Fall 2022 5710 Machine Learning: Assignment 6

Name: Venkata Lakshmi Sasank Tipparaju

Student ID: #700738838

CS5710 – 13469

In Class Programming Activity GitHub Link:
https://github.com/Sasank09/CS5710_13469/tree/main/Assignments/Assignment6

Video Link: https://vimeo.com/771362329/d2d7046883

Question 1: Given points with cords and Distance Matrix for six points

point	x coordinate	y coordinate
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

Table : X-Y coordinates of six points.

	p1	p2	p3	p4	p5	p6
p1	0.0000	0.2357	0.2218	0.3688	0.3421	0.2347
p2	0.2357	0.0000	0.1483	0.2042	0.1388	0.2540
p3	0.2218	0.1483	0.0000	0.1513	0.2843	0.1100
p4	0.3688	0.2042	0.1513	0.0000	0.2932	0.2216
p5	0.3421	0.1388	0.2843	0.2932	0.0000	0.3921
p6	0.2347	0.2540	0.1100	0.2216	0.3921	0.0000

Table : Distance Matrix for Six Points

In **Single Linkage,** the distance between two clusters is the minimum distance between members of the two clusters

We see the points P3, P6 has the least distance "0.1100". So, we will first merge those into a cluster

Re-compute the distance matrix after forming a cluster

Update the distance between the cluster (P3, P6) to P1= Min (dist(P3, P6), P1)) -> Min(dist(P3, P1),dist(P6,P1))

	P1	P2	P3, P6	P4	P5
P1	0				
P2	0.2357	0			
P3, P6	0.2218	0.1483	0		
P4	0.3688	0.2042	0.1513	0	
P5	0.3421	0.1388	0.2843	0.2932	0

Merging – P2 and P5 – (P2, P5)

	P1	P2, P5	P3, P6	P4
P1	0			
P2, P5	0.2357	0		
P3, P6	0.2218	0.1483	0	
P4	0.3688	0.2042	0.1513	0

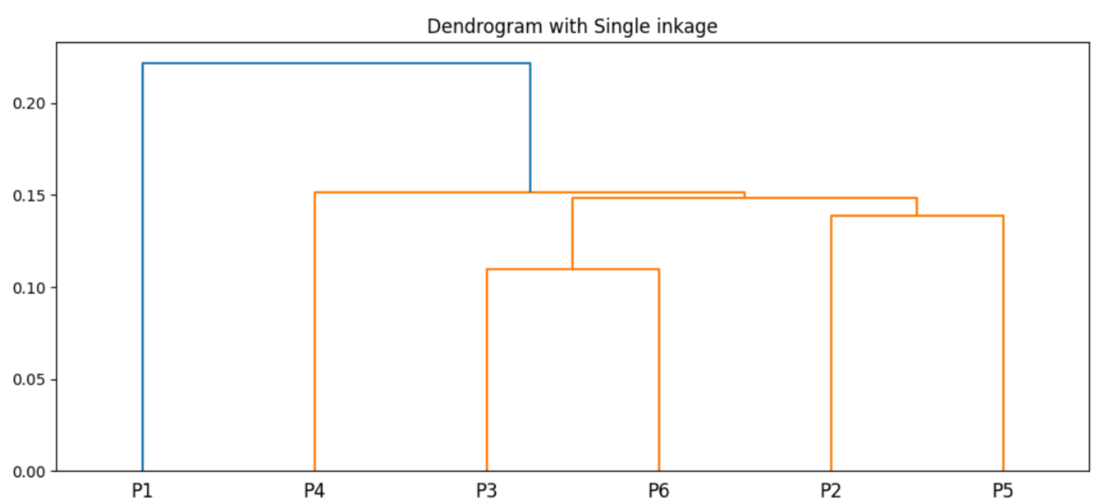
Merging – (P2, P5) and (P3, P6) – (P2, P5, P3, P6)

	P1	P2, P5, P3, P6	P4
P1	0		
P2, P5, P3, P6	0.2218	0	
P4	0.3688	0.1513	0

Merging – P1 and (P2, P5, P3, P6, P4)

	P1	P2, P5, P3, P6, P4
P1	0	
P2, P5, P3, P6, P4	0.2218	0

Dendrogram:



In Complete Linkage, the distance between two clusters is the maximum distance between members of the two clusters

We see the points P3, P6 has the least distance “0.1100”. So, we will first merge those into a cluster

Re-compute the distance matrix after forming a cluster

Update the distance between the cluster (P3, P6) to P1= Max (dist(P3, P6), P1)) -> Max(dist(P3,P1),dist(P6,P1)), Applying same logic over the distance matrix will result as below.

