Machine Learning – Assignment 6

Q1. (Provide only mathematical solutions for this question) Six points with the following attributes are given, calculate and find out clustering representations and dendrogram using Single, complete, and average link proximity function in hierarchical clustering technique.

```
#Q1
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.cluster.hierarchy as shc
from scipy.spatial.distance import squareform, pdist

a = np.array([0.4005,0.2148,0.3457,0.2652,0.0789,0.4548])
b = np.array([0.5306,0.3854,0.3156,0.1875,0.4139,0.3022])

point = ['P1','P2','P3','P4','P5','P6']
data = pd.DataFrame({'Point':point, 'x cordinate':a, 'y cordinate':b})
data = data.set_index('Point')
data
```

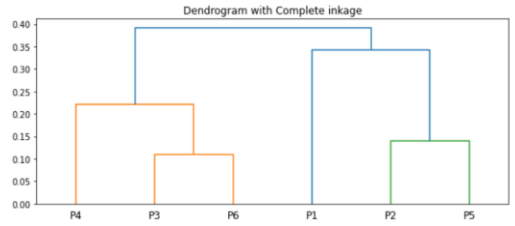
Point	x cordinate	y cordinate
P1	0.4005	0.5306
P2	0.2148	0.3854
P3	0.3457	0.3156
P4	0.2652	0.1875
P5	0.0789	0.4139
P6	0.4548	0.3022

```
[55] dist = pd.DataFrame(squareform(np.round(pdist(data[['x cordinate', 'y cordinate']]),4), 'euclidean'), columns=data.index.values, index=data.index.values)
     dist
```

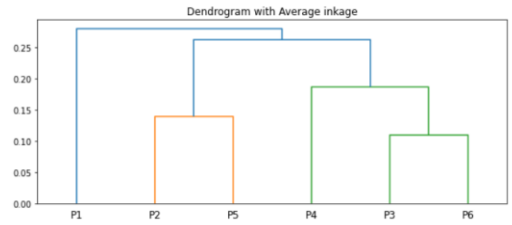
	P1	P2	P3	P4	P5	P6
P1	0.0000	0.2357	0.2219	0.3688	0.3421	0.2348
P2	0.2357	0.0000	0.1483	0.2042	0.1389	0.2540
P3	0.2219	0.1483	0.0000	0.1513	0.2843	0.1099
P4	0.3688	0.2042	0.1513	0.0000	0.2932	0.2216
P5	0.3421	0.1389	0.2843	0.2932	0.0000	0.3921
P6	0.2348	0.2540	0.1099	0.2216	0.3921	0.0000

```
plt.figure(figsize=(10,4))
plt.title("Dendrogram with Single inkage")
dend = shc.dendrogram(shc.linkage(data[['x cordinate', 'y cordinate']], method='single'), labels=data.index)
```

```
plt.figure(figsize=(10,4))
plt.title("Dendrogram with Complete inkage")
dend = shc.dendrogram(shc.linkage(data[['x cordinate', 'y cordinate']], method='complete'), labels=data.index)
```




```
[58] plt.figure(figsize=(10,4))
     plt.title("Dendrogram with Average inkage")
     dend = shc.dendrogram(shc.linkage(data[['x cordinate', 'y cordinate']], method='average'), labels=data.index)
```



Q2: Use CC_GENERAL.csv given in the folder and apply: a) Preprocess the data by removing the categorical column and filling the missing values. b) Apply StandardScaler() and normalize() functions to scale and normalize raw input data. c) Use PCA with K=2 to reduce the input dimensions to two features. d) Apply Agglomerative Clustering with k=2,3,4 and 5 on reduced features and visualize result for each k value using scatter plot. e) Evaluate different variations using Silhouette Scores and Visualize results with a bar chart.

```
[59] ## Q2
     #importing all libraries here for assignment
     import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn import preprocessing,metrics
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import LabelEncoder, StandardScaler
     from sklearn.decomposition import PCA
     from sklearn.cluster import AgglomerativeClustering
     from sklearn.metrics import silhouette_score

     import warnings
     warnings.filterwarnings("ignore")
```

```
[60] dataframe = pd.read_csv('/content/gdrive/MyDrive/Dataset/CC GENERAL.csv')
     dataframe.info()
```

```
dataframe = pd.read_csv('/content/gdrive/MyDrive/Dataset/CC GENERAL.csv')
dataframe.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
```

```
[61] dataframe.head()
```

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

```
[62] dataframe.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FRE
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0.490351	0.202458	0.
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371	0.298336	0.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	0.083333	0.000000	0.
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000	0.083333	0.
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667	0.300000	0.
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000	1.000000	1.

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

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	0.000000	0.083333
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	0.000000	0.000000
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	1.000000	0.000000
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	0.083333	0.000000
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	0.083333	0.000000

