Rohan Nyati

500075940

R177219148

Batch 5 (Ai & Ml)

Assignment 2

1. Extracting data and cleaning:

In [1]:

```
import numpy as np
import pandas as pd
df = pd.read_csv("CC GENERAL.csv")
df.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 [2]:

```
df.isnull().sum()
```

Out[2]:

```
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 [3]:

```
df.describe()
```

Out[3]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLI
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	
std	2081.531879	0.236904	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	
max	19043.138560	1.000000	49039.570000	40761.250000	

In [4]:

```
df.head(10)
```

Out[4]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INS
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	
5	C10006	1809.828751	1.000000	1333.28	0.00	
6	C10007	627.260806	1.000000	7091.01	6402.63	
7	C10008	1823.652743	1.000000	436.20	0.00	
8	C10009	1014.926473	1.000000	861.49	661.49	
9	C10010	152.225975	0.545455	1281.60	1281.60	

In [5]:

```
df = df.drop(['CUST_ID'],axis = 1)
df.head(10)
```

Out[5]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENT
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	
5	1809.828751	1.000000	1333.28	0.00	
6	627.260806	1.000000	7091.01	6402.63	
7	1823.652743	1.000000	436.20	0.00	
8	1014.926473	1.000000	861.49	661.49	
9	152.225975	0.545455	1281.60	1281.60	

2. Data Visualization :

In [6]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

2.1 Univariate Analysis :

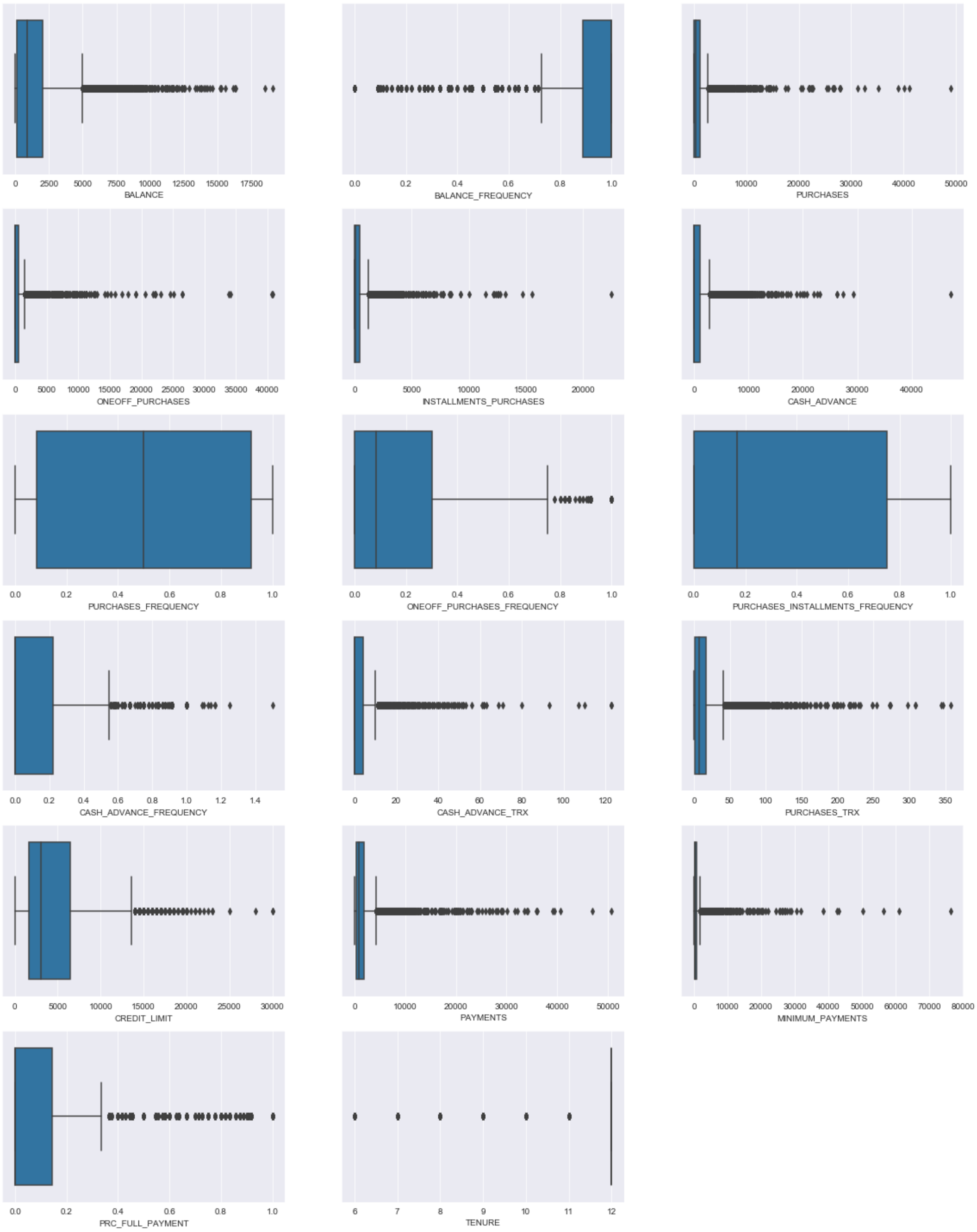
In [9]:

```
sns.set_style("darkgrid")
fig = plt.figure(figsize = (20,30))

plot_feat = df.columns

for i, v in enumerate(plot_feat):
    axes = fig.add_subplot(7, 3, i+1)
    sns.distplot(df[v], kde = False, ax = axes)
```


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

In [10]:

```
fig = plt.figure(figsize = (20,30))

for i, v in enumerate(plot_feat):
    axes = fig.add_subplot(7, 3, i+1)
    sns.boxplot(x = df[v], ax = axes)
```


