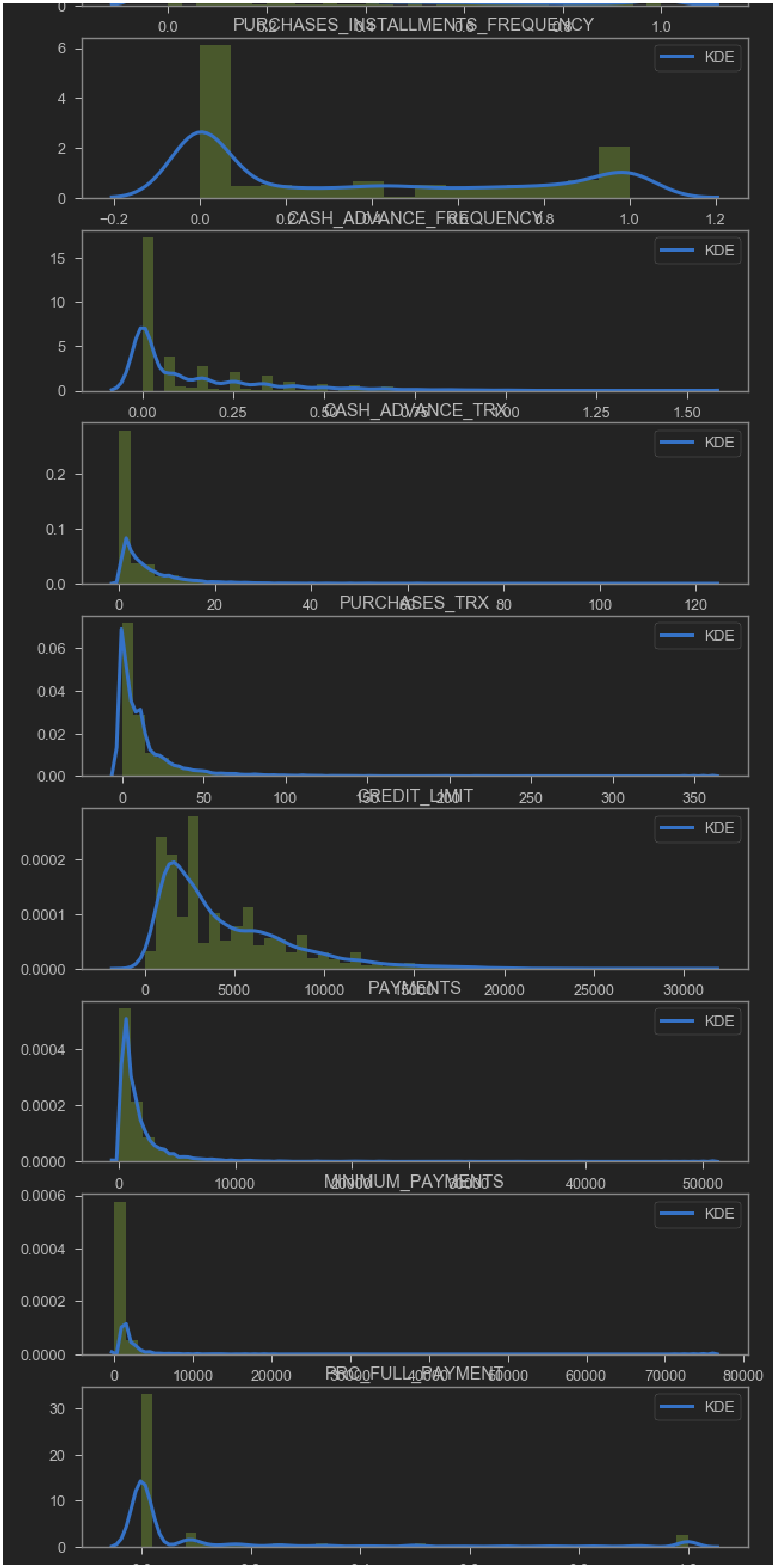
In [28]:
```
creditcard_df.columns
```

```
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')
```

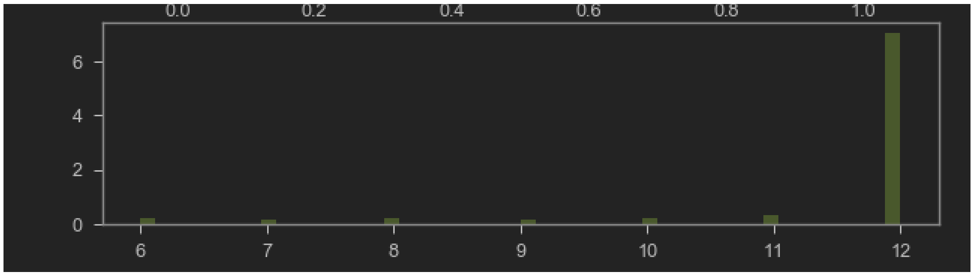
In [30]:
```
# distplot combines the matplotlib.hist function with seaborn kdeplot()
# KDE Plot represents the Kernel Density Estimate
# KDE is used for visualizing the Probability Density of a continuous variable.
# KDE demonstrates the probability density at different values in a continuous variable.

# 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 o
ff puchases or installment purchases frequently
# 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()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\anaconda31\lib\site-packages\statsmodels\nonparametric\kde.py in kdensityfft(X, kernel, bw, weights, gridsize, adjust, clip, cut, retgrid)
    450     try:
--> 451         bw = float(bw)
    452     except:

ValueError: could not convert string to float: 'scott'

During handling of the above exception, another exception occurred:

RuntimeError                              Traceback (most recent call last)
<ipython-input-30-314483609319> in <module>
     15 for i in range(len(creditcard_df.columns)):
     16   plt.subplot(17, 1, i+1)
---> 17   sns.distplot(creditcard_df[creditcard_df.columns[i]], kde_kws={"color": "b", "lw": 3, "label": "KDE"}, hist_kws={"color": "g"})
     18   plt.title(creditcard_df.columns[i])
     19

~\anaconda31\lib\site-packages\seaborn\distributions.py in distplot(a, bins, hist, kde, rug, fit, hist_kws, kde_kws, rug_kws, fit_kws, color, vertical, norm_hist, axlabel, label, ax)
    231     if kde:
    232         kde_color = kde_kws.pop("color", color)
--> 233         kdeplot(a, vertical=vertical, ax=ax, color=kde_color, **kde_kws)
    234         if kde_color != color:
    235             kde_kws["color"] = kde_color

~\anaconda31\lib\site-packages\seaborn\distributions.py in kdeplot(data, data2, shade, vertical, kernel, bw, gridsize, cut, clip, legend, cumulative, shade_lowest, cbar, cbar_ax, cbar_kws, ax, **kwargs)
    703         ax = _univariate_kdeplot(data, shade, vertical, kernel, bw,
    704                                  gridsize, cut, clip, legend, ax,
--> 705                                  cumulative=cumulative, **kwargs)
    706
    707     return ax

~\anaconda31\lib\site-packages\seaborn\distributions.py in _univariate_kdeplot(data, shade, vertical, kernel, bw, gridsize, cut, clip, legend, ax, cumulative, **kwargs)
    293         x, y = _statsmodels_univariate_kde(data, kernel, bw,
    294                                            gridsize, cut, clip,
--> 295                                            cumulative=cumulative)
    296     else:
    297         # Fall back to scipy if missing statsmodels

~\anaconda31\lib\site-packages\seaborn\distributions.py in _statsmodels_univariate_kde(data, kernel, bw, gridsize, cut, clip, cumulative)
    365     fft = kernel == "gau"
    366     kde = smnp.KDEUnivariate(data)
--> 367     kde.fit(kernel, bw, fft, gridsize=gridsize, cut=cut, clip=clip)
    368     if cumulative:
    369         grid, y = kde.support, kde.cdf

~\anaconda31\lib\site-packages\statsmodels\nonparametric\kde.py in fit(self, kernel, bw, fft, weights, gridsize, adjust, cut, clip)
    138             density, grid, bw = kdensityfft(endog, kernel=kernel, bw=bw,
    139                     adjust=adjust, weights=weights, gridsize=gridsize,
--> 140                     clip=clip, cut=cut)
    141         else:
    142             density, grid, bw = kdensity(endog, kernel=kernel, bw=bw,

~\anaconda31\lib\site-packages\statsmodels\nonparametric\kde.py in kdensityfft(X, kernel, bw, weights, gridsize, adjust, clip, cut, retgrid)
    451         bw = float(bw)
    452     except:
--> 453         bw = bandwidths.select_bandwidth(X, bw, kern) # will cross-val fit this pattern?
    454     bw *= adjust
    455

~\anaconda31\lib\site-packages\statsmodels\nonparametric\bandwidths.py in select_bandwidth(x, bw, kernel)
    172         # eventually this can fall back on another selection criterion.
    173         err = "Selected KDE bandwidth is 0. Cannot estiamte density."
--> 174         raise RuntimeError(err)
    175     else:
    176         return bandwidth

RuntimeError: Selected KDE bandwidth is 0. Cannot estiamte density.
```


BALANCE
KDE
BALANCE_FREQUENCY
KDE
PURCHASES
KDE
ONEOFF_PURCHASES
KDE
INSTALLMENTS_PURCHASES
KDE
CASH_ADVANCE
KDE
PURCHASES_FREQUENCY
KDE
ONEOFF_PURCHASES_FREQUENCY
KDE



PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS
PRC_FULL_PAYMENT
KDE



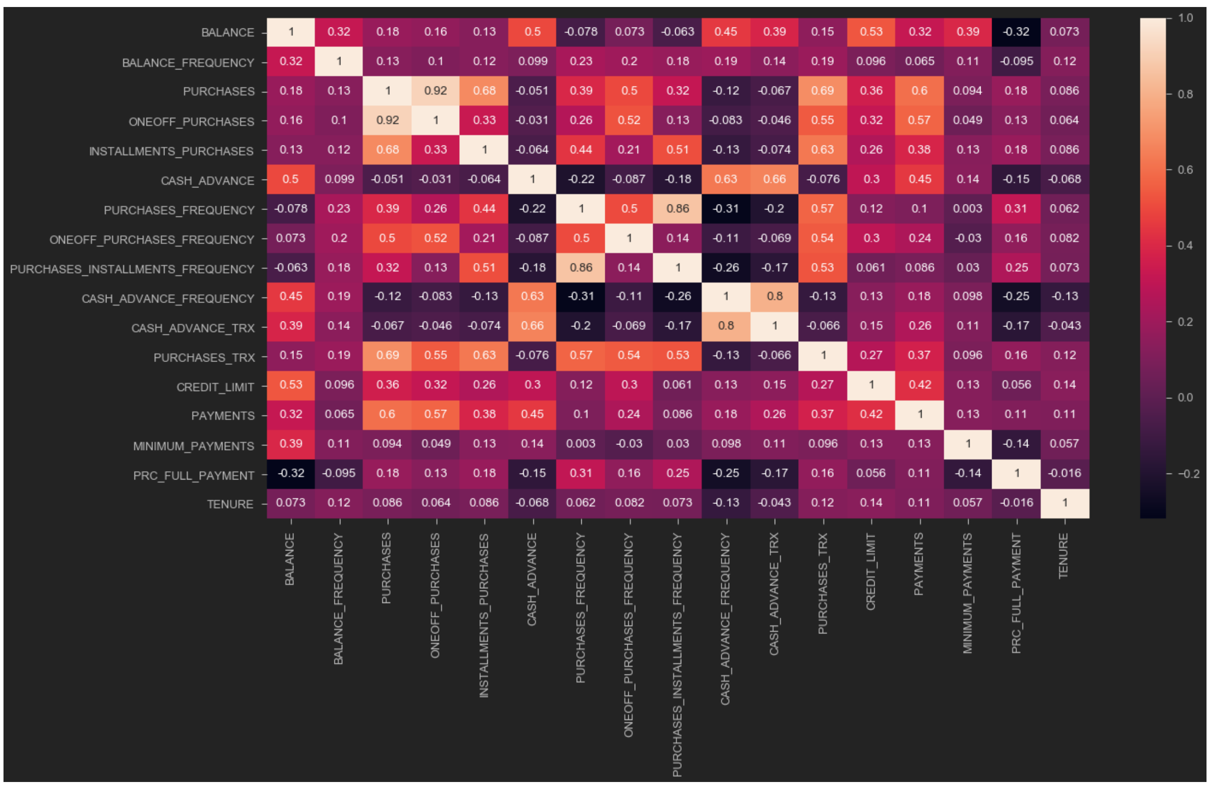
MINI CHALLENGE #5:

- Obtain the correlation matrix between features

In [32]:

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

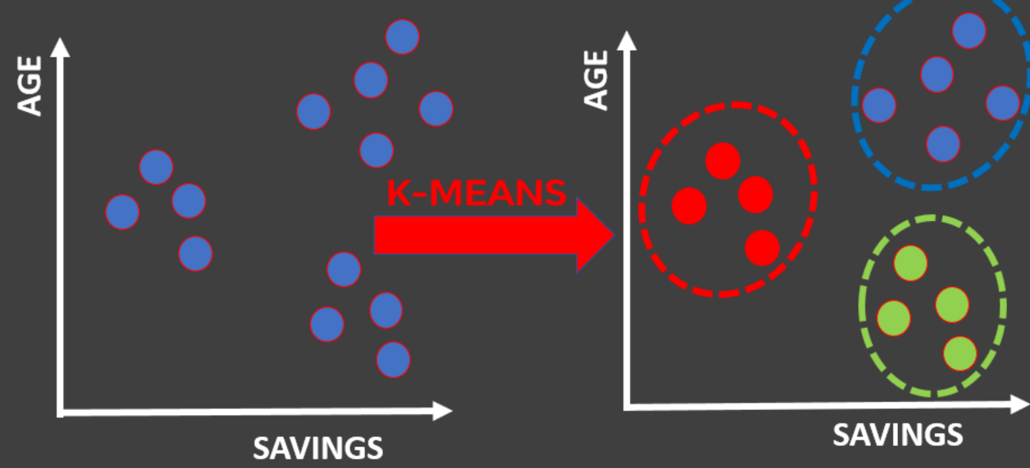
```
<matplotlib.axes._subplots.AxesSubplot at 0x24fb3c41688>
```



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

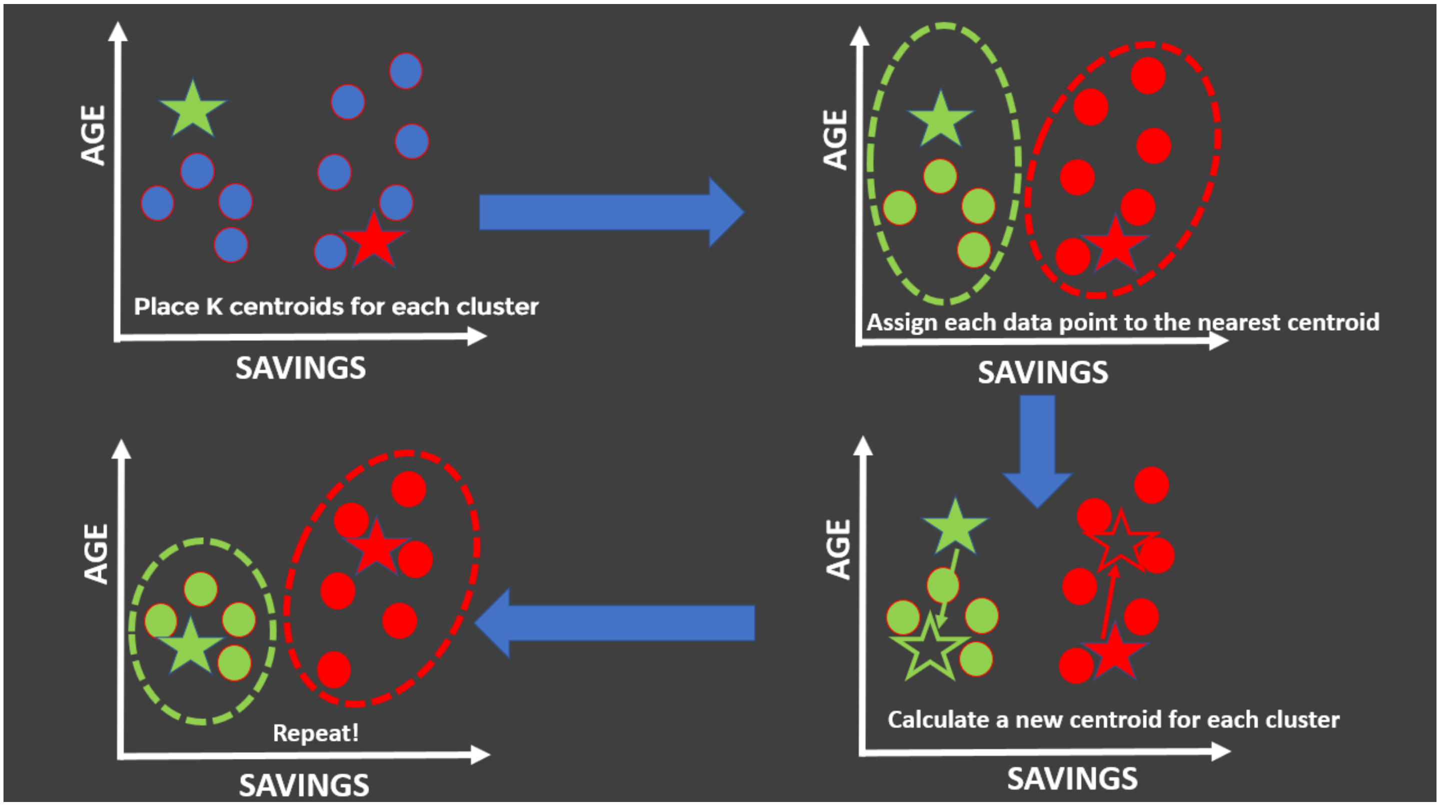
K-MEANS INTUITION

- 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.

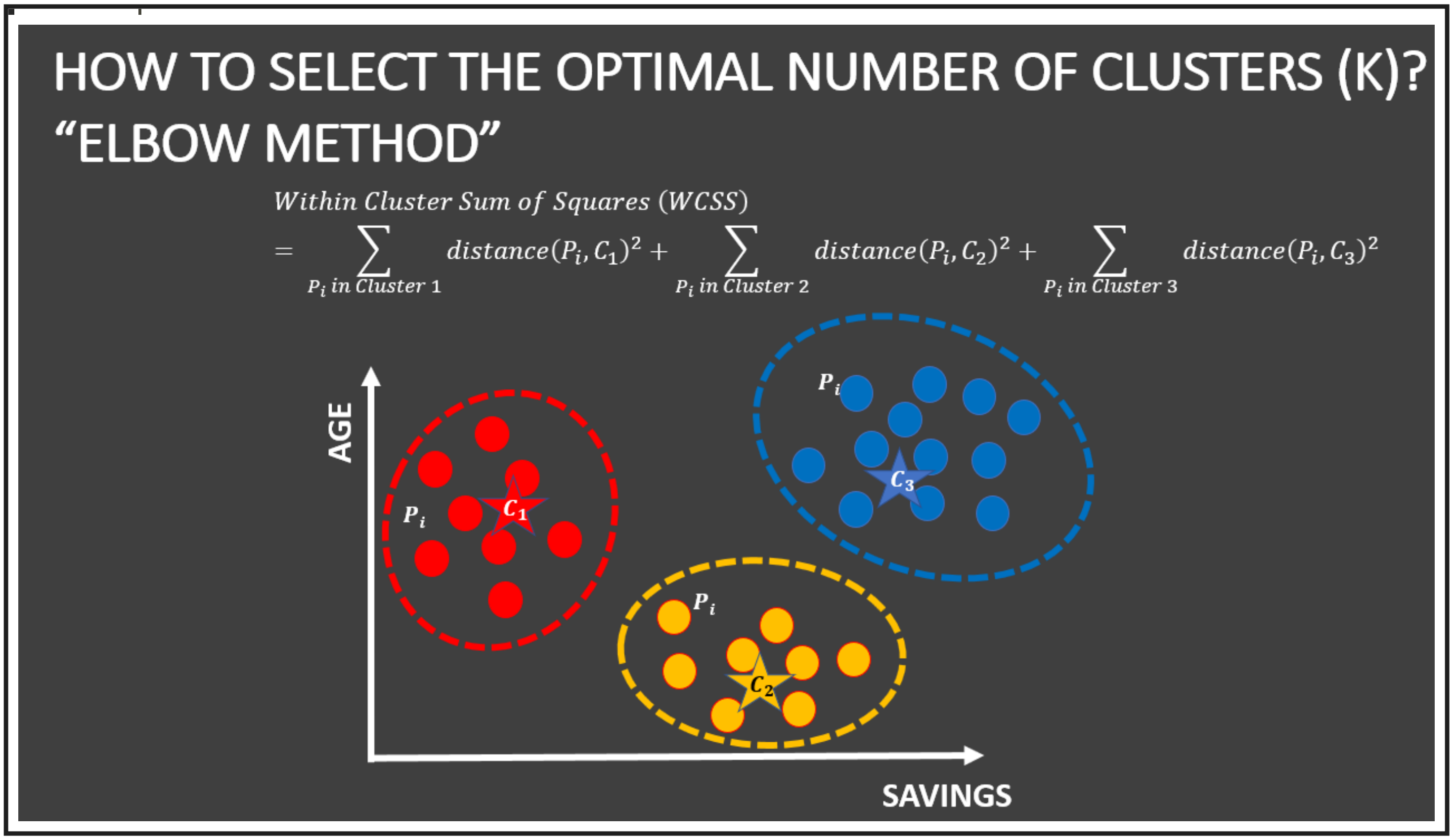


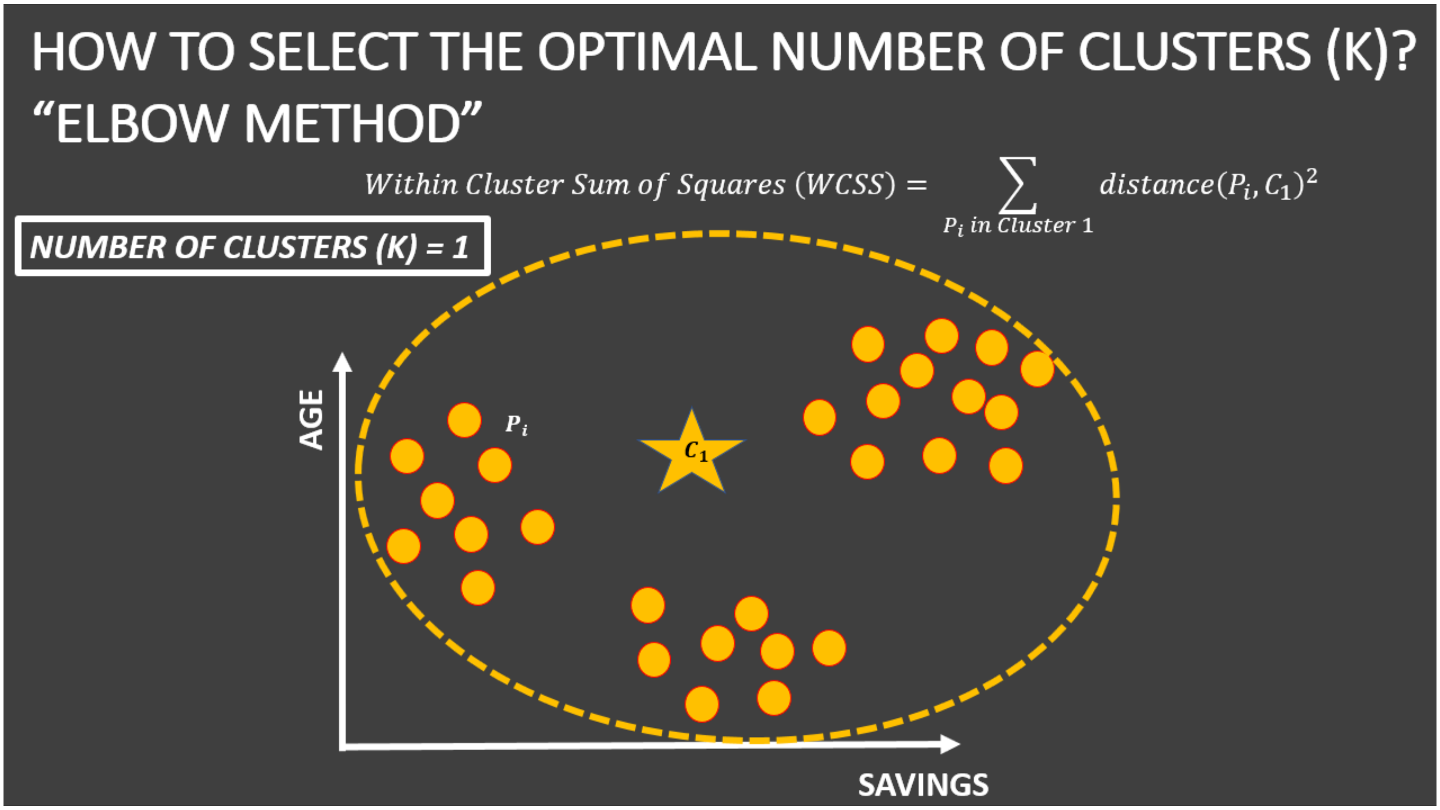
MINI CHALLENGE #6:

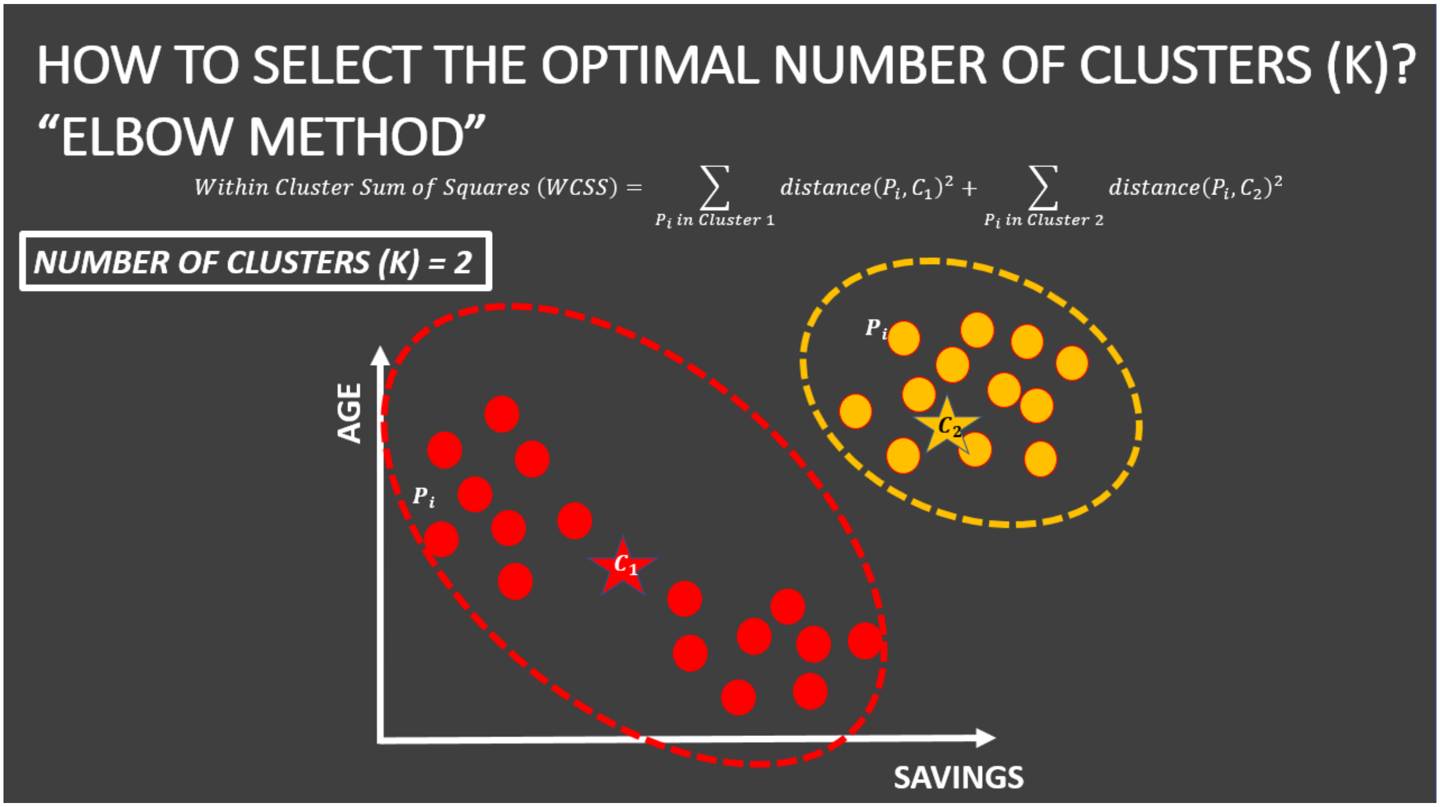
- Which of the following conditions could terminate the K-means clustering algorithm? (choose 2)
  - K-means terminates after a fixed number of iterations is reached
  - K-means terminates when the number of clusters does not increase between iterations
  - K-means terminates when the centroid locations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

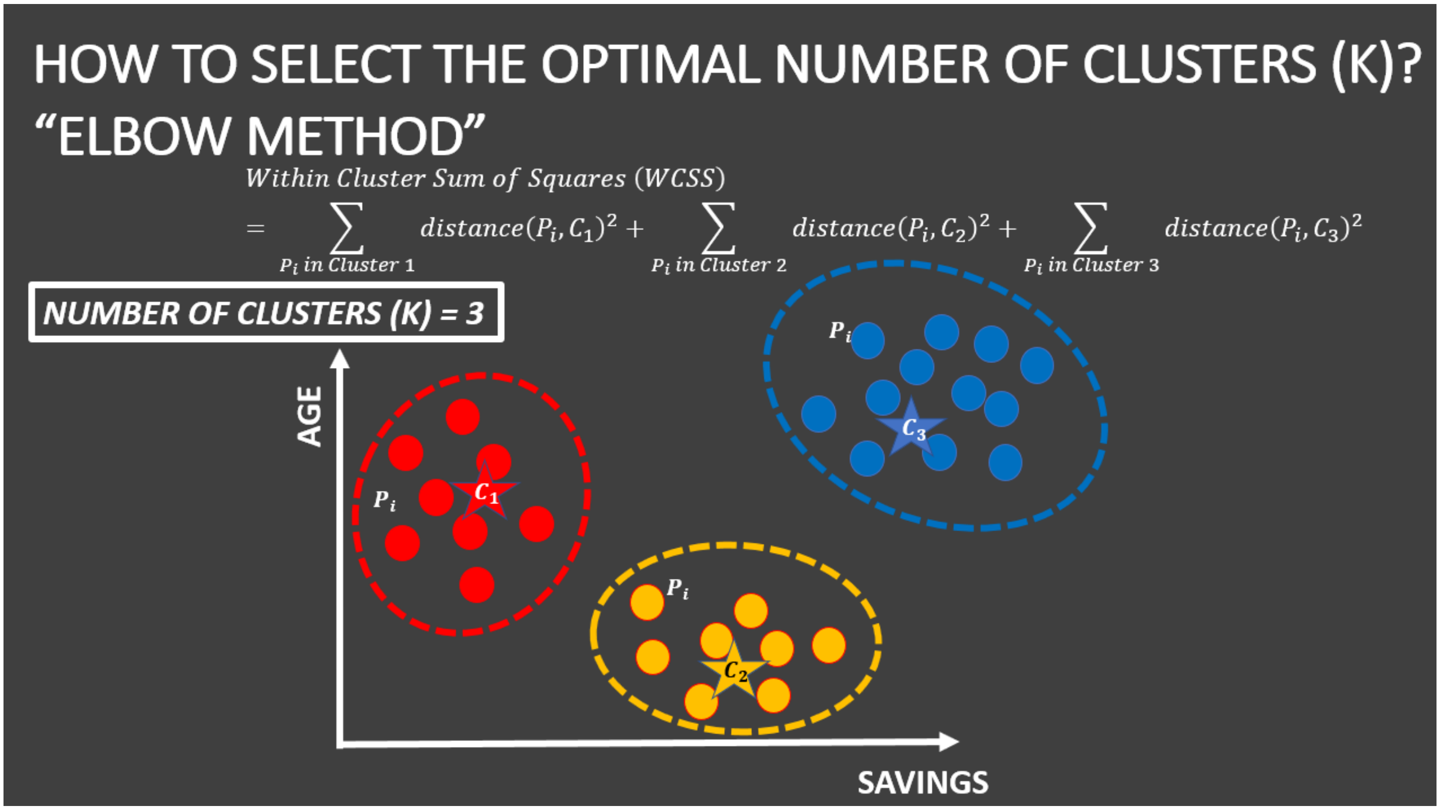
In [ ]:
```
// K-means terminates after a fixed number of iterations is reached
// K-means terminates when the centroid locations do not change between iterations
```

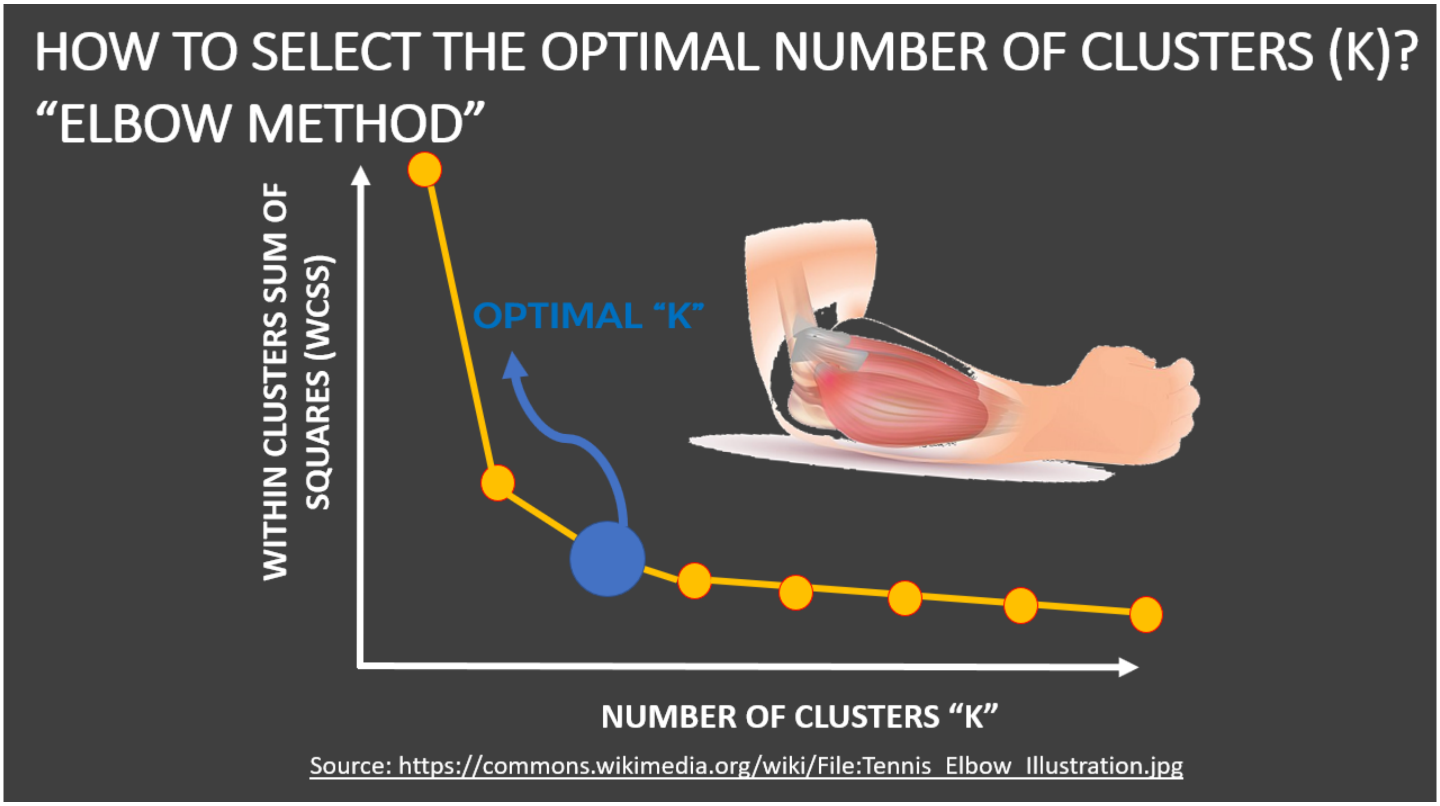
TASK #5: LEARN HOW TO OBTAIN THE OPTIMAL NUMBER OF CLUSTERS (ELBOW METHOD)


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



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://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/)

In [33]:
```
# Let's scale the data first
scaler = StandardScaler()
creditcard_df_scaled = scaler.fit_transform(creditcard_df)
```

In [34]:
```
creditcard_df_scaled.shape
```

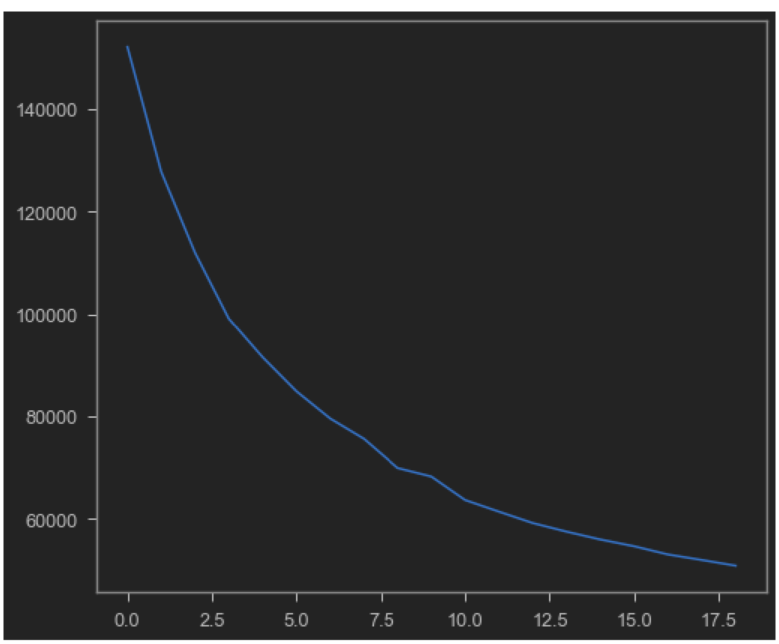
```
(8950, 17)
```

In [38]:
```
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 [39]:
```
# 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 0x24fb3b82308>]
```



MINI CHALLENGE #7:

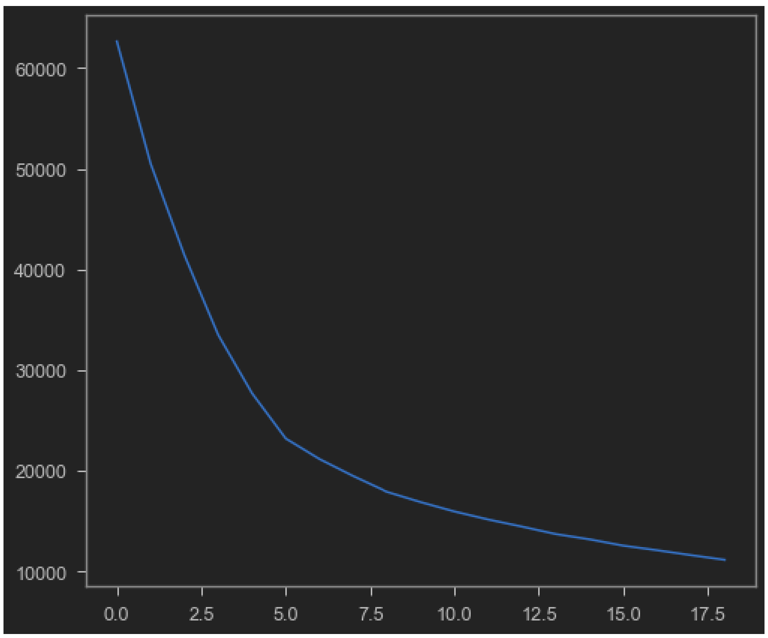
- Let's assume that our data only consists of the first 7 columns of "creditcard_df_scaled", what is the optimal number of clusters would be in this case? modify the code and rerun the cells.

In [43]:
```
creditcard_df_scaled[:, :7].shape
```

```
(8950, 7)
```

In [42]:
```
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 0x24fb5e25888>]
```



TASK #7: APPLY K-MEANS METHOD

In [44]:
```
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	0.009508	0.401739	-0.343430	-0.223474	-0.401364	-0.101925
1	-0.368878	0.333102	-0.041519	-0.230904	0.325868	-0.367172
2	1.488505	0.403475	7.413638	6.553369	5.486972	0.028557
3	-0.335429	-0.342203	-0.283935	-0.208973	-0.287081	0.064839
4	0.143949	0.430997	0.976283	0.923928	0.610965	-0.306762
5	1.675303	0.393689	-0.196573	-0.147680	-0.193575	1.996838
6	-0.701924	-2.134261	-0.306883	-0.230464	-0.302103	-0.322950

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 cha
rges and careful with their money, Cluster with lowest balance ($104) and cash advance ($303), Per
centage of full payment = 23%
# Second customers cluster (revolvers) who use credit card as a loan (most lucrative sector): high
est 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 paymen
t, target for increase credit limit and increase spending habits
# 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	1584.264568	0.972439	269.461234	221.516190	48.119482	765.130009
1	796.685924	0.956179	914.498711	209.184157	705.745881	208.893586
2	4662.671853	0.972850	16842.556892	11469.688108	5372.868784	1038.757441
3	866.307935	0.796206	396.572925	245.585564	151.464308	1114.841673
4	1864.092559	0.979370	3089.048627	2125.968921	963.555897	335.575951
5	5051.477573	0.970532	583.224191	347.318663	236.019764	5166.333011
6	103.479822	0.371684	347.544601	209.914180	137.880042	301.630330

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

```
(8950,)
```

In [50]:
```
labels.max()
```

```
6
```

In [51]:
```
labels.min()
```

```
0
```

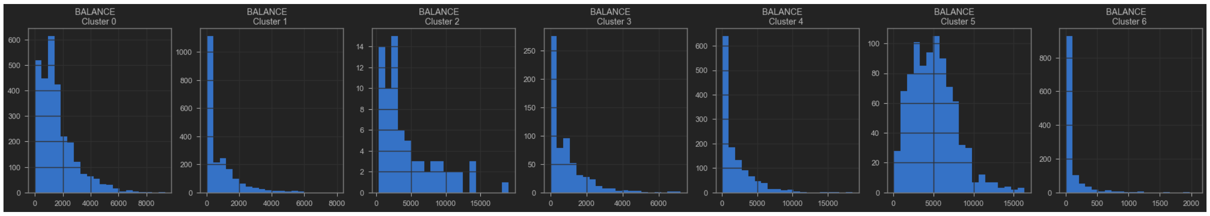
In [52]:
```
y_kmeans = kmeans.fit_predict(creditcard_df_scaled)
y_kmeans
```

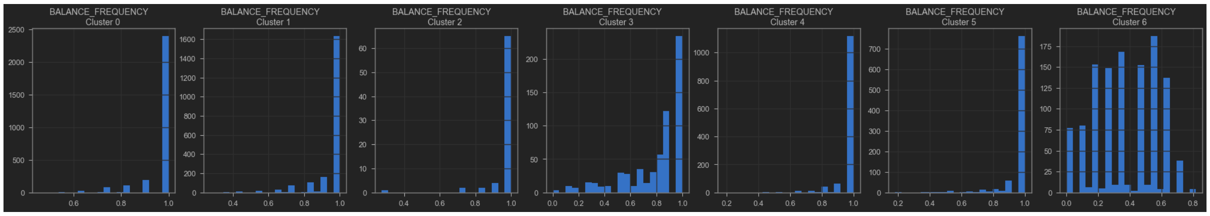
```
array([3, 6, 1, ..., 0, 3, 4])
```

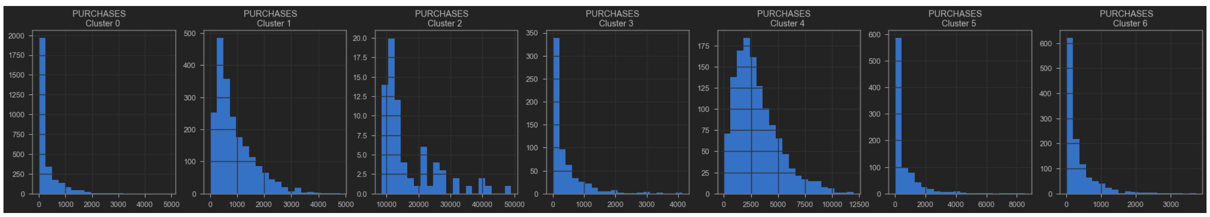
In [53]:
```
# concatenate the clusters labels to our original dataframe
creditcard_df_cluster = pd.concat([creditcard_df, pd.DataFrame({'cluster':labels})], axis = 1)
creditcard_df_cluster.head()
```

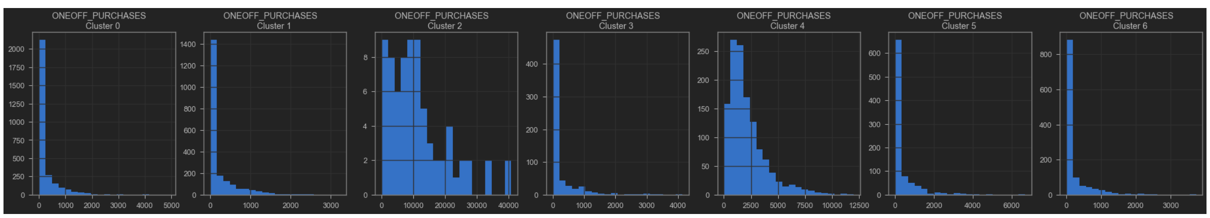
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
0	40.900749	0.818182	95.40	0.00	95.4	0.000000
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017
4	817.714335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

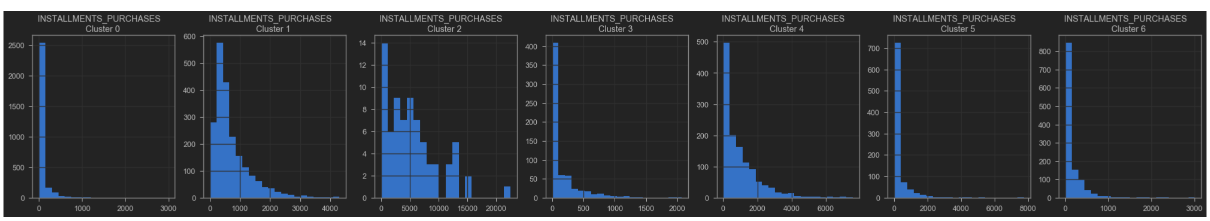
In [54]:
```
# Plot the histogram of various clusters
for i in creditcard_df.columns:
  plt.figure(figsize = (35, 5))
  for j in range(7):
    plt.subplot(1,7,j+1)
    cluster = creditcard_df_cluster[creditcard_df_cluster['cluster'] == j]
    cluster[i].hist(bins = 20)
    plt.title('{}    \nCluster {} '.format(i,j))

  plt.show()
```

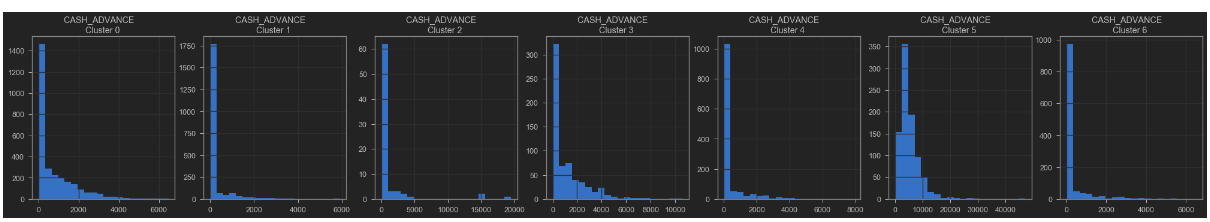

BALANCE Cluster 0
BALANCE Cluster 1
BALANCE Cluster 2
BALANCE Cluster 3
BALANCE Cluster 4
BALANCE Cluster 5
BALANCE Cluster 6

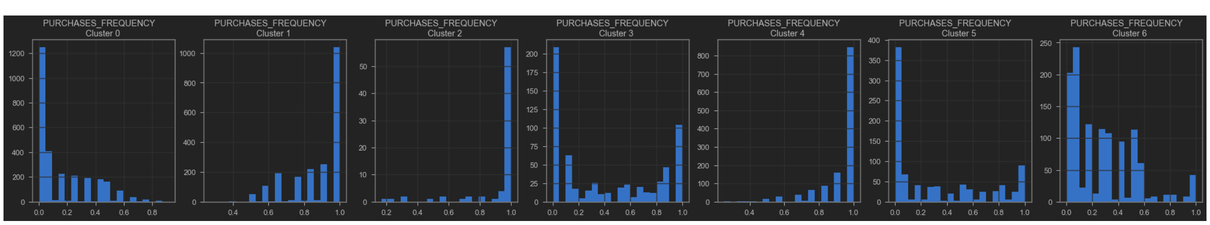

BALANCE_FREQUENCY Cluster 0
BALANCE_FREQUENCY Cluster 1
BALANCE_FREQUENCY Cluster 2
BALANCE_FREQUENCY Cluster 3
BALANCE_FREQUENCY Cluster 4
BALANCE_FREQUENCY Cluster 5
BALANCE_FREQUENCY Cluster 6

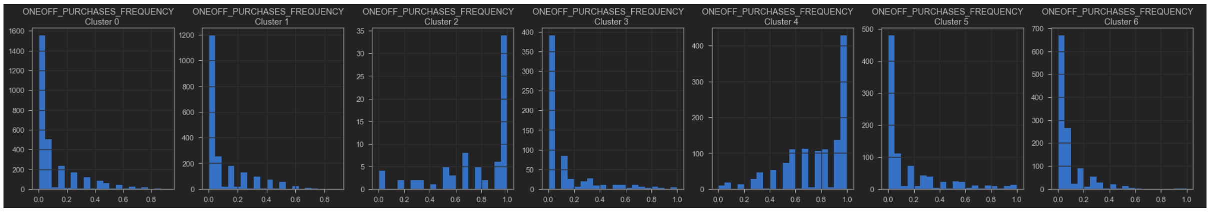

PURCHASES Cluster 0
PURCHASES Cluster 1
PURCHASES Cluster 2
PURCHASES Cluster 3
PURCHASES Cluster 4
PURCHASES Cluster 5
PURCHASES Cluster 6

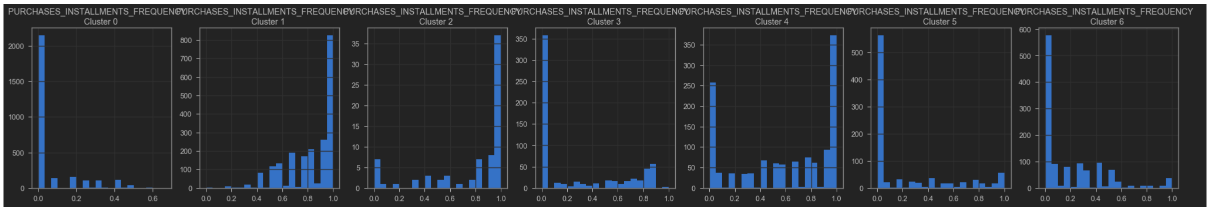

ONEOFF_PURCHASES Cluster 0
ONEOFF_PURCHASES Cluster 1
ONEOFF_PURCHASES Cluster 2
ONEOFF_PURCHASES Cluster 3
ONEOFF_PURCHASES Cluster 4
ONEOFF_PURCHASES Cluster 5
ONEOFF_PURCHASES Cluster 6

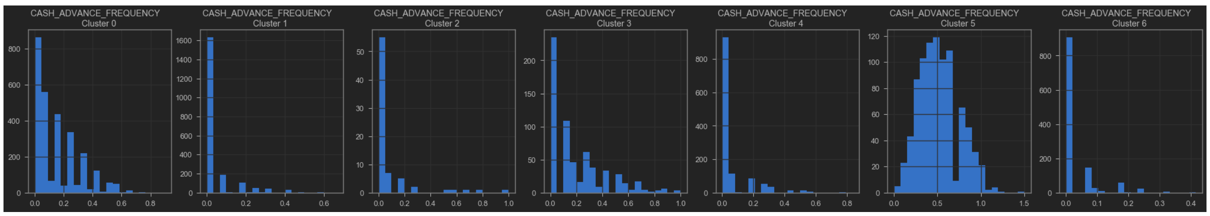

INSTALLMENTS_PURCHASES Cluster 0
INSTALLMENTS_PURCHASES Cluster 1
INSTALLMENTS_PURCHASES Cluster 2
INSTALLMENTS_PURCHASES Cluster 3
INSTALLMENTS_PURCHASES Cluster 4
INSTALLMENTS_PURCHASES Cluster 5
INSTALLMENTS_PURCHASES Cluster 6

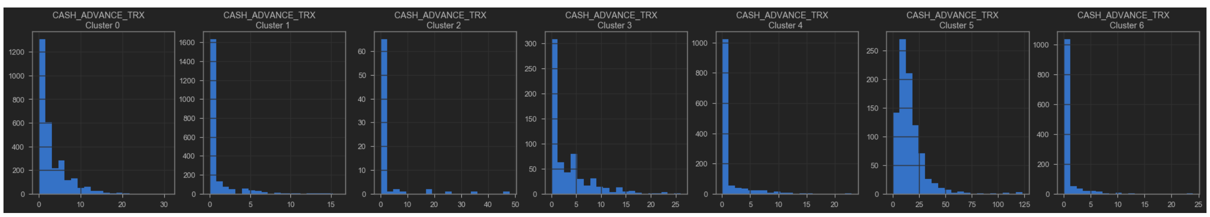

CASH_ADVANCE Cluster 0
CASH_ADVANCE Cluster 1
CASH_ADVANCE Cluster 2
CASH_ADVANCE Cluster 3
CASH_ADVANCE Cluster 4
CASH_ADVANCE Cluster 5
CASH_ADVANCE Cluster 6

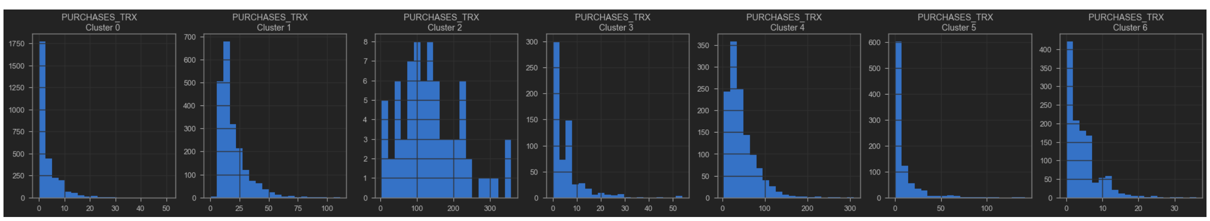

PURCHASES_FREQUENCY Cluster 0
PURCHASES_FREQUENCY Cluster 1
PURCHASES_FREQUENCY Cluster 2
PURCHASES_FREQUENCY Cluster 3
PURCHASES_FREQUENCY Cluster 4
PURCHASES_FREQUENCY Cluster 5
PURCHASES_FREQUENCY Cluster 6

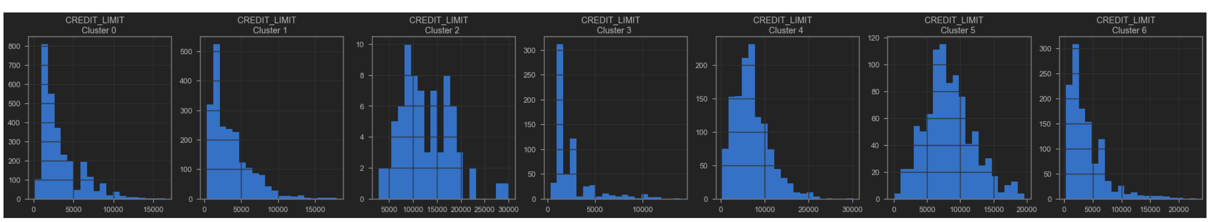

ONEOFF_PURCHASES_FREQUENCY Cluster 0
ONEOFF_PURCHASES_FREQUENCY Cluster 1
ONEOFF_PURCHASES_FREQUENCY Cluster 2
ONEOFF_PURCHASES_FREQUENCY Cluster 3
ONEOFF_PURCHASES_FREQUENCY Cluster 4
ONEOFF_PURCHASES_FREQUENCY Cluster 5
ONEOFF_PURCHASES_FREQUENCY Cluster 6

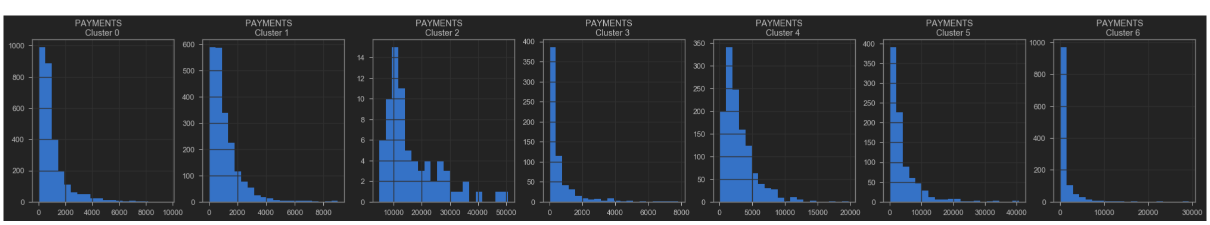

PURCHASES_INSTALLMENTS_FREQUENCY Cluster 0
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 1
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 2
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 3
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 4
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 5
PURCHASES_INSTALLMENTS_FREQUENCY Cluster 6

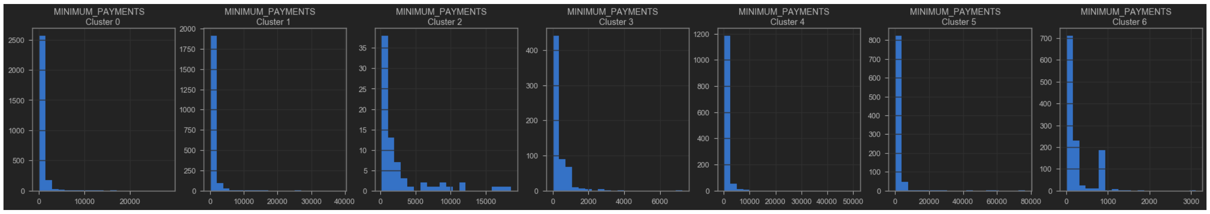

CASH_ADVANCE_FREQUENCY Cluster 0
CASH_ADVANCE_FREQUENCY Cluster 1
CASH_ADVANCE_FREQUENCY Cluster 2
CASH_ADVANCE_FREQUENCY Cluster 3
CASH_ADVANCE_FREQUENCY Cluster 4
CASH_ADVANCE_FREQUENCY Cluster 5
CASH_ADVANCE_FREQUENCY Cluster 6

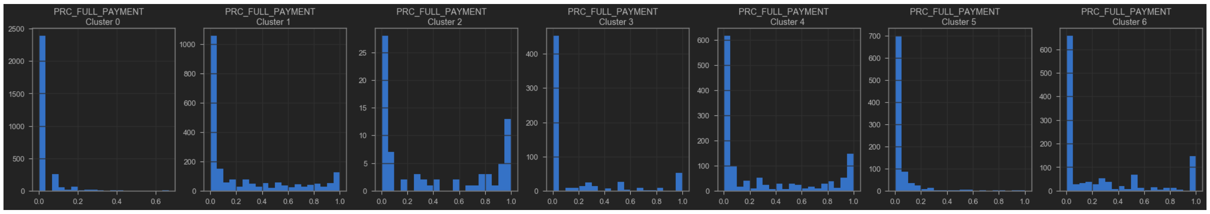

CASH_ADVANCE_TRX Cluster 0
CASH_ADVANCE_TRX Cluster 1
CASH_ADVANCE_TRX Cluster 2
CASH_ADVANCE_TRX Cluster 3
CASH_ADVANCE_TRX Cluster 4
CASH_ADVANCE_TRX Cluster 5
CASH_ADVANCE_TRX Cluster 6

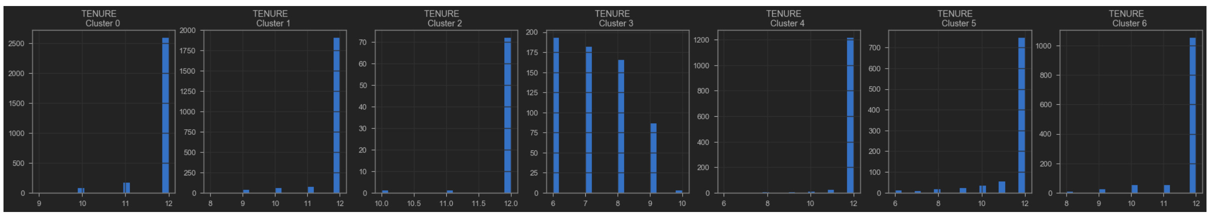

PURCHASES_TRX Cluster 0
PURCHASES_TRX Cluster 1
PURCHASES_TRX Cluster 2
PURCHASES_TRX Cluster 3
PURCHASES_TRX Cluster 4
PURCHASES_TRX Cluster 5
PURCHASES_TRX Cluster 6


CREDIT_LIMIT Cluster 0
CREDIT_LIMIT Cluster 1
CREDIT_LIMIT Cluster 2
CREDIT_LIMIT Cluster 3
CREDIT_LIMIT Cluster 4
CREDIT_LIMIT Cluster 5
CREDIT_LIMIT Cluster 6


PAYMENTS Cluster 0
PAYMENTS Cluster 1
PAYMENTS Cluster 2
PAYMENTS Cluster 3
PAYMENTS Cluster 4
PAYMENTS Cluster 5
PAYMENTS Cluster 6



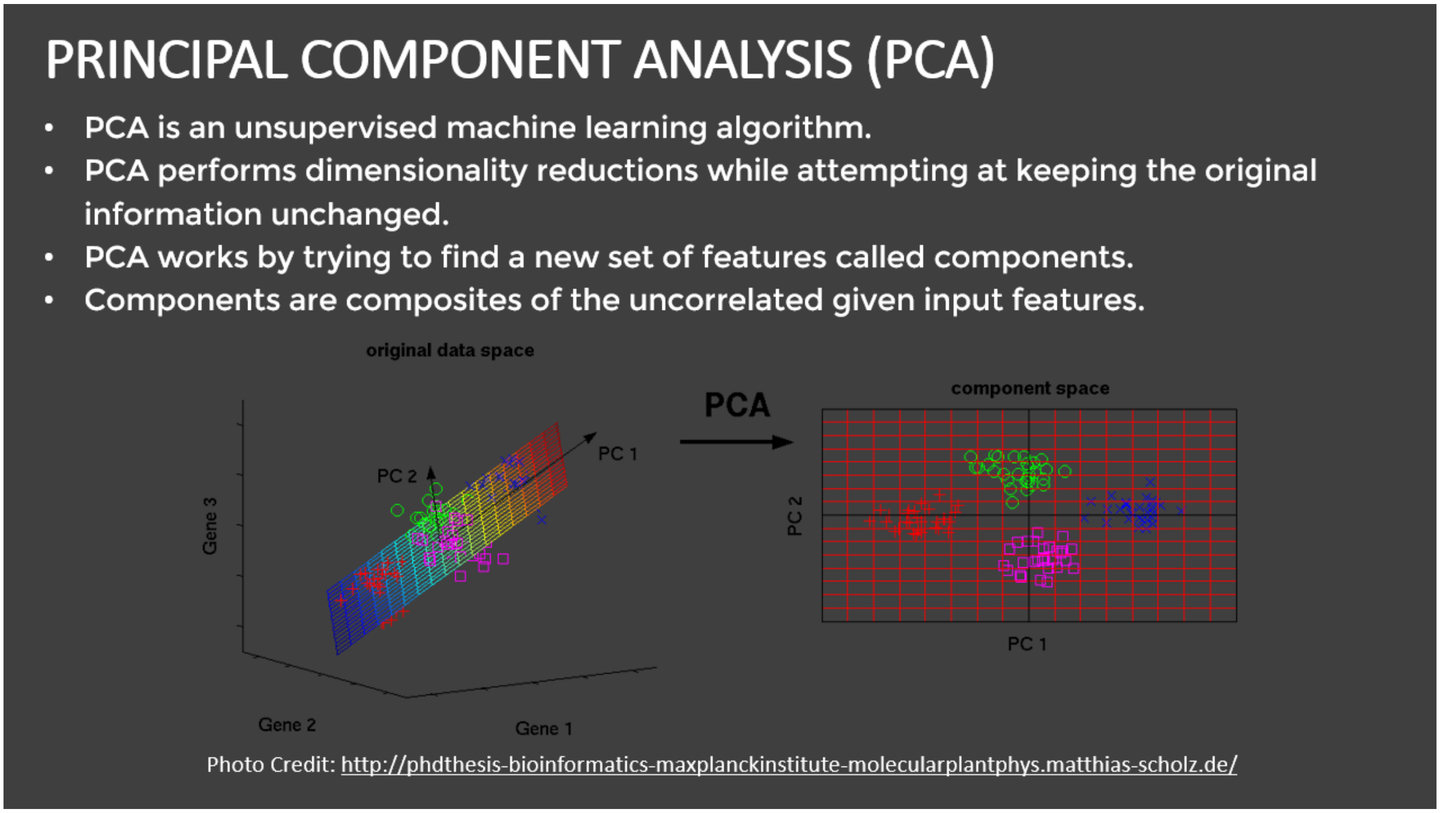




MINI CHALLENGE #8:

- Repeat the same procedure with 8 clusters instead of 7

TASK 8: APPLY PRINCIPAL COMPONENT ANALYSIS AND VISUALIZE THE RESULTS



In [56]:

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

```
array([[-1.68222052, -1.0764523 ],
       [-1.13829595,  2.50646726],
       [ 0.96968604, -0.38350322],
       ...,
       [-0.92620362, -1.81078458],
       [-2.33655245, -0.65797202],
       [-0.55642189, -0.40046605]])
```

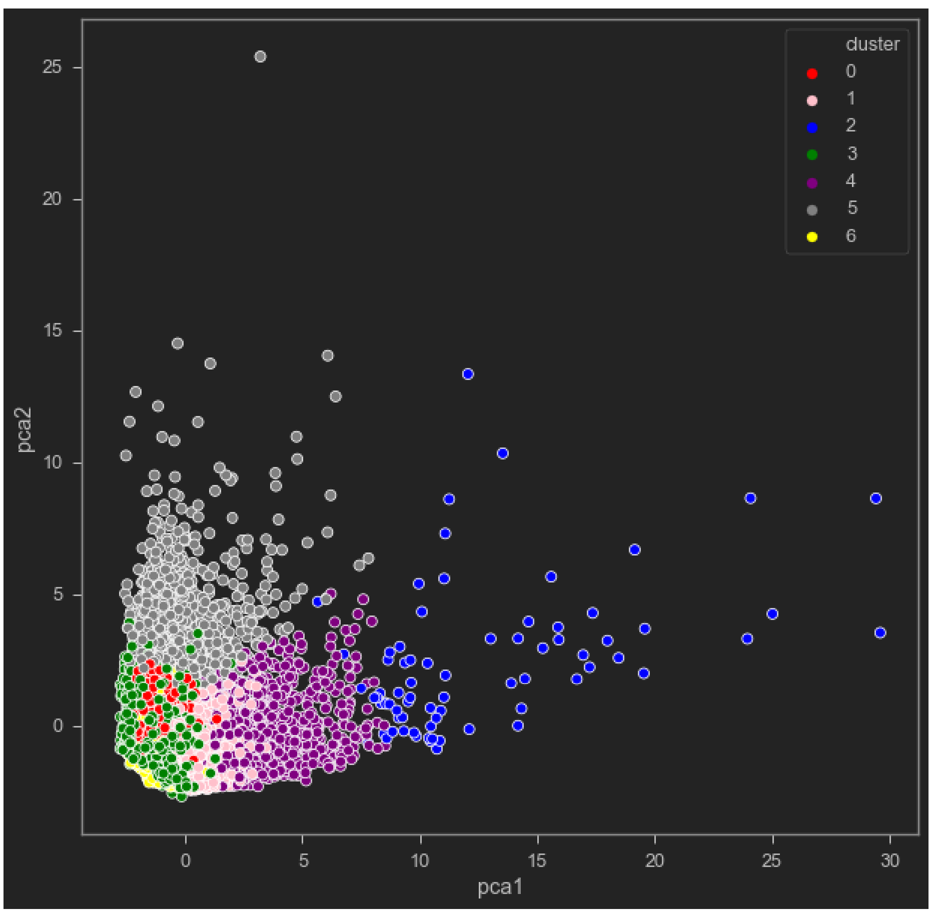
In [57]:
```
# Create a dataframe with the two components
pca_df = pd.DataFrame(data = principal_comp, columns =['pca1','pca2'])
pca_df.head()
```

	pca1	pca2
0	-1.682221	-1.076452
1	-1.138296	2.506467
2	0.969686	-0.383503
3	-0.873628	0.043164
4	-1.599434	-0.688581

In [58]:
```
# 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	-1.682221	-1.076452	0
1	-1.138296	2.506467	5
2	0.969686	-0.383503	4
3	-0.873628	0.043164	0
4	-1.599434	-0.688581	0

In [61]:
```
plt.figure(figsize=(10,10))
ax = sns.scatterplot(x="pca1", y="pca2", hue = "cluster", data = pca_df, palette =['red','pink','b
lue','green','purple','gray','yellow'])
plt.show()
```



MINI CHALLENGE #9:

- Repeat task #7 and #8 with number of clusters = 7 and 4

EXCELLENT JOB! YOU SHOULD BE PROUD OF YOUR NEWLY ACQUIRED SKILLS

MINI CHALLENGE SOLUTIONS

MINI CHALLENGE #1

In [ ]:
```
# Average, minimum and maximum balance amounts
print('The average, minimum and maximum balance amount are:', creditcard_df['BALANCE'].mean(), creditcard_df['BALANCE'].min(), creditcard_df['BALANCE'].max())
```

MINI CHALLENGE #2

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

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

In [ ]:
```
# 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'] == 47137.211760000006]
```

MINI CHALLENGE #3

In [ ]:
```
# Fill up the missing elements with mean of the 'CREDIT_LIMIT'
creditcard_df.loc[(creditcard_df['CREDIT_LIMIT'].isnull() == True), 'CREDIT_LIMIT'] = creditcard_df['CREDIT_LIMIT'].mean()
sns.heatmap(creditcard_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

MINI CHALLENGE #4

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

MINI CHALLENGE #5

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

# 'PURCHASES' have high correlation between one-off purchases, 'installment purchases, purchase transactions, credit limit and payments.
# Strong Positive Correlation between 'PURCHASES_FREQUENCY' and 'PURCHASES_INSTALLMENT_FREQUENCY'
```

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 (True)
  - K-means terminates when the number of clusters does not increase between iterations (False)
  - K-means terminates when the centroid locations do not change between iterations (True)

MINI CHALLENGE #7:

In [ ]:
```
# code modification
kmeans.fit(creditcard_df_scaled[:,:7])
# optimal number of clusters would be = 5
```

MINI CHALLENGE #8 & #9:

- simply change the values requested in the question and rerun the cells