```
# Importing the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import *
from sklearn.decomposition import PCA,KernelPCA
from sklearn.decomposition import TSne
from sklearn.cluster import KMeans
```

```
# Importing the dataset
df = pd.read_csv('CC GENERAL.csv')
```

```
df.shape
```

```
(8950, 18)
```

```
df.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY.1
0	C10001	40.900749	0.818182	95.40	0.00	95.40	0.000000	0.166667	0.000000	0.083333
1	C10002	3464.656763	0.909091	0.00	0.00	0.00	6442.945483	0.000000	0.000000	0.000000
2	C10003	2901.140642	1.000000	773.17	773.17	0.00	0.000000	1.000000	1.000000	0.000000
3	C10004	1494.143518	0.636364	1499.90	1499.90	0.00	205.788017	0.083333	0.083333	0.000000
4	C10005	817.714335	1.000000	16.00	16.00	0.00	0.000000	0.083333	0.083333	0.000000

```
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, int64, object
memory usage: 1.2 MB
```

From the table we can notice that the dataset consists of numerical values except the "CUST_ID" column which represents information about the id of the customer.

```
df.describe().T.
```

```
                                   count         mean          std  ...        50%        75%        max
BALANCE                            8950   1564.474828  2081.531879  ...  873.385231  2054.372848  19043.138560
BALANCE_FREQUENCY                  8950      0.877271     0.236904  ...    1.000000     1.000000      1.000000
PURCHASES                          8950   1003.204834  2136.636089  ...  361.280000  1110.130000  49039.570000
ONEOFF_PURCHASES                   8950    592.437371  1659.826875  ...    0.000000   577.405000  40761.250000
INSTALLMENTS_PURCHASES             8950    411.067645  904.338115  ...   89.000000   468.637500  22500.000000
CASH_ADVANCE                       8950   978.871112  2097.163877  ...  300.093725   578.391739  47137.211760
PURCHASES_FREQUENCY                8950      0.490351     0.401371  ...    0.500000     0.916667      1.000000
ONEOFF_PURCHASES_FREQUENCY         8950      0.202458     0.298336  ...    0.083333     0.300000      1.000000
PURCHASES_INSTALLMENTS_FREQUENCY   8950      0.364437     0.398373  ...    0.166667     0.750000      1.000000
CASH_ADVANCE_FREQUENCY             8950      0.135132     0.200121  ...    0.000000     0.222222      1.500000
CASH_ADVANCE_TRX                   8950      3.248827     6.824647  ...    7.000000    16.000000    123.000000
PURCHASES_TRX                      8950     14.709832    24.857649  ...   12.500000    17.000000    358.000000
CREDIT_LIMIT                       8949   4494.449450  3273.797586  ...  3000.000000  6500.000000  30000.000000
PAYMENTS                           8950   1733.143852  2895.063757  ...  856.901546  1901.134317  50721.483360
MINIMUM_PAYMENTS                   8637    844.906767  2332.792322  ...  312.358706   732.335310  76406.207520
PRC_FULL_PAYMENT                   8950      0.153715     0.292499  ...    0.000000     0.142857      1.000000
TENURE                             8950     11.517318     1.338331  ...   12.000000    12.000000     12.000000

[17 rows x 8 rows]
```

Some descriptive statistics about the data

```
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, int64, object
memory usage: 1.2 MB
```

From the table we can notice that the dataset consists of numerical values except the "CUST_ID" column which represents information about the id of the customer.

```
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
```

```
df[df['CREDIT_LIMIT'].isnull()]
```

```
  CUST_ID  BALANCE  BALANCE_FREQUENCY  PURCHASES  ...  PRC_FULL_PAYMENT  TENURE
5  C10006  817.71              1.000       16.00  ...             0.00      12
```

```
df.fillna(df.mean(),inplace=True)
```

```
df[df['CREDIT_LIMIT'].isnull()]
```

```
Empty DataFrame
Columns: [CUST_ID, 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]
Index: []

[0 rows x 18 columns]
```

Now the data are ready to completely numerical.

```
df.nunique()
```

```
CUST_ID                           8950
BALANCE                           8709
BALANCE_FREQUENCY                   43
PURCHASES                         6203
ONEOFF_PURCHASES                  4014
INSTALLMENTS_PURCHASES            4452
CASH_ADVANCE                      4323
PURCHASES_FREQUENCY                 47
ONEOFF_PURCHASES_FREQUENCY          41
PURCHASES_INSTALLMENTS_FREQUENCY    50
CASH_ADVANCE_FREQUENCY              54
CASH_ADVANCE_TRX                    65
PURCHASES_TRX                      173
CREDIT_LIMIT                      8952
PAYMENTS                          8711
MINIMUM_PAYMENTS                  5782
PRC_FULL_PAYMENT                    47
TENURE                               7
dtype: int64
```

We define a threshold of variance 0.01 and all columns with frequency less than lower unique than this threshold.

```
# Dropping the columns with low variance
df.drop(['BALANCE_FREQUENCY', 'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY', 'PRC_FULL_PAYMENT', 'TENURE'],axis=1,inplace=True)
df
```

	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
0	40.900749	0.818182	95.40	0.00	95.40	0.000000	0.166667	0.000000	0.083333	0.000000	0	2	1000.0	201.802084	139.509787
1	3464.656763	0.909091	0.00	0.00	0.00	6442.945483	0.000000	0.000000	0.000000	0.250000	4	0	7000.0	4103.032597	1072.340217
2	2901.140642	1.000000	773.17	773.17	0.00	0.000000	1.000000	1.000000	0.000000	0.000000	0	12	7500.0	622.066742	627.260523
3	1494.143518	0.636364	1499.90	1499.90	0.00	205.788017	0.083333	0.083333	0.000000	0.083333	1	1	7500.0	NaN	NaN
4	817.714335	1.000000	16.00	16.00	0.00	0.000000	0.083333	0.083333	0.000000	0.000000	0	1	1000.0	678.334763	244.791237
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8945	28.493517	1.000000	291.12	0.00	291.12	0.000000	0.833333	0.000000	0.666667	0.000000	0	11	1000.0	325.594462	48.886365
8946	19.183215	1.000000	300.00	0.00	300.00	0.000000	0.833333	0.000000	0.666667	0.000000	0	12	1000.0	275.861322	NaN
8947	23.398673	0.833333	144.40	0.00	144.40	0.000000	0.833333	0.000000	0.666667	0.000000	0	8	1000.0	381.549516	77.168672
8948	13.457564	0.833333	0.00	0.00	0.00	36.558778	0.000000	0.000000	0.000000	0.166667	2	0	500.0	052.549959	62.772283
8949	372.708075	0.666667	1093.25	1093.25	0.00	127.040008	0.666667	0.666667	0.000000	0.000000	2	23	1200.0	NaN	159.826518

8950 rows * 10 columns.

Normalizing the dataset

```
scaler = MinMaxScaler().fit_transform(df)
df = pd.DataFrame(scaler,columns=df.columns)
```

```
df.shape
```

```
(8950, 10)
```

Now the data are ready to completely normalized.

Removing columns based on correlation

```
df.corr()
```

	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
BALANCE	1.000000	0.322412	0.181261	0.164411	0.126799	0.496692	-0.078314	-0.063269	-0.064693	0.099388	0.073166	0.072693	0.449522	0.322802	0.394672
BALANCE_FREQUENCY	0.322412	1.000000	0.133674	0.104744	0.086799	0.099505	0.229715	0.202415	0.202443	0.132616	0.125066	0.119264	0.096091	0.065008	0.091037
PURCHASES	0.181261	0.133674	1.000000	0.916845	0.679896	-0.051474	0.393017	0.328919	0.603264	-0.057715	0.084744	0.686775	0.356959	0.603264	0.093515
ONEOFF_PURCHASES	0.164411	0.104744	0.916845	1.000000	0.330622	-0.031357	0.264331	0.548293	0.307689	-0.039404	0.086755	0.542207	0.319721	0.524881	0.059026
INSTALLMENTS_PURCHASES	0.126799	0.086799	0.679896	0.330622	1.000000	-0.064244	0.442418	0.205071	0.724089	-0.036771	0.077492	0.359379	0.329672	0.267538	0.097548
CASH_ADVANCE	0.496692	0.099505	-0.051474	-0.031357	-0.064244	1.000000	-0.131799	-0.086754	-0.063004	0.466786	0.301226	-0.123062	0.403615	0.379201	0.315274
PURCHASES_FREQUENCY	-0.078314	0.229715	0.393017	0.264331	0.442418	-0.131799	1.000000	0.501305	0.862921	-0.307294	-0.244089	0.381371	-0.095851	0.255108	0.381422
ONEOFF_PURCHASES_FREQUENCY	-0.063269	0.202415	0.328919	0.548293	0.205071	-0.086754	0.501305	1.000000	0.287124	-0.239614	-0.202071	0.364789	-0.078265	0.190851	0.380086
PURCHASES_INSTALLMENTS_FREQUENCY	-0.064693	0.202443	0.603264	0.307689	0.724089	-0.063004	0.862921	0.287124	1.000000	-0.329672	-0.267172	0.415267	-0.076716	0.255083	0.268160
CASH_ADVANCE_FREQUENCY	0.099388	0.132616	-0.057715	-0.039404	-0.036771	0.466786	-0.307294	-0.239614	-0.329672	1.000000	0.800480	-0.131687	0.133351	0.143773	0.202011
CASH_ADVANCE_TRX	0.073166	0.125066	0.084744	0.086755	0.077492	0.301226	-0.244089	-0.202071	-0.267172	0.800480	1.000000	0.156732	0.132616	0.098788	0.146166
PURCHASES_TRX	0.072693	0.119264	0.686775	0.542207	0.359379	-0.123062	0.381371	0.364789	0.415267	-0.131687	0.156732	1.000000	0.095851	0.255173	0.073226
CREDIT_LIMIT	0.449522	0.096091	0.356959	0.319721	0.329672	0.403615	-0.095851	-0.078265	-0.076716	0.133351	0.132616	0.095851	1.000000	0.421635	0.139018
PAYMENTS	0.322802	0.065008	0.603264	0.524881	0.267538	0.379201	0.255108	0.190851	0.255083	0.143773	0.098788	0.255173	0.421635	1.000000	0.351267
MINIMUM_PAYMENTS	0.394672	0.091037	0.093515	0.059026	0.097548	0.315274	0.381422	0.380086	0.268160	0.202011	0.146166	0.073226	0.139018	0.351267	1.000000
PRC_FULL_PAYMENT	-0.095851	0.070349	0.315772	0.381051	0.239790	-0.268228	0.364043	0.329404	0.415267	-0.131687	-0.133351	0.095851	-0.139018	-0.329672	-0.139018
TENURE	0.072693	0.119264	0.686775	0.542207	0.359379	-0.123062	0.381371	0.364789	0.415267	-0.131687	-0.133351	0.095851	-0.139018	-0.329672	-0.139018

We define the threshold as 0.5 and all remove one variable from the pairs that have higher correlation greater than that. We will remove "CREDIT_LIMIT" for the pair "BALANCE" - "CREDIT_LIMIT" and "PURCHASES_INSTALLMENTS_FREQUENCY" from the pair "PURCHASES_FREQUENCY" - "PURCHASES_INSTALLMENTS_FREQUENCY".

```
df.drop(['CREDIT_LIMIT', 'PURCHASES_INSTALLMENTS_FREQUENCY'],axis=1,inplace=True)
```

```
df.shape
```

```
(8950, 8)
```

Now we have 7 features left and preprocessing is finished. We can move on to PCA.

PCA

```
# Instantiating the data with the help of PCA to be able to visualize the clustering
pca = PCA(n=2)
pca_df = pca.fit_transform(df)
```

```
# Creating a scree plot of principal components
# Importing the required libraries
pca = PCA(random_state=42)
pca.fit(df).transform(df)
var = np.cumsum(np.round(pca.explained_var, decimal = 3))
plt.rcParams['figure.dpi'] = 300
plt.rcParams['figure.figsize'] = (23,10)
pc = np.arange(7) + 1
plt.plot(pc, var)
plt.scatter(pc, var, s=200, c='red')
plt.xlabel('Principal Components')
plt.ylabel('Cumulative Variance')
plt.title('Amount of variance explained by each principal component.')
```

We can see that PC1 and PC2 of together explain more 90% of the variance. So we can use these components to create the model and visualize the data.

```
# Replacing the dimensionality of the data to 2 to be able to visualize the clustering
pca = PCA(n=2)
pca_df = pca.fit_transform(df)
```

K-Means

```
# Creating an elbow plot to check model performance based on the number of clusters
# Importing the required libraries of Kmeans
model = KMeans()
# Fitting the data
visualizer = KelbowVisualizer(model, k=(1,10))
visualizer.fit(pca_df)
# Finalizing the plot
plt.rcParams['figure.dpi'] = 300
plt.rcParams['figure.figsize'] = (23,10)
plt.title('Performance of the model based on the number of clusters', fontsize=17)
plt.xlabel('Number of clusters', fontsize=14)
plt.ylabel('Distortion score', fontsize=14)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show();
```

We can see that starting from k=4, the performance of the model starts to change very slowly. So we will choose 4 clusters to fit our model.

```
# Fitting the data points
clusterer = KMeans(n_clusters=4,random_state=42).fit(pca_df)
centers = clusterer.cluster_centers_
# Predicting the clusters
preds = clusterer.predict(pca_df)
# Storing the value in DataFrame
df['cluster'] = preds
df.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	cluster
0	0.040901	0.818182	0.095400	0.000000	0.095400	0.000000	0.166667	0.000000	0.000000	0.000000	0.000833	1,0
1	0.464966	0.909091	0.000000	0.000000	0.000000	0.643094	0.000000	0.000000	0.250000	0.004000	0.000000	3,0
2	0.290114	1.000000	0.077317	0.077317	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	0.004833	0,0
3	0.149414	0.636364	0.149990	0.149990	0.000000	0.020579	0.083333	0.083333	0.000000	0.001000	0.000833	0,0
4	0.081771	1.000000	0.001600	0.001600	0.000000	0.000000	0.083333	0.083333	0.000000	0.000000	0.000833	0,0

8950 rows * 8 columns.

```
df.shape
```

```
(8950, 8)
```

We can see that dataset has 8 columns after that step.

Visualization of clusters

```
# Visualizing the clusters
plt.figure(figsize=(23,10))
sns.scatterplot(pca_df[:,0], pca_df[:,1], hue='cluster', data=df,palette='Paired',s=50)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Clustered data after applying PCA')
```


Creating a final dataframe, which will contain information about the customer ID and the group they belong to.

```
final_df = pd.DataFrame(df['cluster'],df['CUST_ID'])
final_df[0].sort_values().to_df().head(10)
final_df.shape
```

	CUST_ID	cluster
0	C10001	1
1	C10002	3
2	C10003	0
3	C10004	1
4	C10005	1
5	C10006	1
6	C10007	1
7	C10008	1
8	C10009	1
9	C10010	1

8950 rows * 2 columns.