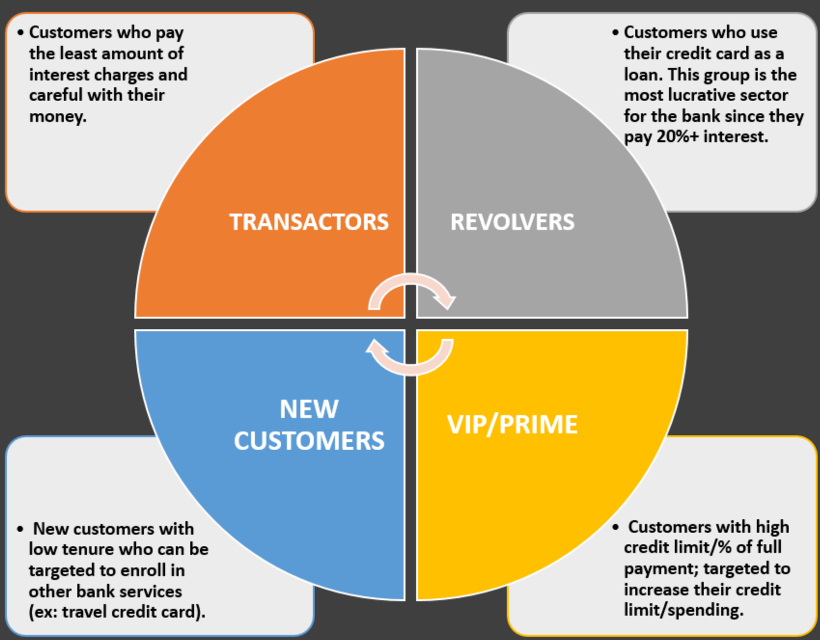
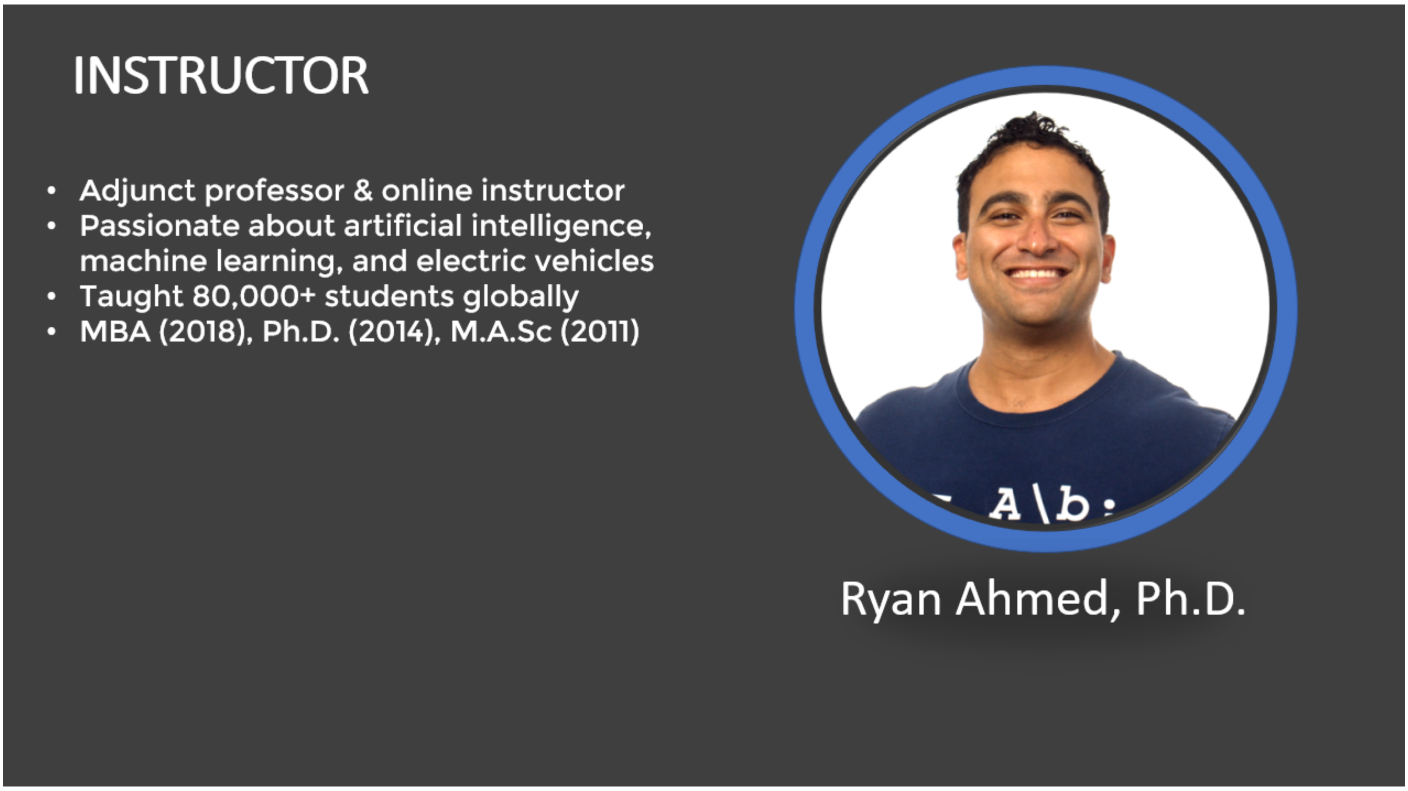
TASK #1: UNDERSTAND THE PROBLEM STATEMENT AND BUSINESS CASE

- 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

In [ ]:





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 [85]:

```
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
```

In [86]:

```
# 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 = frequently purchased, 0 = not frequently
# 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 do
# 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 [87]:

```
creditcard_df
```

Out[87]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENC
0	C10001	40.900749	0.818182	95.40	0.00	95.40	0.000000	0.16666
1	C10002	3202.467416	0.909091	0.00	0.00	0.00	6442.945483	0.0000(
2	C10003	2495.148862	1.000000	773.17	773.17	0.00	0.000000	1.0000(
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.00	205.788017	0.0833:
4	C10005	817.714335	1.000000	16.00	16.00	0.00	0.000000	0.0833:
...	...	...	...	...	...	...	...	
8945	C19186	28.493517	1.000000	291.12	0.00	291.12	0.000000	1.0000(
8946	C19187	19.183215	1.000000	300.00	0.00	300.00	0.000000	1.0000(
8947	C19188	23.398673	0.833333	144.40	0.00	144.40	0.000000	0.8333:
8948	C19189	13.457564	0.833333	0.00	0.00	0.00	36.558778	0.0000(

In [88]:

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

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

MINI CHALLENGE #1:

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

In [ ]:

```

```

In [89]:

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

Out[89]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQ
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	0
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1

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 [ ]:

```

```

In [ ]:

```

```

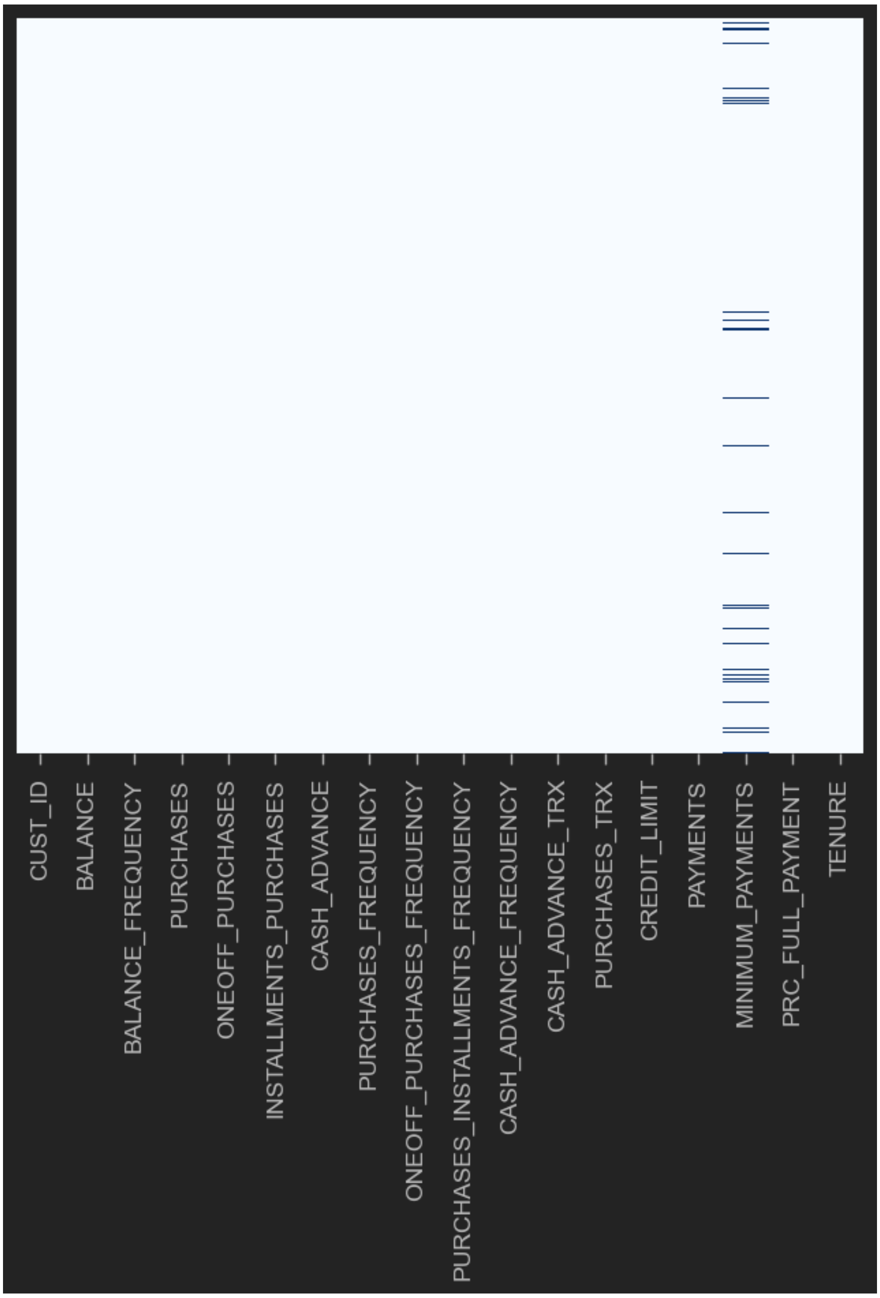
TASK #3: VISUALIZE AND EXPLORE DATASET

In [90]:

```
# 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")
```

Out[90]:

```
<AxesSubplot:>
```



In [91]:

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

Out[91]:

```
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 [92]:

```
# 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()
```

MINI CHALLENGE #3:

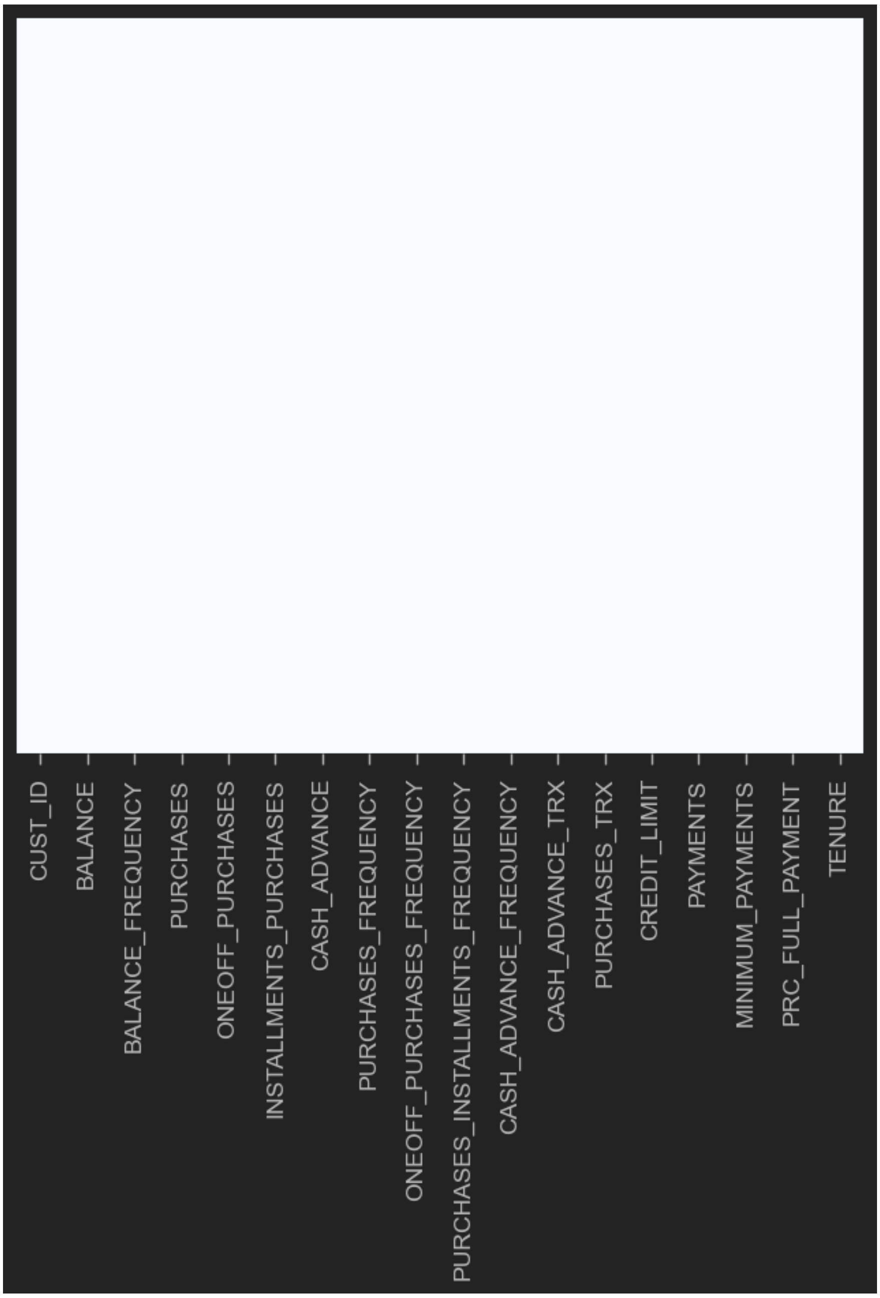
- Fill out missing elements in the "CREDIT_LIMIT" column
- Double check and make sure that no missing elements are present

In [93]:

```
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")
```

Out[93]:

<AxesSubplot:>



In [ ]:

```

```

In [94]:

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

Out[94]:

0

MINI CHALLENGE #4:

- Drop Customer ID column 'CUST_ID' and make sure that the column has been removed from the dataframe

In [95]:

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

Out[95]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENC
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.1666
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.0000
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.0000
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.0833
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.0833

In [ ]:

```python

```

In [96]:

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

Out[96]:

17

In [97]:

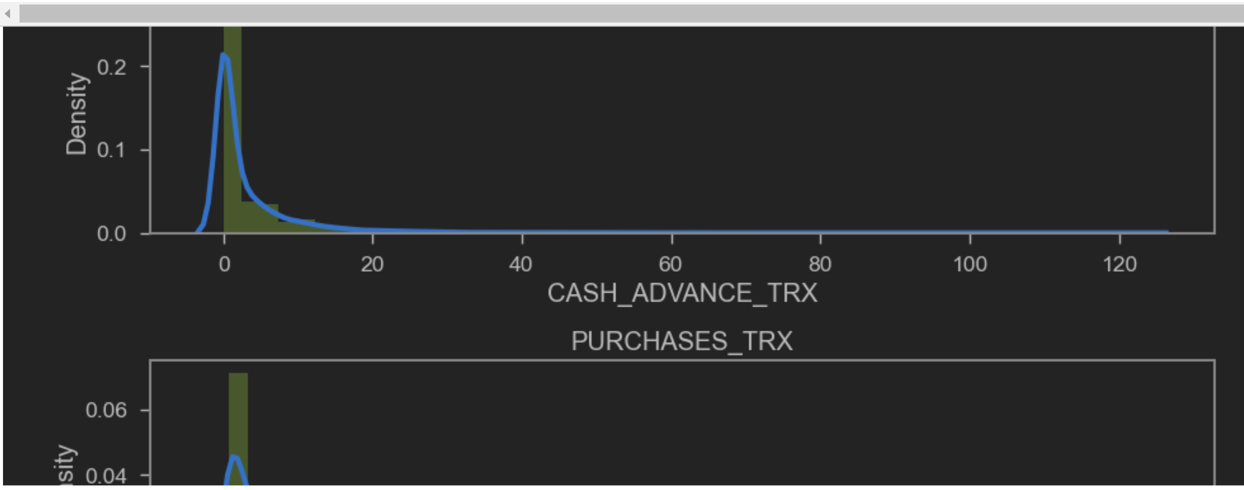
```python
creditcard_df.columns
```

Out[97]:

```
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 [98]:

```python
# 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 puchases or installment purchases
# 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()
```



MINI CHALLENGE #5:

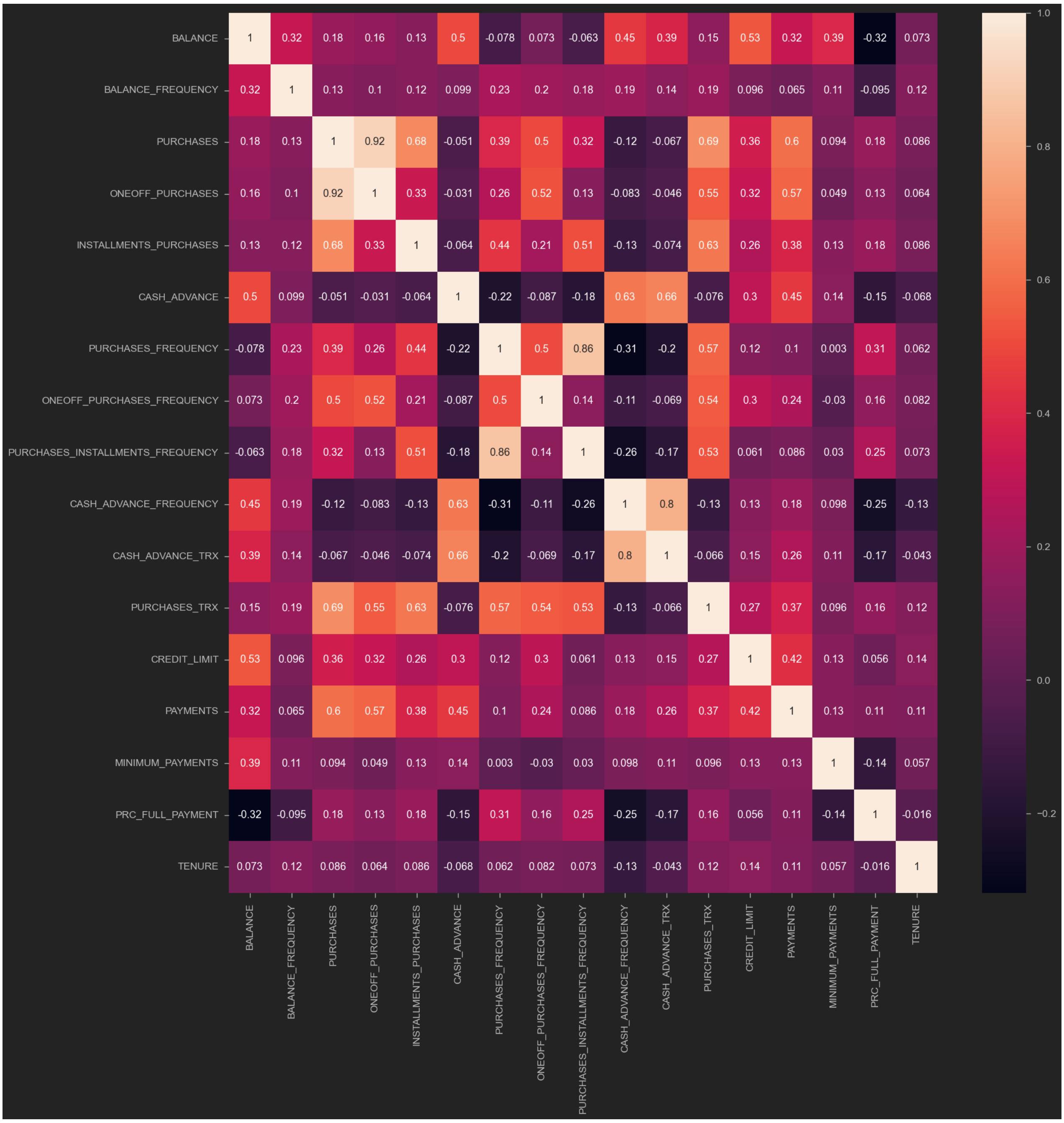
- Obtain the correlation matrix between features

In [99]:

```python
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'
```

Out[99]:

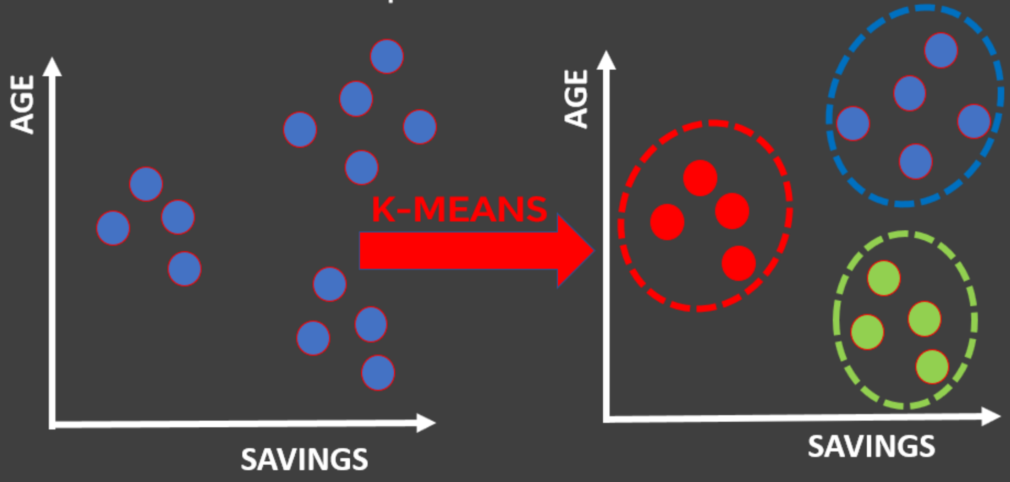
<AxesSubplot:>



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

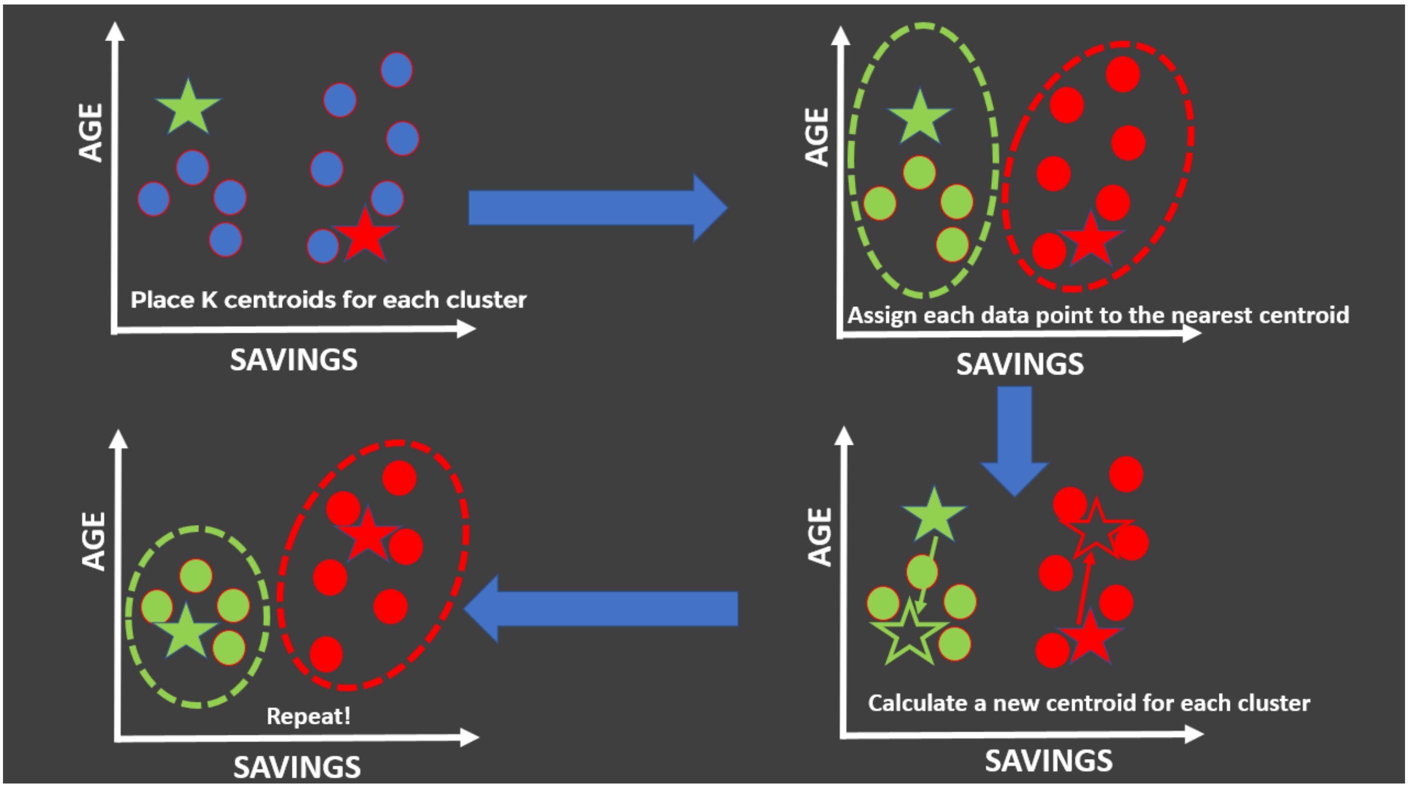
K-MEANS INTUITION

- 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.

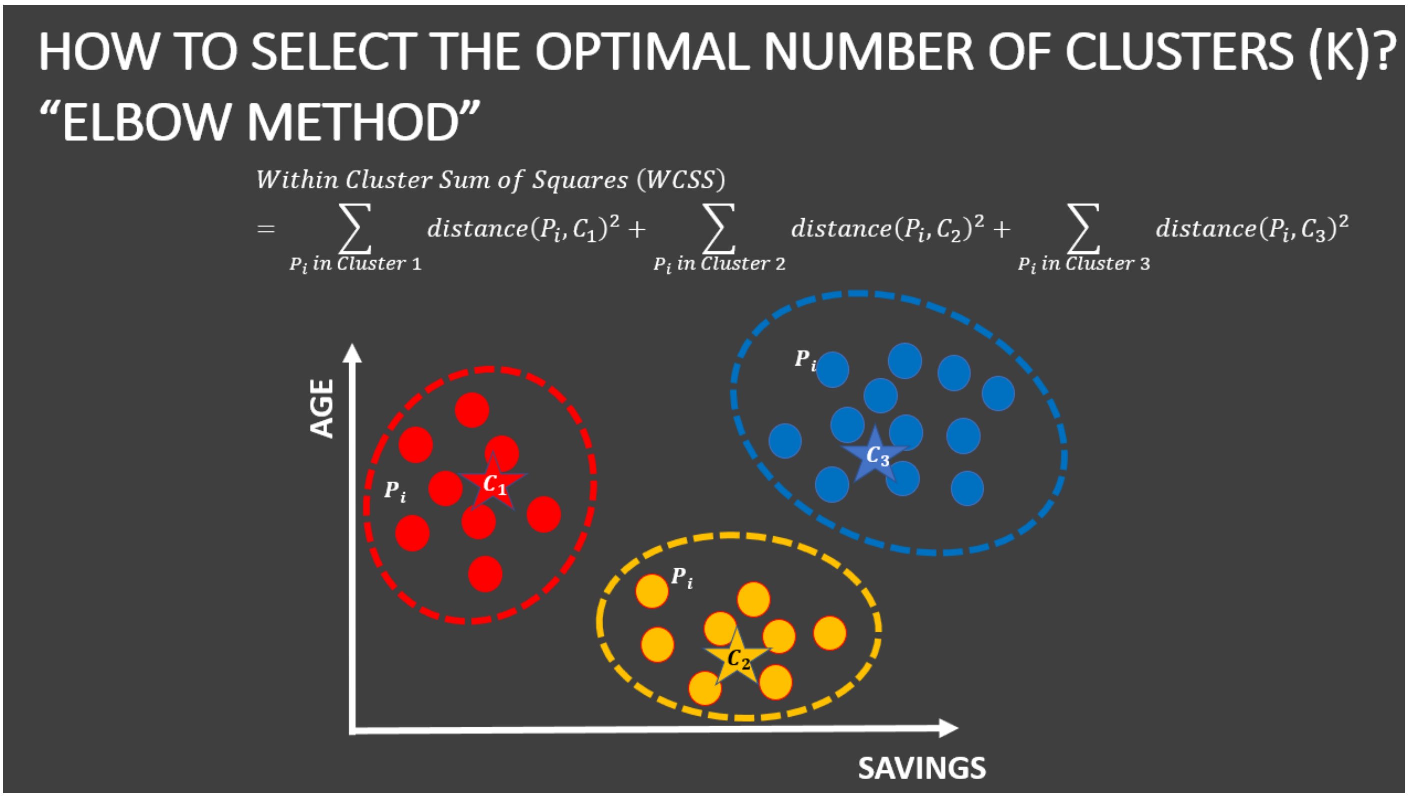


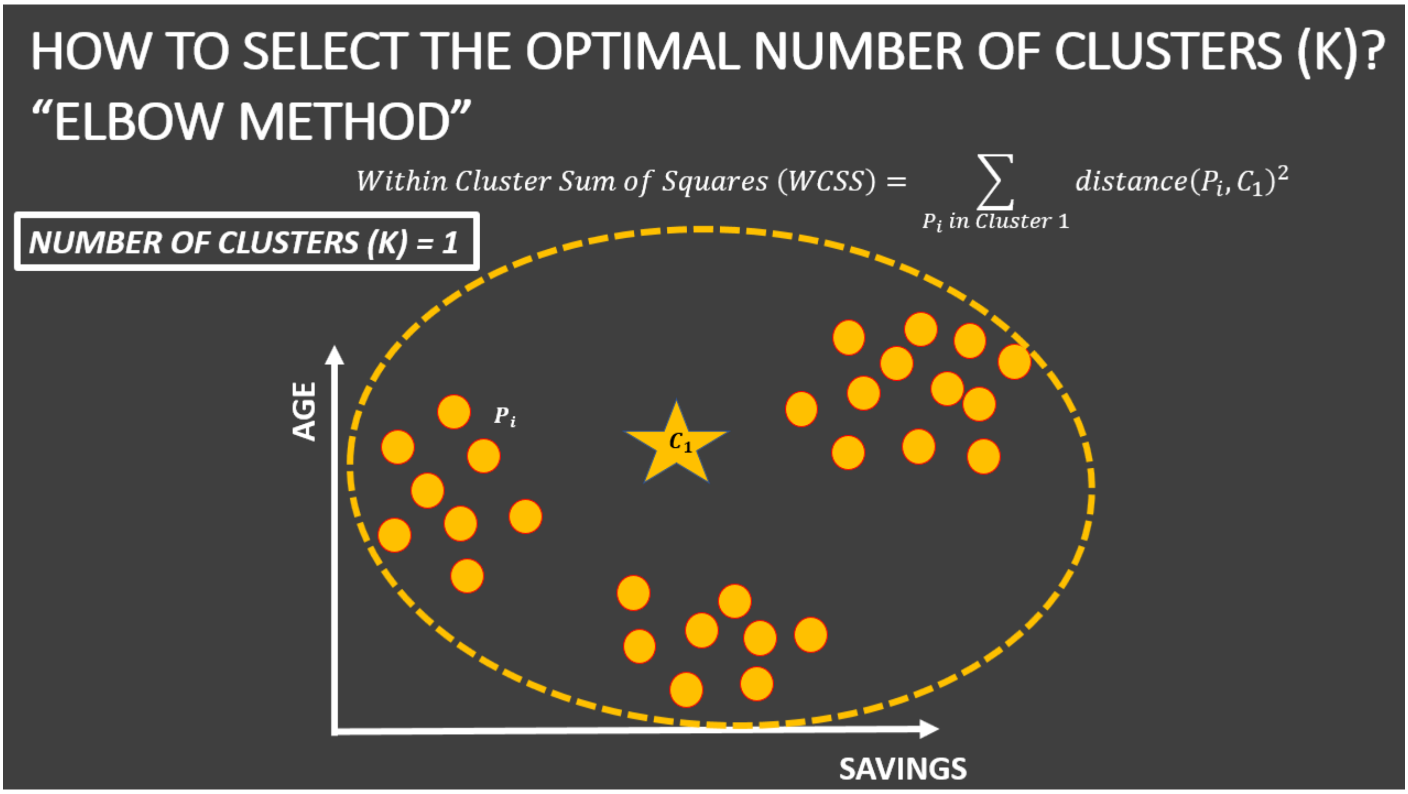
MINI CHALLENGE #6:

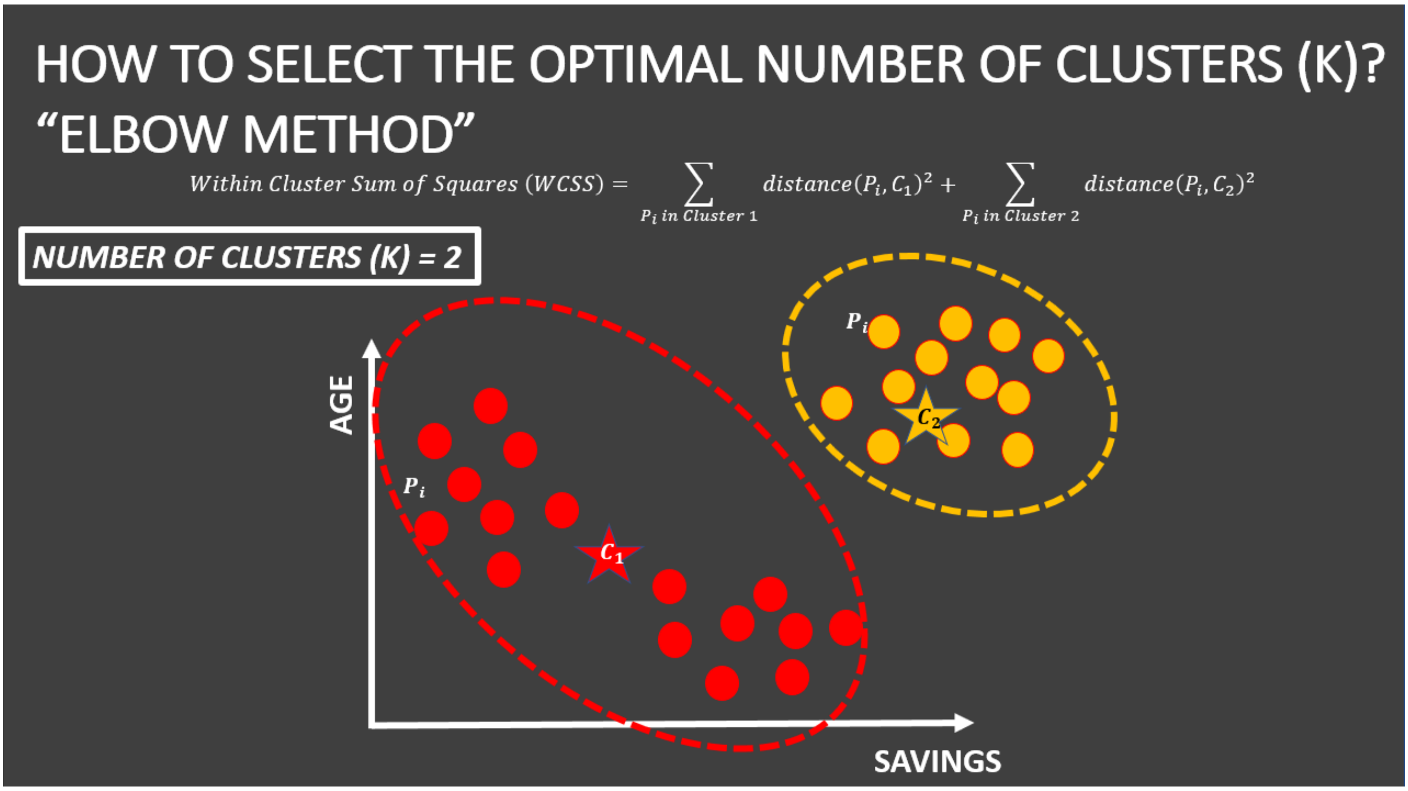
- Which of the following conditions could terminate the K-means clustering algorithm? (choose 2)
  - K-means terminates after a fixed number of iterations is reached
  - K-means terminates when the number of clusters does not increase between iterations
  - K-means terminates when the centroid locations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

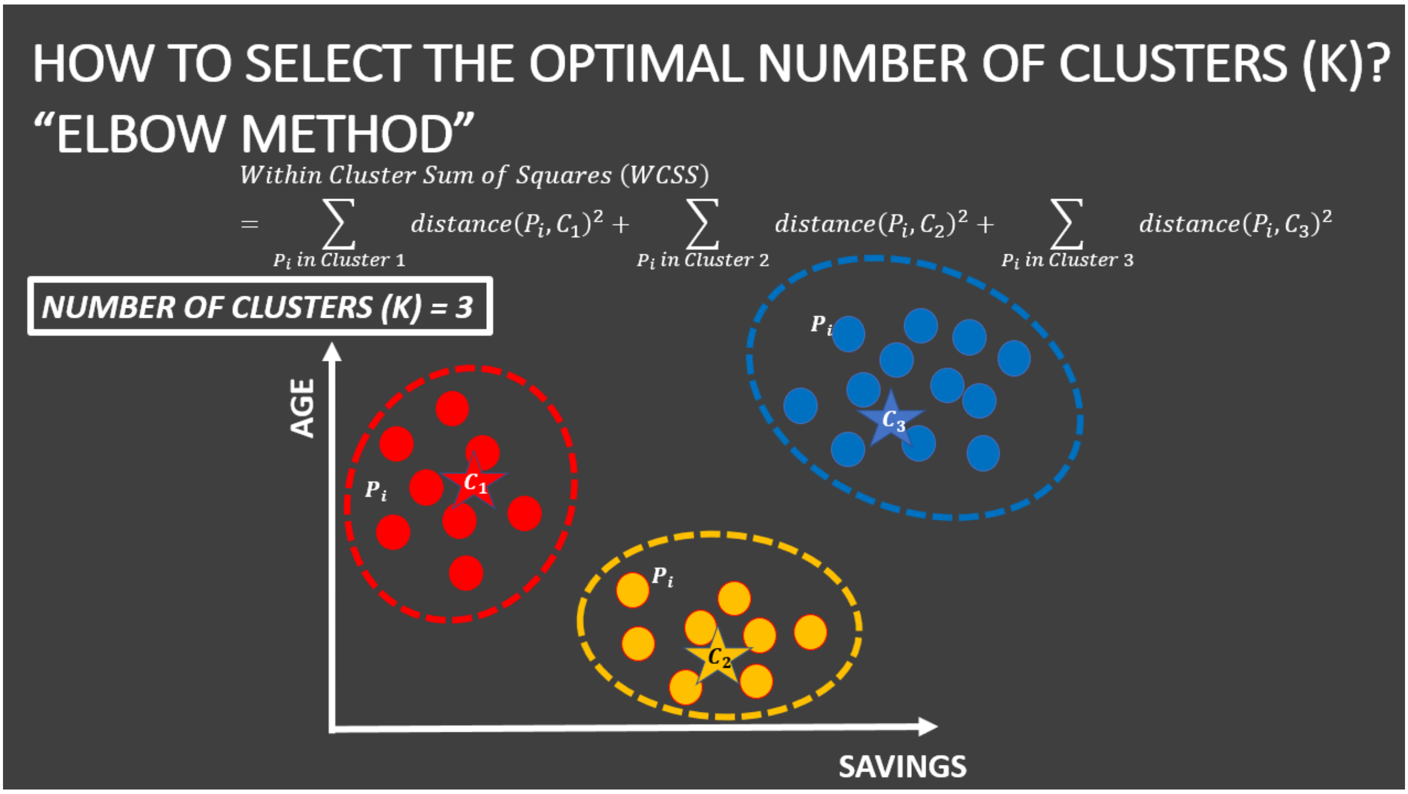
In [ ]:

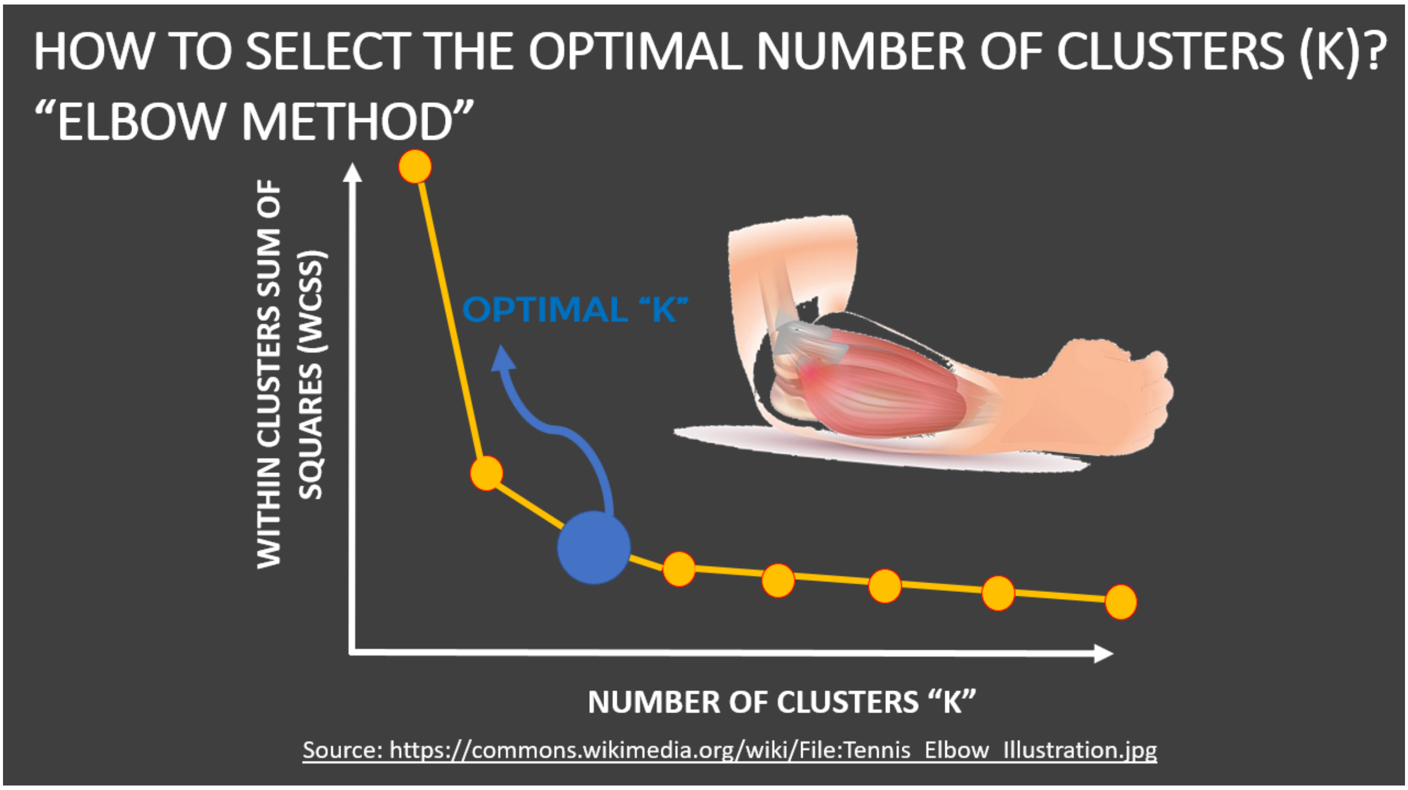
TASK #5: LEARN HOW TO OBTAIN THE OPTIMAL NUMBER OF CLUSTERS (ELBOW METHOD)




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





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 [100]:

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

In [101]:

```
creditcard_df_scaled.shape
```

Out[101]:

```
(8950, 17)
```

In [102]:

```
creditcard_df_scaled
```

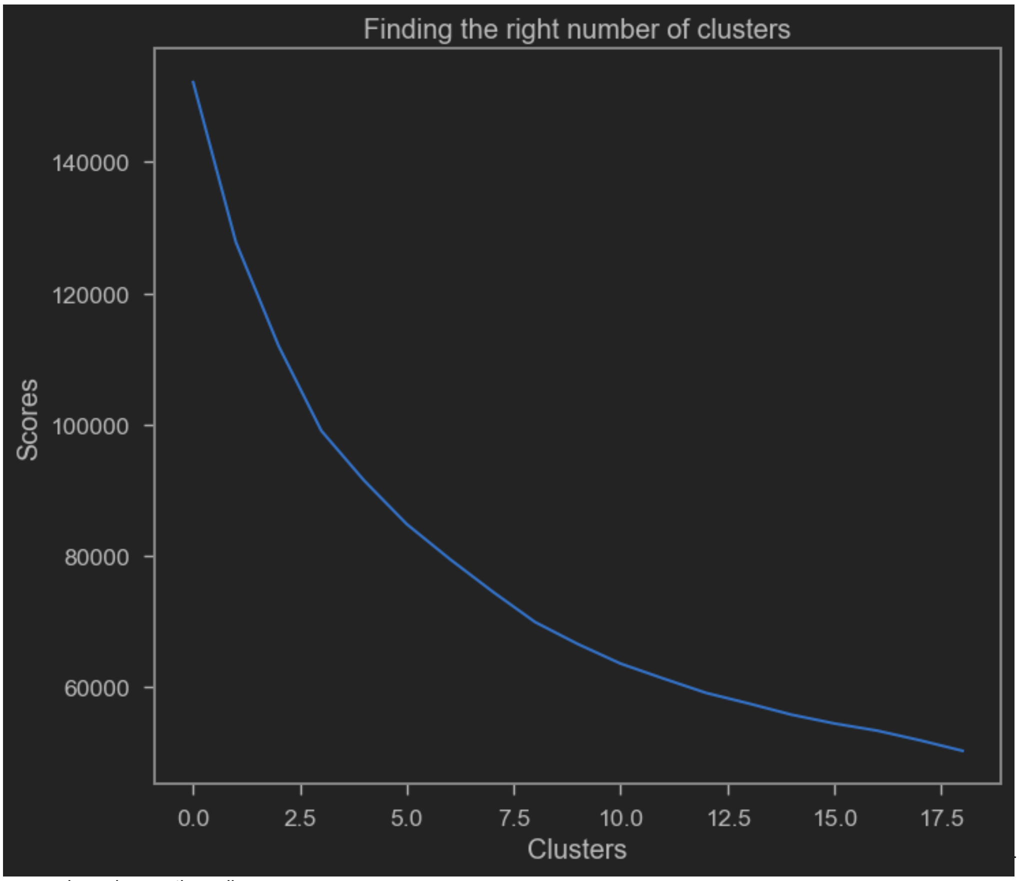
Out[102]:

```
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 [103]:

```
# 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-')
plt.title('Finding the right number of clusters')
plt.xlabel('Clusters')
plt.ylabel('Scores')
plt.show()

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

```
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



of clusters would be in this case? modify the code and rerun the cells.

In [ ]:

```

```

TASK #7: APPLY K-MEANS METHOD

In [104]:

```
kmeans = KMeans(7)
kmeans.fit(creditcard_df_scaled)
labels = kmeans.labels_
```

```
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

In [105]:

```
kmeans.cluster_centers_.shape
```

Out[105]:

```
(7, 17)
```

In [106]:

```

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

Out[106]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
0	-0.701882	-2.133384	-0.306746	-0.230358	-0.301974	-0.323199	-0.546483
1	0.009569	0.401333	-0.343591	-0.223509	-0.401679	-0.101601	-0.814704
2	1.676448	0.394844	-0.196266	-0.147445	-0.193282	1.997830	-0.447815
3	1.488505	0.403475	7.413638	6.553369	5.486972	0.028557	1.072872
4	-0.368652	0.333102	-0.041819	-0.231056	0.325438	-0.366980	0.973176
5	-0.336090	-0.346701	-0.284230	-0.209208	-0.287346	0.064694	-0.196404
6	0.143846	0.431066	0.975990	0.923239	0.611536	-0.306680	1.100349

In [107]:

```
# In order to understand what these numbers mean, let's perform inverse transformation
cluster_centers = scaler.inverse_transform(cluster_centers)
cluster_centers = pd.DataFrame(data = cluster_centers, columns = [creditcard_df.columns])
cluster_centers

# First Customers cluster (Transactors): Those are customers who pay least amount of intrerest charges and careful with their money, Cl
# Second customers cluster (revolvers) who use credit card as a loan (most lucrative sector): highest balance ($5000) and cash advance
# Third customer cluster (VIP/Prime): high credit limit $16K and highest percentage of full payment, target for increase credit limit c
# Fourth customer cluster (low tenure): these are customers with low tenure (7 years), low balance
```

Out[107]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN(
0	103.566909	0.371892	347.837146	210.090875	137.996103	301.107446	0.2710
1	1584.392875	0.972343	269.118389	221.458752	47.834075	765.809207	0.1633
2	5053.860126	0.970806	583.880236	347.709348	236.285253	5168.412968	0.3106
3	4662.671853	0.972850	16842.556892	11469.688108	5372.868784	1038.757441	0.9209
4	797.157201	0.956179	913.858518	208.932467	705.357378	209.297141	0.8809
5	864.932877	0.795140	395.943444	245.195746	151.223889	1114.536873	0.4115
6	1863.877866	0.979386	3088.421697	2124.825567	964.071943	335.749646	0.9319

In [108]:

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

Out[108]:

(8950,)

In [109]:

```
labels.max()
```

Out[109]:

6

In [110]:

```
labels.min()
```

Out[110]:

0

In [111]:

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

```
C:\Users\SOMNATH\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_ini
t` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[111]:

array([2, 0, 4, ..., 5, 5, 5])

In [112]:

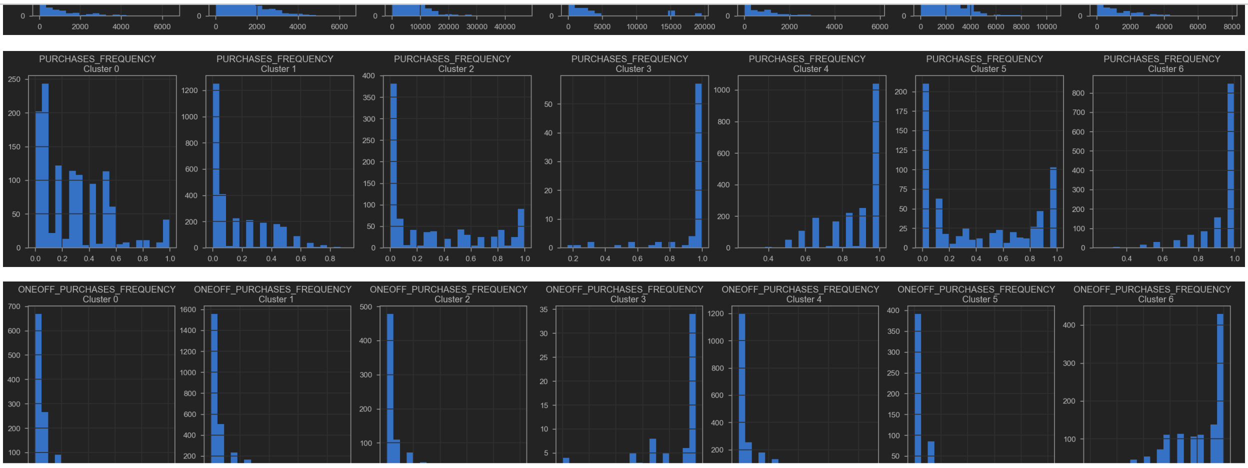
```
# concatenate the clusters labels to our original dataframe
creditcard_df_cluster = pd.concat([creditcard_df, pd.DataFrame({'cluster':labels})], axis = 1)
creditcard_df_cluster.head()
```

Out[112]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN(
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.1666
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.0000
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.0000
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.0833
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.0833

In [113]:

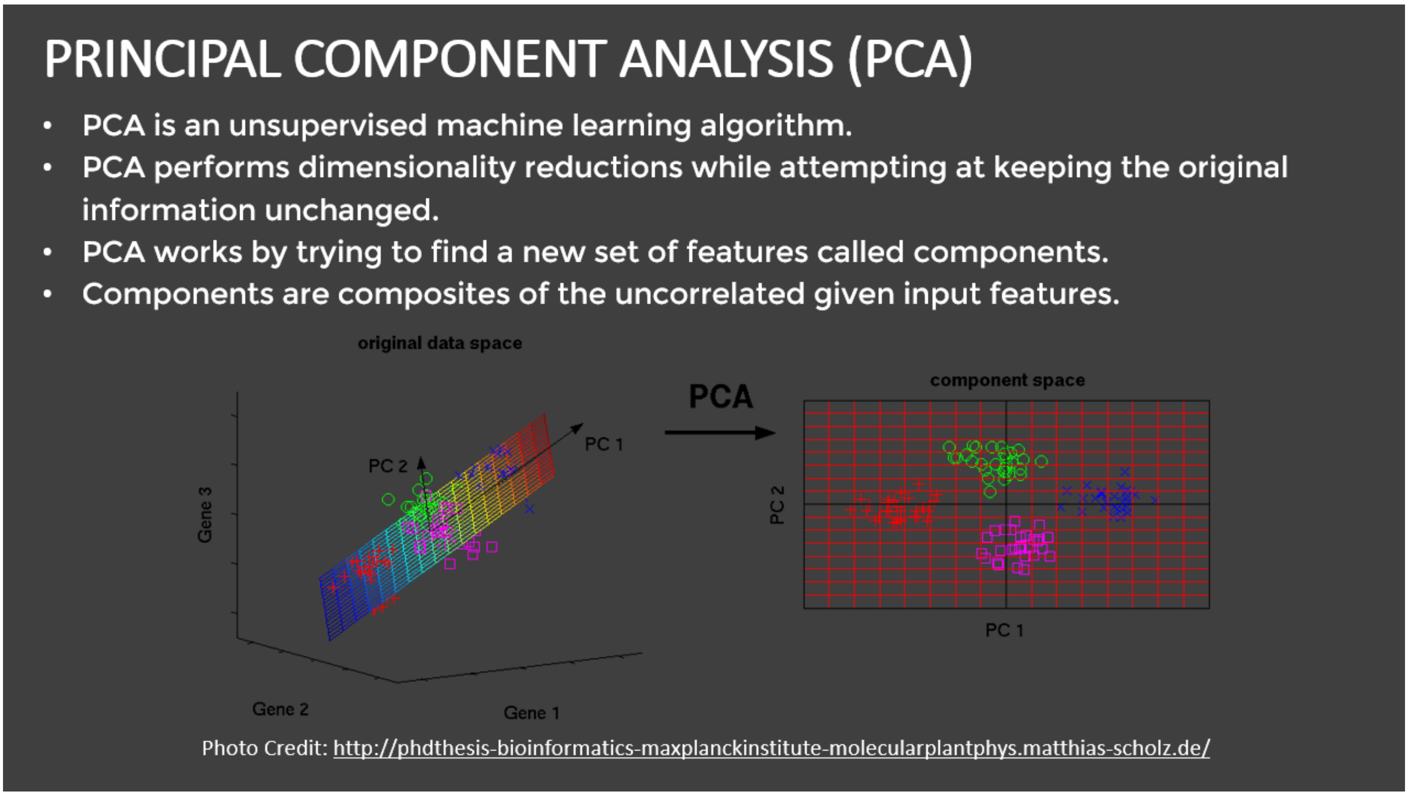
```
# Plot the histogram of various clusters
for i in creditcard_df.columns:
  plt.figure(figsize = (35, 5))
  for j in range(7):
    plt.subplot(1,7,j+1)
    cluster = creditcard_df_cluster[creditcard_df_cluster['cluster'] == j]
    cluster[i].hist(bins = 20)
    plt.title('{}    \nCluster {} '.format(i,j))

  plt.show()
```



MINI CHALLENGE #8:

- Repeat the same procedure with 8 clusters instead of 7

TASK 8: APPLY PRINCIPAL COMPONENT ANALYSIS AND VISUALIZE THE RESULTS



In [114]:

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

Out[114]:

```
array([[-1.6822218 , -1.07644896],
       [-1.13829518,  2.50647535],
       [ 0.96968661, -0.38352372],
       ...,
       [-0.92620587, -1.81078233],
       [-2.336554  , -0.65796235],
       [-0.55641851, -0.40047109]])
```

In [115]:

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

Out[115]:

	pca1	pca2
0	-1.682222	-1.076449
1	-1.138295	2.506475
2	0.969687	-0.383524
3	-0.873628	0.043165
4	-1.599436	-0.688578

In [116]:

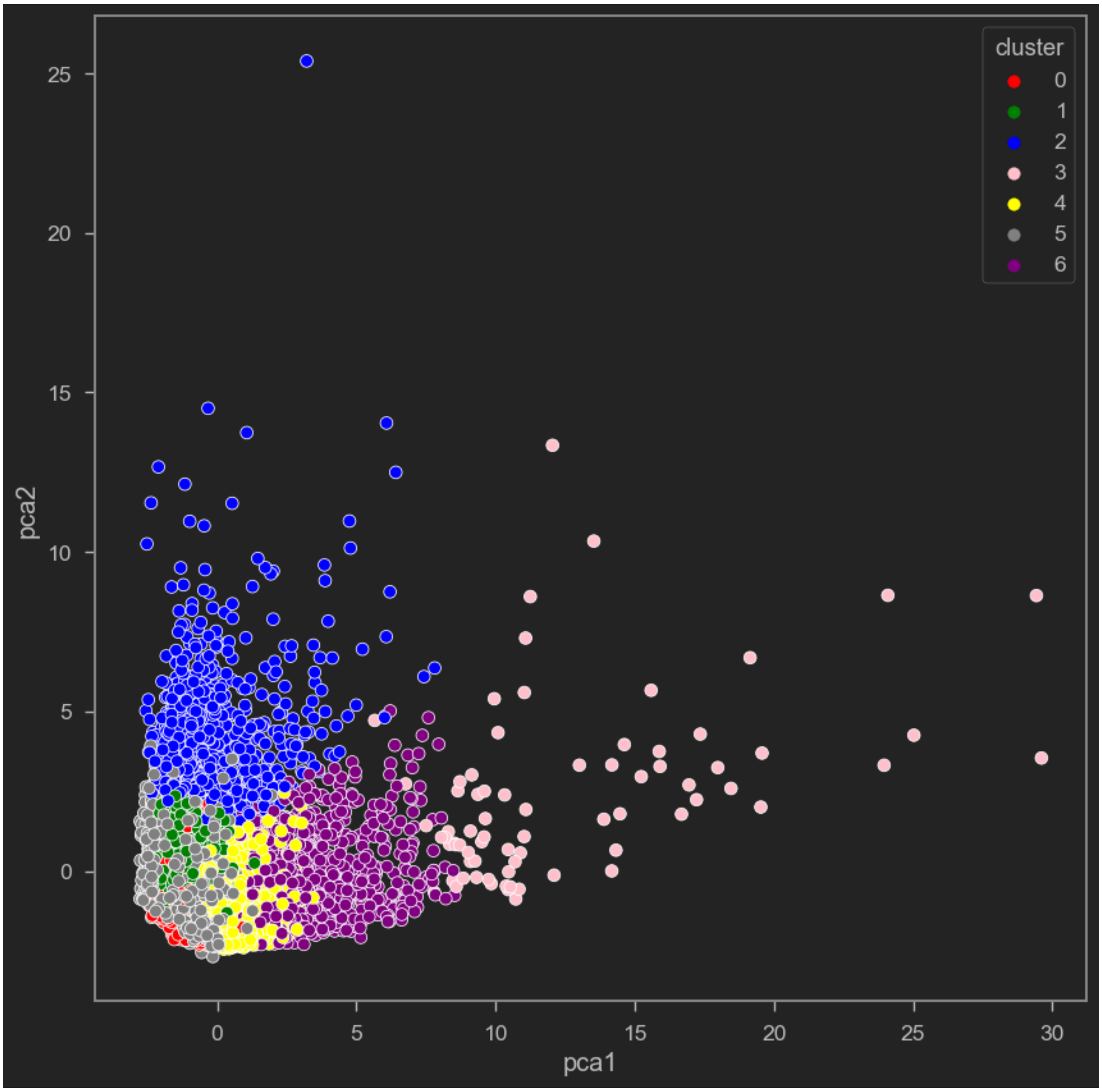
```
# Concatenate the clusters labels to the dataframe
pca_df = pd.concat([pca_df,pd.DataFrame({'cluster':labels})], axis = 1)
pca_df.head()
```

Out[116]:

	pca1	pca2	cluster
0	-1.682222	-1.076449	1
1	-1.138295	2.506475	2
2	0.969687	-0.383524	6
3	-0.873628	0.043165	1
4	-1.599436	-0.688578	1

In [117]:

```
plt.figure(figsize=(10,10))
ax = sns.scatterplot(x="pca1", y="pca2", hue = "cluster", data = pca_df, palette =['red','green','blue','pink','yellow','gray','purple
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 [76]:

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

```
The average, minimum and maximum balance amount are: 1564.4748276781038 0.0 19043.13856
```

MINI CHALLENGE #2

In [77]:

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

Out[77]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUE
550	11547.52001	1.0	49039.57	40761.25	8278.32	558.166886	

In [78]:

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

Out[78]:

```
47137.21176
```

In [80]:

```
# 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'] == 47137.21176]
```

Out[80]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQU
2159	10905.05381	1.0	431.93	133.5	298.43	47137.21176	0.5

MINI CHALLENGE #3

In [81]:

```python
# 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")
```

Out[81]:

```
<AxesSubplot:>
```

MINI CHALLENGE #4

In [ ]:

```python
# 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 [ ]:

```python
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 = 3
```

MINI CHALLENGE #8 & #9:

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