NAME: Tanuja Pasupuleti

700 Number: 700727418

GITHUB_LINK: https://github.com/tanujapasupuleti22/assignment6.git

VIDEO_LINK:
https://drive.google.com/drive/folders/1stIzn1PCssXmx7PhJx-nSOxba39M2TMP?usp=sharing

Question 1

Displayed data below

Question 1

In [ ]:
```
# Attached the Mathematical calculations in the submission Document. Below Code is just for the reference
```

In [ ]:
```
# calculate and find out clustering representations and dendrogram using Single,
# complete, and average link proximity function in hierarchical clustering technique.
```

In [50]:
```
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
```

Out[50]:

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

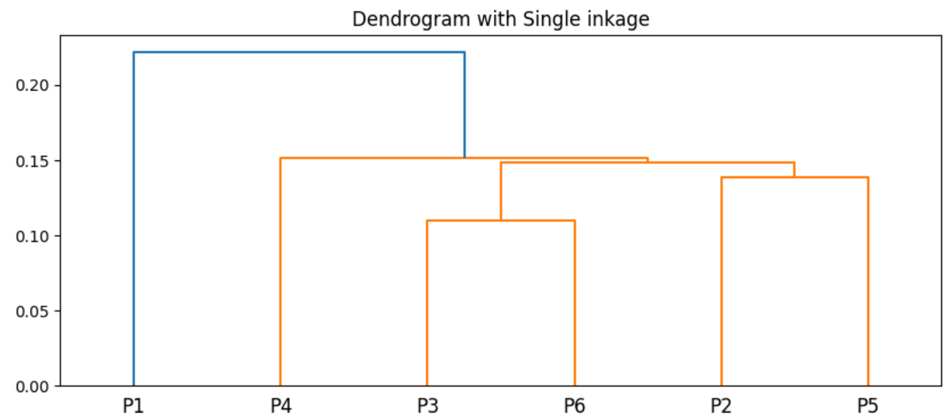
Showing Dendrogram with Single linkage below using dendrogram method

In [90]:
```
dist = pd.DataFrame(squareform(np.round(pdist(data[['x cordinate', 'y cordinate']]),4), 'euclidean'), columns=data.index.values,
dist
```

Out[90]:

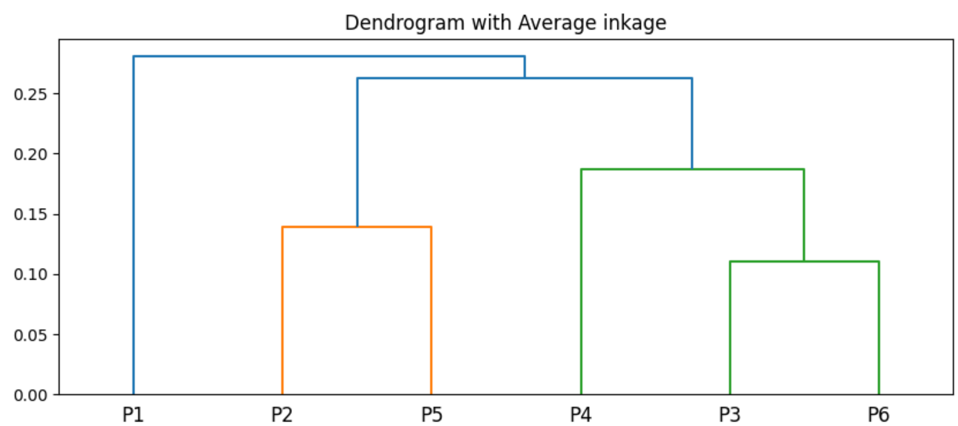
	P1	P2	P3	P4	P5	P6
P1	0.0000	0.2357	0.2219	0.3688	0.3421	0.2348
P2	0.2357	0.0000	0.1483	0.2042	0.1389	0.2540
P3	0.2219	0.1483	0.0000	0.1513	0.2843	0.1099
P4	0.3688	0.2042	0.1513	0.0000	0.2932	0.2216
P5	0.3421	0.1389	0.2843	0.2932	0.0000	0.3921
P6	0.2348	0.2540	0.1099	0.2216	0.3921	0.0000

In [91]:
```
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)
```



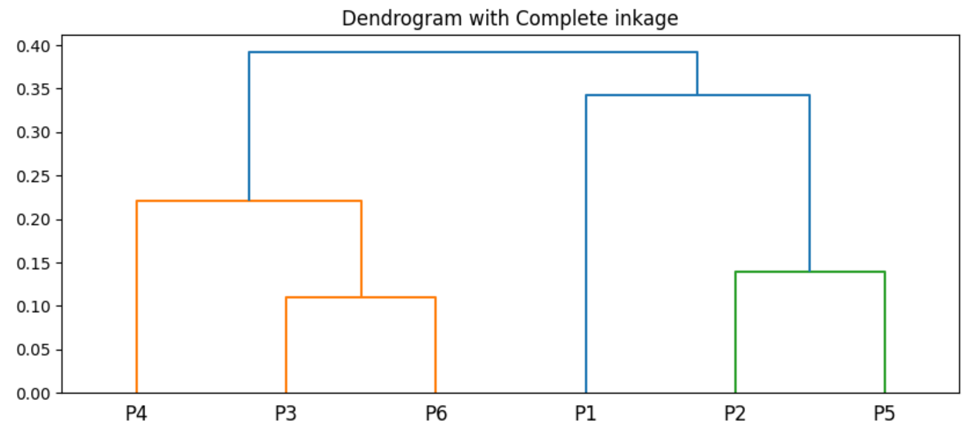
Showing Dendrogram with Average linkage below using dendrogram method

In [92]:
```
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)
```



Showing Dendrogram with Complete linkage below using dendrogram method

In [93]:
```
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)
```



Question 2:

In [55]:
```
#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 [56]:
```
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  
 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
```

First, I have imported the required libraries. Then imported the dataset 'CC GENERAL.csv'. Dataset is also displayed using head () function and there is description of the dataset.

```
In [57]: dataframe.head()
```

Out[57]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083

```
In [58]: dataframe.describe()
```

Out[58]:

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

```
In [59]: df = dataframe.drop(['CUST_ID'], axis=1)
         df.head()
```

Out[59]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEC
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	

For Question 2(a), I have deleted the first column which is 'CUST_ID'. I have checked for the null
values in the dataset there are 2 attributes with the null values. I have used the mean values to fill the null values of those two attributes.

In [60]:
```
df.isnull().any()
```

Out[60]:
```
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 [61]:
```
df.fillna(dataframe.mean(), inplace=True)
df.isnull().any()
```

Out[61]:
```
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
```

Use corr() function to find the correlation among the columns in the Dataframe using the 'Pearson' method. With green gradient.

In [62]:
```
df.corr().style.background_gradient(cmap="Greens")
```

Out[62]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVA
BALANCE	1.000000	0.322412	0.181261	0.164350	0.126469	0.49
BALANCE_FREQUENCY	0.322412	1.000000	0.133674	0.104323	0.124292	0.09
PURCHASES	0.181261	0.133674	1.000000	0.916845	0.679896	-0.05
ONEOFF_PURCHASES	0.164350	0.104323	0.916845	1.000000	0.330622	-0.03
INSTALLMENTS_PURCHASES	0.126469	0.124292	0.679896	0.330622	1.000000	-0.06
CASH_ADVANCE	0.496692	0.099388	-0.051474	-0.031326	-0.064244	1.00
PURCHASES_FREQUENCY	-0.077944	0.229715	0.393017	0.264937	0.442418	-0.21
ONEOFF_PURCHASES_FREQUENCY	0.073166	0.202415	0.498430	0.524891	0.214042	-0.08
PURCHASES_INSTALLMENTS_FREQUENCY	-0.063186	0.176079	0.315567	0.127729	0.511351	-0.17
CASH_ADVANCE_FREQUENCY	0.449218	0.191873	-0.120143	-0.082628	-0.132318	0.62
CASH_ADVANCE_TRX	0.385152	0.141555	-0.067175	-0.046212	-0.073999	0.65
PURCHASES_TRX	0.154338	0.189626	0.689561	0.545523	0.628108	-0.07
CREDIT_LIMIT	0.531267	0.095795	0.356959	0.319721	0.256496	0.30
PAYMENTS	0.322802	0.065008	0.603264	0.567292	0.384084	0.45
MINIMUM_PAYMENTS	0.394282	0.114249	0.093515	0.048597	0.131687	0.13
PRC_FULL_PAYMENT	-0.318959	-0.095082	0.180379	0.132763	0.182569	-0.15
TENURE	0.072692	0.119776	0.086288	0.064150	0.086143	-0.06

For question 2(b), first I have applied the standard scaler. And then I have normalized the data using normalize () function. In above screenshot I have displayed the dataset after the standard scaler and after normalizing to see how dataset changes.

```
In [43]: 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 [63]: #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 simil
         X_normalized = preprocessing.normalize(X_scaled_df)
         # Converting the numpy array into a pandas DataFrame
         X_normalized = pd.DataFrame(X_normalized)
```

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

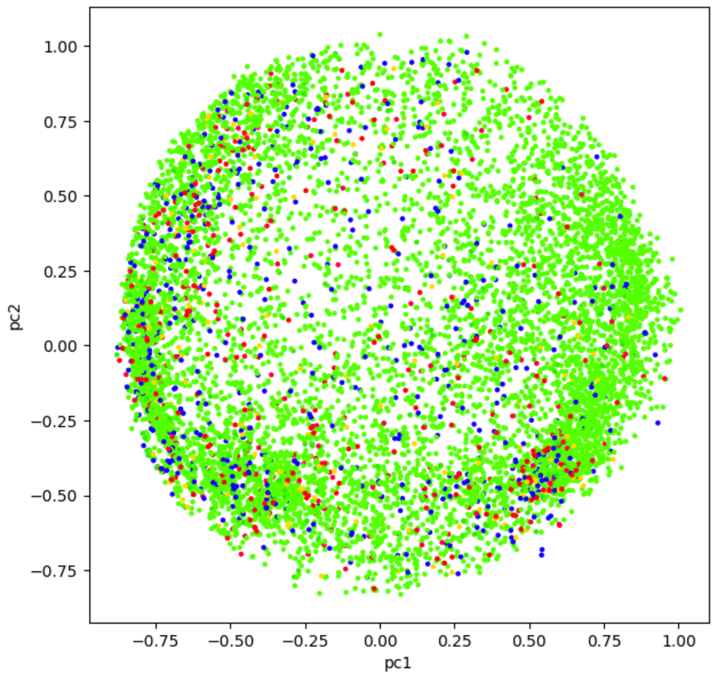
Out[64]:

	P1	P2	TENURE
0	-0.488186	-0.677233	12
1	-0.517295	0.556075	12
2	0.334385	0.287312	12
3	-0.486617	-0.080780	12
4	-0.562175	-0.474770	12

After applying normalizing we get array as an output so I have converted the array into panda dataframe and displayed the dataset named ‘principalDf’.
For Question 2(c), I have implemented PCA where I taken k = 2. So, the dataset x_norm has been transformed into array. I have again transform the array into panda dataframe which has 2 column named ‘P1’,’P2’ and the name of the dataset is x_pca. It is displayed in the screenshot.

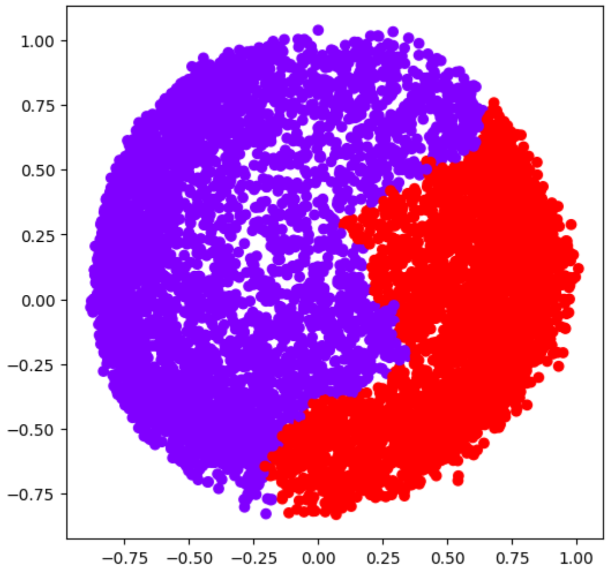
In [65]:
```
plt.figure(figsize=(7,7))
plt.scatter(finalDf['P1'],finalDf['P2'],c=finalDf['TENURE'],cmap='prism', s =5)
plt.xlabel('pc1')
plt.ylabel('pc2')
```