	P1	P2	P3, P6	P4	P5
P1	0				
P2	0.2357	0			
P3, P6	0.2347	0.2540	0		
P4	0.3688	0.2042	0.2216	0	
P5	0.3421	0.1388	0.3921	0.2932	0

Merging – P2 and P5 – (P2, P5)

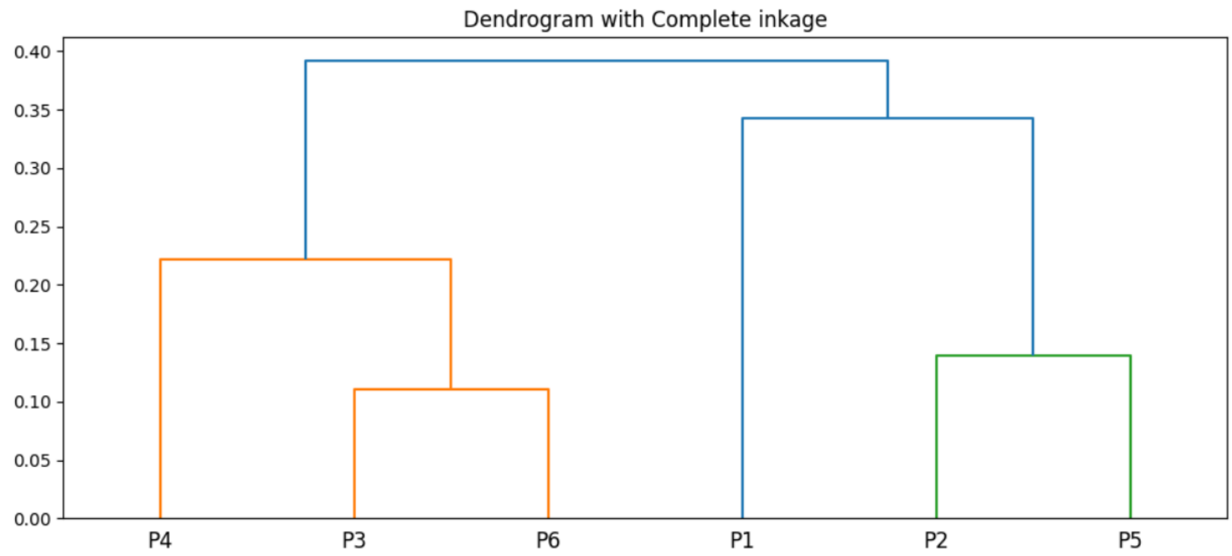
	P1	P2, P5	P3, P6	P4
P1	0			
P2, P5	0.3421	0		
P3, P6	0.2347	0.3921	0	
P4	0.3688	0.2932	0.2216	0

Merging – P4 and P3, P6 – (P4, P3, P6)

	P1	P2, P5	P4, P3, P6
P1	0		
P2, P5	0.3421	0	
P4, P3, P6	0.3688	0.3921	0

Merging P1 and (P2, P5) – (P1, P2, P5)

	P1, P2, P5	P4, P3, P6
P1, P2, P5	0	
P4, P3, P6	0.3921	0



In Average Linkage, the distance between two clusters is the average of all distances between members of the two clusters

We see the points P3, P6 has the least distance "0.1100". So, we will first merge those into a cluster

Re-compute the distance matrix after forming a cluster

Update the distance between the cluster (P3, P6) to P1= Avg (dist(P3, P6), P1)) -> Avg(dist(P3,P1),dist(P6,P1)), Applying same logic over the distance matrix will result as below.

	P1	P2	P3, P6	P4	P5
P1	0				
P2	0.2357	0			
P3, P6	0.2282	0.2011	0		
P4	0.3688	0.2042	0.1864	0	
P5	0.3421	**0.1388**	0.3382	0.2932	0

Merging – P2 and P5 – (P2, P5)

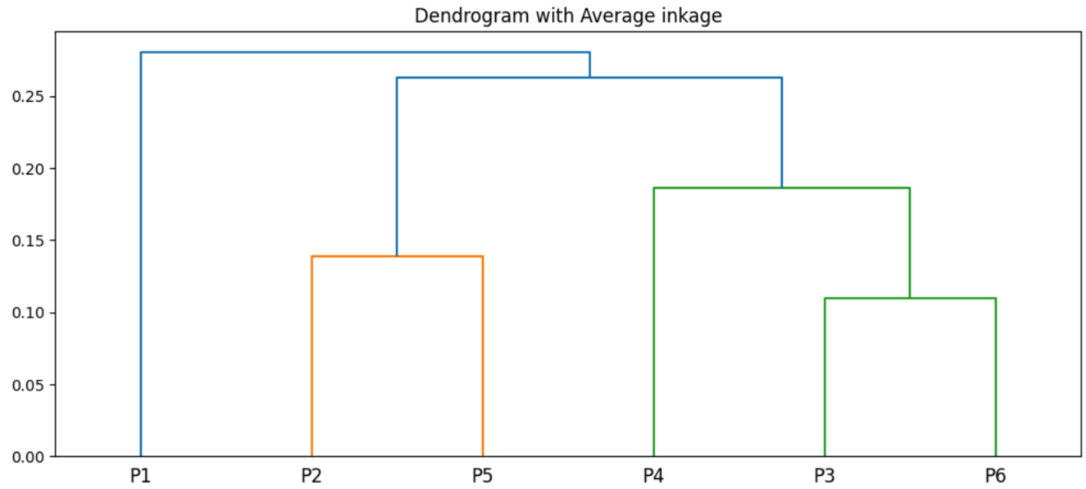
	P1	P2, P5	P3, P6	P4
P1	0			
P2, P5	0.2889	0		
P3, P6	0.2282	0.2696	0	
P4	0.3421	0.2487	**0.1864**	0

Merging – P4 and (P3, P6)

	P1	P2, P5	P4, P3, P6
P1	0		
P2, P5	0.2889	0	
P4, P3, P6	0.2851	0.2591	0

Merging – (P2, P5) and (P3, P4, P6)

	P1	P2, P5, P3, P6, P4
P1	0	
P2, P5, P3, P6, P4	0.2870	0



Question 2:

#Imported all the libraries required for Q2 – and loaded the data using pandas read_csv from CC GENERAL file

In [1]:
```
## 2) 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.
```

In [2]:
```
#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")
```

In [3]:
```
dataframe = pd.read_csv('CC GENERAL.csv')
dataframe.info()
```

Used Dataframe.info () method to check datatypes of the file. Since all columns mostly are numeric and only CUST_ID is not required, dropped the CUST_ID from data frame


```
<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
```

In [4]:
```
dataframe.head()
```

Out[4]:

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

In [5]:
```
dataframe.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONE(
nt	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	
an	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0.490351	
td	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371	
in	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	0.083333	
%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000	
%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667	
ax	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000	

In [6]:
```
df = dataframe.drop(['CUST_ID'], axis=1)
df.head()
```

Out[6]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUE
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.16
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.00
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.00
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.08
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.08

In [7]:
```
df.isnull().any()
```


Checked if Dataframe contains any null values and replaced with Mean() of that columns

Out[7]:
```
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
```

In [8]:
```
df.fillna(dataframe.mean(), inplace=True)
df.isnull().any()
```

Out[8]:
```
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
```

In [9]:
```
df.corr().style.background_gradient(cmap="Greens")
```


#Displayed the correlation and X and Y and preprocessed the data and standardized with StandardScalar() from sklearn

Out[9]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	C
BALANCE	1.000000	0.322412	0.181261	0.164350	0.126469	
BALANCE_FREQUENCY	0.322412	1.000000	0.133674	0.104323	0.124292	
PURCHASES	0.181261	0.133674	1.000000	0.916845	0.679896	
ONEOFF_PURCHASES	0.164350	0.104323	0.916845	1.000000	0.330622	
INSTALLMENTS_PURCHASES	0.126469	0.124292	0.679896	0.330622	1.000000	
CASH_ADVANCE	0.496692	0.099388	-0.051474	-0.031326	-0.064244	
PURCHASES_FREQUENCY	-0.077944	0.229715	0.393017	0.264937	0.442418	
ONEOFF_PURCHASES_FREQUENCY	0.073166	0.202415	0.498430	0.524891	0.214042	
PURCHASES_INSTALLMENTS_FREQUENCY	-0.063186	0.176079	0.315567	0.127729	0.511351	
CASH_ADVANCE_FREQUENCY	0.449218	0.191873	-0.120143	-0.082628	-0.132318	
CASH_ADVANCE_TRX	0.385152	0.141555	-0.067175	-0.046212	-0.073999	
PURCHASES_TRX	0.154338	0.189626	0.689561	0.545523	0.628108	
CREDIT_LIMIT	0.531267	0.095795	0.356959	0.319721	0.256496	
PAYMENTS	0.322802	0.065008	0.603264	0.567292	0.384084	
MINIMUM_PAYMENTS	0.394282	0.114249	0.093515	0.048597	0.131687	
PRC_FULL_PAYMENT	-0.318959	-0.095082	0.180379	0.132763	0.182569	
TENURE	0.072692	0.119776	0.086288	0.064150	0.086143	

In [10]:

```
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)
```

In [11]:

```
#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
```


After preprocessing, utilized normalize () method to perform normalization on the raw data

Applied PCA on the normalized data to reduce the features to two columns with P1 and P2 and visualized the plot using scatter

```
X_normalized = preprocessing.normalize(X_scaled_df)
# Converting the numpy array into a pandas DataFrame
X_normalized = pd.DataFrame(X_normalized)
```

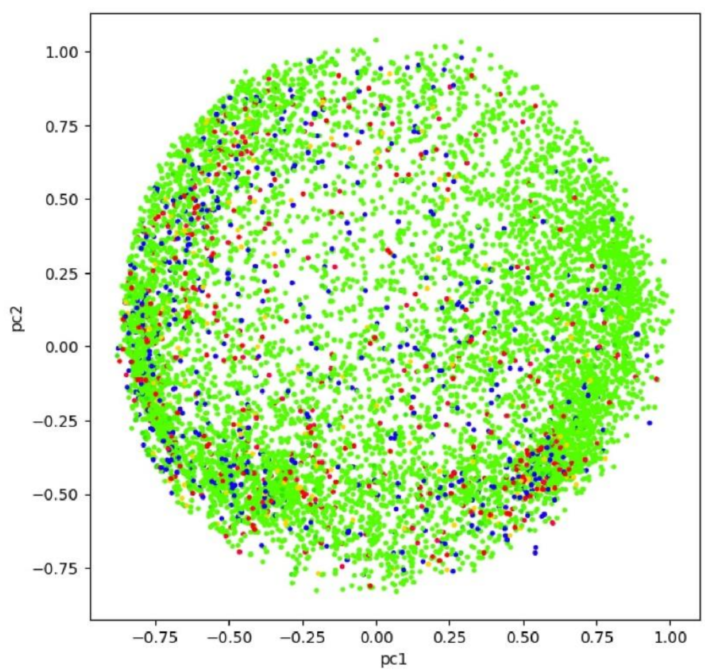
In [12]:
```
pca2 = PCA(n_components=2)
principalComponents = pca2.fit_transform(X_normalized)

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

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

Out[12]:

	P1	P2	TENURE
0	-0.488186	-0.677233	12
1	-0.517294	0.556075	12
2	0.334384	0.287312	12
3	-0.486616	-0.080781	12
4	-0.562175	-0.474770	12

In [13]:
```
plt.figure(figsize=(7,7))
plt.scatter(finalDf['P1'],finalDf['P2'],c=finalDf['TENURE'],cmap='prism', s =5)
plt.xlabel('pc1')
plt.ylabel('pc2')
```

Out[13]: `Text(0, 0.5, 'pc2')`

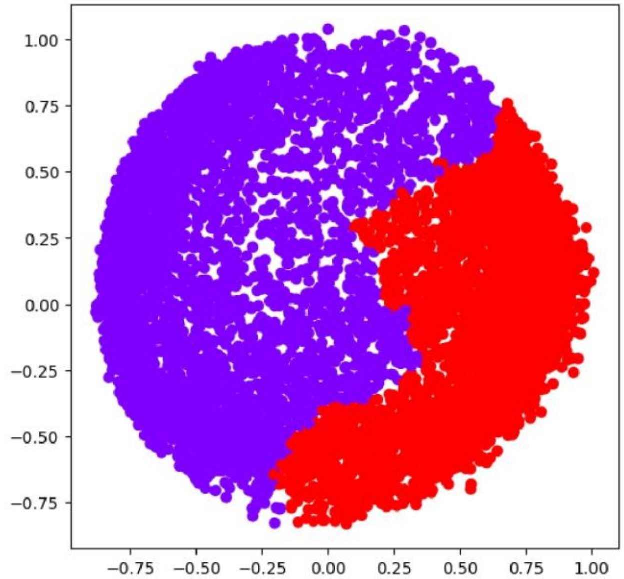


In [14]:
```
ac2 = AgglomerativeClustering(n_clusters = 2)

# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(principalDf['P1'], principalDf['P2'],
```