0 2500 5000 7500 10000 12500 15000 17500
BALANCE
0.0 0.2 0.4 0.6 0.8 1.0
BALANCE_FREQUENCY
0 10000 20000 30000 40000 50000
PURCHASES
0 5000 10000 15000 20000 25000 30000 35000 40000
ONEOFF_PURCHASES
0 5000 10000 15000 20000
INSTALLMENTS_PURCHASES
0 10000 20000 30000 40000
CASH_ADVANCE
0.0 0.2 0.4 0.6 0.8 1.0
PURCHASES_FREQUENCY
0.0 0.2 0.4 0.6 0.8 1.0
ONEOFF_PURCHASES_FREQUENCY
0.0 0.2 0.4 0.6 0.8 1.0
PURCHASES_INSTALLMENTS_FREQUENCY
0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4
CASH_ADVANCE_FREQUENCY
0 20 40 60 80 100 120
CASH_ADVANCE_TRX
0 50 100 150 200 250 300 350
PURCHASES_TRX
0 5000 10000 15000 20000 25000 30000
CREDIT_LIMIT
0 10000 20000 30000 40000 50000
PAYMENTS
0 10000 20000 30000 40000 50000 60000 70000 80000
MINIMUM_PAYMENTS
0.0 0.2 0.4 0.6 0.8 1.0
PRC_FULL_PAYMENT
6 7 8 9 10 11 12
TENURE

Imputing null values :

In [12]:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='median')

X = df['MINIMUM_PAYMENTS'].values.reshape(-1,1)
X = imputer.fit_transform(X)

df['MINIMUM_PAYMENTS_NEW'] = X
```

In [13]:

```
X2 = df['CREDIT_LIMIT'].values.reshape(-1,1)
X2 = imputer.fit_transform(X2)

df['CREDIT_LIMIT_NEW'] = X2
```

In [14]:

```
df = df.drop(['CREDIT_LIMIT','MINIMUM_PAYMENTS'],axis = 1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 17 columns):
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
PAYMENTS                            8950 non-null float64
PRC_FULL_PAYMENT                    8950 non-null float64
TENURE                              8950 non-null int64
MINIMUM_PAYMENTS_NEW                8950 non-null float64
CREDIT_LIMIT_NEW                    8950 non-null float64
dtypes: float64(14), int64(3)
memory usage: 1.2 MB
```

In [15]:

```
df.isnull().sum().sum()
```

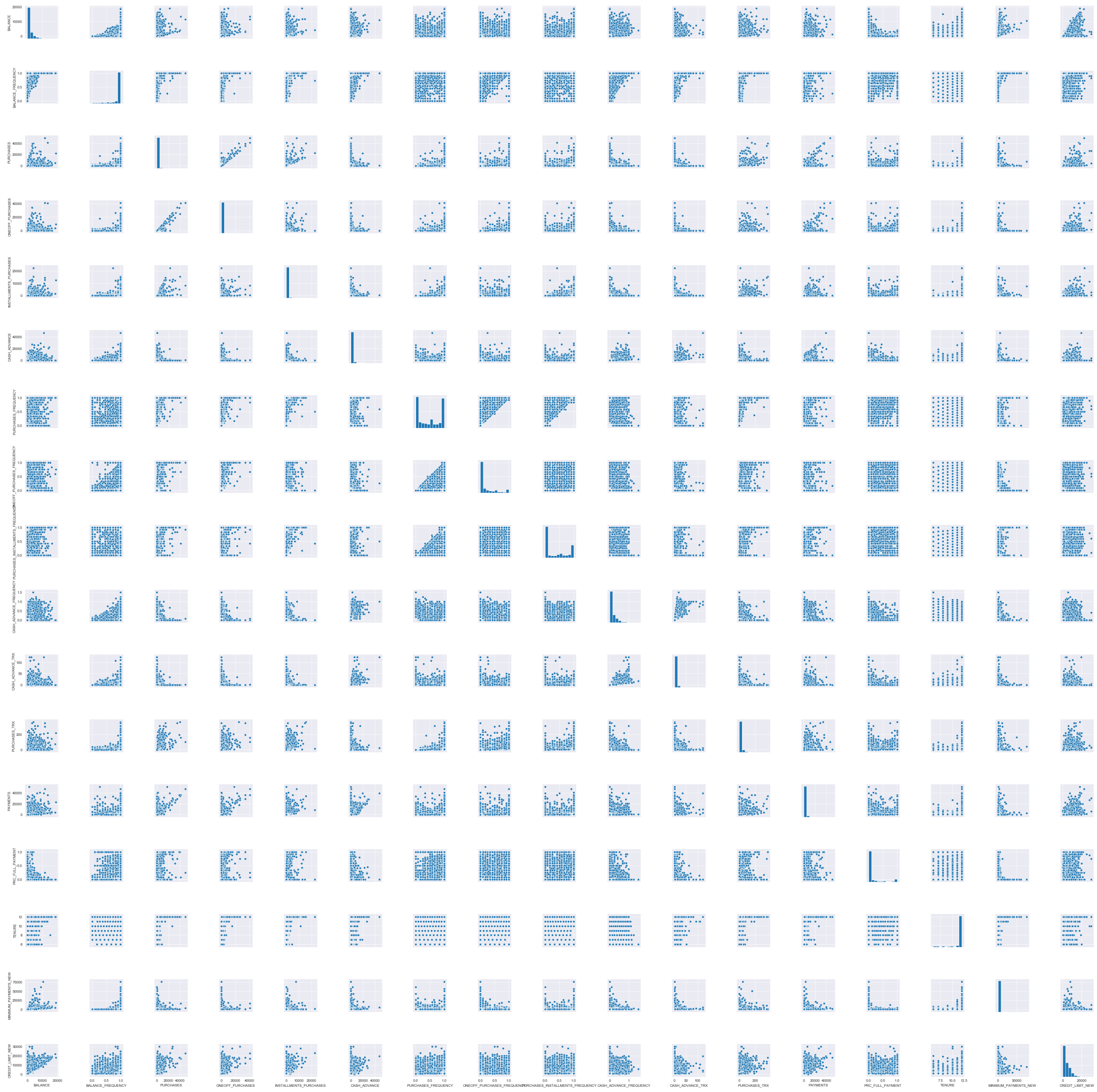
Out[15]:

