Credit card Dataset for clustering

In this notebook, I am trying Log Transformation + Kernel PCA as preprocessing for all clustering algorithms and here I have chosen number of commponents = 10 in Kernel PCA so the results of clustering was different from the approach of not selecting the compoments

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('CC GENERAL.csv')
```

EDA

```
df1=df
df1.head(10)
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PU
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	

```
df1.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
```

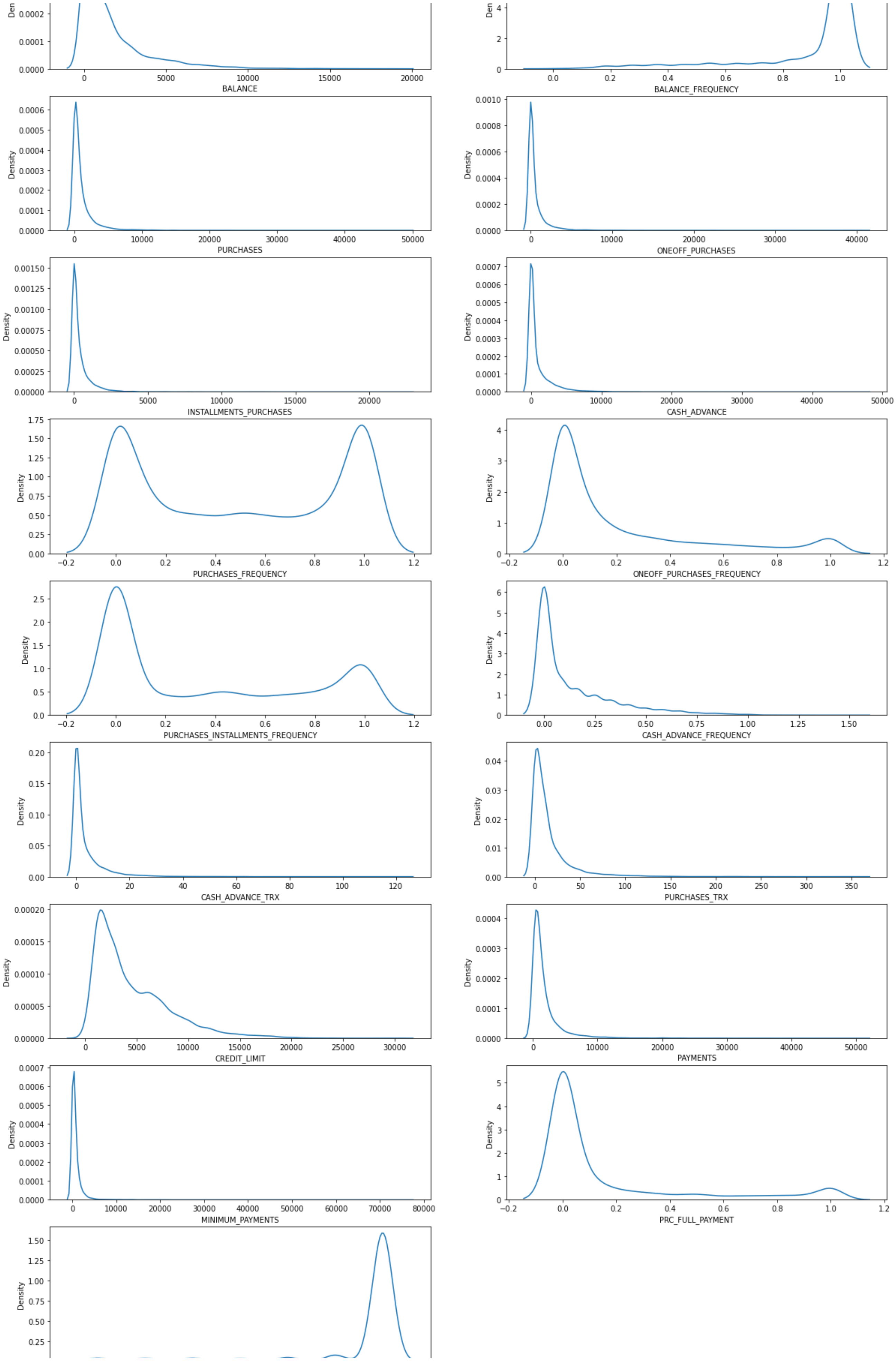
```
#Drop null values and drop cust_id column
df1.dropna(inplace=True)
df1.drop('CUST_ID', axis=1, inplace=True)
```

