K-Means Clustering For Machine Learning Real Life Project

A case study, I have been hired as a consultant to a bank in New York. The bank has extensive data on their customers for the past 6 months. the Marketing team at the bank want to launch a targeted ad campaign by dividing their customers into at least 3 distinctive groups.

In a case study like this the best algorithm to use K-means Algorithm.

Clustering is the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different. We try to find subgroups within the data such that data points in each cluster are as similar as possible according to a similarity measure such as euclidean-based distance or correlation-based distance. Clustering is used in market segmentation; where we try to find customers that are similar to each other whether in terms of behaviors or attributes etc

K-Means Algorithm is an algorithm that tries to partition the dataset into *K*-defined distinct non-overlapping subgroups (clusters) where each data point belongs to one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum.

The dataset constitutes of 18 columns and 8950 rows. 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 [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
```

In [2]:
```
Marketing = pd.read_csv("Marketing_data.csv")
```

In [3]:
```
Marketing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
CUST_ID                             8950 non-null object
BALANCE                             8950 non-null float64
BALANCE_FREQUENCY                   8950 non-null float64
PURCHASES                           8950 non-null float64
ONEOFF_PURCHASES                    8950 non-null float64
INSTALLMENTS_PURCHASES              8950 non-null float64
CASH_ADVANCE                        8950 non-null float64
PURCHASES_FREQUENCY                 8950 non-null float64
ONEOFF_PURCHASES_FREQUENCY          8950 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY    8950 non-null float64
CASH_ADVANCE_FREQUENCY              8950 non-null float64
CASH_ADVANCE_TRX                    8950 non-null int64
PURCHASES_TRX                       8950 non-null int64
CREDIT_LIMIT                        8949 non-null float64
PAYMENTS                            8950 non-null float64
MINIMUM_PAYMENTS                    8637 non-null float64
PRC_FULL_PAYMENT                    8950 non-null float64
TENURE                              8950 non-null int64
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

In [4]:
```
Marketing.describe()
```

Out[4]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	C
count	8950.000000	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	0.490351	
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371	
min	0.000000	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	0.083333	
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000	
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667	
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000	

From the above we can see the maximum balance a customer has in the account is 19043 and minimum is 0 The maximum amount of purchases is 49039 and mean of purchases is 1003 the describe of a dataset gives all the information overview of the dataset

In [5]:
```
Marketing.head()
```

Out[5]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENC
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.16666
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.0000(
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.0000(
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.0833:
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.0833:

In [6]:
```python
#Lets see who made a one off purchase of $40761.25

Marketing[Marketing ["ONEOFF_PURCHASES"] == 40761.25]
```

Out[6]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEI
550	C10574	11547.52001	1.0	49039.57	40761.25	8278.32	558.166886	

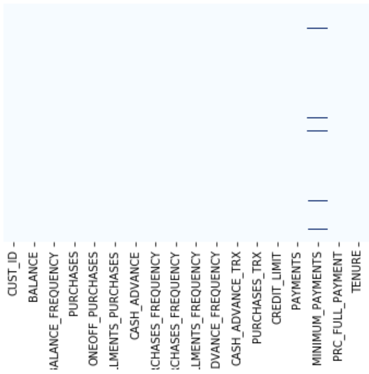
In [7]:
```python
Marketing["BALANCE"].max()
```

Out[7]:
```
19043.13856
```

VISUALIZE AND EXPLORE DATASET

In [8]:
```python
#Lets see if we have any missing data
sns.heatmap(Marketing.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1debe048>
```



In [9]:
```python
#Another way to check out missing values
Marketing.isnull().sum()
```

Out[9]:
```
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 [10]:
```python
#fill up the missing elements of the minimum payments with the mean
Marketing.loc[(Marketing['MINIMUM_PAYMENTS'].isnull() == True), 'MINIMUM_PAYMENTS'] = Marketing['MINIMUM_PAYMENTS'].mea
```

In [12]:
```python
#fill up the missing elements for credit limit
Marketing.loc[(Marketing['CREDIT_LIMIT'].isnull() == True), 'CREDIT_LIMIT'] = Marketing['CREDIT_LIMIT'].mean()
```

In [14]:
```
#to check for duplicate values
Marketing.duplicated().sum()
```
Out[14]: 0

In [15]:
```
#to drop customer id since its not relevant for our analysis
Marketing.drop("CUST_ID", axis = 1, inplace = True)
```

In [16]:
```
Marketing.head()
```
Out[16]:

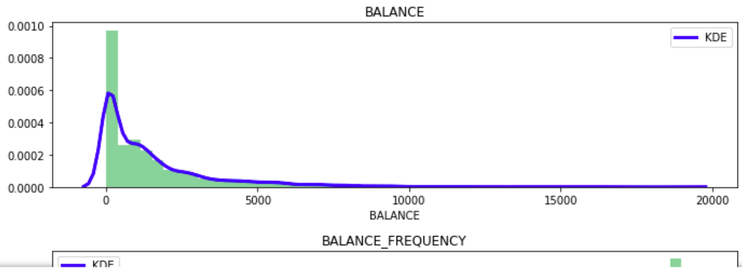
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOF
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	

In [17]:
```
n= len(Marketing.columns)
n
```
Out[17]: 17

In [18]:
```
plt.figure(figsize=(10,50))
for i in range(len(Marketing.columns)):
  plt.subplot(17, 1, i+1)
  sns.distplot(Marketing[Marketing.columns[i]], kde_kws={"color": "b", "lw": 3, "label": "KDE"}, hist_kws={"color": "g"
  plt.title(Marketing.columns[i])

