Install Requirements

```
!pip install pycaret
import pycaret
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
```

```
import pycaret
pycaret.__version__
```

```
'3.3.0'
```

```
from pycaret.clustering import *
```

Load the Dataset

```
file_path = '/content/drive/My Drive/YU-ML-Proj-2/Credit_Card.csv'
credits_data = pd.read_csv(file_path)
credits_data.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	O
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	

Setting up the Environmment

```
#init setup
s = setup(credits_data, ignore_features = ['CUST_ID'], session_id = 123)
credits_data.head()
```

	Description	Value
0	Session id	123
1	Original data shape	(8950, 18)
2	Transformed data shape	(8950, 17)
3	Ignore features	1
4	Numeric features	17
5	Rows with missing values	3.5%
6	Preprocess	True
7	Imputation type	simple
8	Numeric imputation	mean
9	Categorical imputation	mode
10	CPU Jobs	-1
11	Use GPU	False
12	Log Experiment	False
13	Experiment Name	cluster-default-name
14	USI	7945

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
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	

Create Model

```
# to check all the available models
models()
```

	Name	Reference
ID		
kmeans	K-Means Clustering	sklearn.cluster._kmeans.KMeans
ap	Affinity Propagation	sklearn.cluster._affinity_propagation.Affinity...
meanshift	Mean Shift Clustering	sklearn.cluster._mean_shift.MeanShift
sc	Spectral Clustering	sklearn.cluster._spectral.SpectralClustering
hclust	Agglomerative Clustering	sklearn.cluster._agglomerative.AgglomerativeCl...
dbscan	Density-Based Spatial Clustering	sklearn.cluster._dbscan.DBSCAN
optics	OPTICS Clustering	sklearn.cluster._optics.OPTICS
birch	Birch Clustering	sklearn.cluster._birch.Birch

```
# train Kmeans Model
kmeans = create_model('kmeans')
print(kmeans)
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.3968	2675.3726	1.3211	0	0	0

```
KMeans(n_clusters=4, random_state=123)
```

```
#train Affinity Propagation mMdel
agglo = create_model('hclust')
print(agglo)
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.3815	2215.9591	1.4904	0	0	0

AgglomerativeClustering(n_clusters=4)

Comparision of Different Models

```
models = ['kmeans', 'ap', 'birch', 'dbscan', 'hclust','meanshift', 'optics', 'sc']

for model_name in models:
    print(f"Creating model: {model_name}")
    model = create_model(model_name)
    print(model)
    print("**********************************************************************************")
```

Creating model: kmeans

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.3968	2675.3726	1.3211	0	0	0

KMeans(n_clusters=4, random_state=123)
****
Creating model: ap

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.1844	693.4131	0.9937	0	0	0

AffinityPropagation()
****
Creating model: birch

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.3815	2215.9591	1.4904	0	0	0

Birch(n_clusters=4)
****
Creating model: dbscan

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0	0	0	0	0	0

DBSCAN(n_jobs=-1)
****
Creating model: hclust

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.3815	2215.9591	1.4904	0	0	0

AgglomerativeClustering(n_clusters=4)
****
Creating model: meanshift

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.4270	152.3064	0.6082	0	0	0

MeanShift(n_jobs=-1)
****
Creating model: optics

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	-0.5428	6.7718	1.3894	0	0	0

OPTICS(n_jobs=-1)
****
Creating model: sc

```
# train Kmeans Model
kmeans = create_model('kmeans', num_clusters=3)
print(kmeans)
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.4523	5621.0578	0.9864	0	0	0

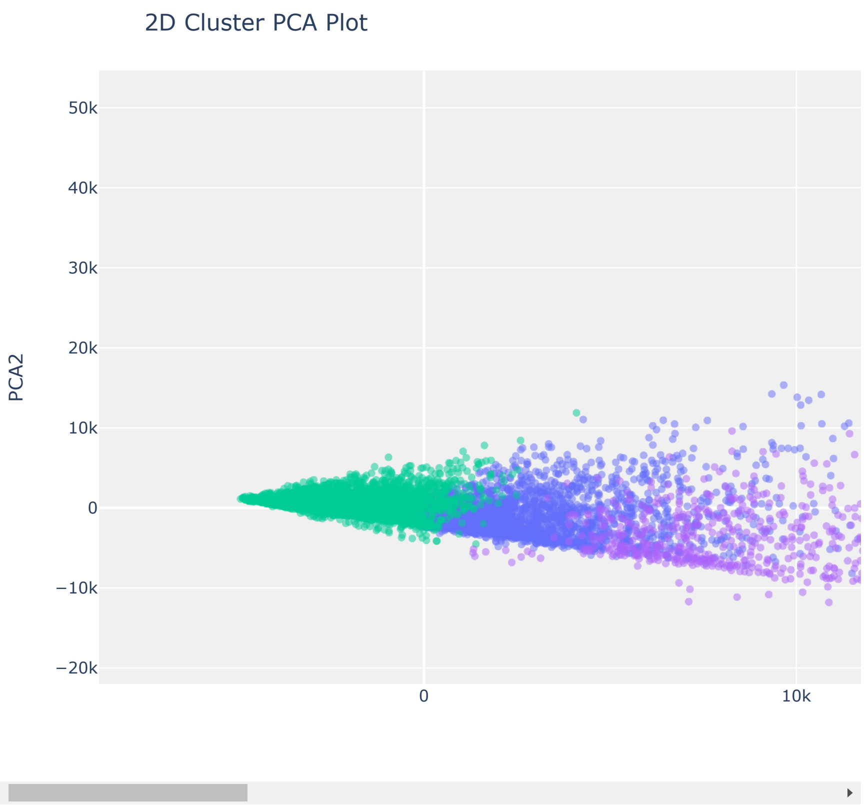
```
KMeans(n_clusters=3, random_state=123)
```

Analyze results

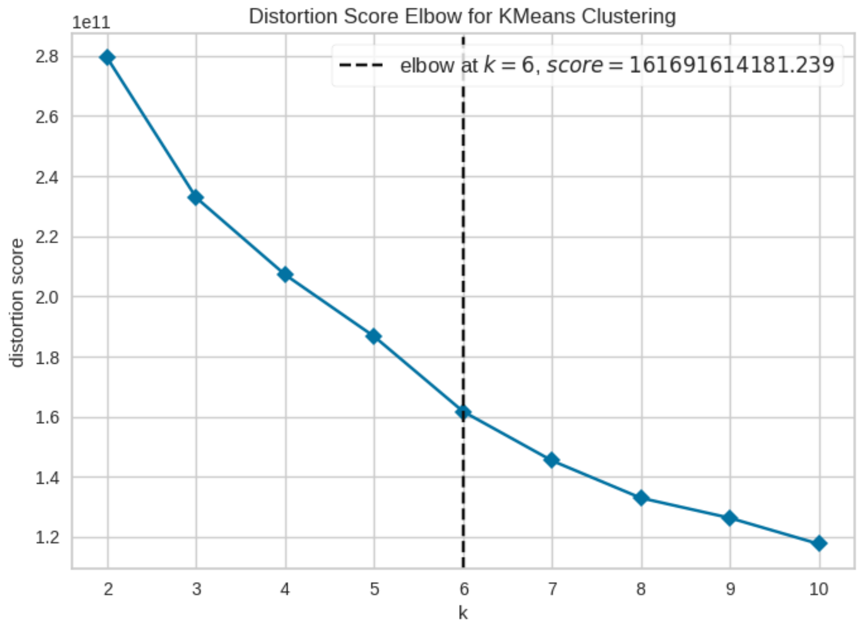
```
#Assign the cluster labels to the dataset to analyze the results
kmean_results = assign_model(kmeans)
kmean_results.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHAS
0	40.900749	0.818182	95.400002	0.000000	95.400(
1	3202.467529	0.909091	0.000000	0.000000	0.000(
2	2495.148926	1.000000	773.169983	773.169983	0.000(
3	1666.670532	0.636364	1499.000000	1499.000000	0.000(
4	817.714355	1.000000	16.000000	16.000000	0.000(

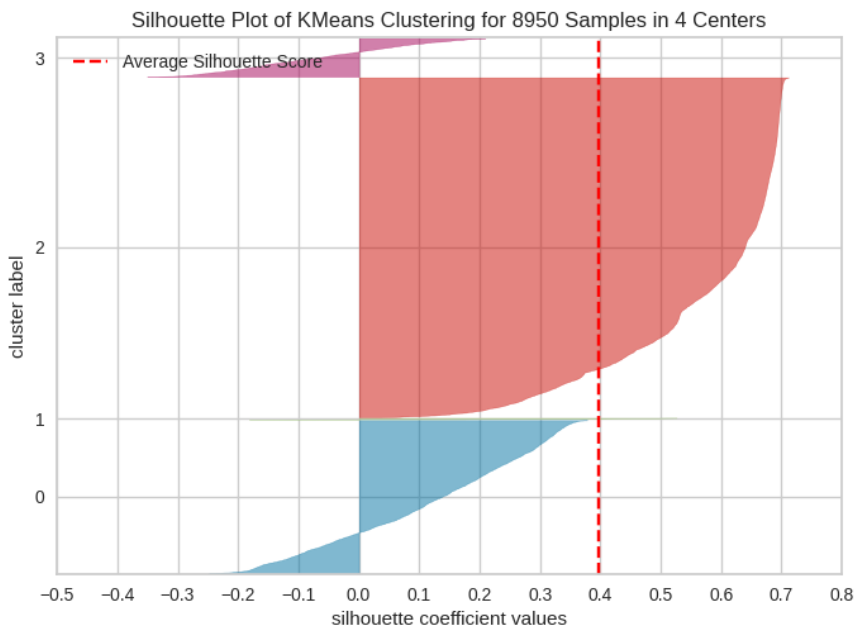
```
#Analyzing using PCA plot
plot_model(kmeans)
```



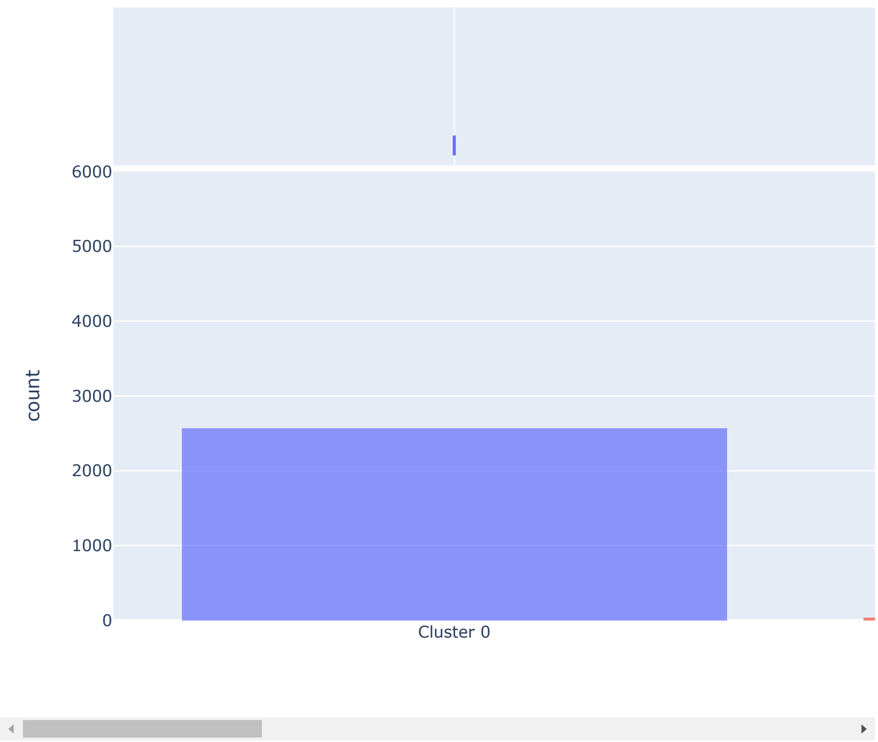

```
##Distrortion  Score Elbow for finding the Optimal number of clusters
plot_model(kmeans, plot = 'elbow')
```



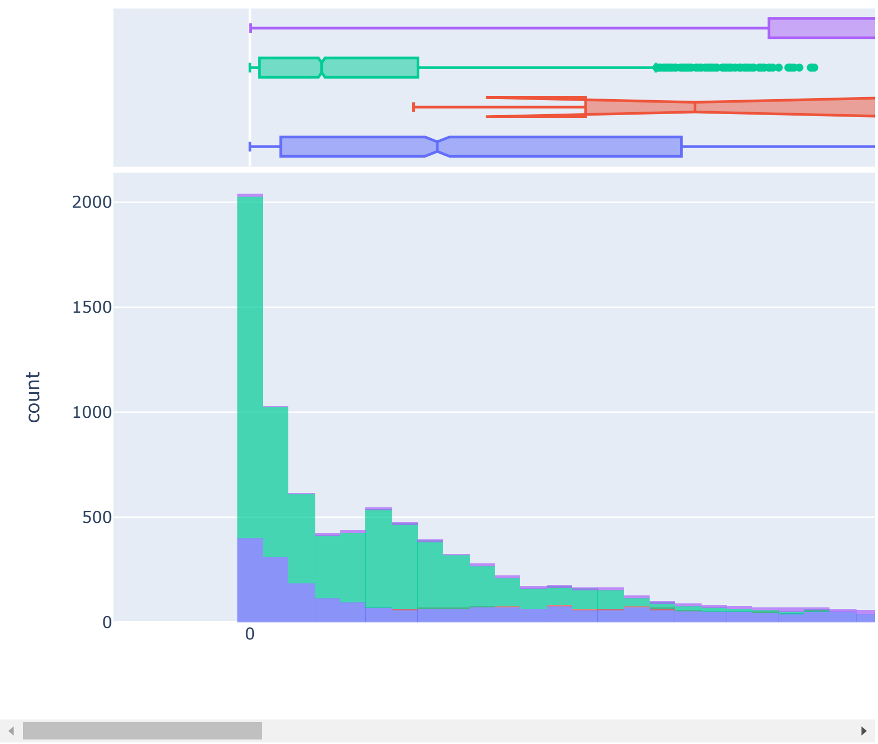

```
#Analyze using the silhouette plot
plot_model(kmeans, plot = 'silhouette')
```




```
#Analyze the distribution of clusters
plot_model(kmeans, plot = 'distribution')
```




```
#Analyze the distribution of features
plot_model(kmeans, plot = 'distribution', feature = 'BALANCE')
```



#Preprocessing and Parameter Tunning

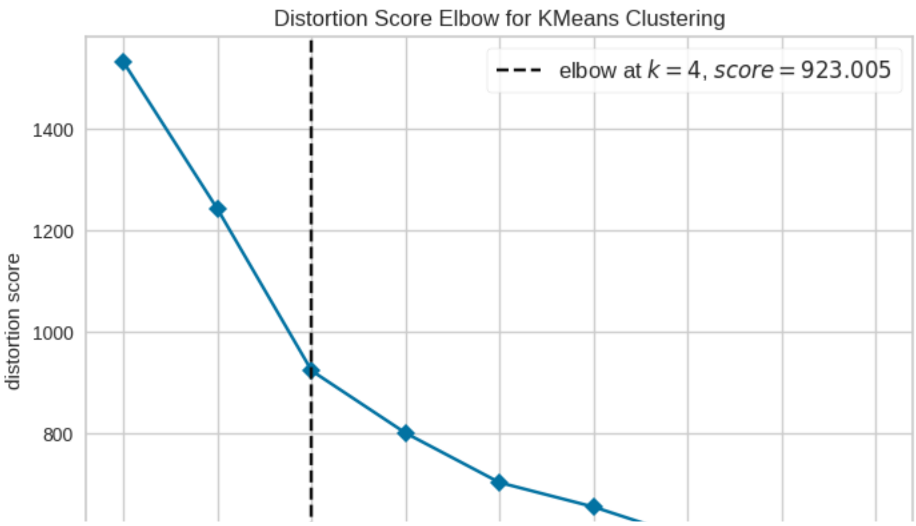
```
drop_features = ['CUST_ID', 'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY','CASH_ADVANCE_FREQUENCY','PRC_FULL_PAYMENT
s = setup(credits_data, normalize = True, ignore_features=drop_features, normalize_method = 'minmax', session_id = 123)
```

	Description	Value
0	Session id	123
1	Original data shape	(8950, 18)
2	Transformed data shape	(8950, 13)
3	Ignore features	5
4	Numeric features	13
5	Rows with missing values	3.5%
6	Preprocess	True
7	Imputation type	simple
8	Numeric imputation	mean
9	Categorical imputation	mode
10	Normalize	True
11	Normalize method	minmax
12	CPU Jobs	-1
13	Use GPU	False
14	Log Experiment	False
15	Experiment Name	cluster-default-name
16	USI	21ce

#Analyzing Models after preprocessing and parameter tunning

```
model = create_model('kmeans')
plot_model(model,'elbow')
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.4688	6065.2136	0.8892	0	0	0




```
models = ['kmeans', 'ap', 'birch', 'dbscan', 'hclust','meanshift', 'optics', 'sc']

for model_name in models:
    print(f"Creating model: {model_name}")
    model = create_model(model_name, num_clusters=3)
    print(model)
    print("*********************************************************************************")
```

```
Creating model: kmeans
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.4523	5621.0578	0.9864	0	0	0

```
KMeans(n_clusters=3, random_state=123)
**********************************************************************************
Creating model: ap
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.1897	1195.1314	1.3301	0	0	0

```
AffinityPropagation()
**********************************************************************************
Creating model: birch
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.4348	7423.7306	0.9552	0	0	0

```
Birch()
**********************************************************************************
Creating model: dbscan
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.5335	58.1533	1.4026	0	0	0

```
DBSCAN(n_jobs=-1)
**********************************************************************************
Creating model: hclust
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.3578	4188.6019	1.2753	0	0	0

```
AgglomerativeClustering(n_clusters=3)
**********************************************************************************
Creating model: meanshift
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.3946	634.5767	1.2488	0	0	0

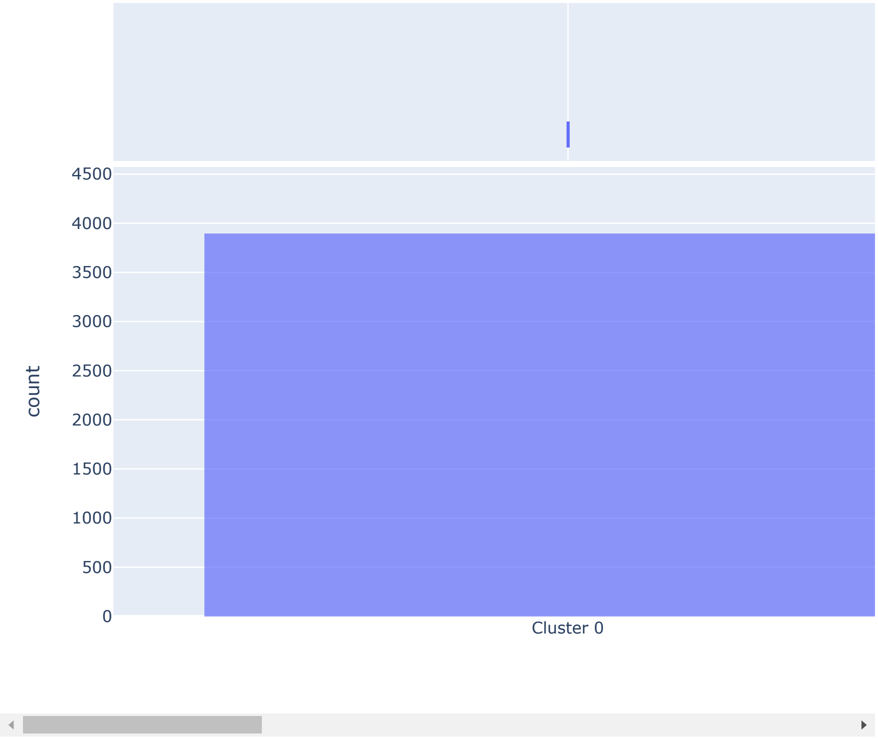
```
MeanShift(n_jobs=-1)
**********************************************************************************
Creating model: optics
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness

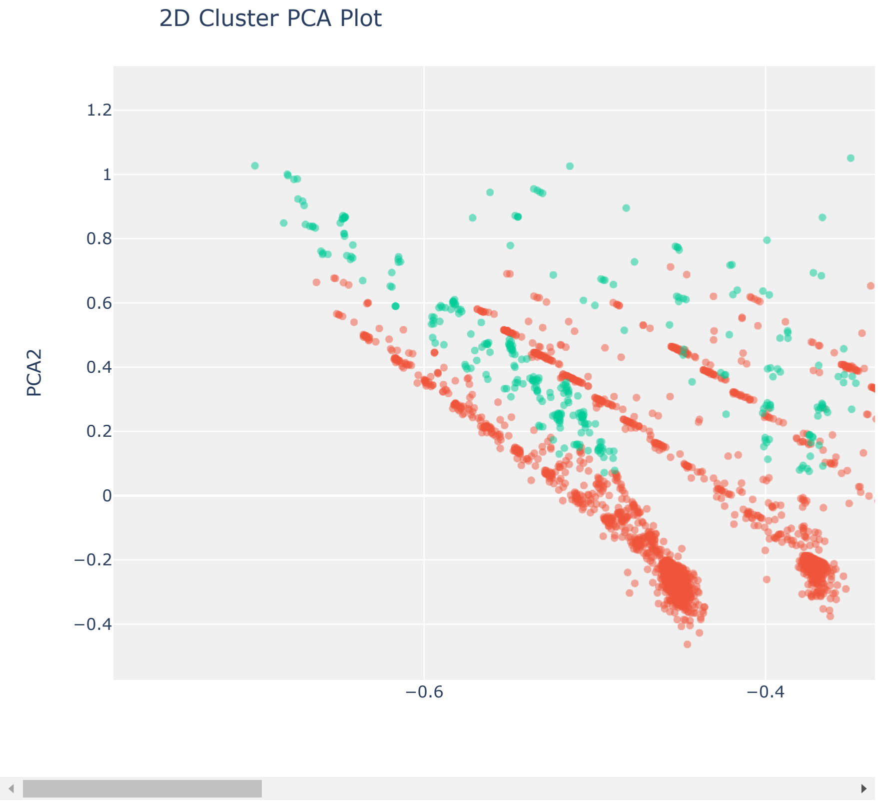
```
# train Kmeans Model
kmeans = create_model('kmeans', num_clusters=3)
print(kmeans)
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.4523	5621.0578	0.9864	0	0	0

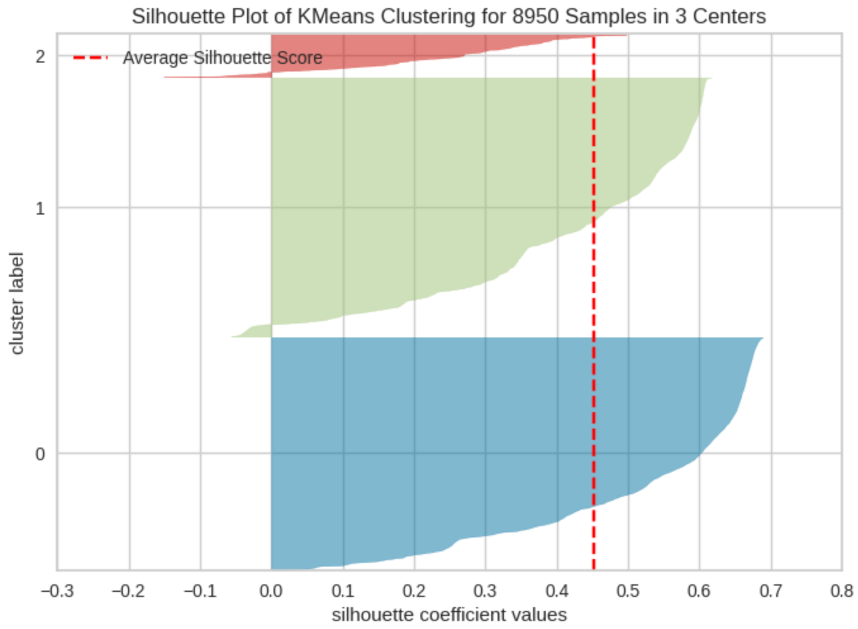
```
#Analyze the distribution of clusters
plot_model(kmeans, plot = 'distribution')
```



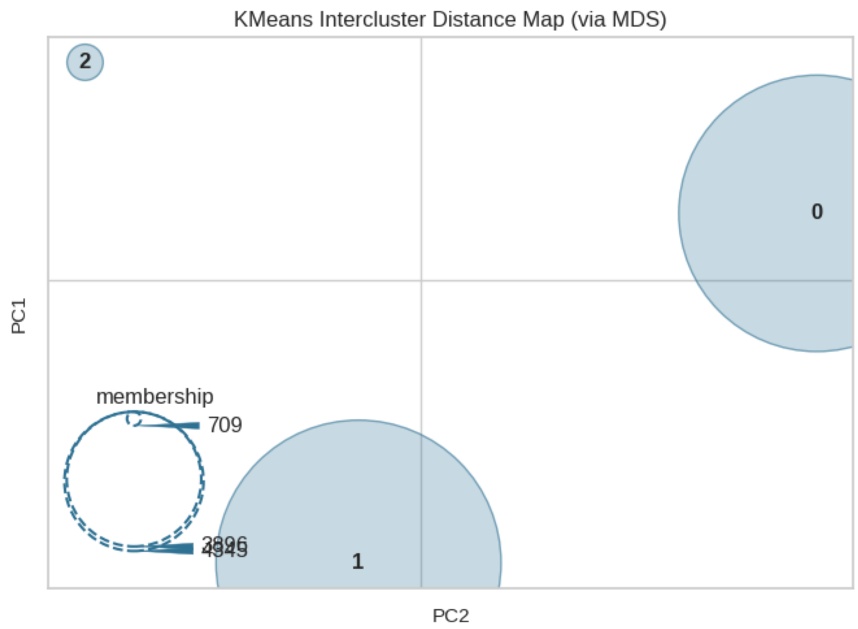

```
plot_model(kmeans)
```




```
plot_model(kmeans, plot = 'silhouette')
```




```
plot_model(kmeans, plot = 'distance')
```




```
#Assign a cluster to a respective data
assign_model(kmeans)
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	CASH_
0	40.900749	0.818182	95.400002	0.000000	95.400002	0.000000	0.166667	
1	3202.467529	0.909091	0.000000	0.000000	0.000000	6442.945312	0.000000	
2	2495.148926	1.000000	773.169983	773.169983	0.000000	0.000000	1.000000	
3	1666.670532	0.636364	1499.000000	1499.000000	0.000000	205.788010	0.083333	
4	817.714355	1.000000	16.000000	16.000000	0.000000	0.000000	0.083333	
...	...	...	...	...	...	...	...	
8945	28.493517	1.000000	291.119995	0.000000	291.119995	0.000000	1.000000	
8946	19.183214	1.000000	300.000000	0.000000	300.000000	0.000000	1.000000	
8947	23.398672	0.833333	144.399994	0.000000	144.399994	0.000000	0.833333	
8948	13.457564	0.833333	0.000000	0.000000	0.000000	36.558777	0.000000	

Saving the Model

```
save_model(kmeans,'project-2')
```

```
Transformation Pipeline and Model Successfully Saved
(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(include=['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'],
                                     transformer=SimpleImputer())),
                 ('categorical_imputer',
                  TransformerWrapper(include=[],
                                     transformer=SimpleImputer(strategy='most_frequent'))),
                 ('trained_model', KMeans(n_clusters=4, random_state=123))]),
 'project-2.pkl')
```

```
# reduce number of clusters to 3
kmeans2 = create_model('kmeans', num_clusters=3)
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.4659	3079.5131	1.1896	0	0	0

```
unique_labels = np.unique(kmeans2.labels_)
print("Unique cluster labels:", unique_labels)
```

```
Unique cluster labels: [0 1 2]
```

```
#train Agglomerative clustering
hclust = create_model('hclust',num_clusters=3)
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.3883	2491.3826	1.0138	0	0	0

Asign Model

```
kmeans_cluster = assign_model(kmeans)
kmeans_cluster
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PUR(
0	40.900749	0.818182	95.400002	0.000000	95.4
1	3202.467529	0.909091	0.000000	0.000000	0.(
2	2495.148926	1.000000	773.169983	773.169983	0.(
3	1666.670532	0.636364	1499.000000	1499.000000	0.(
4	817.714355	1.000000	16.000000	16.000000	0.(
...	...	...	...	...	
8945	28.493517	1.000000	291.119995	0.000000	291.
8946	19.183214	1.000000	300.000000	0.000000	300.(
8947	23.398672	0.833333	144.399994	0.000000	144.:

Start coding or generate with AI.

This notebook is to analyze the Customer's Credit Card data and apply Unsupervised Machine Learning techniques to understand the hidden relationship within the dataset so that appropriate customer base can be identified for marketing or promition purpose to increase sales/revenue etc.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd;
dfCCOrig=pd.read_csv("/content/drive/MyDrive/YU-ML-Proj-1/Week6/Credit_Card.csv")
dfCCOrig.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
```

From above, it was observed that all available variables are of numeric data type except one (CUST_ID)

Perform Null checks (% of Null values)

```
dfCCOrigLen=len(dfCCOrig)
df1=(dfCCOrig.isnull().sum()/dfCCOrigLen)*100
df2=df1[df1.values>0]
df2.sort_values(ascending=False)
```

```
MINIMUM_PAYMENTS    3.497207
CREDIT_LIMIT        0.011173
dtype: float64
```

Perform EDA on Numeric variables

```
dfCCNumeric= dfCCOrig.drop(['CUST_ID', 'BALANCE_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY','CASH_ADVAN
```

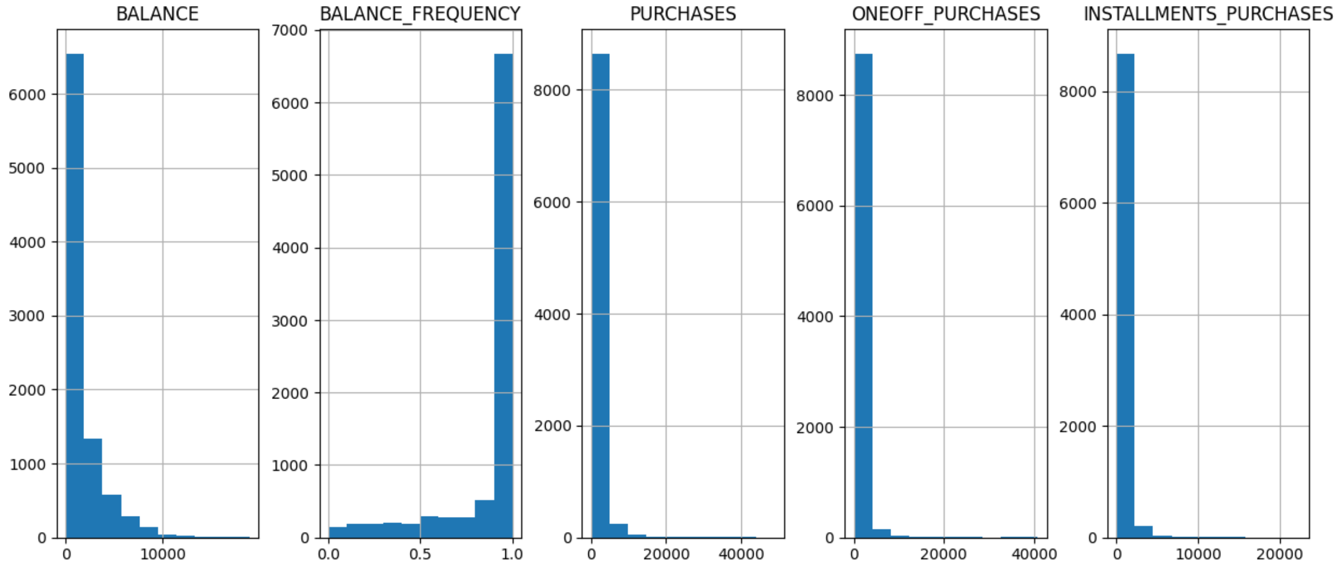
```
dfCCNumeric.info()
```

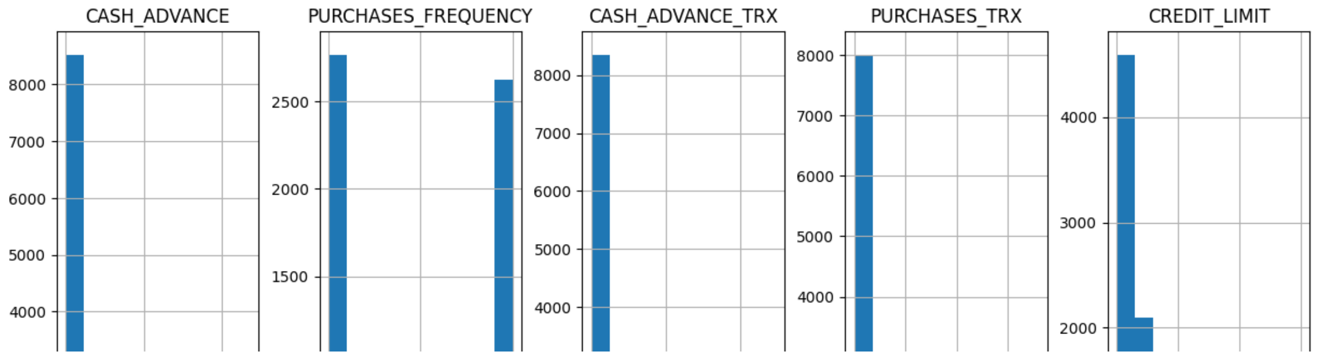
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   BALANCE                 8950 non-null   float64
 1   PURCHASES               8950 non-null   float64
 2   ONEOFF_PURCHASES        8950 non-null   float64
 3   INSTALLMENTS_PURCHASES  8950 non-null   float64
 4   CASH_ADVANCE            8950 non-null   float64
```

```
 5   PURCHASES_FREQUENCY     8950 non-null   float64
 6   CASH_ADVANCE_TRX        8950 non-null   int64
 7   PURCHASES_TRX           8950 non-null   int64
 8   CREDIT_LIMIT            8949 non-null   float64
 9   PAYMENTS                8950 non-null   float64
 10  MINIMUM_PAYMENTS        8637 non-null   float64
 11  TENURE                  8950 non-null   int64
dtypes: float64(9), int64(3)
memory usage: 839.2 KB
```

```
dfCCOrig.drop(['CUST_ID', 'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY','CASH_ADVANCE_FREQUENCY','PRC_FULL_PAYMENT'].
```

```
array([[<Axes: title={'center': 'BALANCE'}>,
        <Axes: title={'center': 'BALANCE_FREQUENCY'}>,
        <Axes: title={'center': 'PURCHASES'}>,
        <Axes: title={'center': 'ONEOFF_PURCHASES'}>,
        <Axes: title={'center': 'INSTALLMENTS_PURCHASES'}>],
       [<Axes: title={'center': 'CASH_ADVANCE'}>,
        <Axes: title={'center': 'PURCHASES_FREQUENCY'}>,
        <Axes: title={'center': 'CASH_ADVANCE_TRX'}>,
        <Axes: title={'center': 'PURCHASES_TRX'}>,
        <Axes: title={'center': 'CREDIT_LIMIT'}>],
       [<Axes: title={'center': 'PAYMENTS'}>,
        <Axes: title={'center': 'MINIMUM_PAYMENTS'}>,
        <Axes: title={'center': 'TENURE'}>, <Axes: >, <Axes: >],
       [<Axes: >, <Axes: >, <Axes: >, <Axes: >, <Axes: >]], dtype=object)
```

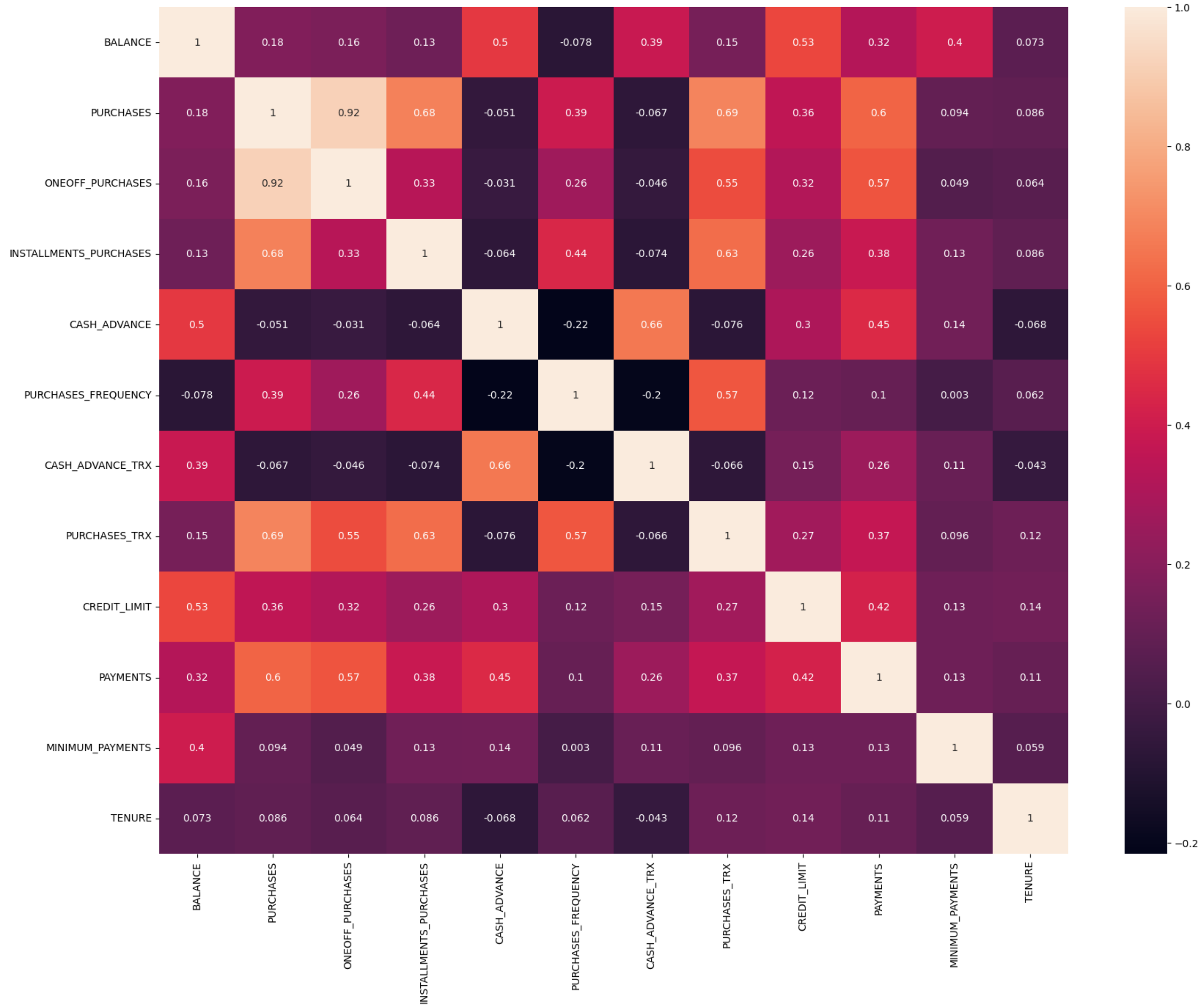




Histogram to understand the corelation

```
import seaborn as sns
from matplotlib import pyplot as plt
plt.subplots(figsize=(20,15))
corr=dfCCNumeric.corr()
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns, annot=True, )
```

```
<Axes: >
```



Perform Imputation to fill the missing or null values for above 2 variables

MINIMUM_PAYMENTS
CREDIT_LIMIT

```
dfCCPP=dfCCOrig
dfmeanMP=dfCCPP["MINIMUM_PAYMENTS"].mean().round(0).astype(int)
dfCCPP["MINIMUM_PAYMENTS"].fillna(dfmeanMP, inplace=True)

dfmeanCL=dfCCPP["CREDIT_LIMIT"].mean().round(0).astype(int)
dfCCPP["CREDIT_LIMIT"].fillna(dfmeanCL, inplace=True)
```

Null Check after Imputation (We can see there is no null data in teh dataset after imputation is done)

```
dfCCOrigLen=len(dfCCPP)
df1=(dfCCPP.isnull().sum()/dfCCOrigLen)*100
df2=df1[df1.values>0]
df2.sort_values(ascending=False)
```

```
Series([], dtype: float64)
```

```
from sklearn import preprocessing

dfCCPPSub=dfCCPP.drop(['CUST_ID'], axis=1)

dfCCPP_norm = preprocessing.normalize(dfCCPPSub)
```

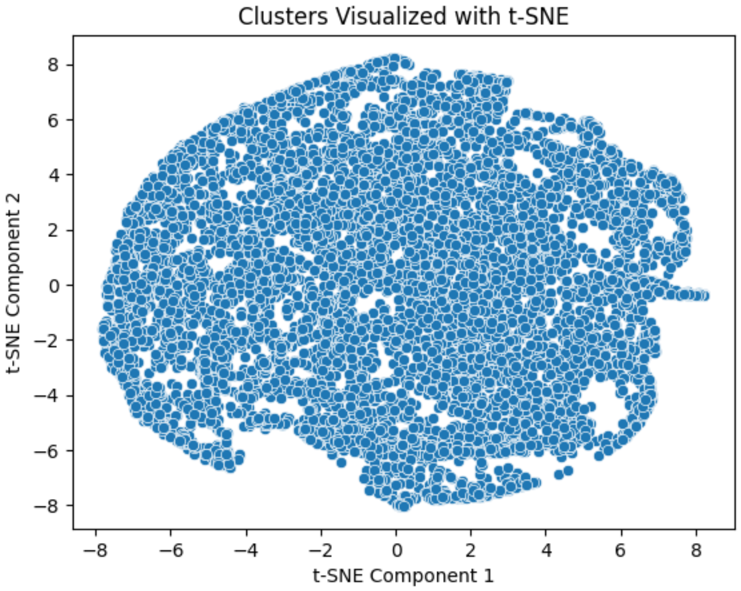
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 4, random_state = 0, n_init='auto')
kmeans.fit(dfCCPP_norm)
```

```
KMeans
KMeans(n_clusters=4, n_init='auto', random_state=0)
```

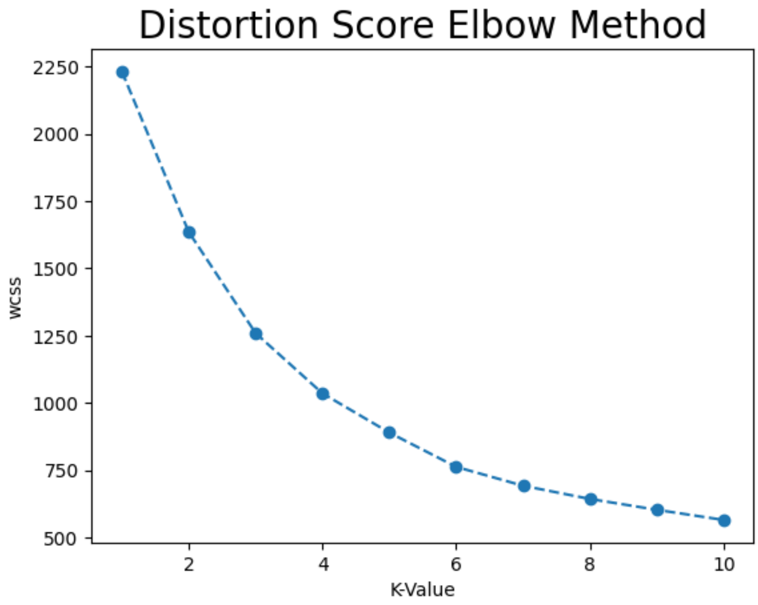
```
kmeans.labels_
```

```
array([1, 2, 1, ..., 1, 1, 0], dtype=int32)
```

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2,  perplexity=30, learning_rate=0.1, n_iter=2000)
X_tsne = tsne.fit_transform(dfCCPP_norm)

# Add the cluster information to the reduced data
df_tsne = pd.DataFrame(data=X_tsne, columns=['t-SNE Component 1', 't-SNE Component 2'])

# Plotting
sns.scatterplot(x='t-SNE Component 1', y='t-SNE Component 2', data=df_tsne)
plt.title('Clusters Visualized with t-SNE')
plt.show()
```




```
wcss= []

for i in range(1,11):
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=300, random_state=42)
    km.fit(dfCCPP_norm)
    wcss.append(km.inertia_)

plt.plot(range(1,11),wcss, marker='o', linestyle='--')
plt.title('Distortion Score Elbow Method', fontsize =20)
plt.xlabel('K-Value')
plt.ylabel('wcss')
plt.show()
```




```
n_clusters = 3
# Instantiate the KMeans model
kmeans = KMeans(n_clusters=n_clusters )

# Fit the model to the scaled data
kmeans.fit(dfCCPP_norm)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
▾       KMeans
KMeans(n_clusters=3)
```

```
kmeans.labels_
```

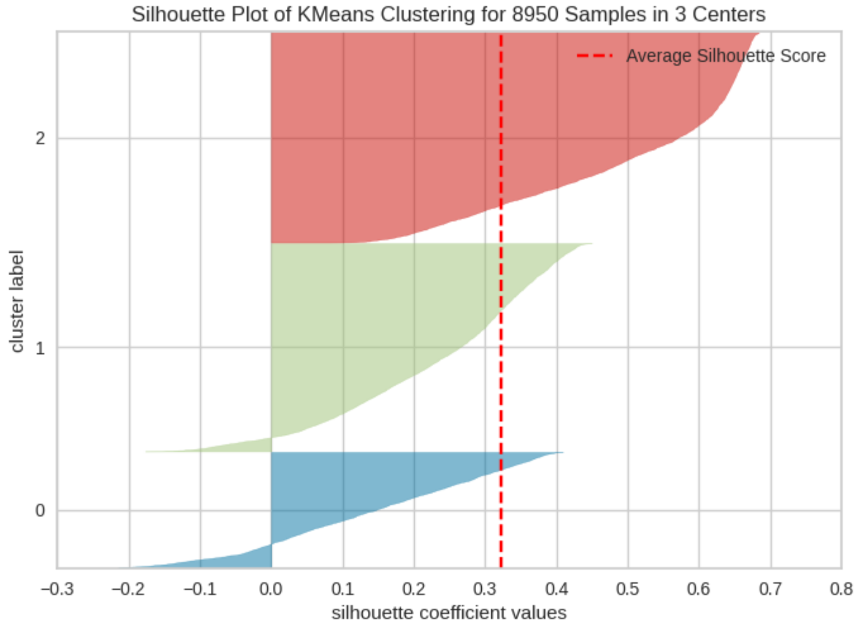
```
array([2, 1, 2, ..., 2, 2, 0], dtype=int32)
```

```
from sklearn.metrics import silhouette_samples, silhouette_score
score = silhouette_score(dfCCPP_norm, kmeans.labels_, metric='euclidean')
```

```
score
```

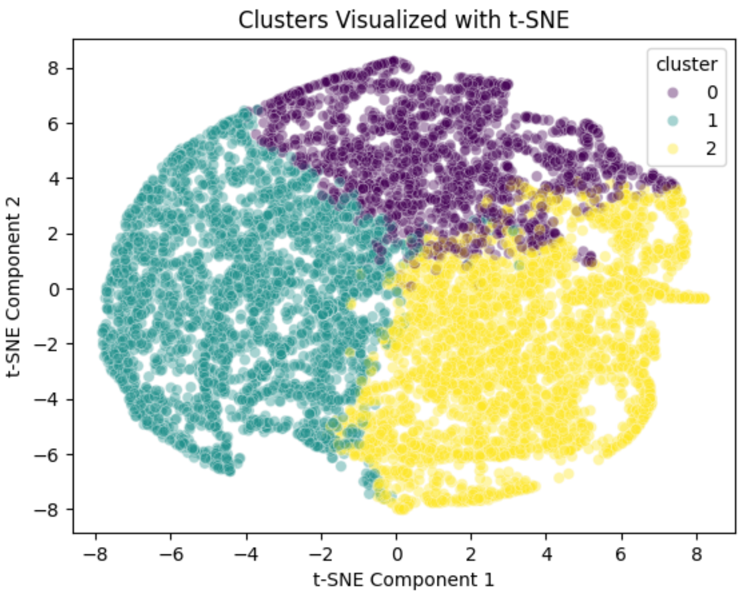
```
0.32222630149805565
```

```
from yellowbrick.cluster import SilhouetteVisualizer
visualizer = SilhouetteVisualizer(kmeans, colors='yellowbrick')

visualizer.fit(dfCCPP_norm)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
```




```
<Axes: title={'center': 'Silhouette Plot of KMeans Clustering for 8950 Samples in 3
Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>
```

```
df_tsne['cluster'] = kmeans.labels_

# Plotting
sns.scatterplot(x='t-SNE Component 1', y='t-SNE Component 2', hue='cluster', data=df_tsne, palette='viridis', alpha=0.4 )
plt.title('Clusters Visualized with t-SNE')
plt.show()
```




```
pca_labels = kmeans.fit_predict(dfCCPP_norm)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
```

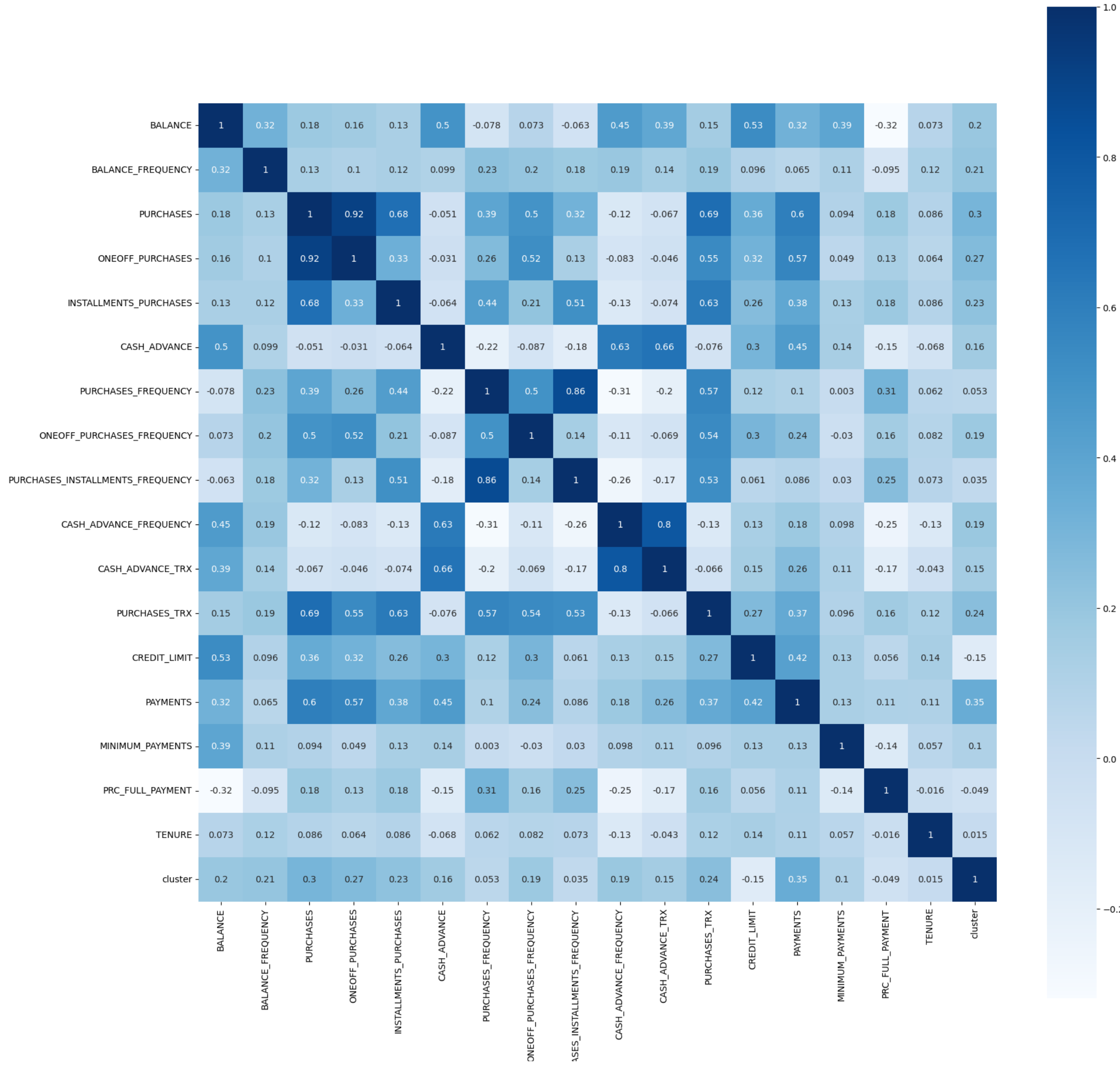
```
dfCCOrig['cluster']=pca_labels
```

```
dfCCOrig.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	(
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	

```
corr = dfCCOrig.corr()
plt.figure(figsize = (20, 20))
sns.heatmap(corr, square = True, annot = True, cmap = 'Blues')
```

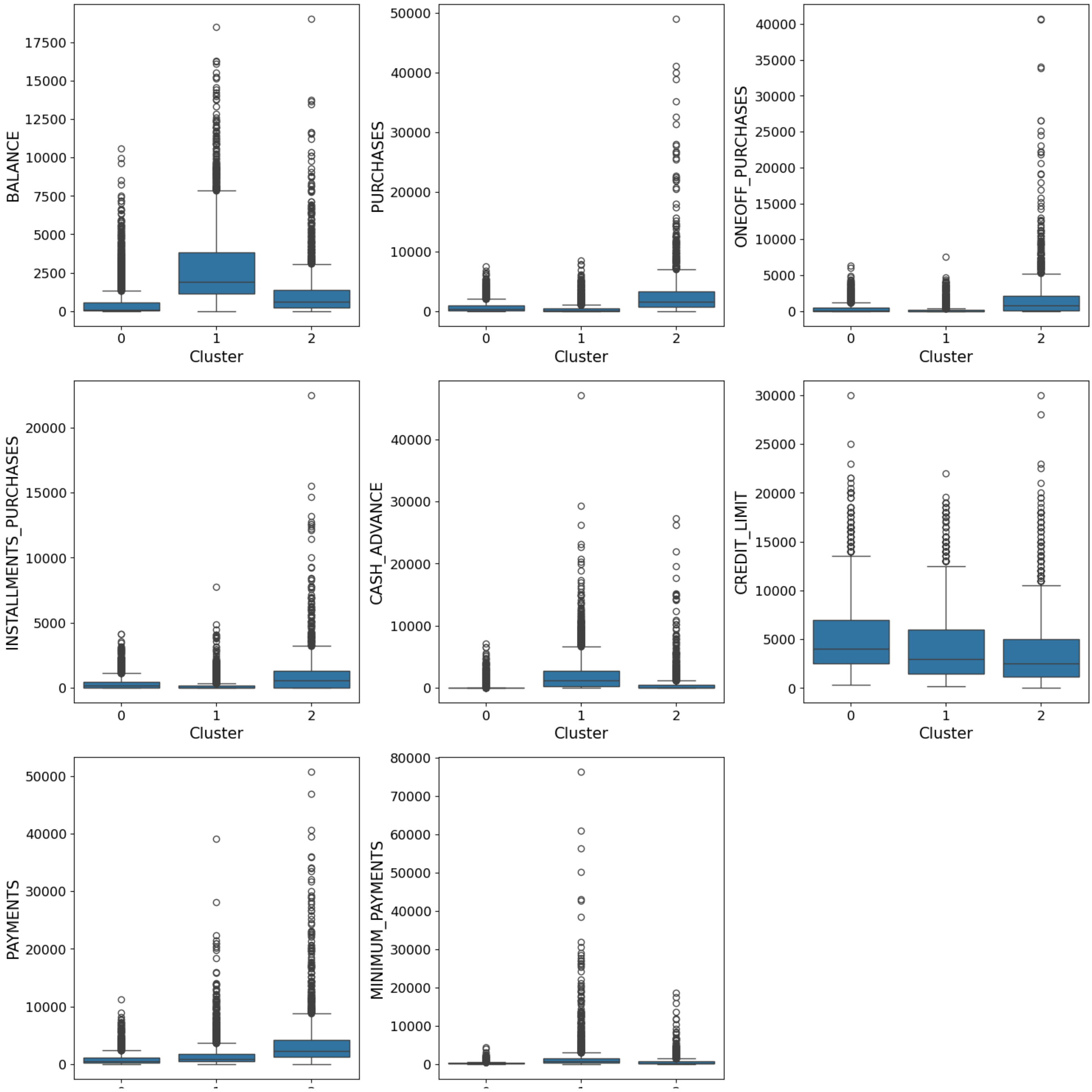
```
<ipython-input-20-9d33e8e2f301>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  corr = dfCCOrig.corr()
<Axes: >
```




```
cols_imp = list(corr[(corr['cluster'] <=-0.15) | (corr['cluster'] >=0.15)].index)
cols_imp
```

```
['BALANCE',
 'BALANCE_FREQUENCY',
 'PURCHASES',
 'ONEOFF_PURCHASES',
 'INSTALLMENTS_PURCHASES',
 'CASH_ADVANCE',
 'ONEOFF_PURCHASES_FREQUENCY',
 'CASH_ADVANCE_FREQUENCY',
 'CASH_ADVANCE_TRX',
 'PURCHASES_TRX',
```

```
      'CREDIT_LIMIT',
      'PAYMENTS',
      'cluster']


cols_imp=['BALANCE',
 'PURCHASES',
 'ONEOFF_PURCHASES',
 'INSTALLMENTS_PURCHASES',
 'CASH_ADVANCE',
 'CREDIT_LIMIT',
 'PAYMENTS',
'MINIMUM_PAYMENTS',
'TENURE'
 'cluster']




plt.figure(figsize = (15, 20))
for i, col in enumerate(cols_imp[:-1]):
  if i+1 < 16:
    ax = plt.subplot(4, 3, i+1)
    sns.boxplot(x = dfCCOrig['cluster'], y = dfCCOrig[col])
    plt.xlabel("Cluster", fontsize = 15)
    plt.ylabel(col, fontsize = 15)
    plt.xticks(fontsize = 13)
    plt.yticks(fontsize = 13)


plt.tight_layout()
plt.show()
```




```
dfCluster0=dfCCOrig[dfCCOrig['cluster']==0]
dfCluster1=dfCCOrig[dfCCOrig['cluster']==1]
dfCluster2=dfCCOrig[dfCCOrig['cluster']==2]
```

```
dfCluster0.shape
```

```
(3530, 19)
```

```
dfCluster1.shape
```

```
(3495, 19)
```

```
dfCluster2.shape
```

```
(1925, 19)
```

```
dfCluster0S=dfCluster0[['BALANCE','PURCHASES', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'PAYMENTS', 'MINIMUM_PAYMENTS','CREDIT_LIM
df0=dfCluster0S.describe().loc[['mean' ]].round(2)
df0['MINIMUM_PAYMENT_TO_CC_LIMIT(%)']=((df0['MINIMUM_PAYMENTS']/df0['CREDIT_LIMIT'])*100).round(2)
df0['BALANCE_TO_CC_LIMIT(%)']=((df0['BALANCE']/df0['CREDIT_LIMIT'])*100).round(2)
df0['CASH_ADVANCE_TO_CC_LIMIT(%)']=((df0['CASH_ADVANCE']/df0['CREDIT_LIMIT'])*100).round(2)
df0['TOTAL_SPENDING']=(df0['PURCHASES']+df0['CASH_ADVANCE']).round(2)
df0['SPEND_To_CC_LIMIT(%)']=((df0['TOTAL_SPENDING']/df0['CREDIT_LIMIT'])*100).round(2)
df0.insert(0,'Cluster_Name', 'Cluster 0')
```

```
dfCluster1S=dfCluster1[['BALANCE','PURCHASES', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'PAYMENTS', 'MINIMUM_PAYMENTS','CREDIT_LIMI
df1=dfCluster1S.describe().loc[['mean']].round(2)
df1['MINIMUM_PAYMENT_TO_CC_LIMIT(%)']=((df1['MINIMUM_PAYMENTS']/df1['CREDIT_LIMIT'])*100).round(2)
df1['BALANCE_TO_CC_LIMIT(%)']=((df1['BALANCE']/df1['CREDIT_LIMIT'])*100).round(2)
df1['TOTAL_SPENDING']=(df1['PURCHASES']+df1['CASH_ADVANCE']).round(2)
df1['CASH_ADVANCE_TO_CC_LIMIT(%)']=((df1['CASH_ADVANCE']/df1['CREDIT_LIMIT'])*100).round(2)
df1['SPEND_To_CC_LIMIT(%)']=((df1['TOTAL_SPENDING']/df1['CREDIT_LIMIT'])*100).round(2)
df1.insert(0,'Cluster_Name', 'Cluster 1')
```