Name : Garimella Suresh Babu
Student Id: 700741367
User Id :SXG13670@ucmo.edu

GitHub Link:-
https://github.com/Suresh-Garimella/Heirarchial_Clustering_Algorithms

Video Link:-

https://github.com/Suresh-Garimella/Heirarchial_Clustering_Algorithms/blob/main/Screen_Recording.mkv

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.

Importing the required Libraries Namely Padas,Seaborn,sklearn ,numpy and matplotlib.

```
In [43]: #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")
```

Then read the CC GENERAL.csv file using pandas into a data frame.

In [44]:
```python
df = pd.read_csv('CC GENERAL.csv')
```

In [45]:
```python
df.head()
```

Out[45]:

	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

During Preprocessing we can see that the CUST_ID column is of no use , so deleting the column.

In [46]:
```python
del df['CUST_ID']

df.head()
```

Out[46]:

	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	

Checking if the data frame has any values using isnull.().any() function .

In [47]:
```
df.isnull().any()
```

Out[47]:
```
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
```

Replacing the null values with their respective means and checking the Data frame.
Here we can see that now the data frame has no null values.

In [48]:
```
mean1=df['CREDIT_LIMIT'].mean()
mean2=df['MINIMUM_PAYMENTS'].mean()
df['CREDIT_LIMIT'].fillna(value=mean1, inplace=True)
df['MINIMUM_PAYMENTS'].fillna(value=mean2, inplace=True)
```

In [49]:
```
df.isnull().any()
```

Out[49]:
```
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 pandas we can get the Correlation matrix using corr() function.
Background_gradient visualization using pandas and Heat map visualization using Seaborn.

In [50]:
```python
# This plots the correlation matrix using the color gradient "YLOrRd"
df.corr().style.background_gradient(cmap="YlOrRd")
```

Out[50]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN
BALANCE	1.000000	0.322412	0.181261	0.164350	0.126469	0.496692	-0.077
:E_FREQUENCY	0.322412	1.000000	0.133674	0.104323	0.124292	0.099388	0.229
PURCHASES	0.181261	0.133674	1.000000	0.916845	0.679896	-0.051474	0.393
F_PURCHASES	0.164350	0.104323	0.916845	1.000000	0.330622	-0.031326	0.264
TS_PURCHASES	0.126469	0.124292	0.679896	0.330622	1.000000	-0.064244	0.442
:ASH_ADVANCE	0.496692	0.099388	-0.051474	-0.031326	-0.064244	1.000000	-0.215
ES_FREQUENCY	-0.077944	0.229715	0.393017	0.264937	0.442418	-0.215507	1.000
ES_FREQUENCY	0.073166	0.202415	0.498430	0.524891	0.214042	-0.086754	0.501
TS_FREQUENCY	-0.063186	0.176079	0.315567	0.127729	0.511351	-0.177070	0.862
:E_FREQUENCY	0.449218	0.191873	-0.120143	-0.082628	-0.132318	0.628522	-0.308
_ADVANCE_TRX	0.385152	0.141555	-0.067175	-0.046212	-0.073999	0.656498	-0.203
JRCHASES_TRX	0.154338	0.189626	0.689561	0.545523	0.628108	-0.075850	0.568
CREDIT_LIMIT	0.531267	0.095795	0.356959	0.319721	0.256496	0.303983	0.119
PAYMENTS	0.322802	0.065008	0.603264	0.567292	0.384084	0.453238	0.103
IUM_PAYMENTS	0.394282	0.114249	0.093515	0.048597	0.131687	0.139223	0.002
FULL_PAYMENT	-0.318959	-0.095082	0.180379	0.132763	0.182569	-0.152935	0.305
TENURE	0.072692	0.119776	0.086288	0.064150	0.086143	-0.068312	0.061

b) Apply StandardScaler() and normalize() functions to scale and normalize raw input data.

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing.

It basically helps to normalize the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.

Here we are using standard scalar feature scaling technique to normalize the data.

In [51]:
```python
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)
```

We can use normalize() to normalize the data.

In [52]:
```python
#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)
```

c) Use PCA with K=2 to reduce the input dimensions to two features.

Then transformed the data using PCA with 2 components that means the final dataset has only 2 columns excluding the final attribute.

Principal component analysis can be broken down into five steps. I'll go through each step, providing logical explanations of what PCA is doing and simplifying mathematical concepts such as standardization, covariance, eigenvectors and eigenvalues without focusing on how to compute them.

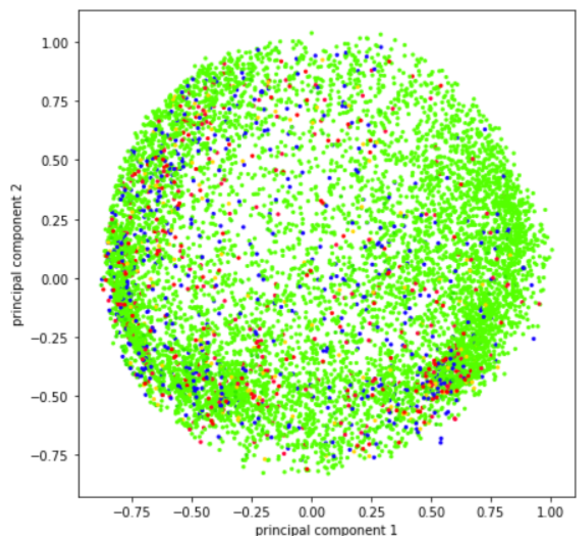
```
In [53]: pca2 = PCA(n_components=2)
         principalComponents = pca2.fit_transform(X_normalized)

         principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])

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

Out[53]:

	principal component 1	principal component 2	TENURE
0	-0.488186	-0.677233	12
1	-0.517294	0.556075	12
2	0.334384	0.287313	12
3	-0.486617	-0.080780	12
4	-0.562175	-0.474770	12

Visualization of the DataFrame After Performing PCA.

```
In [57]: plt.figure(figsize=(7,7))
         plt.scatter(finalDf['principal component 1'],finalDf['principal component 2'],c=finalDf['TENURE'],cmap='prism', s =5)
         plt.xlabel('principal component 1')
         plt.ylabel('principal component 2')

Out[57]: Text(0, 0.5, 'principal component 2')
```



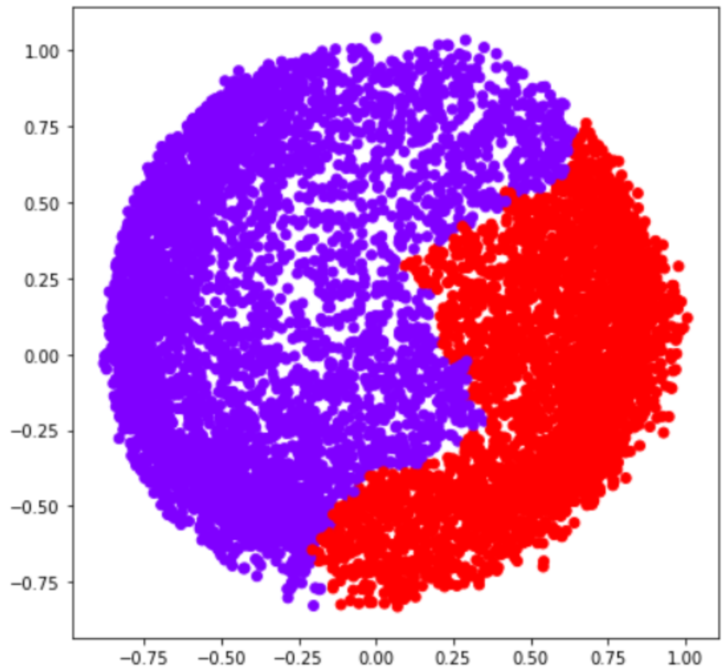
d) Apply Agglomerative Clustering with k=2,3,4 and 5 on reduced features and visualize the result for each k value using scatter plot.

Agglomerative Clustering is a type of hierarchical clustering algorithm. It is an unsupervised machine learning technique that divides the population into several clusters such that data points in the same cluster are more similar and data points in different clusters are dissimilar.

Here K states Number of clusters the Algorithm should form.

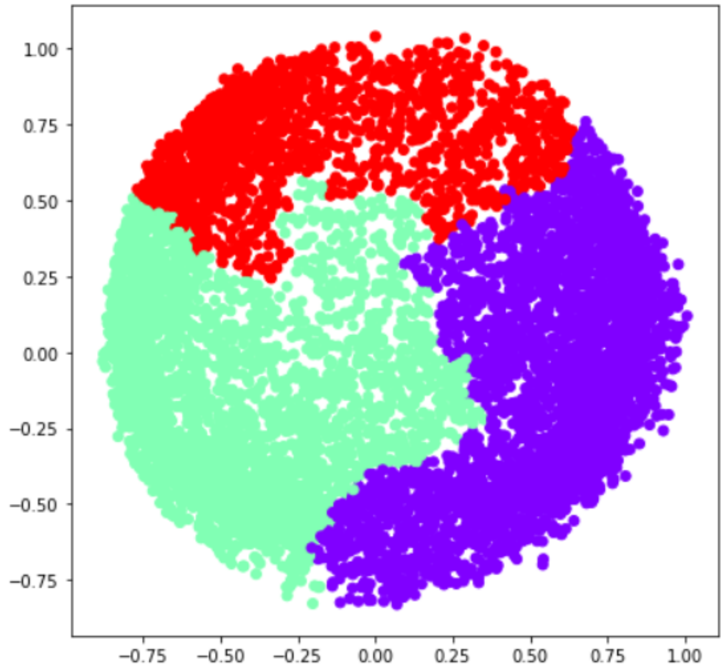
When K=2 i.e Agglomerative Clustering with 2 clusters and its scatter plot.

In [58]:
```
ac2 = AgglomerativeClustering(n_clusters = 2)

# Visualizing the clustering
plt.figure(figsize =(7, 7))
plt.scatter(principalDf['principal component 1'], principalDf['principal component 2'],
           c = ac2.fit_predict(principalDf), cmap ='rainbow')
plt.show()
```



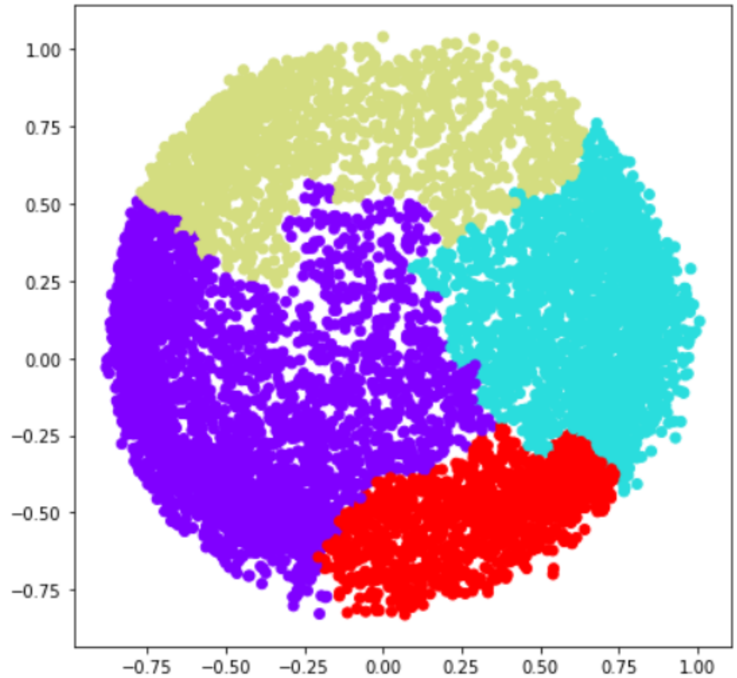
When K=3 i.e Agglomerative Clustering with 3 clusters and its scatter plot.

```
In [59]: ac3 = AgglomerativeClustering(n_clusters = 3)

         # Visualizing the clustering
         plt.figure(figsize =(7, 7))
         plt.scatter(principalDf['principal component 1'], principalDf['principal component 2'],
                     c = ac3.fit_predict(principalDf), cmap ='rainbow')
         plt.show()
```



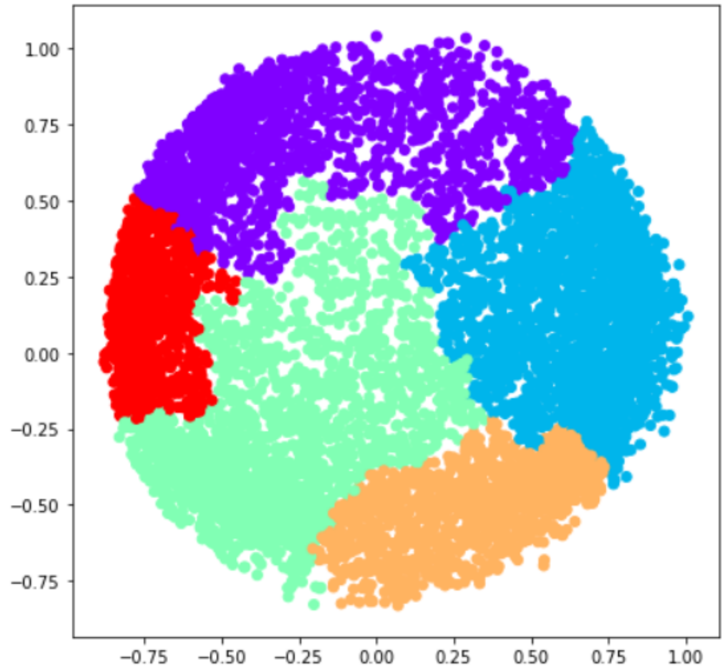
When K=4 i.e Agglomerative Clustering with 3 clusters and its scatter plot.

```
In [60]: ac4 = AgglomerativeClustering(n_clusters = 4)

         # Visualizing the clustering
         plt.figure(figsize =(7, 7))
         plt.scatter(principalDf['principal component 1'], principalDf['principal component 2'],
                    c = ac4.fit_predict(principalDf), cmap ='rainbow')
         plt.show()
```



When K=5 i.e Agglomerative Clustering with 3 clusters and its scatter plot.

In [62]:
```
ac5 = AgglomerativeClustering(n_clusters = 5)

# Visualizing the clustering
plt.figure(figsize =(7, 7))
plt.scatter(principalDf['principal component 1'], principalDf['principal component 2'],
            c = ac5.fit_predict(principalDf), cmap ='rainbow')
plt.show()
```



e) Evaluate different variations using Silhouette Scores and Visualize results with a bar chart.

Silhouette Score is a cluster validating coefficient. Silhouette Coefficient or silhouette score is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1.

Silhouette score is the difference between the point and the nearest cluster that the point is not part of the cluster. 1: Means clusters are well apart from each other and clearly distinguished. 0: Means clusters are indifferent, or we can say that the distance between clusters is not significant. -1: Means clusters are assigned in the wrong way.

In [64]:
```python
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([2, 3, 4, 5], silhouette_scores)
plt.xlabel('No of clusters', fontsize = 20)
plt.ylabel('S(i)', fontsize = 20)
plt.show()
```