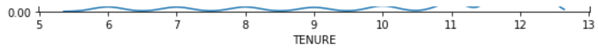
```
#Check number of null values
df1.isna().sum()
```

```
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
```

Now we'll see how each column in the dataframe is distributed.

```
plt.figure(figsize=(20,35))
for i, col in enumerate(df1.columns):
  if df1[col].dtype!='object':
    ax = plt.subplot(9, 2, i+1)
    sns.kdeplot(df1[col], ax=ax)
    plt.xlabel(col)
```

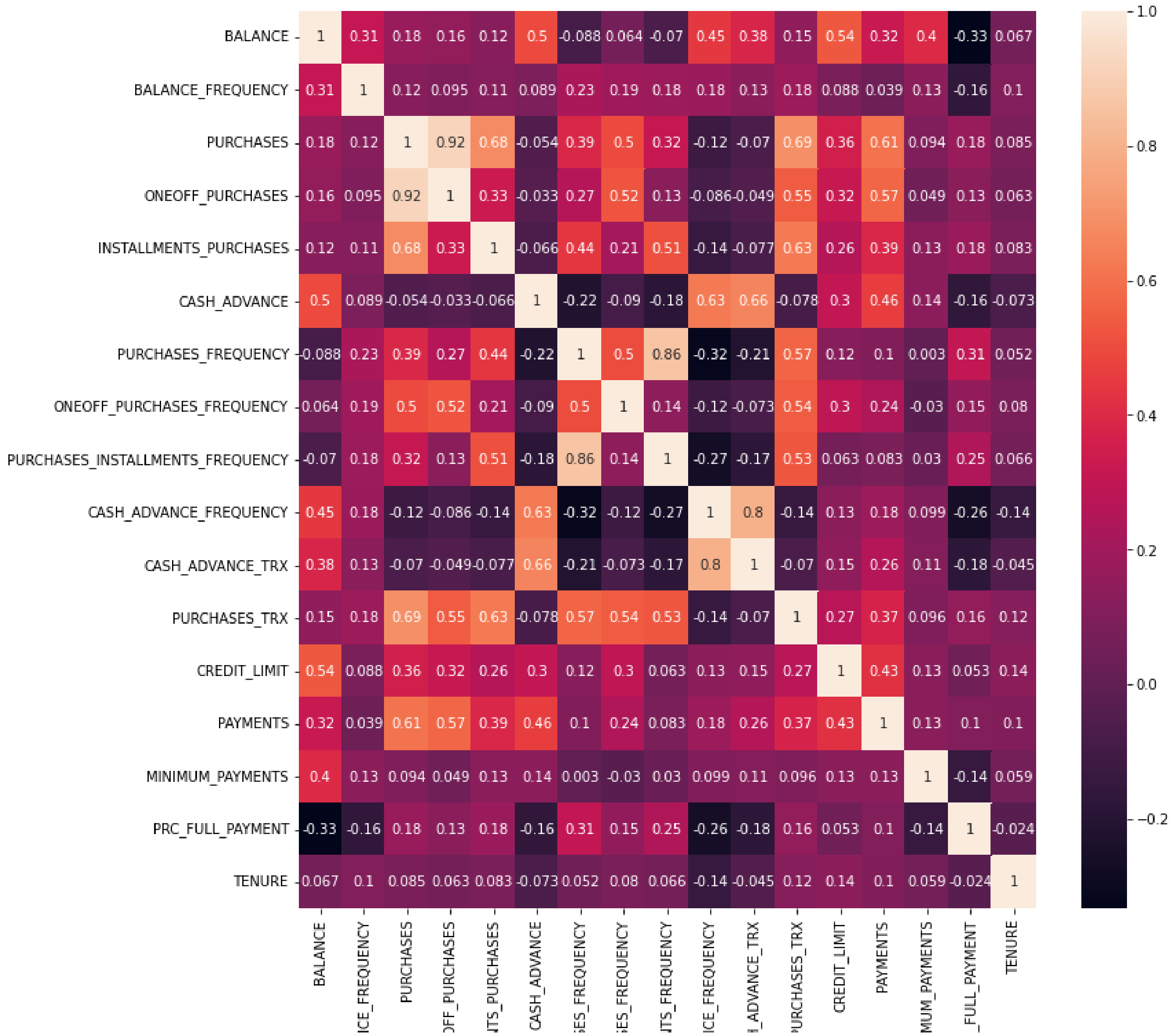

BALANCE
BALANCE_FREQUENCY
PURCHASES
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS
PRC_FULL_PAYMENT
Density




```
df1.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PUR
count	8636.000000	8636.000000	8636.000000	8636.000000	8636.
mean	1601.224893	0.895035	1025.433874	604.901438	420.
std	2095.571300	0.207697	2167.107984	1684.307803	917.
min	0.000000	0.000000	0.000000	0.000000	0.
25%	148.095189	0.909091	43.367500	0.000000	0.
50%	916.855459	1.000000	375.405000	44.995000	94.
75%	2105.195853	1.000000	1145.980000	599.100000	484.
max	19043.138560	1.000000	49039.570000	40761.250000	22500.

```
plt.figure(figsize=(12,12))
sns.heatmap(df1.corr(), annot=True)
plt.show()
```



Log Transformation to deal with skewness in dataset

```
for col in df1.columns:
    df1[col] = np.log(1 + df1[col])
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(df1)
```

KERNEL PCA:

PCA is a linear method. That is it can only be applied to datasets which are linearly separable. It does an excellent job for datasets, which are linearly separable. But, if we use it to non-linear datasets, we might get a result which may not be the optimal dimensionality reduction. Kernel PCA uses a kernel function to project dataset into a higher dimensional feature space, where it is linearly separable. It is similar to the idea of Support Vector Machines.

```
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components = 10, kernel = 'rbf')
Res_kpca = kpca.fit_transform(df1)
df_PCA = pd.DataFrame(Res_kpca)
df_PCA.head(10)
```

	0	1	2	3	4	5	6	7	
0	-0.316655	0.518660	0.085757	0.035720	-0.015652	-0.012605	-0.353939	0.060512	0.
1	0.569986	0.071253	0.111611	0.004157	-0.027734	0.586436	-0.003635	0.008141	0.
2	-0.102019	-0.177922	-0.303104	0.504025	-0.085828	0.016659	0.026615	0.083910	-0.
3	-0.044340	-0.042467	-0.205327	0.217165	-0.012058	-0.072369	0.016535	0.046219	0.
4	-0.186224	0.187854	0.009408	-0.041324	0.023448	0.000349	0.579882	0.073158	-0.
5	-0.181012	-0.397703	0.272450	-0.102479	0.057468	-0.007480	-0.114004	0.428021	-0.
6	-0.268727	0.331246	0.071467	-0.028587	0.016550	-0.001523	0.640642	0.048052	-0.
7	-0.270377	-0.460093	0.328936	0.082712	-0.018538	0.004373	0.030306	-0.187372	0.
8	-0.104386	-0.175720	-0.327202	0.577536	-0.136997	0.021489	-0.010554	0.035044	-0.
9	-0.215729	0.240570	0.031458	-0.039280	0.021769	0.000437	0.616565	0.066846	-0.

Use TSNE Algorithm for embedding (Moving from 4-dim space to 2-dim space)

```
from sklearn.manifold import TSNE

tsne_projection = TSNE(n_components=2,
                       perplexity=50,
                       n_iter=10**4,
                       early_exaggeration = 12,
                       init='random',
                       random_state=42).fit_transform(df_PCA)
tsne_projection = pd.DataFrame(tsne_projection)
tsne_projection
```

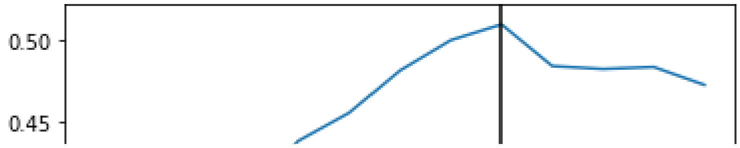
```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: Th
  FutureWarning,
```

	0	1
0	-59.407619	66.873520
1	10.531981	-74.045578
2	-25.901361	21.385733
3	-22.532774	1.789568
4	2.054819	82.750305
...	...	...
8631	-8.750680	-22.030550
8634	-43.100754	-27.655609

KMeansClustering Algorithm

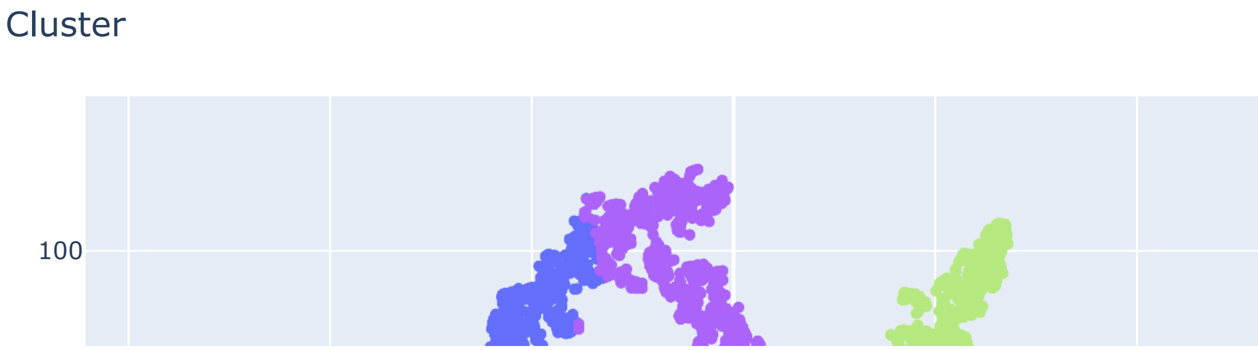
```
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

import numpy as np
range_n_cluster = list(range(2,15))
silhoutte_score = []
best_cluster_model = None

for n_clusters in range_n_cluster:
    cluster_model  = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = cluster_model.fit(df_PCA).labels_

    silhouette_avg = silhouette_score(df_PCA, cluster_labels)
    silhoutte_score += [silhouette_avg]

    if silhouette_avg >= np.max(silhoutte_score):
        best_cluster_model = cluster_model


plt.plot(range_n_cluster, silhoutte_score)
plt.axvline(best_cluster_model.n_clusters, color='black')
```

```
<matplotlib.lines.Line2D at 0x7f5e6d638750>
```

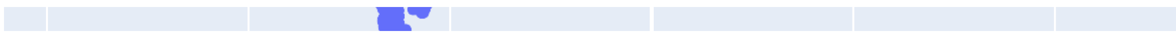



```
cluster_labels = ["cluster" + str(label)
                  for label in best_cluster_model.labels_]
```

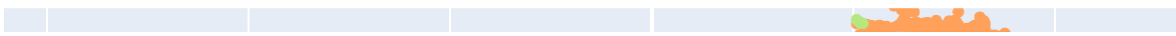
```
import plotly.express as px
fig = px.scatter(x=tsne_projection[0],
                 y=tsne_projection[1],
                 text=tsne_projection.index,
                 color=cluster_labels)
fig.update_traces(textposition='top center')
fig.update_layout(height=800, width=800, title_text='Cluster')
fig.show()
```



AgglomerativeClustering



Choose number of clusters that give the best silhouette_score



```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
import numpy as np
range_n_cluster = list(range(2,15))
silhoutte_score = []
best_cluster_model = None

for n_clusters in range_n_cluster:
    cluster_model  = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward')
    cluster_labels = cluster_model.fit_predict(df_PCA)

    silhouette_avg = silhouette_score(df_PCA, cluster_labels)
    silhoutte_score += [silhouette_avg]

    if silhouette_avg >= np.max(silhoutte_score):
        best_cluster_model = cluster_model
```

```
plt.plot(range_n_cluster, silhoutte_score)
plt.axvline(best_cluster_model.n_clusters, color='black')
```

```
<matplotlib.lines.Line2D at 0x7f5e69a83e10>
```

```
cluster_labels = ["cluster" + str(label)
                  for label in best_cluster_model.labels_]
```

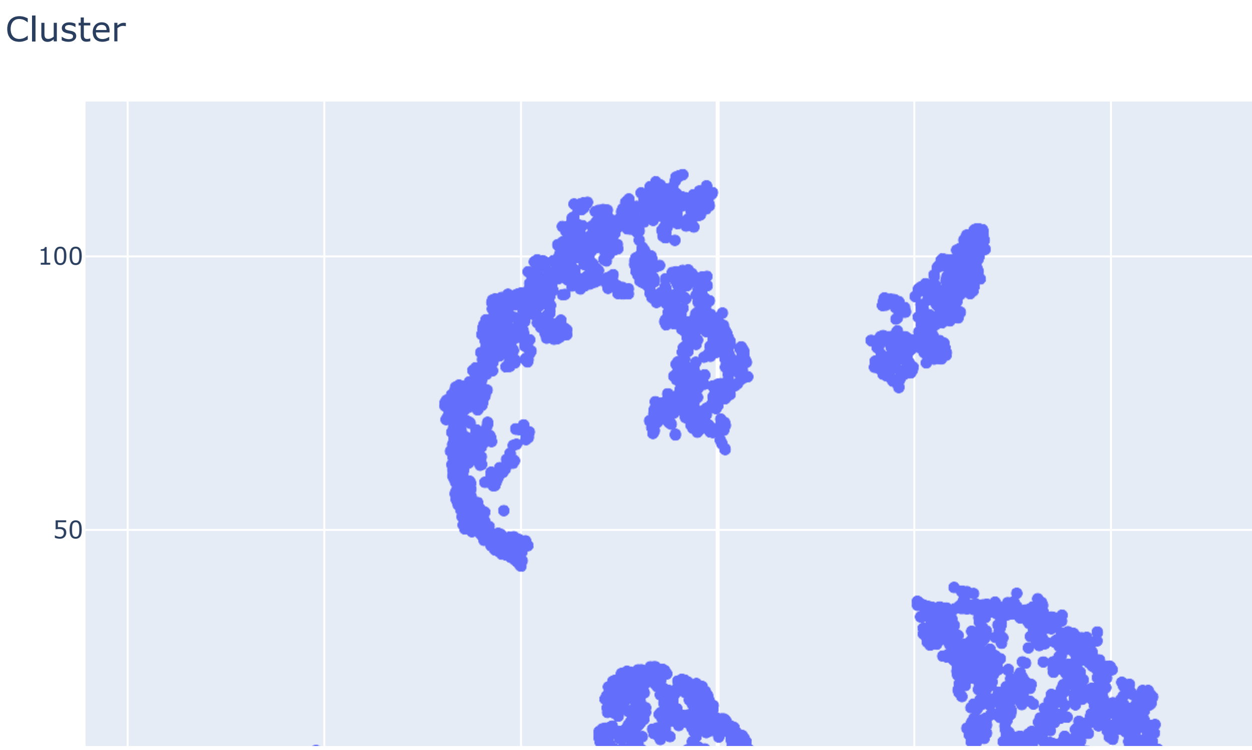
```
import plotly.express as px
fig = px.scatter(x=tsne_projection[0],
                 y=tsne_projection[1],
                 text=tsne_projection.index,
                 color=cluster_labels)
fig.update_traces(textposition='top center')
fig.update_layout(height=800, width=800, title_text='Cluster')
fig.show()
```

DBSCAN

```
from sklearn.cluster import DBSCAN
db_default = DBSCAN(eps = 1.2, min_samples = 5).fit(df_PCA)
labels = db_default.labels_
```

```
cluster_labels = ["cluster (" + str(label) + ")"
                  for label in labels]
```

```
import plotly.express as px
fig = px.scatter(x=tsne_projection[0],
                 y=tsne_projection[1],
                 text=tsne_projection.index,
                 color=cluster_labels)
fig.update_traces(textposition='top center')
fig.update_layout(height=800, width=800, title_text='Cluster')
fig.show()
```

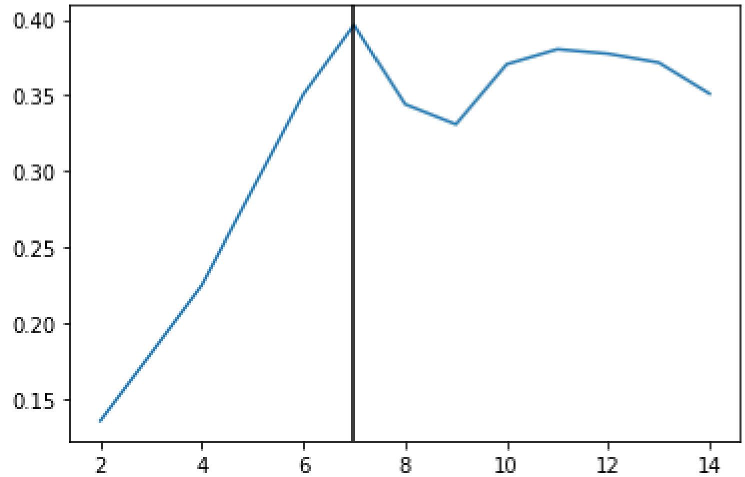


Expectation-Maximization (EM) Algorithm

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaled_data = scaler.fit_transform(df_PCA)
xs = pd.DataFrame(scaled_data, columns = df_PCA.columns)
```

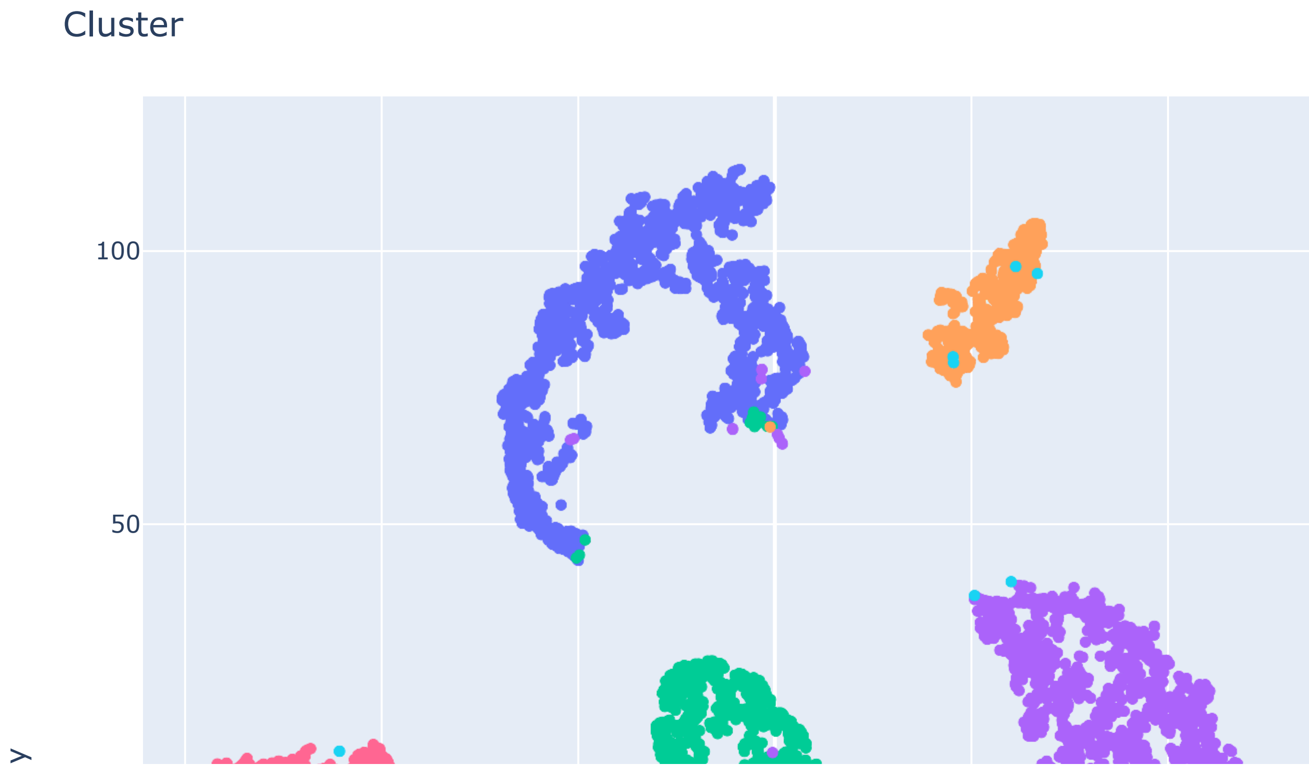
```
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
import numpy as np
range_n_cluster = list(range(2,15))
silhoutte_score = []
best_cluster_model = None

for n_clusters in range_n_cluster:
    cluster_model  = GaussianMixture(n_components = n_clusters)
    cluster_labels = cluster_model.fit_predict(xs)

    silhouette_avg = silhouette_score(xs, cluster_labels)
    silhoutte_score += [silhouette_avg]

    if silhouette_avg >= np.max(silhoutte_score):
        best_cluster_model = cluster_model
        labels = cluster_labels


plt.plot(range_n_cluster, silhoutte_score)
plt.axvline(best_cluster_model.n_components, color='black')
```

```
<matplotlib.lines.Line2D at 0x7f5e697d9a50>
```



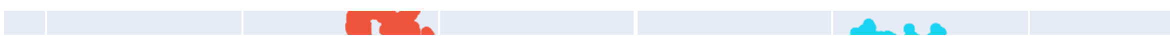
```
cluster_labels = ["cluster (" + str(label) + ")"
                  for label in labels]
```

```
import plotly.express as px
fig = px.scatter(x=tsne_projection[0],
                 y=tsne_projection[1],
                 text=tsne_projection.index,
                 color=cluster_labels)
fig.update_traces(textposition='top center')
fig.update_layout(height=800, width=800, title_text='Cluster')
fig.show()
```

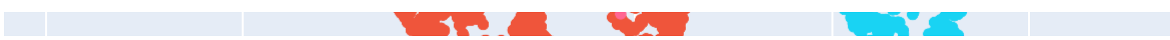


Isolated Random Forest

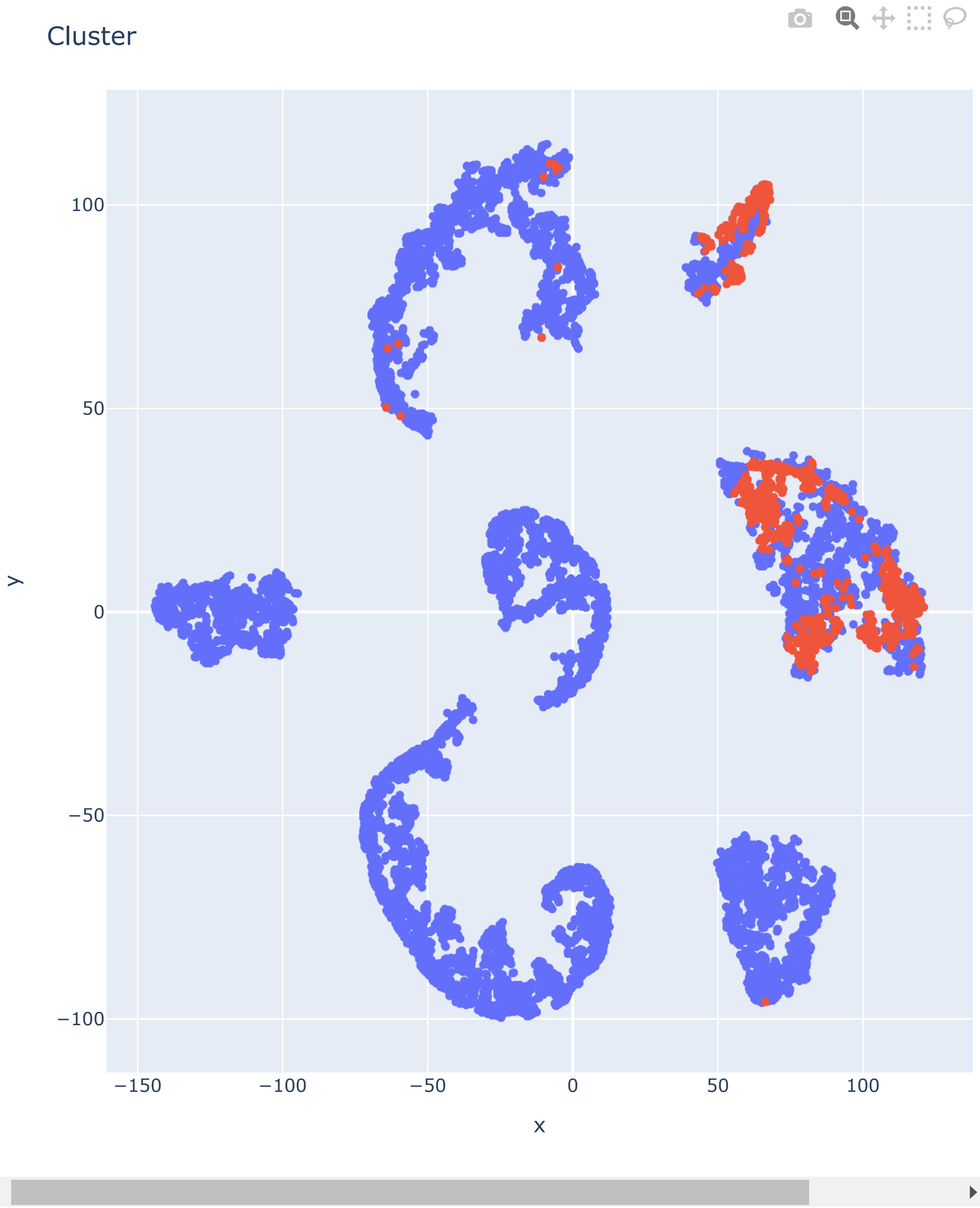
```
from sklearn.ensemble import IsolationForest
model = IsolationForest(n_estimators=100, contamination=0.1, max_features=0.7)
labels = model.fit_predict(df_PCA)
```



```
cluster_labels = ["cluster (" + str(label) + ")"
                  for label in labels]
```



```
import plotly.express as px
fig = px.scatter(x=tsne_projection[0],
                 y=tsne_projection[1],
                 text=tsne_projection.index,
                 color=cluster_labels)
fig.update_traces(textposition='top center')
fig.update_layout(height=800, width=800, title_text='Cluster')
fig.show()
```


Cluster
100
50
0
−50
−100
y
−150
−100
−50
0
50
100
x