plt.tight_layout()
```

```
/Users/mac/anaconda3/lib/python3.7/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tuple sequence
for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be
interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

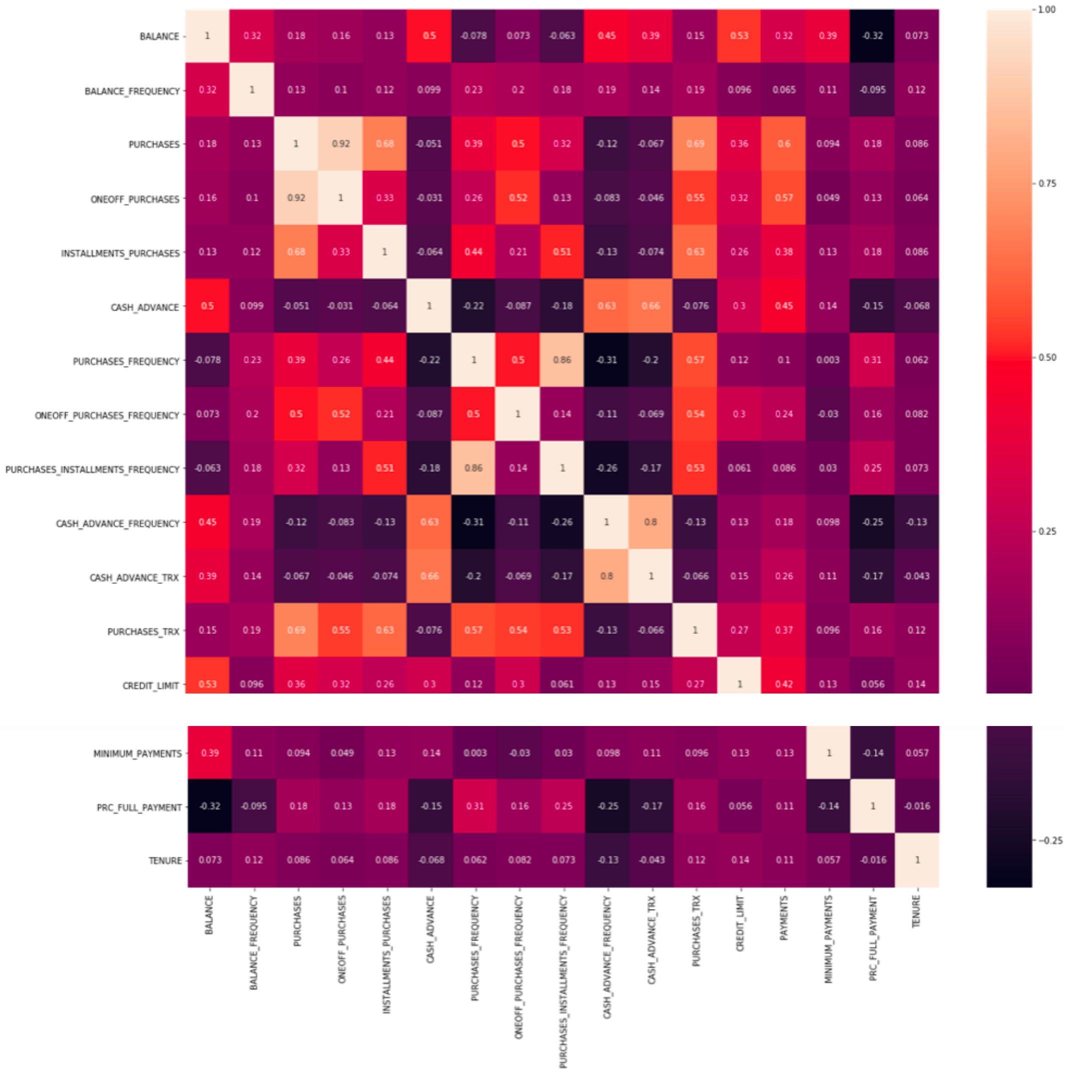


Kernel Density plot (KDE) is used for visualizing the Probability Density of a continous variable You can take note that the Purchases Frequency have two distinct group of customers

In [19]:
```
correlations = Marketing.corr()
```

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

Out[20]: `<matplotlib.axes._subplots.AxesSubplot at 0x1a17072160>`



The Heat map shows the correlations between two points Purchases is highly correlating with One_off purchase Purchases is also correlating with Payments

FIND THE OPTIMAL NUMBER OF CLUSTERS USING ELBOW METHOD

In [21]:
```python
# Let's scale the data first
scaler = StandardScaler()
Marketing_scaled = scaler.fit_transform(Marketing)
```

In [22]:
```python
Marketing_scaled.shape
```

Out[22]: `(8950, 17)`

In [23]:
```python
Marketing_scaled
```

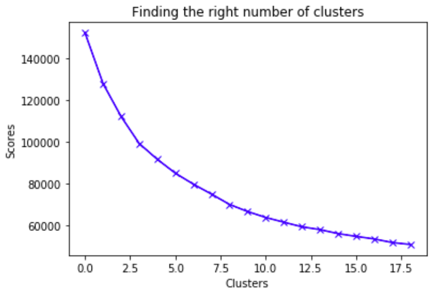
Out[23]:
```
array([[-0.73198937, -0.24943448, -0.42489974, ..., -0.31096755,
        -0.52555097,  0.36067954],
       [ 0.78696085,  0.13432467, -0.46955188, ...,  0.08931021,
         0.2342269 ,  0.36067954],
       [ 0.44713513,  0.51808382, -0.10766823, ..., -0.10166318,
        -0.52555097,  0.36067954],
       ...,
       [-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 [24]:
```python
scores_1 = []

range_values = range(1, 20)

for i in range_values:
  kmeans = KMeans(n_clusters = i)
  kmeans.fit(Marketing_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()
```



APPLY K-MEANS ALGORITHM

In [25]:
```python
kmeans = KMeans(8)
kmeans.fit(Marketing_scaled)
labels = kmeans.labels_
```

In [26]:
```python
kmeans.cluster_centers_.shape
```

Out[26]: (8, 17)

In [27]:
```python
cluster_centers = pd.DataFrame(data = kmeans.cluster_centers_, columns = [Marketing.columns])
cluster_centers
```

Out[27]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_
0	0.006954	0.402035	-0.343349	-0.224522	-0.399248	-0.104866	-0.808895	
1	0.141575	0.430102	0.952037	0.901883	0.594139	-0.306634	1.095364	
2	-0.701894	-2.135494	-0.307095	-0.230581	-0.302387	-0.322957	-0.547410	
3	1.374491	0.412070	7.177493	6.384202	5.239545	0.016050	1.082454	
4	-0.335506	-0.348076	-0.284525	-0.208973	-0.288475	0.065539	-0.198735	
5	-0.378843	0.329833	-0.042241	-0.233129	0.328255	-0.368349	0.979143	
6	1.290312	0.454124	-0.039755	-0.268864	0.399233	-0.000938	0.022469	
7	1.647350	0.392553	-0.202804	-0.147838	-0.208009	2.006220	-0.455365	

In [28]:
```python
cluster_centers = scaler.inverse_transform(cluster_centers)
cluster_centers = pd.DataFrame(data = cluster_centers, columns = [Marketing.columns])
cluster_centers
```

Out[28]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOF
0	1578.948977	0.972509	269.635445	219.777461	50.032730	758.962013	0.165702	
1	1859.151768	0.979158	3037.245611	2089.378377	948.340047	335.845655	0.929973	
2	103.540821	0.371392	347.092201	209.719739	137.622715	301.615215	0.270648	
3	4425.362379	0.974886	16338.028250	11188.905375	5149.122875	1012.529590	0.924792	
4	866.148306	0.794815	395.311749	245.585564	150.203132	1116.308792	0.410589	
5	775.944610	0.955405	912.955381	205.490418	707.904353	206.426870	0.883328	
6	4250.150569	0.984848	918.267222	146.178056	772.089167	976.903069	0.499369	
7	4993.295040	0.970263	569.910045	347.057345	222.967582	5186.008272	0.307590	

In [29]:
```
labels.shape
```
Out[29]: (8950,)

In [30]:
```
labels.max()
```
Out[30]: 7

In [31]:
```
labels.min()
```
Out[31]: 0

In [32]:
```
y_kmeans = kmeans.fit_predict(Marketing_scaled)
y_kmeans
```
Out[32]: array([3, 0, 2, ..., 5, 5, 5], dtype=int32)

In [33]:
```
Marketing_cluster = pd.concat([Marketing, pd.DataFrame({'cluster':labels})], axis = 1)
Marketing_cluster.head()
```
Out[33]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOF
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	

In [34]:
```
# Plot the histogram of various clusters
for i in Marketing.columns:
  plt.figure(figsize = (35, 5))
  for j in range(8):
    plt.subplot(1,8,j+1)
    cluster = Marketing_cluster[Marketing_cluster['cluster'] == j]
    cluster[i].hist(bins = 20)
    plt.title('{}    \nCluster {} '.format(i,j))

  plt.show()
```