```
[64] df.isnull().any()
```

```
BALANCE                           False
BALANCE_FREQUENCY                 False
PURCHASES                         False
ONEOFF_PURCHASES                  False
INSTALLMENTS_PURCHASES            False
CASH_ADVANCE                      False
PURCHASES_FREQUENCY               False
ONEOFF_PURCHASES_FREQUENCY        False
PURCHASES_INSTALLMENTS_FREQUENCY  False
CASH_ADVANCE_FREQUENCY            False
CASH_ADVANCE_TRX                  False
PURCHASES_TRX                     False
CREDIT_LIMIT                       True
PAYMENTS                          False
MINIMUM_PAYMENTS                   True
PRC_FULL_PAYMENT                  False
TENURE                            False
dtype: bool
```

```
df.fillna(dataframe.mean(), inplace=True)
df.isnull().any()
```

```
BALANCE                           False
BALANCE_FREQUENCY                 False
PURCHASES                         False
ONEOFF_PURCHASES                  False
INSTALLMENTS_PURCHASES            False
CASH_ADVANCE                      False
PURCHASES_FREQUENCY               False
ONEOFF_PURCHASES_FREQUENCY        False
PURCHASES_INSTALLMENTS_FREQUENCY  False
CASH_ADVANCE_FREQUENCY            False
CASH_ADVANCE_TRX                  False
PURCHASES_TRX                     False
CREDIT_LIMIT                      False
PAYMENTS                          False
MINIMUM_PAYMENTS                  False
PRC_FULL_PAYMENT                  False
TENURE                            False
dtype: bool
```

```
[66] df.corr().style.background_gradient(cmap="Greens")
```

EOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
0.073166	-0.063186	0.449218	0.385152	0.154338	0.531267	0.322802	0.394282	-0.318959	0.072692
0.202415	0.176079	0.191873	0.141555	0.189626	0.095795	0.065008	0.114249	-0.095082	0.119776
0.498430	0.315567	-0.120143	-0.067175	0.689561	0.356959	0.603264	0.093515	0.180379	0.086288
0.524891	0.127729	-0.082628	-0.046212	0.545523	0.319721	0.567292	0.048597	0.132763	0.064150
0.214042	0.511351	-0.132318	-0.073999	0.628108	0.256496	0.384084	0.131687	0.182569	0.086143
-0.086754	-0.177070	0.628522	0.656498	-0.075850	0.303983	0.453238	0.139223	-0.152935	-0.068312
0.501343	0.862934	-0.308478	-0.203478	0.568430	0.119778	0.103464	0.002976	0.305802	0.061506
1.000000	0.142329	-0.111716	-0.069088	0.544869	0.295030	0.243537	-0.029963	0.157531	0.082466
0.142329	1.000000	-0.262958	-0.169207	0.529975	0.060752	0.085551	0.029590	0.250087	0.073275
-0.111716	-0.262958	1.000000	0.799561	-0.131168	0.132616	0.183192	0.097898	-0.249773	-0.133372
-0.069088	-0.169207	0.799561	1.000000	-0.066157	0.149699	0.255278	0.109185	-0.169784	-0.043421
0.544869	0.529975	-0.131168	-0.066157	1.000000	0.272877	0.370832	0.095858	0.162066	0.121874
0.295030	0.060752	0.132616	0.149699	0.272877	1.000000	0.421852	0.125134	0.055671	0.139034
0.243537	0.085551	0.183192	0.255278	0.370832	0.421852	1.000000	0.125046	0.112138	0.106136
-0.029963	0.029590	0.097898	0.109185	0.095858	0.125134	0.125046	1.000000	-0.139674	0.057257
0.157531	0.250087	-0.249773	-0.169784	0.162066	0.055671	0.112138	-0.139674	1.000000	-0.016486
0.082466	0.073275	-0.133372	-0.043421	0.121874	0.139034	0.106136	0.057257	-0.016486	1.000000