Out[65]: `Text(0, 0.5, 'pc2')`

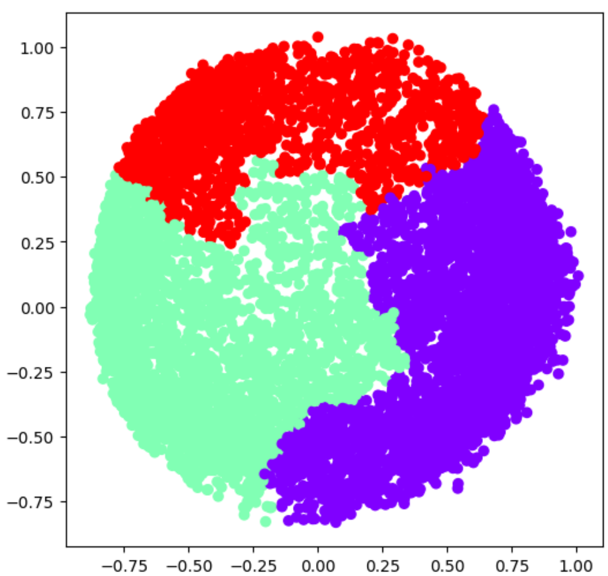


For question 2(d), I have implemented agglomerative clustering using sklearn library. Where the number of clusters is 2. Also, the output has been displayed using the scatterplot. 2 different cluster has been displayed using two different colors.

In [66]:
```
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()
```

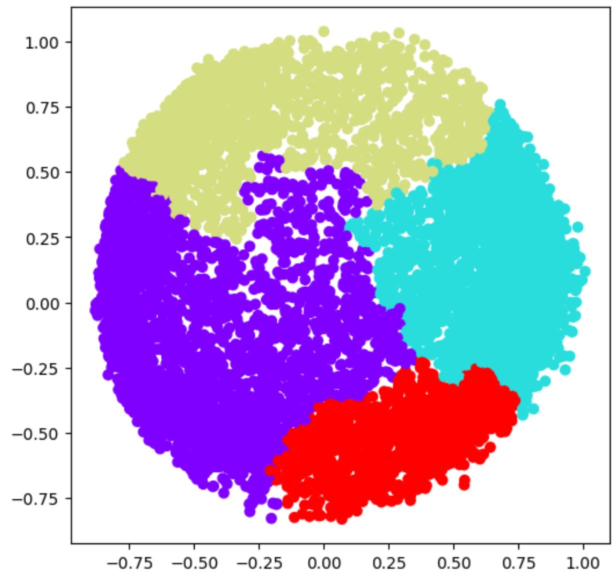


In [67]:
```
ac3 = AgglomerativeClustering(n_clusters = 3)

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



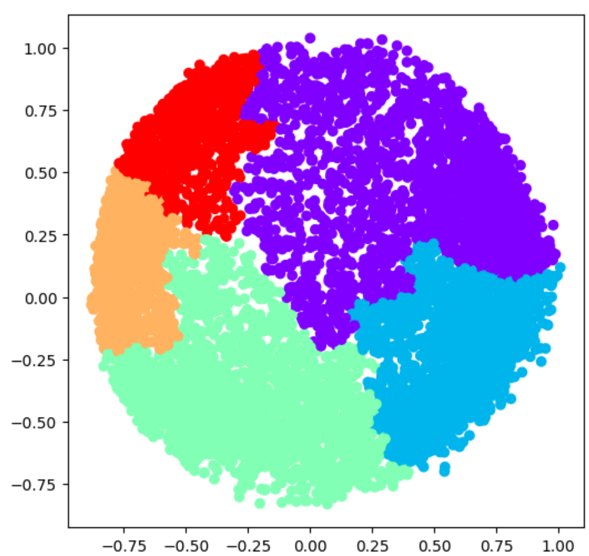
Above I have implemented the agglomerative cluster where number of clusters is 3. Three different colors represent three clusters.

```
In [68]: 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()
```



Above is the implementation of the agglomerative cluster with number of the cluster 4.

```
In [87]: 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()
```



Above is the implementation of agglomerative cluster where number of cluster is 5.

For question 2(e), first I have calculated the silhouette score for all clusters model named "S2,S3,S4,S5" and added to the list named "ss".
I have used the bar graph to represent the silhouette score of each model. In bar graph y-axis represent the silhouette score and x-axis represent cluster models.

```
In [88]: 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()
```