Assignment-6

GitHub link: https://github.com/SUSHMASHETTY6/Assignment-6

Question 1: Mathematical Solution

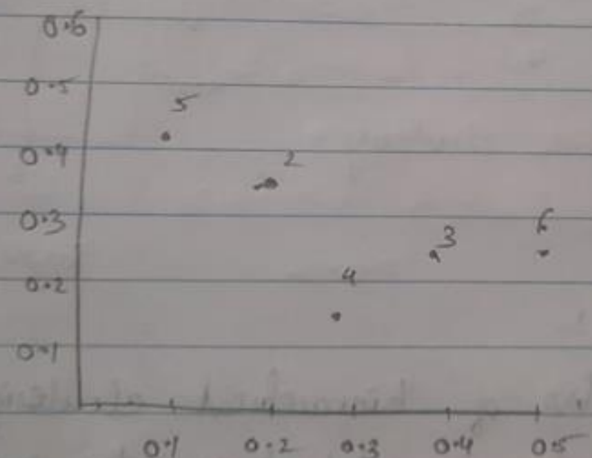
Q1) Calculate and find out clustering representation and dendrogram using Single, complete, and average link proximity function in hierarchical clustering technique.

| 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 |

| | 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 |

By Single link :-

* For single link hierarchical clustering, the proximity of two clusters is minimum of the distance between any two points in 2 different clusters
* The single link technique is good for non elliptical shapes, but sensitive to noise & outliers.
* Applying single link technique to our example data set of six points.



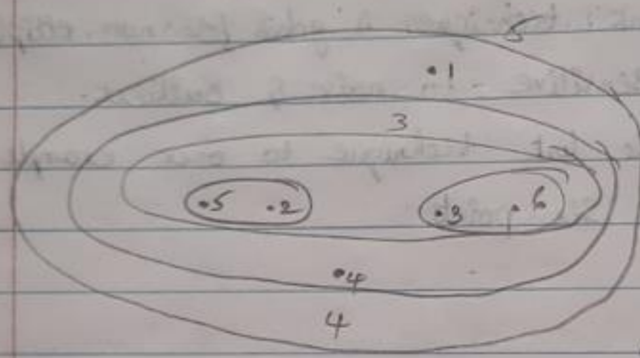

→ from table 1, we can observe distance between P3 & P6 is 0.11

→ The height at which two clusters are merged can be represented as distance between two clusters. Distance between clusters {3,6} & {2,5} is given by

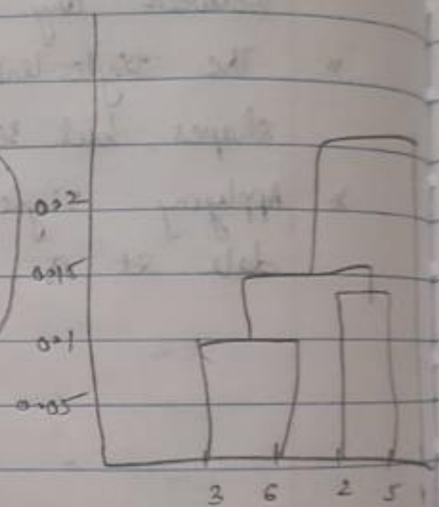
$$\text{dist}(\{3,6\},\{2,5\}) = \min(\text{dist}(3,2),\ \text{dist}(6,2),\ \text{dist}(3,5),\ \text{dist}(6,5))$$

$$\Rightarrow \min(0.15,\ 0.25,\ 0.28,\ 0.39)$$

$$\Rightarrow 0.15$$




single link clustering




Single link dendrogram

Complete link :-

→ In Complete link of hierarchical clustering, the proximity of two clusters is defined as " the maximum of the distance between any two points in two different clusters.

→ Complete link is less susceptible to noise & outliers, but it can break large clusters & its favours globular shapes.

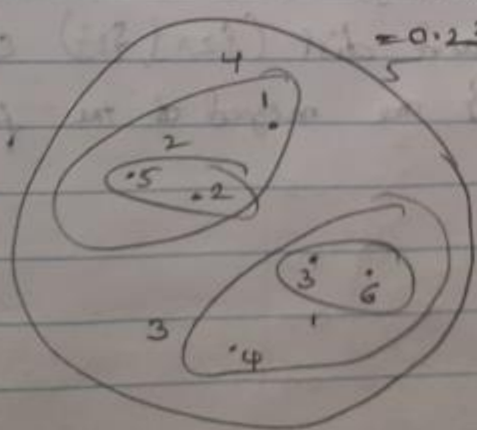
→ Below figure shows results of Applying max to the Sample data set of six points -

→ Points 3 & 6 are merged first.
{3, 6} is merged with {4} instead of {2,5} or {1} this is Because

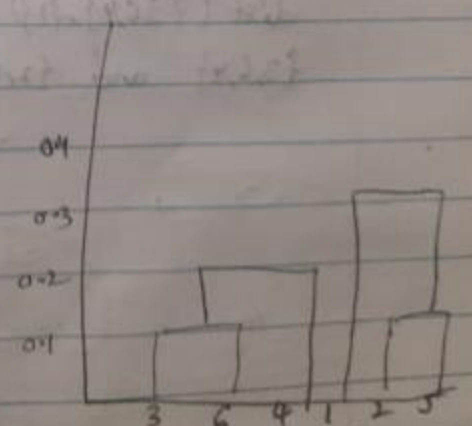
$$dist(\{3,6\}, \{4\}) = \max(dist(3,4), dist(6,4))$$
$$= \max(0.15, 0.22)$$
$$= 0.22$$

$$dist(\{3,6\}, \{2,5\}) = \max(dist(3,2), dist(6,2), dist(3,5), dist(6,5))$$
$$= \max(0.15, 0.25, 0.28, 0.39)$$
$$= 0.39$$

$$dist(\{3,6\}, \{1\}) = \max(dist(3,1), dist(6,1))$$
$$= \max(0.22, 0.23)$$
$$= 0.23$$




complete link clustering




Complete link dendrogram

Average link :-

Below fig shows results after applying the group average approach to sample data of six points.

→ we calculate the distance between some clusters

→ proximity = proximity $(c_i, c_j) = \dfrac{\sum_{x \in c_i, y \in c_j} \text{proximity}(x, y)}{m_i \times m_j}$

$\text{dist}(\{3,6,4\},\{1\}) = (0.22 + 0.37 + 0.23) / (3 \times 1)$

$= 0.28$

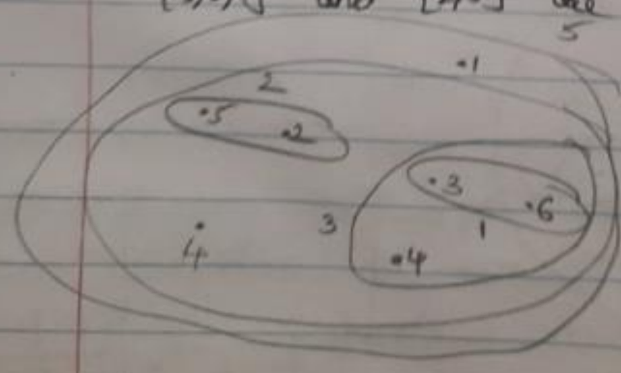
$\text{dist}(\{2,5\},\{1\}) = (0.24 + 0.34) / (2 \times 1)$

$= 0.29$

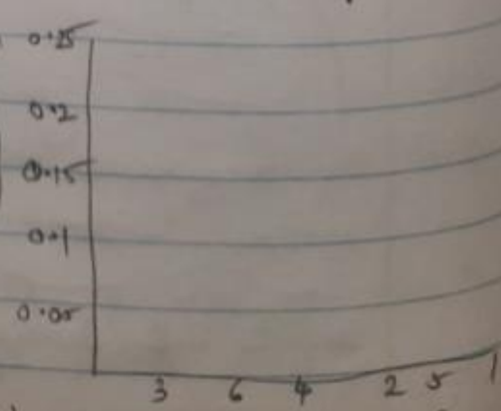
$\text{dist}(\{3,6,4\},\{2,5\}) = (0.15 + 0.28 + 0.25 + 0.39 + 0.20 + 0.2$ …

$= 0.26$

Here, Because $\text{dist}(\{3,6,4\},\{2,5\})$ is smaller than $\text{dist}(\{3,6,4\},\{1\})$ and $\text{dist}(\{2,5\},\{1\})$ clusters $\{3,6,4\}$ and $\{2,5\}$ are merged at the fourth stage




Group average clusters




5) group average dendrogram

→ Average version of hierarchical clustering, the proximity of two cluster is defined as the average pairwise proximity among all pair of points in the different clusters.

proximity proximity $(C_i, C_j)$ of cluster $C_i$ and $C_j$ which are of size $m_i$ and $m_j$ respectively

$$\text{Proximity}(C_i, C_j) = \sum_{x \in C_i} \frac{\text{Proximity}(x, y)}{m_i \times m_j}$$

Question 2: Screenshots:

In [52]:
```python
#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 [53]:
```python
dataframe = pd.read_csv('CC General.csv')
dataframe.info()
```

Reads the csv file

```
<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
```

# dataframe.head(): This function returns the first n rows for the object based on position.

In [54]:
```python
dataframe.head()
```

Out[54]:

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

The describe() method returns description of the data in the DataFrame.

If the DataFrame contains numerical data, the description contains this information for each column.

In [55]:
```
dataframe.describe()
```

Out[55]:

| | BALANCE | BALANCE_FREQUENCY | PURCHASES | ONEOFF_PURCHASES | INSTALLMENTS_PURCHASES | CASH_ADVANCE | PURCHASES_FREQUEN |
|---|---|---|---|---|---|---|---|
| count | 8950.000000 | 8950.000000 | 8950.000000 | 8950.000000 | 8950.000000 | 8950.000000 | 8950.000 |
| mean | 1564.474828 | 0.877271 | 1003.204834 | 592.437371 | 411.067645 | 978.871112 | 0.490 |
| std | 2081.531879 | 0.236904 | 2136.634782 | 1659.887917 | 904.338115 | 2097.163877 | 0.401 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000 |
| 25% | 128.281915 | 0.888889 | 39.635000 | 0.000000 | 0.000000 | 0.000000 | 0.083 |
| 50% | 873.385231 | 1.000000 | 361.280000 | 38.000000 | 89.000000 | 0.000000 | 0.500 |
| 75% | 2054.140036 | 1.000000 | 1110.130000 | 577.405000 | 468.637500 | 1113.821139 | 0.916 |
| max | 19043.138560 | 1.000000 | 49039.570000 | 40761.250000 | 22500.000000 | 47137.211760 | 1.000 |

*# Remove rows or columns by specifying label names and corresponding axis(in this case its 1)*

In [56]:
```
df = dataframe.drop(['CUST_ID'], axis=1)
df.head()
```

Out[56]:

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

# returns Boolean value

In [57]:
```
df.isnull().any()
```

Out[57]:
```
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
```

# modifies if any values are true

In [58]:
```
df.fillna(dataframe.mean(), inplace=True)
df.isnull().any()
```

Out[58]:
```
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
```

# color grid variances

In [59]:
```
df.corr().style.background_gradient(cmap="Greens")
```

| | BALANCE | BALANCE_FREQUENCY | PURCHASES | ONEOFF_PURCHASES | INSTALLMENTS_PURCHASES |
|---|---|---|---|---|---|
| **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 [69]:
```
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 [70]:
```
#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
X_normalized = preprocessing.normalize(X_scaled_df)
# Converting the numpy array into a pandas DataFrame
X_normalized = pd.DataFrame(X_normalized)
```

# Dataframe →principal components(P1,P2,Tenure)

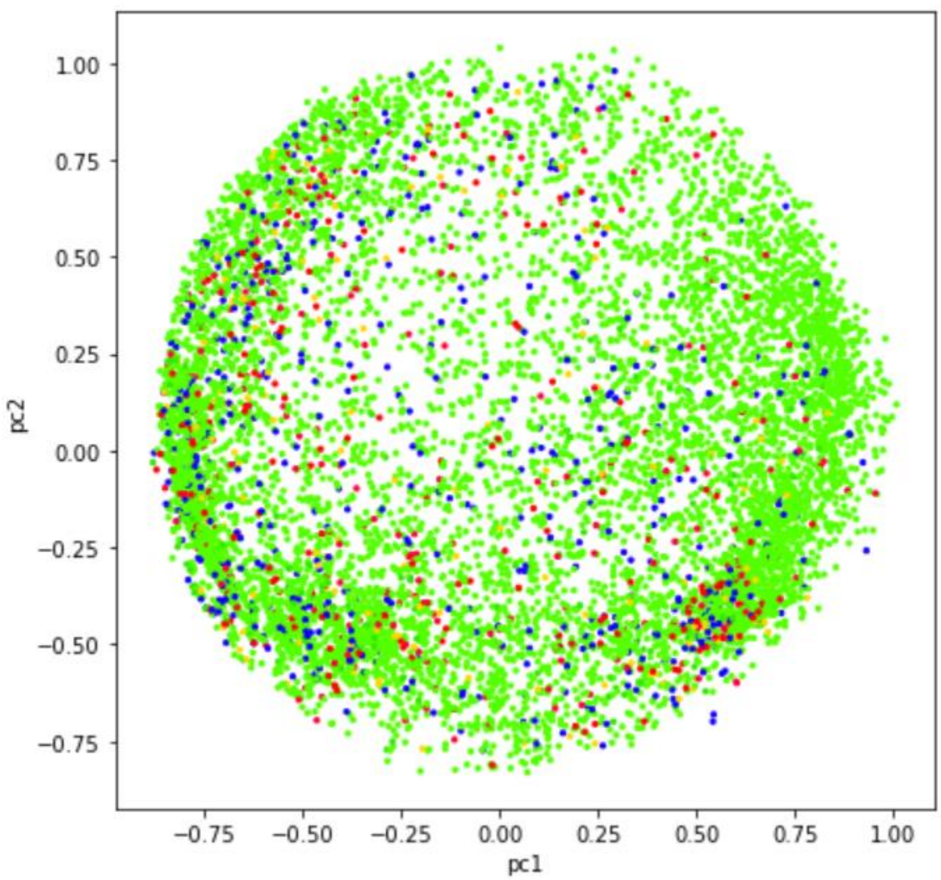
In [62]:
```
pca2 = PCA(n_components=2)
principalComponents = pca2.fit_transform(X_normalized)

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

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

Out[62]:

| | P1 | P2 | TENURE |
|---|---|---|---|
| **0** | -0.488186 | -0.677233 | 12 |
| **1** | -0.517294 | 0.556075 | 12 |
| **2** | 0.334384 | 0.287312 | 12 |
| **3** | -0.486616 | -0.080780 | 12 |
| **4** | -0.562175 | -0.474770 | 12 |

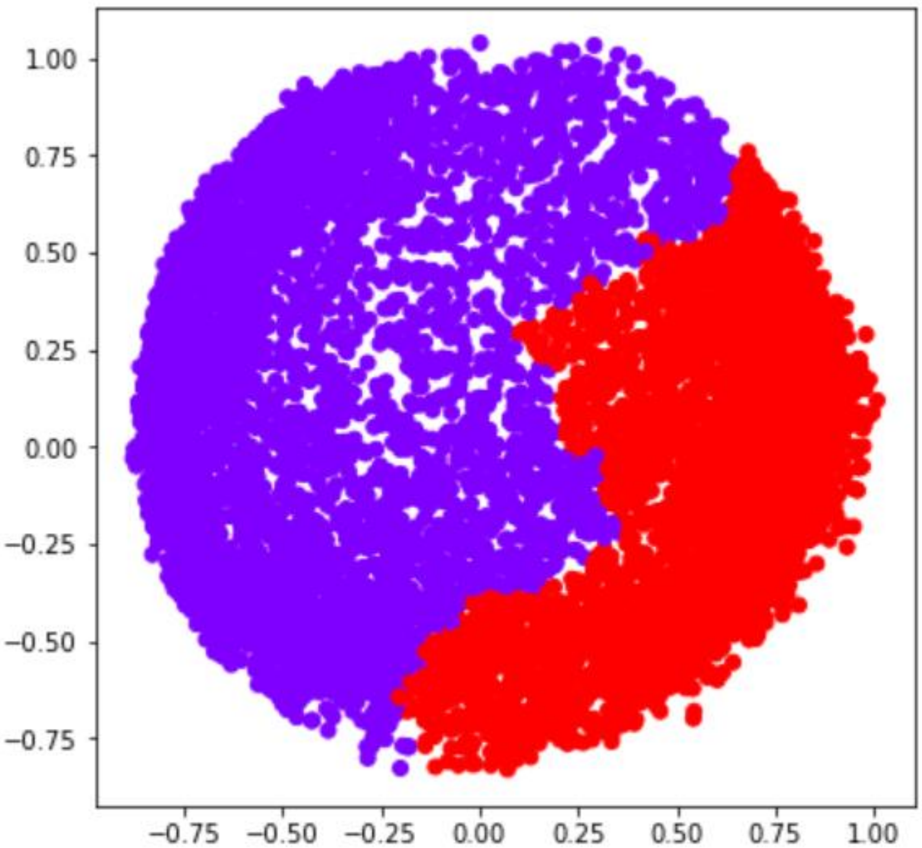
In [63]:
```
plt.figure(figsize=(7,7))
plt.scatter(finalDf['P1'],finalDf['P2'],c=finalDf['TENURE'],cmap='prism', s =5)
plt.xlabel('pc1')
plt.ylabel('pc2')
```

Out[63]: Text(0, 0.5, 'pc2')



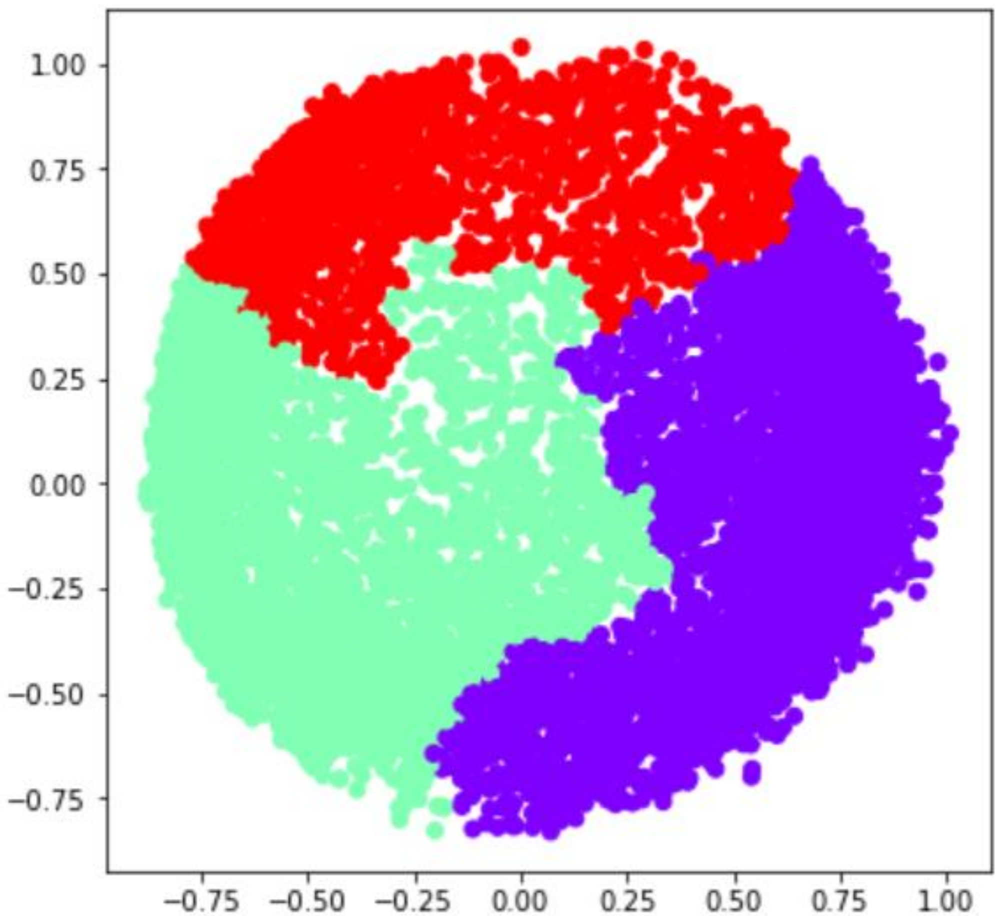

# Agglomertative clustering(n=2)

In [64]:
```
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()
```




# Agglomertative clustering(n=3)

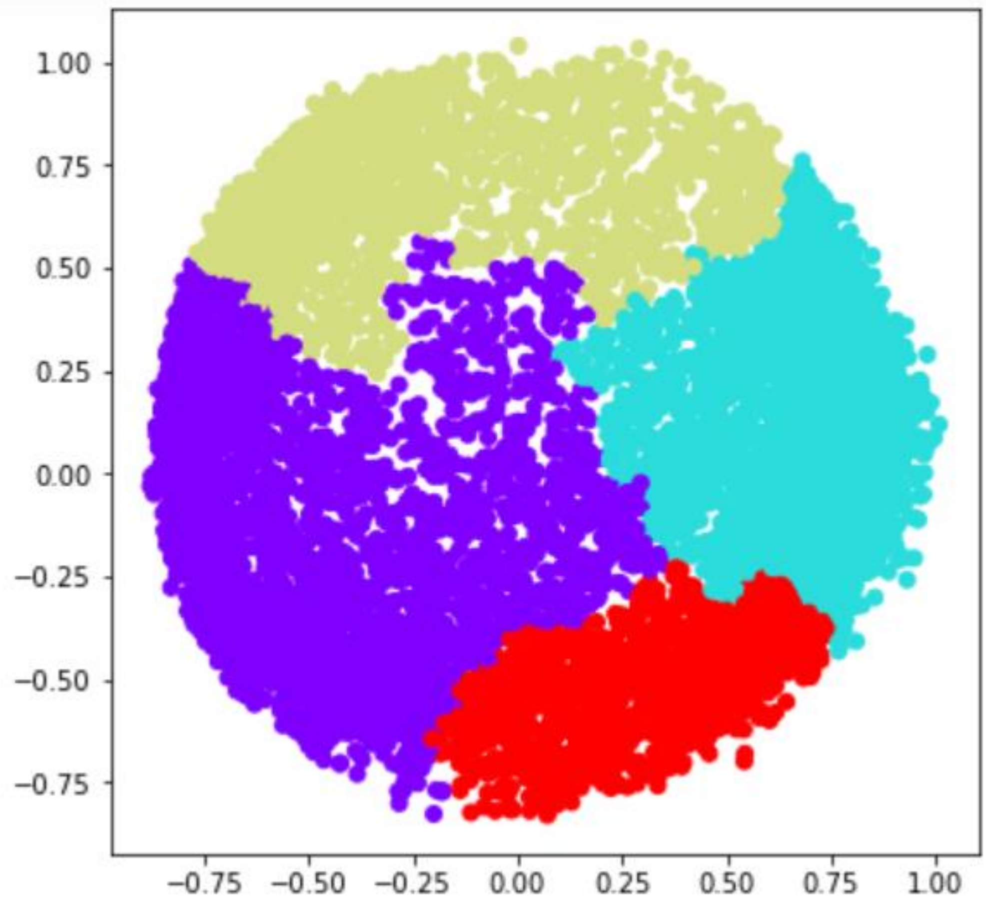
In [65]:
```
ac3 = AgglomerativeClustering(n_clusters = 3)

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




# Agglomertative clustering(n=4)

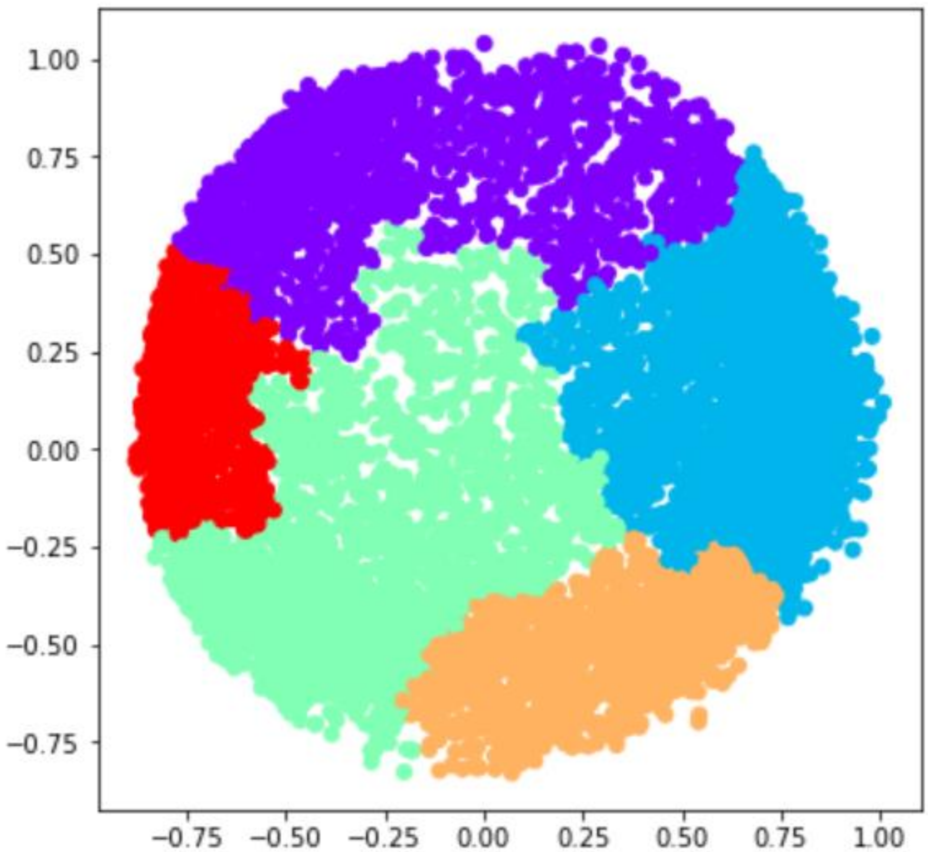
In [66]:

```
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()
```



# Agglomertative clustering(n=5)

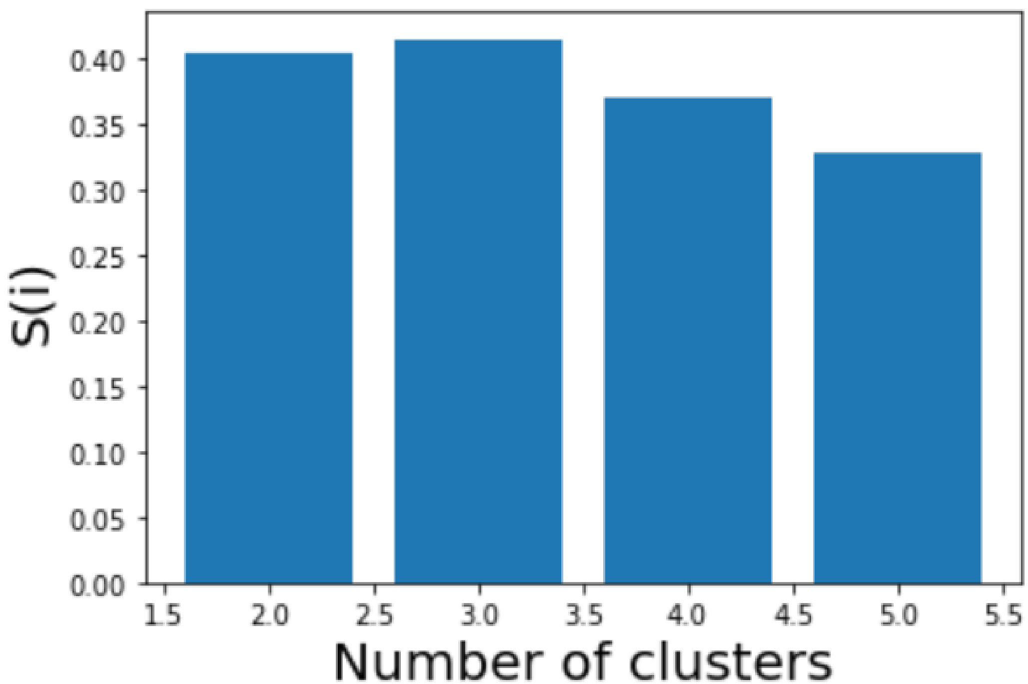
In [67]:
```
ac5 = AgglomerativeClustering(n_clusters = 5)

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




# Appending the silhouette score of the different models to the list and then plotting a bar graph to compare the results

In [68]:

```
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()
```



S(i)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
Number of clusters