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
import warnings
warnings.filterwarnings('ignore')
# 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
```

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 =
# 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 purch
# ONEOFF_PURCHASES_FREQUENCY: How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = n
# PURCHASES_INSTALLMENTS_FREQUENCY: How frequently purchases in installments are being done (1 = frequently don
# 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.head()
```

Out[3]:

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

In [4]:
```
# Let's apply info and get additional insights on our dataframe
# 18 features with 8950 points
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 [5]:
```
creditcard_df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
BALANCE	8950.0	1564.474828	2081.531879	0.000000	128.281915	873.385231	2054.140036	19043.13856
BALANCE_FREQUENCY	8950.0	0.877271	0.236904	0.000000	0.888889	1.000000	1.000000	1.00000
PURCHASES	8950.0	1003.204834	2136.634782	0.000000	39.635000	361.280000	1110.130000	49039.57000
ONEOFF_PURCHASES	8950.0	592.437371	1659.887917	0.000000	0.000000	38.000000	577.405000	40761.25000
INSTALLMENTS_PURCHASES	8950.0	411.067645	904.338115	0.000000	0.000000	89.000000	468.637500	22500.00000
CASH_ADVANCE	8950.0	978.871112	2097.163877	0.000000	0.000000	0.000000	1113.821139	47137.21176
PURCHASES_FREQUENCY	8950.0	0.490351	0.401371	0.000000	0.083333	0.500000	0.916667	1.00000
ONEOFF_PURCHASES_FREQUENCY	8950.0	0.202458	0.298336	0.000000	0.000000	0.083333	0.300000	1.00000
PURCHASES_INSTALLMENTS_FREQUENCY	8950.0	0.364437	0.397448	0.000000	0.000000	0.166667	0.750000	1.00000
CASH_ADVANCE_FREQUENCY	8950.0	0.135144	0.200121	0.000000	0.000000	0.000000	0.222222	1.50000
CASH_ADVANCE_TRX	8950.0	3.248827	6.824647	0.000000	0.000000	0.000000	4.000000	123.00000
PURCHASES_TRX	8950.0	14.709832	24.857649	0.000000	1.000000	7.000000	17.000000	358.00000
CREDIT_LIMIT	8949.0	4494.449450	3638.815725	50.000000	1600.000000	3000.000000	6500.000000	30000.00000
PAYMENTS	8950.0	1733.143852	2895.063757	0.000000	383.276166	856.901546	1901.134317	50721.48336
MINIMUM_PAYMENTS	8637.0	864.206542	2372.446607	0.019163	169.123707	312.343947	825.485459	76406.20752
PRC_FULL_PAYMENT	8950.0	0.153715	0.292499	0.000000	0.000000	0.000000	0.142857	1.00000
TENURE	8950.0	11.517318	1.338331	6.000000	12.000000	12.000000	12.000000	12.00000

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

In [6]:
```
creditcard_df.describe().loc[['mean','min','max'],['BALANCE']]
```

Out[6]:

	BALANCE
mean	1564.474828
min	0.000000
max	19043.138560

In [7]:
```
creditcard_df.loc[:,'BALANCE'].min()
```

Out[7]: 0.0

In [8]:
```
creditcard_df.loc[:,'BALANCE'].max()
```

Out[8]: 19043.13856

In [9]:
```
creditcard_df.loc[:,'BALANCE'].mean()
```

Out[9]: 1564.4748276781038

In [10]:
```
# 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 generall
# Average credit limit ~ 4500
# Percent of full payment is 15%
# Average tenure is 11 years
```

- 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 [11]:
```
creditcard_df[creditcard_df['CASH_ADVANCE']==creditcard_df['CASH_ADVANCE'].max()]
```

Out[11]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
2159	C12226	10905.05381	1.0	431.93	133.5	298.43	47137.21176

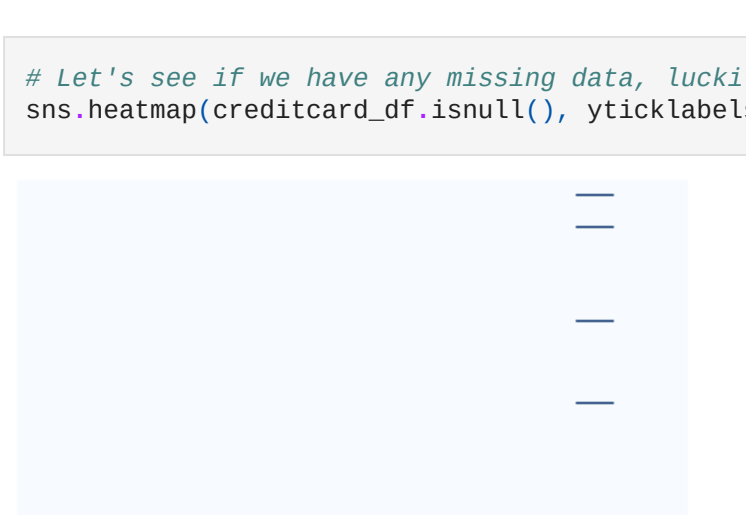
In [12]:
```
creditcard_df[creditcard_df['ONEOFF_PURCHASES']==creditcard_df['ONEOFF_PURCHASES'].max()]
```

Out[12]:

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

VISUALIZE AND EXPLORE DATASET

In [13]:
```
# 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");
```



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

Out[14]:
```
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 [15]:
```
# Fill up the missing elements with mean of the 'MINIMUM_PAYMENT'
creditcard_df.loc[(creditcard_df['MINIMUM_PAYMENTS'].isnull() == True), 'MINIMUM_PAYMENTS'] = creditcard_df['MI
```

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

In [16]:
```
creditcard_df['CREDIT_LIMIT'].fillna(creditcard_df['CREDIT_LIMIT'].mean() , inplace=True)
```

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

Out[17]:
```
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 [18]:
```
# Let's see if we have duplicated entries in the data
creditcard_df.duplicated().sum()
```

Out[18]: 0

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

In [19]:
```
creditcard_df.drop(columns='CUST_ID', inplace=True)
```

In [20]:
```
creditcard_df.head()
```

Out[20]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_
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 [21]:
```
n = len(creditcard_df.columns)
n
```

Out[21]: 17

In [22]:
```
creditcard_df.columns
```

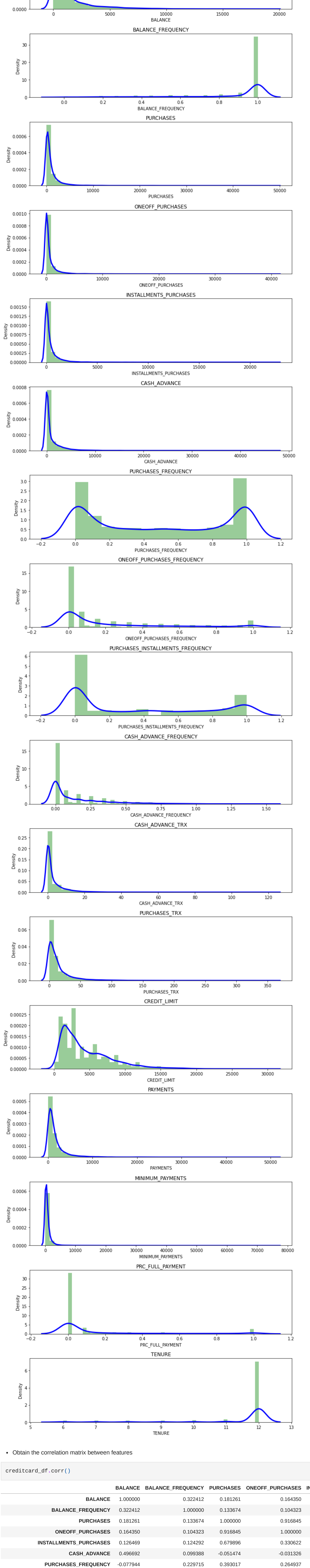
Out[22]:
```
Index(['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES',
       'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FREQUENCY',
       'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY',
       'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX',
       'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT',
       'TENURE'],
      dtype='object')
```

In [23]:
```
# 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 o
# 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

plt.tight_layout()
```



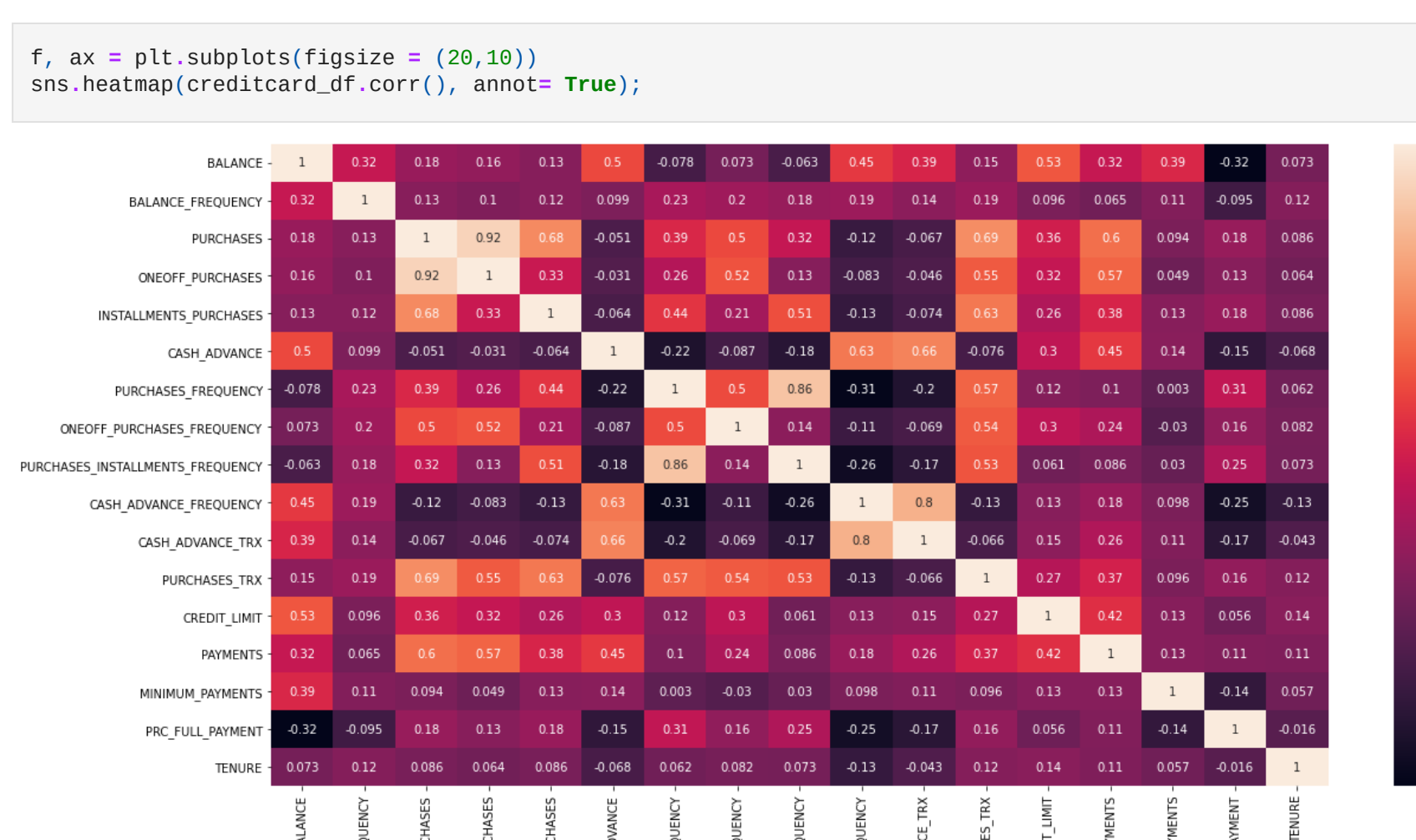
- Obtain the correlation matrix between features

In [24]:
```
creditcard_df.corr()
```

Out[24]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
BALANCE	1.000000	0.322412	0.181261	0.164350	0.12
BALANCE_FREQUENCY	0.322412	1.000000	0.133674	0.104323	0.12
PURCHASES	0.181261	0.133674	1.000000	0.916845	0.67
ONEOFF_PURCHASES	0.164350	0.104323	0.916845	1.000000	0.33
INSTALLMENTS_PURCHASES	0.126469	0.124292	0.679896	0.330622	1.00
CASH_ADVANCE	0.496692	0.099388	-0.051474	-0.031326	-0.06
PURCHASES_FREQUENCY	-0.077944	0.229715	0.393017	0.264937	0.44
ONEOFF_PURCHASES_FREQUENCY	0.073166	0.202415	0.498430	0.524891	0.21
PURCHASES_INSTALLMENTS_FREQUENCY	-0.063186	0.176079	0.315567	0.127729	0.51
CASH_ADVANCE_FREQUENCY	0.449218	0.191873	-0.120143	-0.082628	-0.13
CASH_ADVANCE_TRX	0.385152	0.141555	-0.067175	-0.046212	-0.07
PURCHASES_TRX	0.154338	0.189626	0.689561	0.545523	0.62
CREDIT_LIMIT	0.531267	0.095795	0.356959	0.319721	0.25
PAYMENTS	0.322802	0.065008	0.603264	0.567292	0.38
MINIMUM_PAYMENTS	0.394282	0.114249	0.093515	0.048597	0.13
PRC_FULL_PAYMENT	-0.318959	-0.095082	0.180379	0.132763	0.18
TENURE	0.072692	0.119776	0.086288	0.064150	0.08

In [25]:
```
f, ax = plt.subplots(figsize = (20,10))
sns.heatmap(creditcard_df.corr(), annot= True);
```



In [ ]:

Kmeans

FIND THE OPTIMAL NUMBER OF CLUSTERS USING ELBOW METHOD

In [26]:
```
# Let's scale the data first
scaler = StandardScaler()
creditcard_df_scaled = scaler.fit_transform(creditcard_df)
```

In [27]:
```
creditcard_df_scaled.shape
```

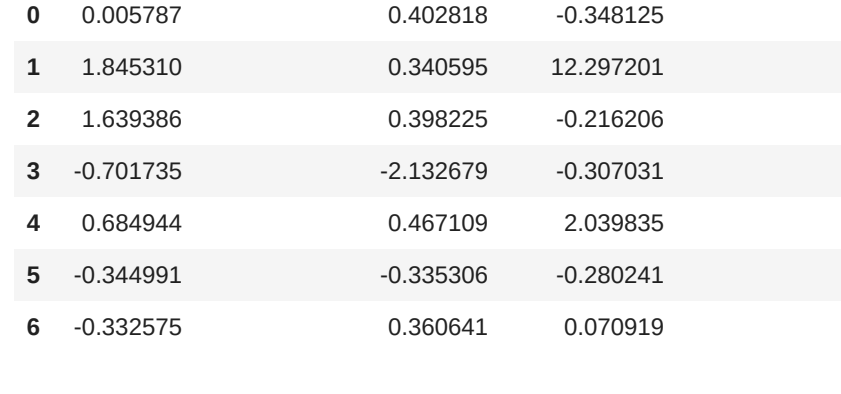
Out[27]: (8950, 17)

In [28]:
```
creditcard_df_scaled
```

Out[28]:
```
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 [29]:
```
# 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 = []
for i in range(1,20):
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(creditcard_df_scaled)
    scores_1.append(kmeans.inertia_)

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



In [ ]:

APPLY K-MEANS METHOD

In [30]:
```
kmeans = KMeans(7)
kmeans.fit(creditcard_df_scaled)
labels = kmeans.labels_
```

In [31]:
```
kmeans.cluster_centers_.shape #centroid for clusters
```

Out[31]: (7, 17)

In [32]:
```
cluster_centers = pd.DataFrame(data = kmeans.cluster_centers_, columns = [creditcard_df.columns])
cluster_centers
```

Out[32]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FR
0	0.005787	0.402818	-0.348125	-0.230430	-0.399565	-0.109991	
1	1.845310	0.340595	12.297201	12.823670	5.516158	0.272530	
2	1.639386	0.398225	-0.216206	-0.151410	-0.233121	1.949346	
3	-0.701735	-2.132679	-0.307031	-0.232805	-0.298160	-0.322394	
4	0.684944	0.467109	2.039835	1.677136	1.741851	-0.218039	
5	-0.344991	-0.335306	-0.280241	-0.207459	-0.281139	0.057798	
6	-0.332575	0.360641	0.070919	-0.043771	0.247802	-0.367446	

In [33]:
```
# 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 care
# Second customers cluster (revolvers) who use credit card as a loan (most lucrative sector): highest balance (
# Third customer cluster (VIP/Prime): high credit limit $16K and highest percentage of full payment, target for
# Fourth customer cluster (low tenure): these are customers with low tenure (7 years), low balance
```

Out[33]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_
0	1576.520837	0.972695	259.431159	209.969968	49.746070	748.214436	
1	5405.330935	0.957955	27276.363750	21877.102917	5399.260833	1550.378389	
2	4976.717437	0.971607	541.277522	341.127987	200.259567	5066.741502	
3	103.873507	0.372059	347.227614	206.028687	141.445644	302.796560	
4	2990.128786	0.987925	5361.342712	3376.139251	1986.201797	521.633217	
5	846.405138	0.797840	404.465878	248.098605	156.837492	1100.076741	
6	872.247331	0.962703	1154.723818	519.785977	635.152149	208.319403	

In [34]:
```
labels.shape # Labels associated to each data point
```

Out[34]: (8950,)

In [35]:
```
labels.max()
```

Out[35]: 6

In [36]:
```
labels.min()
```

Out[36]: 0

In [37]:
```
y_kmeans = kmeans.fit_predict(creditcard_df_scaled)
y_kmeans
```

Out[37]: array([4, 2, 6, ..., 3, 3, 3], dtype=int32)

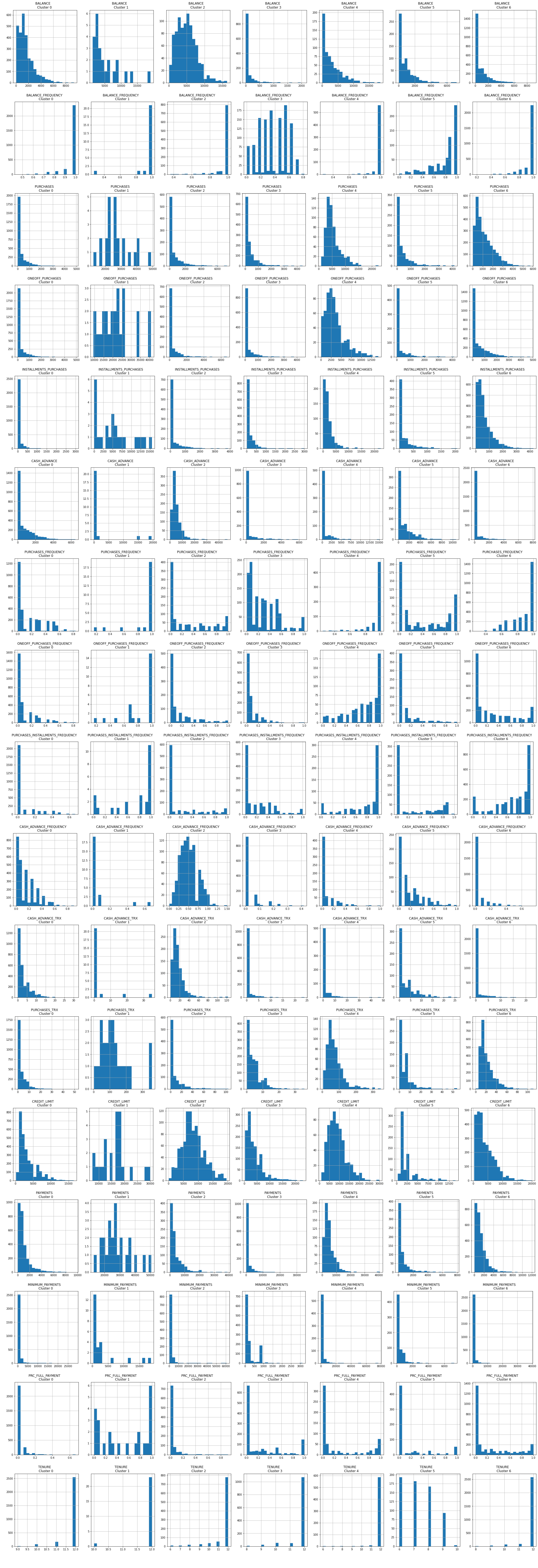
In [38]:
```
# concatenate the clusters labels to our original dataframe
creditcard_df_cluster = pd.concat([creditcard_df, pd.DataFrame({'cluster':labels})], axis = 1)
creditcard_df_cluster.head()
```

Out[38]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_
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 [39]:
```
# 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()
```



APPLY PRINCIPAL COMPONENT ANALYSIS AND VISUALIZE THE RESULTS

In [40]:
```
# Obtain the principal components
pca = PCA(n_components=2)
principal_comp = pca.fit_transform(creditcard_df_scaled)
```

In [41]:
```
# Create a dataframe with the two components
pca_df = pd.DataFrame(data = principal_comp, columns =['pca1','pca2'])
pca_df.head()
```

Out[41]:

	pca1	pca2
0	-1.682216	-1.076444
1	-1.138281	2.506504
2	0.969683	-0.383523
3	-0.873629	0.043164
4	-1.599429	-0.688572

In [42]:
```
# Concatenate the clusters labels to the dataframe
pca_df = pd.concat([pca_df,pd.DataFrame({'cluster':labels})], axis = 1)
pca_df.head()
```

Out[42]:

	pca1	pca2	cluster
0	-1.682216	-1.076444	0
1	-1.138281	2.506504	2
2	0.969683	-0.383523	6
3	-0.873629	0.043164	0
4	-1.599429	-0.688572	0

In [43]:
```
plt.figure(figsize=(10,10))
ax = sns.scatterplot(x="pca1", y="pca2", hue = "cluster", data = pca_df, palette =['red','green','blue','pink',
plt.show()
```