```
[67] x = df.iloc[:,0:-1]
     y = df.iloc[:,-1]


     scaler = preprocessing.StandardScaler()
     scaler.fit(x)
     X_scaled_array = scaler.transform(x)
     X_scaled_df = pd.DataFrame(X_scaled_array, columns = x.columns)
```

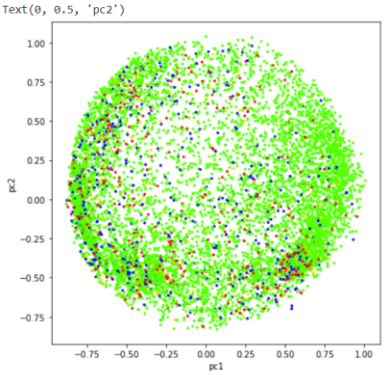
```
#Normalization is the process of scaling individual samples to have unit norm.
#This process can be useful if you plan to use a quadratic form such as the dot-product or any other kernel to quantify the similarity of any pair of samples.
X_normalized = preprocessing.normalize(X_scaled_df)
# Converting the numpy array into a pandas DataFrame
X_normalized = pd.DataFrame(X_normalized)
```

```
[69] pca2 = PCA(n_components=2)
     principalComponents = pca2.fit_transform(X_normalized)

     principalDf = pd.DataFrame(data = principalComponents, columns = ['P1', 'P2'])

     finalDf = pd.concat([principalDf, df[['TENURE']]], axis = 1)
     finalDf.head()
```

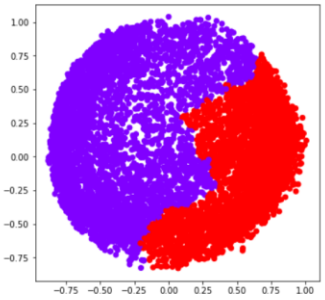
	P1	P2	TENURE
0	-0.488186	-0.677233	12
1	-0.517294	0.556075	12
2	0.334384	0.287313	12
3	-0.486616	-0.080780	12
4	-0.562175	-0.474770	12

```
[70] plt.figure(figsize=(7,7))
     plt.scatter(finalDf['P1'],finalDf['P2'],c=finalDf['TENURE'],cmap='prism', s =5)
     plt.xlabel('pc1')
     plt.ylabel('pc2')
```

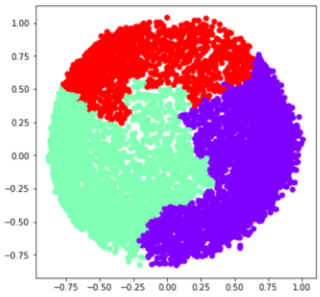
```
Text(0, 0.5, 'pc2')
```



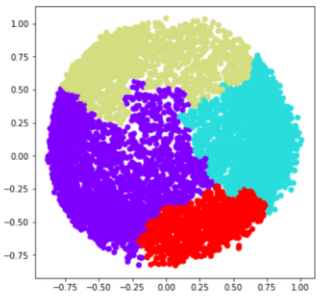
```
[71] ac2 = AgglomerativeClustering(n_clusters = 2)

     # Visualizing the clustering
     plt.figure(figsize =(6, 6))
     plt.scatter(principalDf['P1'], principalDf['P2'],
                 c = ac2.fit_predict(principalDf), cmap ='rainbow')
     plt.show()
```



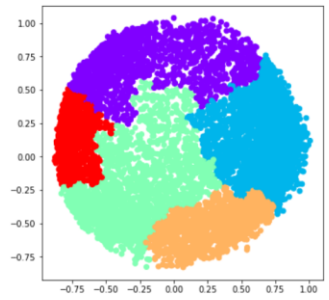

```
[72] ac3 = AgglomerativeClustering(n_clusters = 3)

     # Visualizing the clustering
     plt.figure(figsize =(6, 6))
     plt.scatter(principalDf['P1'], principalDf['P2'],
                 c = ac3.fit_predict(principalDf), cmap ='rainbow')
     plt.show()
```




```
[73] ac4 = AgglomerativeClustering(n_clusters = 4)

     # Visualizing the clustering
     plt.figure(figsize =(6, 6))
     plt.scatter(principalDf['P1'], principalDf['P2'],
                 c = ac4.fit_predict(principalDf), cmap ='rainbow')
     plt.show()
```



[74]
```
ac5 = AgglomerativeClustering(n_clusters = 5)

# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(principalDf['P1'], principalDf['P2'],
            c = ac5.fit_predict(principalDf), cmap ='rainbow')
plt.show()
```



[75]
```
k = [2, 3, 4, 5]

# Appending the silhouette scores of the different models to the list
silhouette_scores = []
silhouette_scores.append(
        silhouette_score(principalDf, ac2.fit_predict(principalDf)))
silhouette_scores.append(
        silhouette_score(principalDf, ac3.fit_predict(principalDf)))
silhouette_scores.append(
        silhouette_score(principalDf, ac4.fit_predict(principalDf)))
silhouette_scores.append(
        silhouette_score(principalDf, ac5.fit_predict(principalDf)))


# Plotting a bar graph to compare the results
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 20)
plt.ylabel('S(i)', fontsize = 20)
plt.show()
```