```
0
```

2.2 Bivariate Analysis

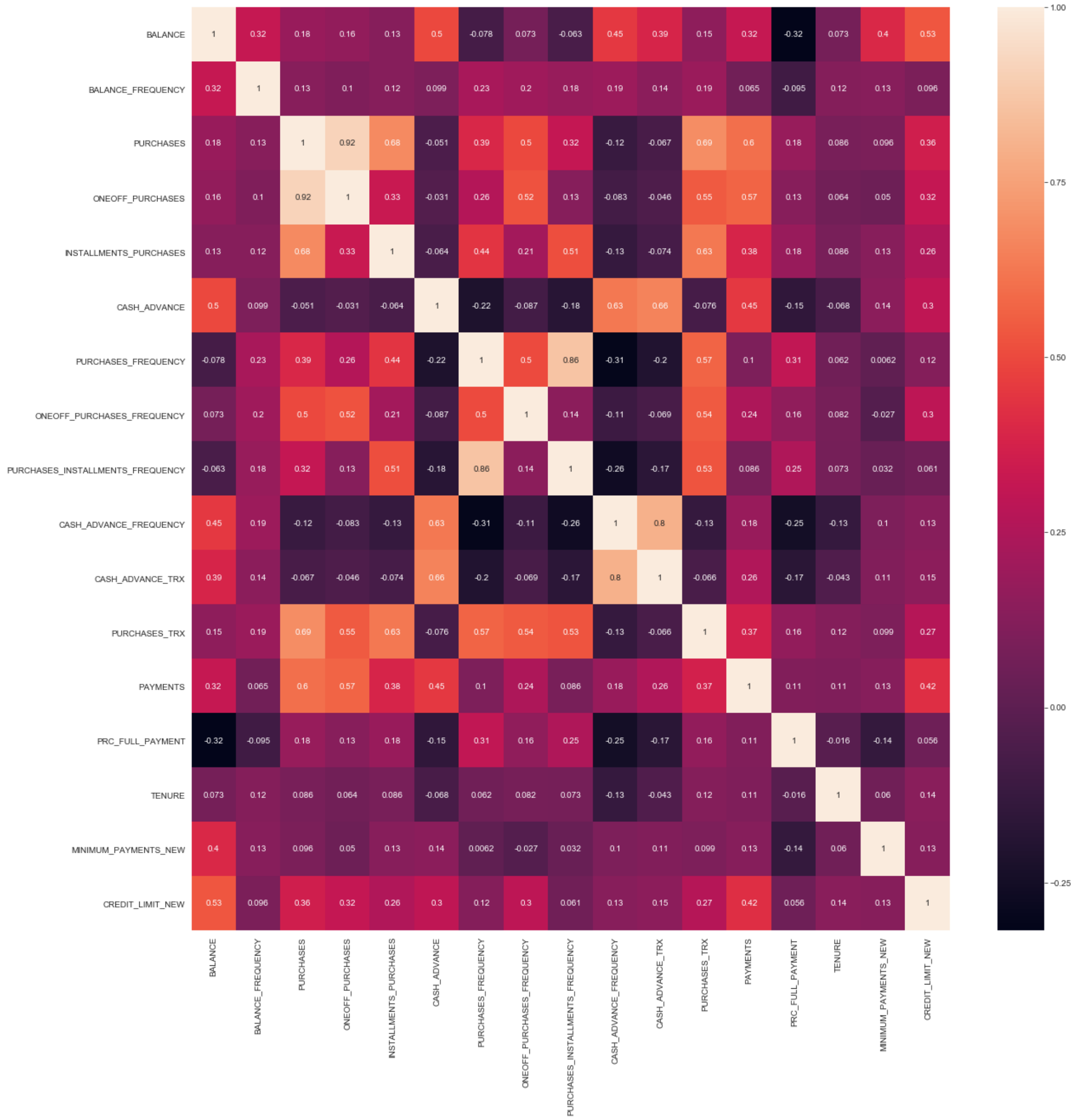
In [16]:

```
sns.pairplot(df)
plt.show()
```



In [17]:

```
plt.figure(figsize=(20,20))
corr_df = df.corr()
sns.heatmap(corr_df,annot=True)
plt.show()
```



3. Model Building :

In [18]:

```
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df,test_size=0.2,random_state=42)
```

3.1 Normalizing the values:

In [19]:

```
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
train_df = mm.fit_transform(train_df)
test_df = mm.transform(test_df)
```

In [20]:

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
train_df = pt.fit_transform(train_df)
test_df = pt.transform(test_df)
```

3.2 K-MEANS :

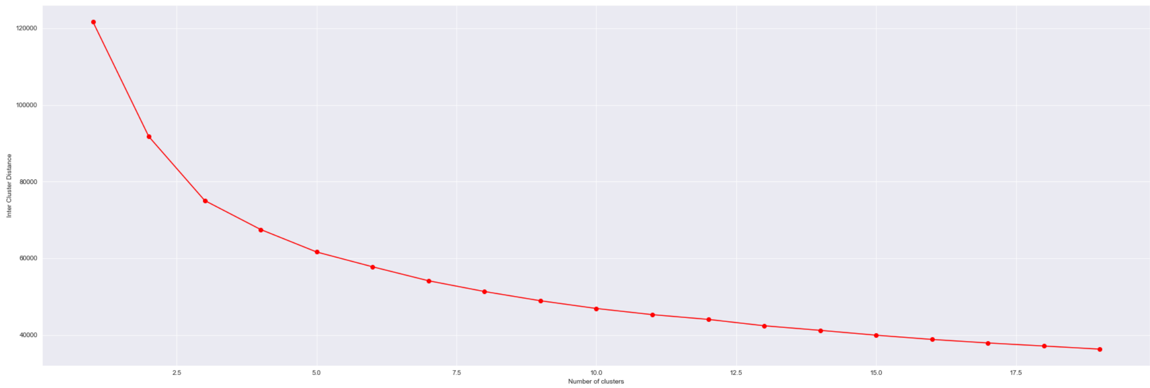
In [21]:

```
from sklearn.cluster import KMeans
```