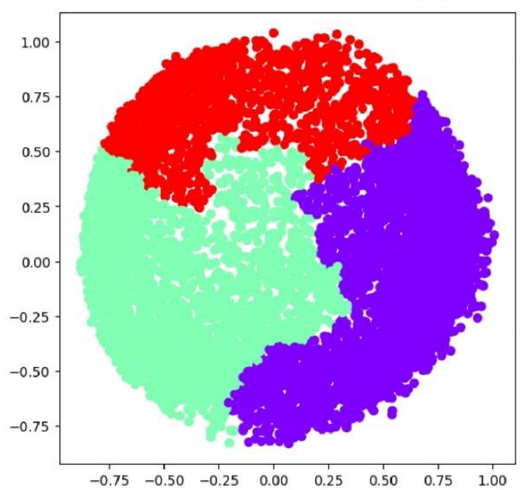

Performed Agglomerative Clustering on the principal data after applying PCA

- **Performed clustering with k value 2,3,4,5 and visualized them**

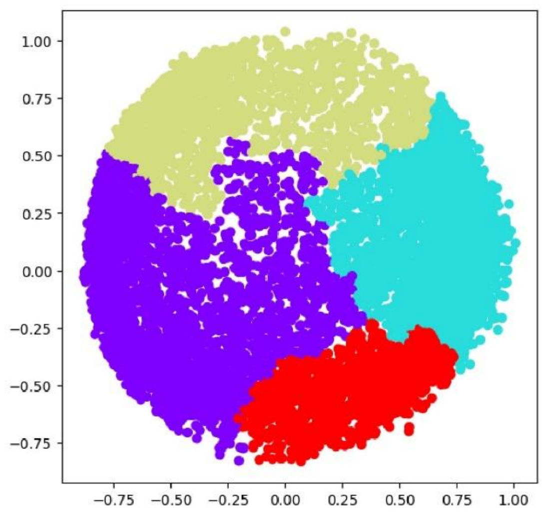

```
           c = ac2.fit_predict(principalDf), cmap ='rainbow')
plt.show()
```



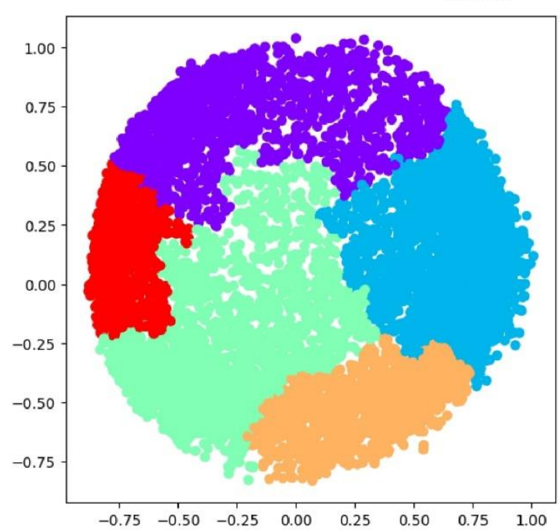

```
In [15]: ac3 = AgglomerativeClustering(n_clusters = 3)

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


```
In [16]: 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()
```



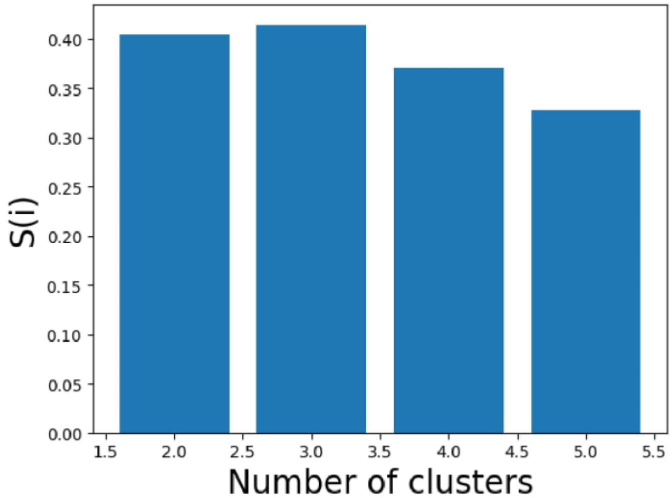
In [17]:
```
ac5 = AgglomerativeClustering(n_clusters = 5)

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

In [18]:
```
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)))
```


Calculated the Silhouette Score for each Agglomerative Clusters – k 2,3,4,5 and visualized in bar chat

```python
# 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()
```



In [ ]: