Assignment – 6

Name: Satya Ishyanth Kadali

Student id: #700735513

CRN: 11813, Subject code: CS 5710

Single Link Proximity:

Single Linkage is defined as the distance between two clusters is the minimum distance between members of the two clusters.

	P1	P2	P3	P4	P5	P6
P1	0	0.2357	0.2218	0.3688	0.3421	0.2347
P2	0.2357	0	0.1483	0.2042	0.1388	0.254
P3	0.2218	0.1483	0	0.1513	0.2843	0.11
P4	0.3688	0.2042	0.1513	0	0.2932	0.2216
P5	0.3421	0.1388	0.2843	0.2932	0	0.3921
P6	0.2347	0.254	0.11	0.2216	0.3921	0

Smallest distance from above data is 0.11

P3 and P6 forms first cluster

	P1	P2	P36	P4	P5
P1	0	0.2357	0.2218	0.3688	0.3421
P2	0.2357	0	0.1483	0.2042	0.1388
P36	0.2218	0.1483	0	0.1513	0.2843
P4	0.3688	0.2042	0.1513	0	0.2932
P5	0.3421	0.1388	0.2843	0.2932	0

Smallest distance from above data is 0.1388

P2 and P5 forms 2nd cluster

	P1	**P25**	**P36**	**P4**
P1	0	0.2357	0.2218	0.3688
P25	0.2357	0	0.1483	0.2042
P36	0.2218	0.1483	0	0.1513
P4	0.3688	0.2042	0.1513	0

Smallest distance from above data is 0.1483

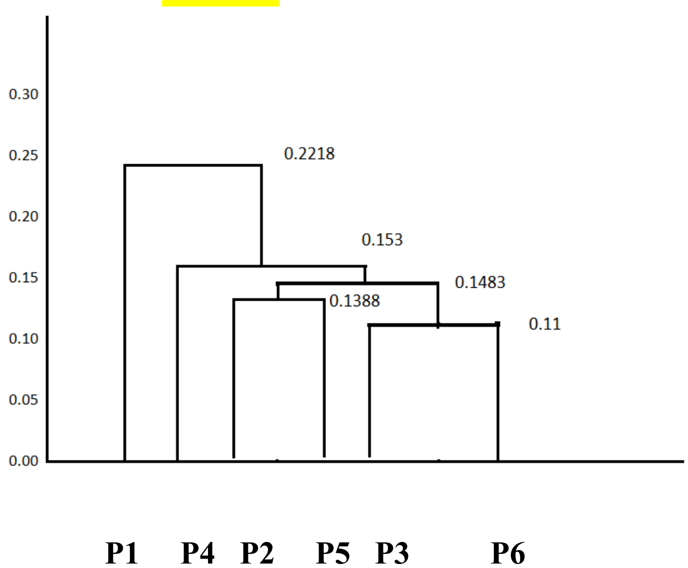
P25 and P36 forms 3rdcluster.

	P1	**P(25)(36)**	**P4**
P1	0	0.2218	0.3688
P(25)(36)	0.2218	0	0.1513
P4	0.3688	0.1513	0

Smallest distance from above data is 0.1513

P(25)(36)and P4 forms 4thcluster

	P1	**P4(25)(36)**
P1	0	0.2218
P4(25)(36)	0.2218	0



Complete Link Proximity:

Complete Linkage is defined as the distance between two clusters is the maximum distance between members of the two clusters

	P1	**P2**	**P3**	**P4**	**P5**	**P6**
P1	0	0.2357	0.2218	0.3688	0.3421	0.2347
P2	0.2357	0	0.1483	0.2042	0.1388	0.254
P3	0.2218	0.1483	0	0.1513	0.2843	0.11
P4	0.3688	0.2042	0.1513	0	0.2932	0.2216
P5	0.3421	0.1388	0.2843	0.2932	0	0.3921
P6	0.2347	0.254	0.11	0.2216	0.3921	0

Smallest distance from above data is 0.11

P3 and P6 forms first cluster

	P1	**P2**	**P36**	**P4**	**P5**
P1	0	0.2357	0.2347	0.3688	0.3421
P2	0.2357	0	0.254	0.2042	0.1388
P36	0.2347	0.254	0	0.2216	0.3921
P4	0.3688	0.2042	0.2216	0	0.2932
P5	0.3421	0.1388	0.3921	0.2932	0

Smallest distance from above data is 0.1388

P2 and P5 forms 2nd cluster

	P1	**P25**	**P36**	**P4**
P1	0	0.3421	0.2347	0.3688
P25	0.3421	0	0.3921	0.2932
P36	0.2347	0.3921	0	0.2216
P4	0.3688	0.2932	0.2216	0

Smallest distance from above data is 0.2216

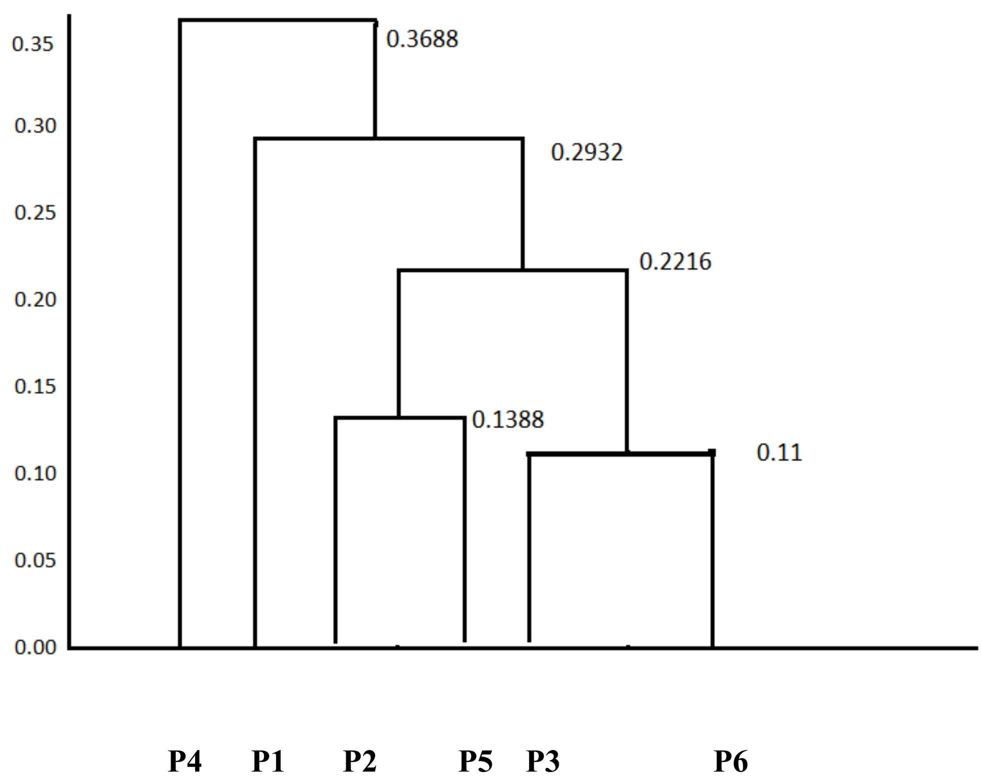
P25 and P36 forms 3rdcluster

	P1	**P(25)(36)**	**P4**
P1	0	0.3421	0.3688
P(25)(36)	0.3421	0	0.2932
P4	0.3688	0.2932	0

Smallest distance from above data is 0.2932

P(25)(36)and P1 forms 4th cluster

	P1(25)(36)	**P4**
P1(25)(36)	0	0.1483
P4	0.3688	0



Average Link Proximity:

Average Linkage is defined as the distance between two clusters is the average of all distances between members of the two clusters

	P1	P2	P3	P4	P5	P6
P1	0	0.2357	0.2218	0.3688	0.3421	0.2347
P2	0.2357	0	0.1483	0.2042	0.1388	0.254
P3	0.2218	0.1483	0	0.1513	0.2843	0.11
P4	0.3688	0.2042	0.1513	0	0.2932	0.2216
P5	0.3421	0.1388	0.2843	0.2932	0	0.3921
P6	0.2347	0.254	0.11	0.2216	0.3921	0

Smallest distance from above data is 0.11

P3 and P6 forms first cluster

	P1	P2	P36	P4	P5
P1	0	0.2357	0.22825	0.3688	0.3421
P2	0.2357	0	0.20115	0.2042	0.1388
P36	0.22825	0.20115	0	0.18645	0.3382
P4	0.3688	0.2042	0.18645	0	0.2932
P5	0.3421	0.1388	0.3382	0.2932	0

Smallest distance from above data is 0.1388

P2 and P5 forms 2nd cluster

	P1	P25	P36	P4
P1	0	0.2889	0.2347	0.3688
P25	0.2889	0	0.269675	0.2487
P36	0.2347	0.269675	0	0.18645
P4	0.3688	0.2487	0.18645	0

Smallest distance from above data is 0.18645

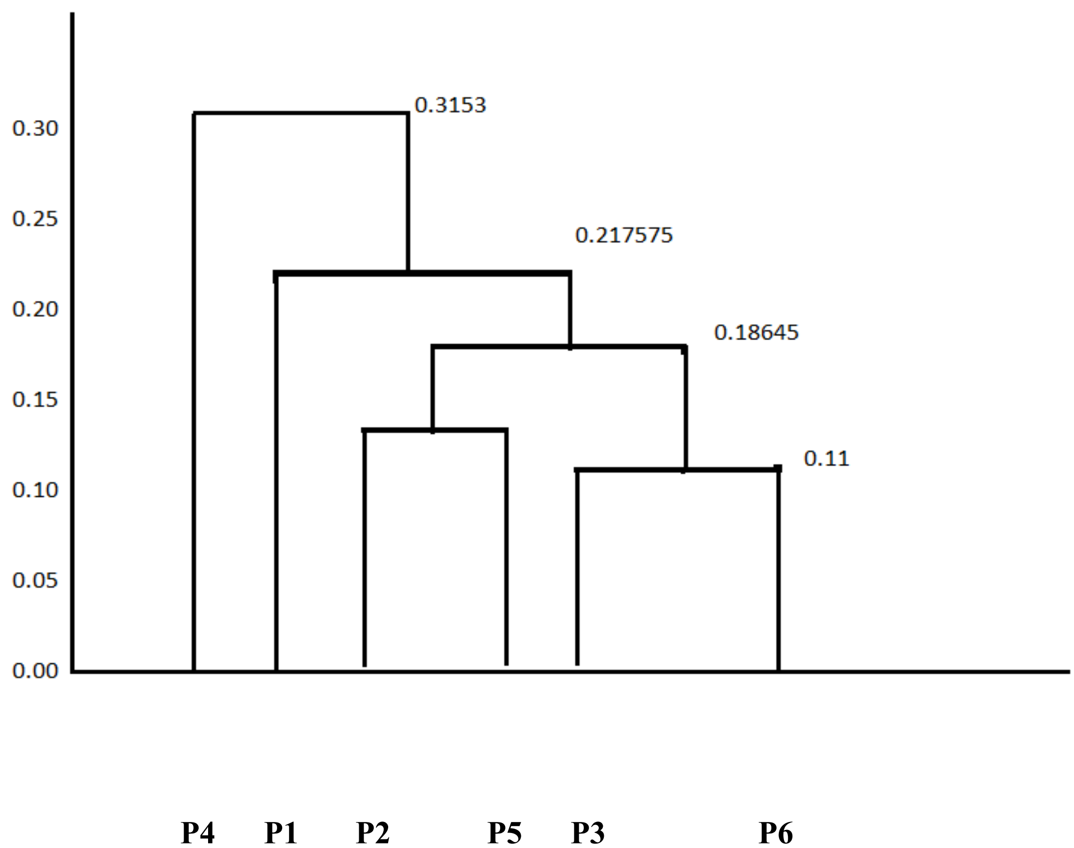
P25 and P36 forms 3rdcluster

	P1	P(25)(36)	P4
P1	0	0.2618	0.3688
P(25)(36)	0.2618	0	0.217575
P4	0.3688	0.217575	0

Smallest distance from above data is 0.217575

P(25)(36)and P1 forms 4thcluster

	P1(25)(36)	P4
P1(25)(36)	0	0.3153
P4	0.3153	0



```
Question-2
```

In [27]:
```
#importing libraries
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 [8]:
```
#reading the data into pandas dataframe
dataframe = pd.read_csv('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
```

Importing Libraries

Feeding the data to the pandas dataframe.

Describing data to find the statistics of the columns

Dropping the CUST_ID column as the data is in alpha numeric, which is hard to compute with the algorithm.

Later, checking the null values in the columns and replacing the null values by taking the mean of that column.

In [9]:
```python
#getting first five rows
dataframe.head()
```

Out[9]:

	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	(
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 [10]:
```python
#Statistics of the data
dataframe.describe()
```

Out[10]:

	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

In [11]:
```python
#dropping the column
df = dataframe.drop(['CUST_ID'], axis=1)
df.head()
```

Out[11]:

	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

In [12]:
```python
#checking the column if having any null values
df.isnull().any()
```

Out[12]:
```
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 [24]:
#Replacing the null values with mean
df.fillna(dataframe.mean(), inplace=True)
df.isnull().any()
```

```
Out[24]: 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 [14]:
df.corr().style.background_gradient(cmap="Oranges")
```

Out[14]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQ
BALANCE	1.000000	0.322412	0.181261	0.164350	0.126469	0.496692	-0
E_FREQUENCY	0.322412	1.000000	0.133674	0.104323	0.124292	0.099388	0
PURCHASES	0.181261	0.133674	1.000000	0.916845	0.679896	-0.051474	0
F_PURCHASES	0.164350	0.104323	0.916845	1.000000	0.330622	-0.031326	0
S_PURCHASES	0.126469	0.124292	0.679896	0.330622	1.000000	-0.064244	0
ASH_ADVANCE	0.496692	0.099388	-0.051474	-0.031326	-0.064244	1.000000	-0
S_FREQUENCY	-0.077944	0.229715	0.393017	0.264937	0.442418	-0.215507	1
S_FREQUENCY	0.073166	0.202415	0.498430	0.524891	0.214042	-0.086754	0
S_FREQUENCY	-0.063186	0.176079	0.315567	0.127729	0.511351	-0.177070	0
E_FREQUENCY	0.449218	0.191873	-0.120143	-0.082628	-0.132318	0.628522	-0
ADVANCE_TRX	0.385152	0.141555	-0.067175	-0.046212	-0.073999	0.656498	-0
RCHASES_TRX	0.154338	0.189626	0.689561	0.545523	0.628108	-0.075850	0
CREDIT_LIMIT	0.531267	0.095795	0.356959	0.319721	0.256496	0.303983	0
PAYMENTS	0.322802	0.065008	0.603264	0.567292	0.384084	0.453238	0
UM_PAYMENTS	0.394282	0.114249	0.093515	0.048597	0.131687	0.139223	0
ULL_PAYMENT	-0.318959	-0.095082	0.180379	0.132763	0.182569	-0.152935	0
TENURE	0.072692	0.119776	0.086288	0.064150	0.086143	-0.068312	0

By using cmap gradient, here we can see the highest values as the darker color and lowest values as the lighter color.

Now preprocessing the data by Scaling and Normalizing to get the data into specific range of values.

Reducing the input dimensions to 2 features by using Principal Component Analysis.

In [26]:

```
#Using Standard Scaler
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 [16]:

```
#using Normalization to rescaling real-valued numeric attributes into a 0 to 1 range.
X_normalized = preprocessing.normalize(X_scaled_df)
# Converting the numpy array into a pandas DataFrame
X_normalized = pd.DataFrame(X_normalized)
```

In [17]:

```
#Reducing the input dimensions to 2 features
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()
```

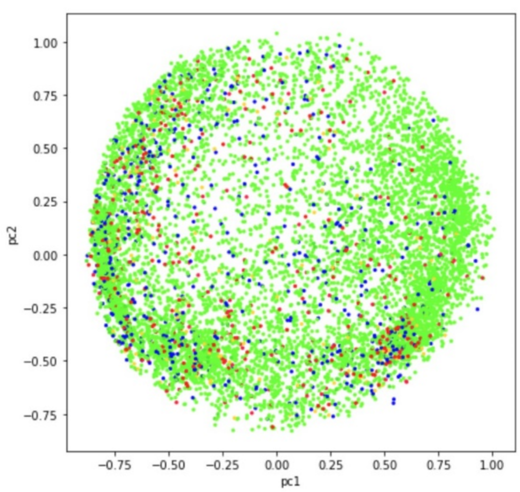
Out[17]:

	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 [18]:

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

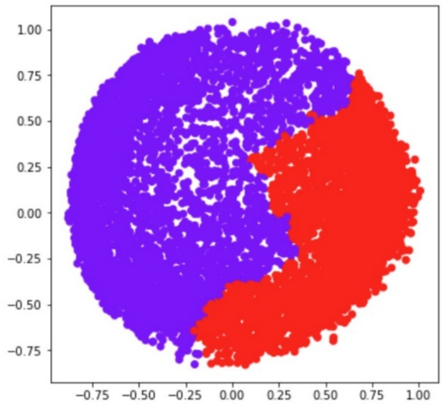
Out[18]: Text(0, 0.5, 'pc2')



By using Agglomerative clustering we are taking n_clusters as 2,3,4,5.

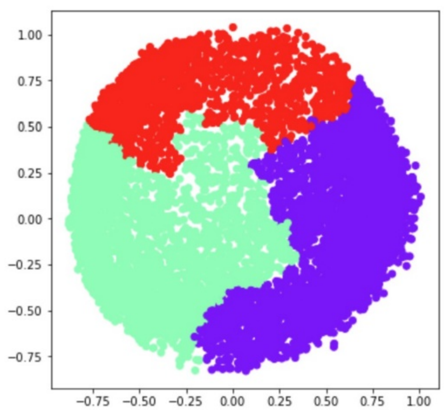
In [19]:

```python
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()
```



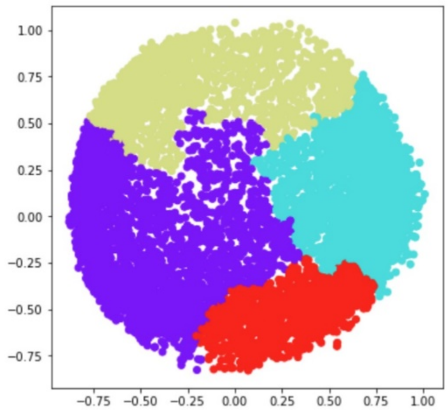
In [20]:

```python
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()
```



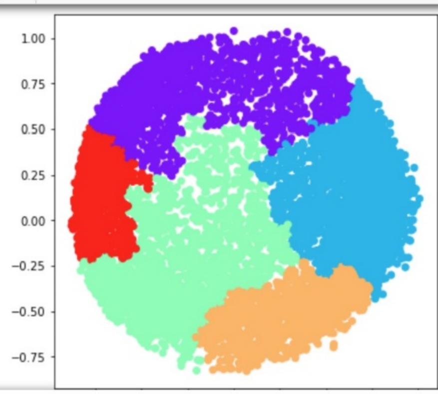
In [21]:

```python
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()
```



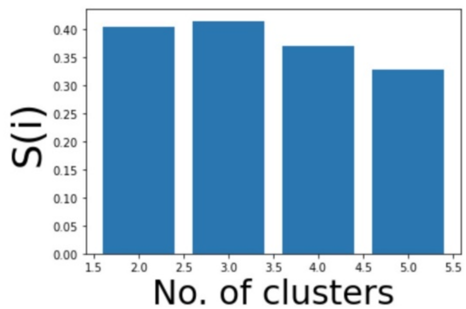
In [22]:

```python
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 [23]:

```python
#Silhouette Scores

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('No. of clusters', fontsize = 30)
plt.ylabel('S(i)', fontsize = 35)
plt.show()
```



Here we are finding Silhouette scores to calculate the performance for each no. of clusters.

Now we are comparing the Silhouette scores in the bar graph and finding the more number of clusters the less similarities the clusters have in between.

Taking 3 number of clusters is optimal.