Finding optimum number of clusters for grouping:

In [22]:

```
interclusterdistance = []

for clusters in range(1,20):
    km = KMeans(n_clusters = clusters,init ='k-means++', max_iter=300,random_state=42)
    km.fit(train_df)
    interclusterdistance.append(km.inertia_)

#plotting the values
plt.figure(figsize=(30,10))
plt.plot(range(1, 20), interclusterdistance, marker='o', color='r')
plt.xlabel('Number of clusters')
plt.ylabel('Inter Cluster Distance')
plt.show()
```



In [23]:

```
km = KMeans(n_clusters = 6,init ='k-means++', max_iter=300,random_state=42)
km.fit(train_df)
y_pred = km.predict(train_df)
```

In [24]:

```
cluster_df = pd.DataFrame(train_df,columns = df.columns)
cluster_df['clusters'] = y_pred
cluster_df.head(10)
```

Out[24]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	-0.937132	-1.267216	-0.029433	0.338526	
1	1.291880	0.630684	-1.038959	-0.770843	
2	-0.465038	0.630684	-1.038959	-0.770843	
3	-1.148135	-1.698717	1.370433	-0.659492	
4	1.388866	0.630684	-0.912679	-0.576602	
5	-0.978519	-1.972925	-0.513848	-0.770843	
6	-1.132653	0.630684	-0.284999	-0.770843	
7	0.784516	0.630684	1.955706	2.101295	
8	0.270674	0.630684	-1.002768	-0.714616	
9	-0.669240	-1.065671	0.219068	-0.005898	

In [25]:

```
cluster_df['clusters'].value_counts()
```

Out[25]:

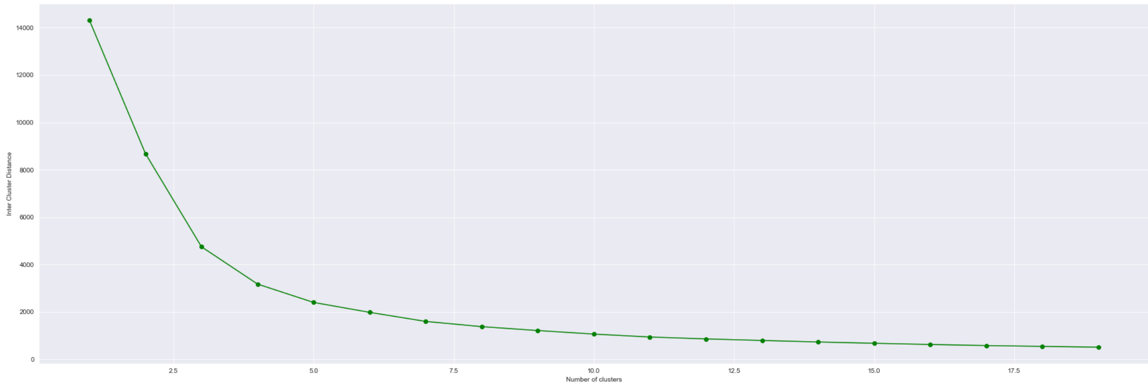
```
2    1617
1    1509
5    1394
4    1002
0     911
3     727
Name: clusters, dtype: int64
```

In [26]:

```
X = cluster_df[['BALANCE','PURCHASES']].to_numpy()
```

In [27]:

```
interclusterdistance = []

for clusters in range(1,20):
    km = KMeans(n_clusters = clusters,init ='k-means++', max_iter=300,random_state=42)
    km.fit(X)
    interclusterdistance.append(km.inertia_)

#plotting the values
plt.figure(figsize=(30,10))
plt.plot(range(1, 20), interclusterdistance, marker='o', color='g')
plt.xlabel('Number of clusters')
plt.ylabel('Inter Cluster Distance')
plt.show()
```



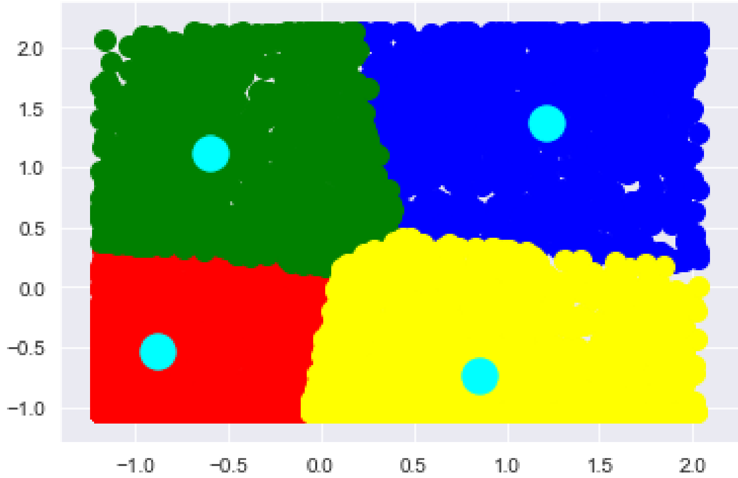
Considering only BALANCE variable, we can choose k = 4

In [28]:

```
km = KMeans(n_clusters = 4,init ='k-means++', max_iter=300,random_state=42)
km.fit(X)
y_balance_pred = km.predict(X)
```

In [29]:

```
plt.scatter(X[y_balance_pred==0, 0], X[y_balance_pred==0, 1], s=100, c='red', label = 'Cluster 1')
plt.scatter(X[y_balance_pred==1, 0], X[y_balance_pred==1, 1], s=100, c='blue', label = 'Cluster 2')
plt.scatter(X[y_balance_pred==2, 0], X[y_balance_pred==2, 1], s=100, c='green', label ='Cluster 3')
plt.scatter(X[y_balance_pred==3, 0], X[y_balance_pred==3, 1], s=100, c='yellow', label ='Cluster 4')

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], s=300, c='cyan', label = 'Centroids')
plt.show()
```



3.2 DBSCAN :

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density. K-Means is not capable of creating clusters of arbitary shape. This is were DBSCAN is helpful.

In [30]:

```
from sklearn.cluster import DBSCAN
```

In [31]:

```
dbscan = DBSCAN(eps=2,min_samples=6)
dbscan.fit(train_df)
y_dbscan_pred = dbscan.labels_
y_dbscan_pred
```

Out[31]:

```
array([ 1,  0,  0, ...,  0,  0, -1], dtype=int64)
```

In [32]:

```
dbscan_df = pd.DataFrame(train_df,columns = df.columns)
dbscan_df['clusters'] = y_dbscan_pred
dbscan_df.head(10)
```

Out[32]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS
0	-0.937132	-1.267216	-0.029433	0.338526	
1	1.291880	0.630684	-1.038959	-0.770843	
2	-0.465038	0.630684	-1.038959	-0.770843	
3	-1.148135	-1.698717	1.370433	-0.659492	
4	1.388866	0.630684	-0.912679	-0.576602	
5	-0.978519	-1.972925	-0.513848	-0.770843	
6	-1.132653	0.630684	-0.284999	-0.770843	
7	0.784516	0.630684	1.955706	2.101295	
8	0.270674	0.630684	-1.002768	-0.714616	
9	-0.669240	-1.065671	0.219068	-0.005898	

In [33]:

```
dbscan_df['clusters'].value_counts()
```

Out[33]:

```
 0    5904
 1     876
-1     361
 2       8
 4       6
 3       5
Name: clusters, dtype: int64
```

In [34]:

```
X = dbscan_df[['BALANCE','PURCHASES']].to_numpy()
```

In [35]:

```
dbscan = DBSCAN(eps=0.075,min_samples=2)
dbscan.fit(X)
y_dbscan_pred = dbscan.labels_
y_dbscan_pred
```

Out[35]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [36]:

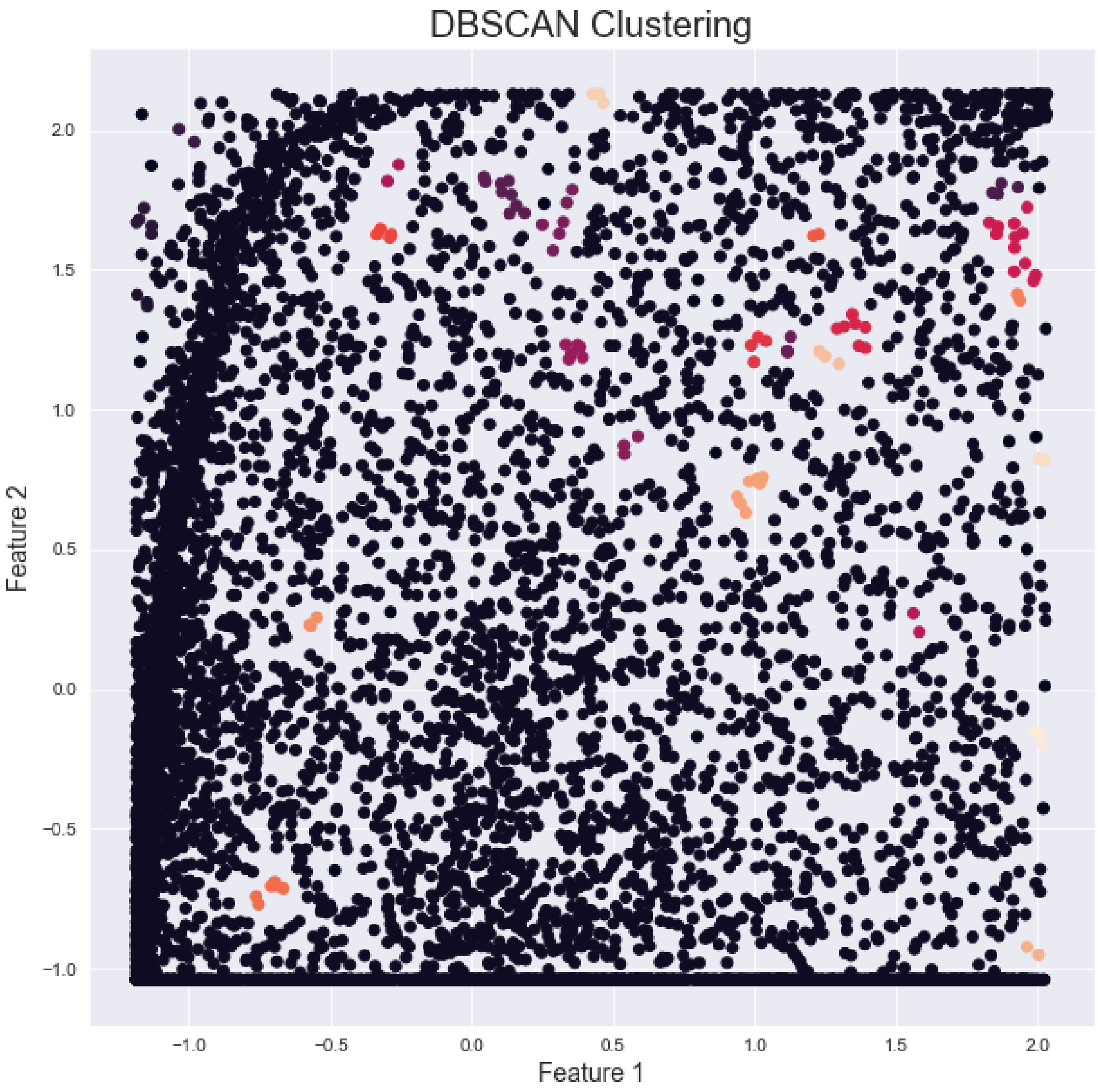
```
dbscan_df['clusters'] = y_dbscan_pred
dbscan_df['clusters'].value_counts()
```

Out[36]:

```
 0     7031
-1       24
 12      12
 5        9
 20       7
 13       7
 17       6
 7        6
 9        6
 2        5
 15       4
 4        4
 14       4
 8        3
 1        3
 23       3
 19       3
 6        3
 22       3
 25       3
 21       2
 10       2
 24       2
 18       2
 16       2
 3        2
 11       2
Name: clusters, dtype: int64
```

In [37]:

```
plt.figure(figsize=(10,10))
plt.scatter(dbscan_df['BALANCE'],dbscan_df['PURCHASES'],c=dbscan_df['clusters'])
plt.title('DBSCAN Clustering',fontsize=20)
plt.xlabel('Feature 1',fontsize=14)
plt.ylabel('Feature 2',fontsize=14)
plt.show()
```

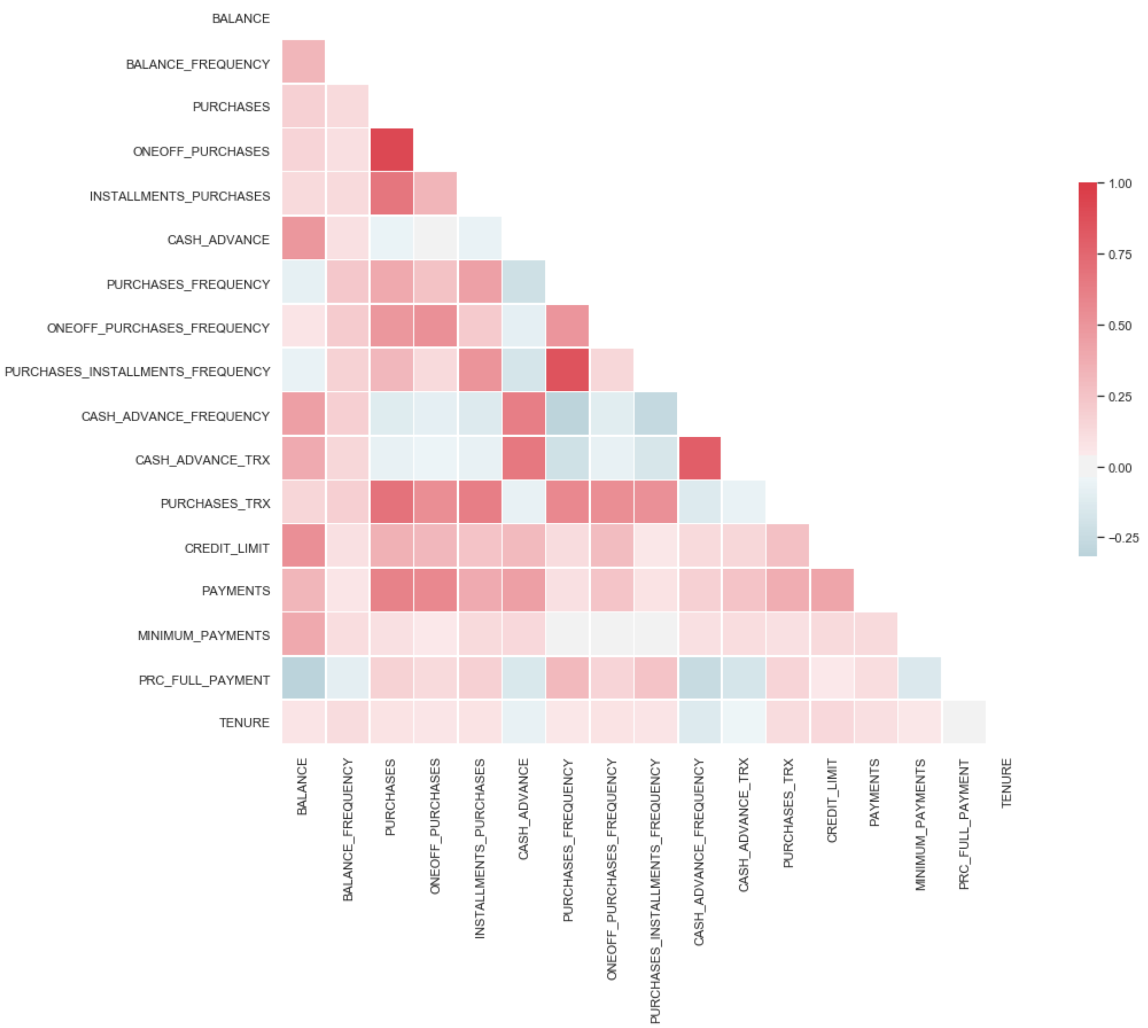


In [41]:

```
sns.set(style = "white")
cor_matrix = df.corr()

mask = np.triu(np.ones_like(cor_matrix, dtype = np.bool))

plt.figure(figsize = (15, 12))

cmap = sns.diverging_palette(220, 10, as_cmap = True)

sns.heatmap(cor_matrix, mask = mask, cmap = cmap, center = 0,
            square = True, linewidths = .5, cbar_kws = {"shrink": .5})
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x23bce88e5f8>
```



In [42]:

```
cols =  ['BALANCE',
         'PURCHASES',
         'ONEOFF_PURCHASES',
         'INSTALLMENTS_PURCHASES',
         'CASH_ADVANCE',
         'CASH_ADVANCE_TRX',
         'PURCHASES_TRX',
         'CREDIT_LIMIT',
         'PAYMENTS',
         'MINIMUM_PAYMENTS',
        ]

df[cols] = np.log(1 + df[cols])

df.head()
```

Out[42]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	3.735304	0.818182	4.568506	0.000000	
1	8.071989	0.909091	0.000000	0.000000	
2	7.822504	1.000000	6.651791	6.651791	
3	7.419183	0.636364	7.313220	7.313220	
4	6.707735	1.000000	2.833213	2.833213	

In [43]:

```
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)

scaled_df = pd.DataFrame(scaled_df, index = df.index, columns = df.columns)

scaled_df.describe()
```

Out[43]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTAI
count	8.950000e+03	8.950000e+03	8.950000e+03	8.950000e+03	
mean	-2.666861e-17	1.209548e-14	1.379306e-15	1.040682e-14	
std	1.000056e+00	1.000056e+00	1.000056e+00	1.000056e+00	
min	-3.060633e+00	-3.703271e+00	-1.679855e+00	-9.870896e-01	
25%	-6.455634e-01	4.904486e-02	-4.097152e-01	-9.870896e-01	
50%	3.039373e-01	5.180838e-01	3.403734e-01	1.414854e-01	
75%	7.284269e-01	5.180838e-01	7.246132e-01	9.722184e-01	
max	1.834341e+00	5.180838e-01	2.023087e+00	2.283062e+00	

Agglomerative Clustering

In [44]:

```
cov = scaled_df.cov()
cov_inv = pd.DataFrame(np.linalg.inv(cov.values), cov.columns, cov.index)

dist = distance.pdist(scaled_df, 'mahalanobis', VI = cov_inv)
dist_mat = distance.squareform(dist)
dist_mat.shape
```

Out[44]:

```
(8950, 8950)
```

In [45]:

```
def clustering_linkage(dist, link):
    hier = linkage(dist, link)
    hier = np.around(hier, decimals = 2)
    fig = plt.figure(figsize=(25, 10))
    dn = dendrogram(hier, p = 30, truncate_mode = 'lastp')
    plt.show()
    return hier
```

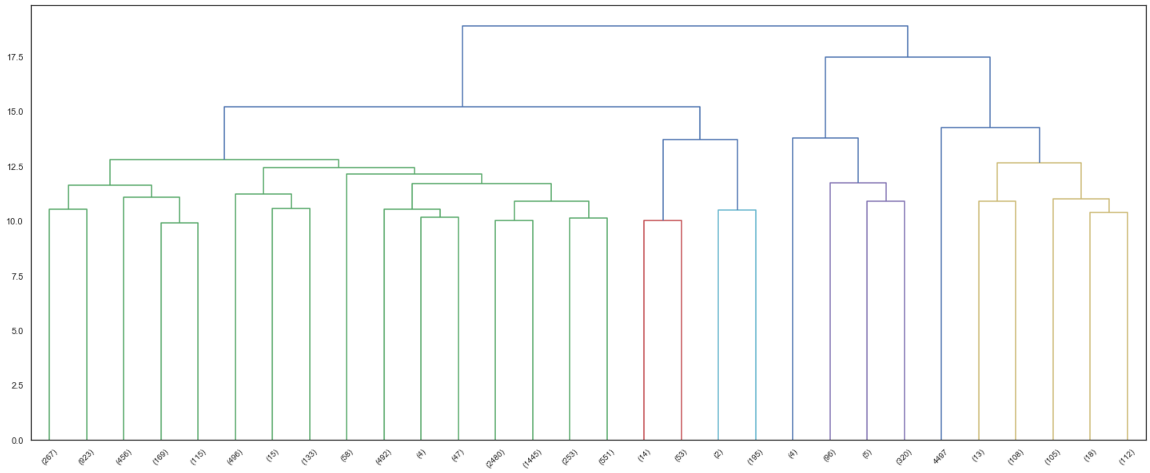
In [46]:

```
def get_distances(hier):
    distances = hier[-20:,2]
    num_clus = np.arange(20, 0, -1)


    d = {'Number of Clusters': num_clus, 'Distance between merged Clusters': distances
}
    df_dist = pd.DataFrame(d)

    plt.figure(figsize=(15, 8))
    sns.barplot(x = "Number of Clusters", y = "Distance between merged Clusters", data
= df_dist)
```

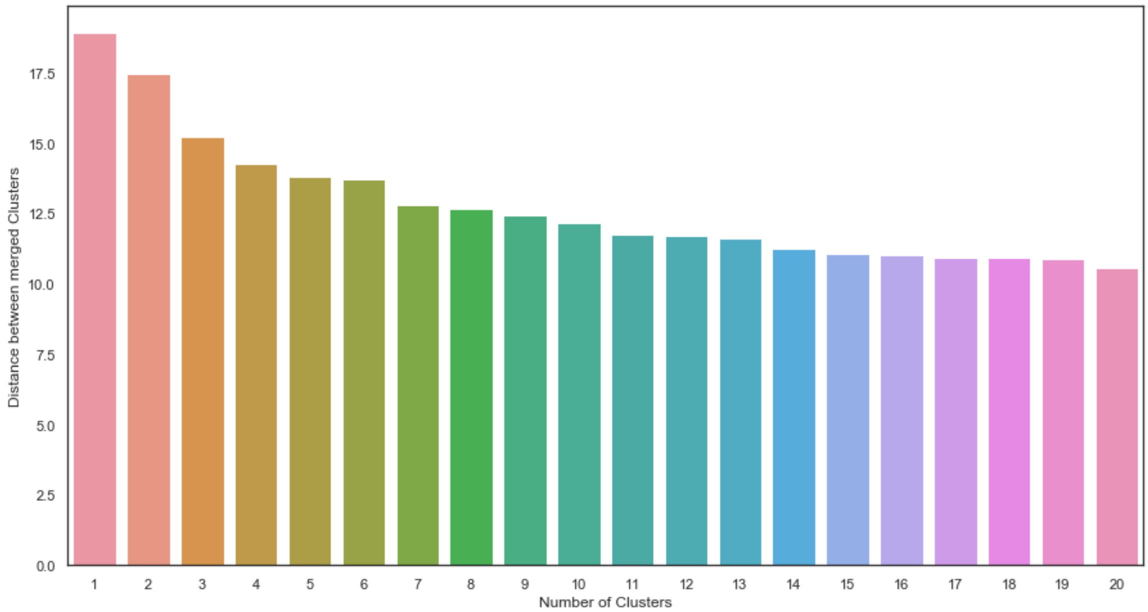
Complete linkage

In [47]:

```
hier_com = clustering_linkage(dist, 'complete')
print("This is what the linkage algorithm returns:")
hier_com
get_distances(hier_com)
```



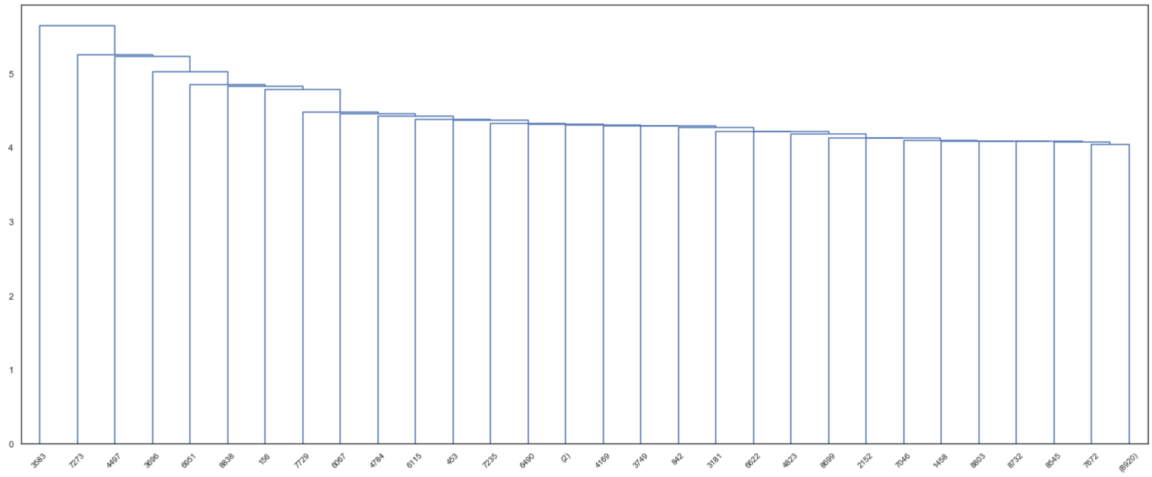
This is what the linkage algorithm returns:

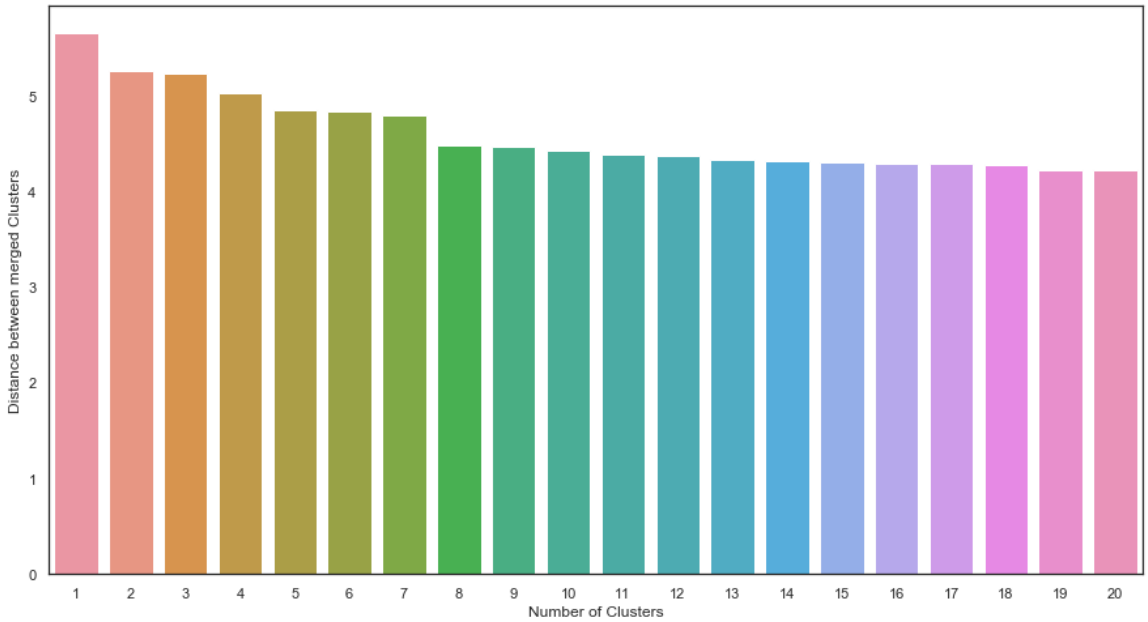


Single linkage

In [48]:

```
hier_sin = clustering_linkage(dist, 'single')
get_distances(hier_sin)
```





In [ ]: