# Machine Learning
# Assignment -6

**Student: Sowjanya Sunkavalli**
**ID: 700731896**

---

## 1. Calculate and find out clustering representations and dendrogram using Single, complete, and average link proximity function in hierarchical clustering technique.

For this solution we have considered the given table of points and pointed them in a graphical representation.

And also, Euclidian distance between each point is also given in the table.

a. Single Link Proximity Function:
   The distance between two clusters is the minimum distance between members of the two clusters.
   In this approach we consider the closest points and merge them and use the **minimum distance between members of the two** clusters to update the distance between cluster to other points and update the distance table and repeat these steps until we left with only two clusters.
b. Complete Link Proximity Function:
   The distance between two clusters is the maximum distance between members of the two clusters
   In this approach we consider the closest points and merge them and **use the distance between two clusters is the maximum distance between members of the two clusters** to update the distance between cluster to other points and update the distance table and repeat these steps until we left with only two clusters.

c. Average Link Proximity Function:
   The distance between two clusters is the average of all distances between members of the two clusters.
   In this approach we consider the closest points and merge them and use the **distance between two clusters is the average of all distances between members of the two clusters** to update the distance between cluster to other points and update the distance table and repeat these steps until we left with only two clusters.

   We have derived the Cluster Graphical representation and also the corresponding dendrograms for the given points.

   **Question 1 is handwritten and submitted as another PDF file**

## 2. Use CC_GENERAL.csv given in the folder and apply:

```
In [1]:
import pandas as pd # importing pandas as pd
import numpy as np # imporring numpy as np
from sklearn.preprocessing import StandardScaler,normalize # importing StandardScaler, normalize from sklearn.preprocessing
from sklearn.decomposition import PCA # importing PCA from sklearn.decomposition
from sklearn.metrics import silhouette_score # importing silhouette_score from sklearn.metrics
from matplotlib import pyplot as plt # importing pyplot as plt from matplotlib
import warnings # importing warinings
warnings.filterwarnings('ignore') # to ignore all the warnings
```

```
In [2]:
data = pd.read_csv('CC GENERAL.csv') # reading CC GENERAL.csv and storing it as data
data.head() # displaying first 5 rows of data
```

Out[2]:

| | CUST_ID | BALANCE | BALANCE_FREQUENCY | PURCHASES | ONEOFF_PURCHASES | INSTALLMENTS_PURCHASES | CASH_ADVANCE | PURCHASES_FREQ |
|---|---|---|---|---|---|---|---|---|
| 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 | |

Importing all the required packages and libraries. Which includes following modules
PCA module from sklearn.decomposition library, StandardScaler from sklearn.preprocessing library, train_test_split module from sklearn.model_selection library, LogisticRegression module from sklearn.linear_model library, accuracy_score from sklearn.metrics library, accuracy_score from sklearn.metrics library, Warnings library and filtering Warnings to Ignore

Reading CC_GENERAL.csv to data frame and assigning it to data using read_csv () method from Pandas.
Displaying the first five rows of the data frame using head () function.

### a. Preprocess the data by removing the categorical column and filling the missing values.

```
In [3]:
data.fillna(data.mean(),axis=0,inplace=True) # replacing all the NaN values with means of respective columns
X = data.drop(columns=['CUST_ID','TENURE']) # dropping 'TENURE','CUST_ID' columns and storing in X
X.head() # displaying first 5 rows of X
```

Out[3]:

| | 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 |

Replacing all the NaN values with the mean values of respective columns
Dropping 'TENURE' ,'CUST_ID' from data and storing it as X (features)
Displaying the first five rows of X using head() function.

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

```
In [4]: scaler = StandardScaler() # creating a StandardScalar to scale the data
        scaler.fit(X) # feeding the data to the scalar
        X_scaled_array = scaler.transform(X) # scaling the data using Standard Scalar function
        X_scaled = pd.DataFrame(X_scaled_array, columns = X.columns) # creating a new dataframe with scaled data
        X_scaled.head() # displaying first 5 rows of X_scaled
```

Out[4]:

| | BALANCE | BALANCE_FREQUENCY | PURCHASES | ONEOFF_PURCHASES | INSTALLMENTS_PURCHASES | CASH_ADVANCE | PURCHASES_FREQUENCY | ON |
|---|---|---|---|---|---|---|---|---|
| 0 | -0.731989 | -0.249434 | -0.424900 | -0.356934 | -0.349079 | -0.466786 | -0.806490 | |
| 1 | 0.786961 | 0.134325 | -0.469552 | -0.356934 | -0.454576 | 2.605605 | -1.221758 | |
| 2 | 0.447135 | 0.518084 | -0.107668 | 0.108889 | -0.454576 | -0.466786 | 1.269843 | |
| 3 | 0.049099 | -1.016953 | 0.232058 | 0.546189 | -0.454576 | -0.368653 | -1.014125 | |
| 4 | -0.358775 | 0.518084 | -0.462063 | -0.347294 | -0.454576 | -0.466786 | -1.014125 | |

```
In [5]: X_normal_array = normalize(X_scaled) # normalizing the scaled data using normalize() function
        X_normal = pd.DataFrame(X_normal_array,columns = X.columns) # creating a new dataframe with normalized data
        X_normal.head() # displaying first 5 rows of X_scaled
```

Out[5]:

| | BALANCE | BALANCE_FREQUENCY | PURCHASES | ONEOFF_PURCHASES | INSTALLMENTS_PURCHASES | CASH_ADVANCE | PURCHASES_FREQUENCY | ON |
|---|---|---|---|---|---|---|---|---|
| 0 | -0.315690 | -0.107575 | -0.183249 | -0.153937 | -0.150549 | -0.201314 | -0.347820 | |
| 1 | 0.221051 | 0.037731 | -0.131893 | -0.100260 | -0.127687 | 0.731894 | -0.343182 | |
| 2 | 0.127349 | 0.147556 | -0.030665 | 0.031013 | -0.129468 | -0.132945 | 0.361664 | |
| 3 | 0.020828 | -0.431402 | 0.098441 | 0.231699 | -0.192836 | -0.156386 | -0.430202 | |
| 4 | -0.153387 | 0.221496 | -0.197546 | -0.148479 | -0.194345 | -0.199565 | -0.433569 | |

Creating a StandaradScaler model and assigning it to scaler
Feeding the X(feature) data to the StandardScaler to get the scaled data
Creating a new dataframe with scaled X data and storing it as X_scaled
Displaying first five rows of scaled data
Feeding the scaled data to normalize () function to get normalized data
Creating a new dataframe with normalized X data and storing it as X_normal
Displaying the first five rows of normalized data

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

```
In [6]: pca2 = PCA(n_components=2) # creating PCA with no of components is 2
        principalComponents = pca2.fit_transform(X_normal) # giving the data to PCA
        # Creating new dataframe from the result obtained from PCA
        principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])
        final_data = pd.concat([principalDf, data[['TENURE']]], axis = 1) # concatinating the PCA values and target values
        final_data.head() # displaying first 5 rows of final_data
```

Out[6]:

| | 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 | -0.486616 | -0.080780 | 12 |
| 4 | -0.562175 | -0.474770 | 12 |

Creating the object for the PCA class with n_components = 2
Now feed the normalized data to the object to get the reduced features (2 principle components)
Creating a new data frame with the PCA data
Concat the target column to the PCA data and assign it to final_data
Now display the first five rows of the final_data

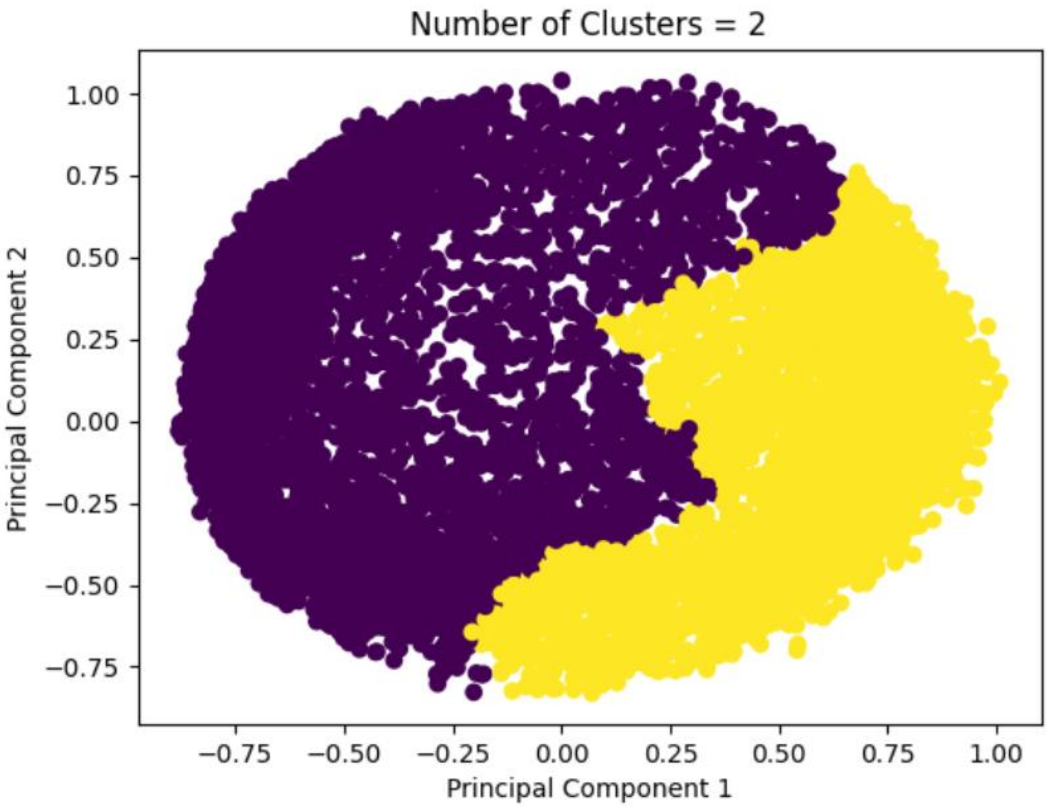
## d. Apply Agglomerative Clustering with k=2,3,4 and 5 on reduced features and visualize result for each k value using scatter plot

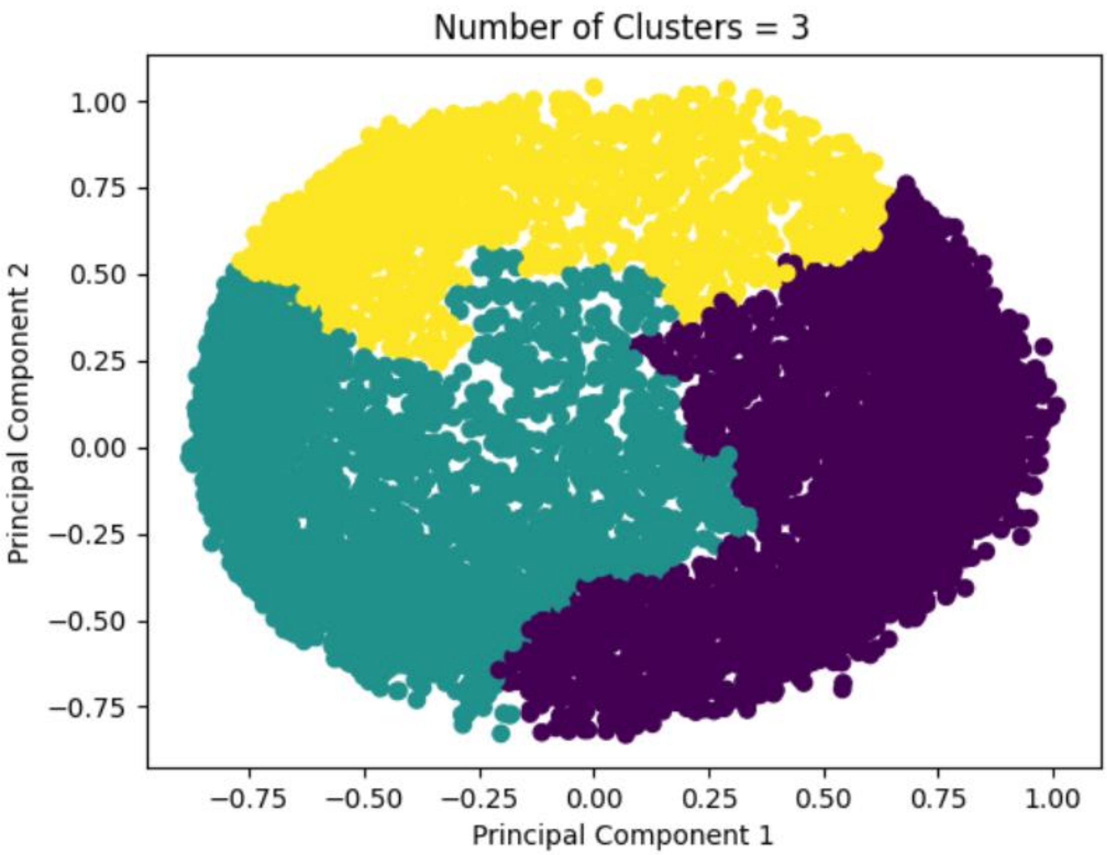
In [7]:
```
X_pca = final_data.drop('TENURE',axis=1) # dropping 'TENURE' from finalcc and storing as X_pca
y_pca = final_data['TENURE'] # considering 'TENURE' and storing as y_pca
from sklearn.cluster import AgglomerativeClustering # importing AgglomerativeClustering from sklearn.cluster
k = [2,3,4,5] # creating the list with number of clusters required
result = [] # creating an empty list
for i in k: # looping through k
    cluster = AgglomerativeClustering(i).fit(X_pca,y_pca) # feeding data to cluster with i value
    y_pred = cluster.fit_predict(X_pca) # predicting the target column from the X_pca
    result.append(y_pred) # appending the predicted values to result list
# plotting a scatter plot for each value of k
for i in result: # looping through each predicted result from the result list
    plt.scatter(X_pca['principal component 1'],X_pca['principal component 2'],c = i) # ploting scatter plot
    plt.xlabel('Principal Component 1') # labelling x-axis
    plt.ylabel('Principal Component 2') # labelling y-axis
    plt.title(f'Number of Clusters = {len(np.unique(i))}') # giving title for each scatter plot
    plt.show() # displaying the graph
```

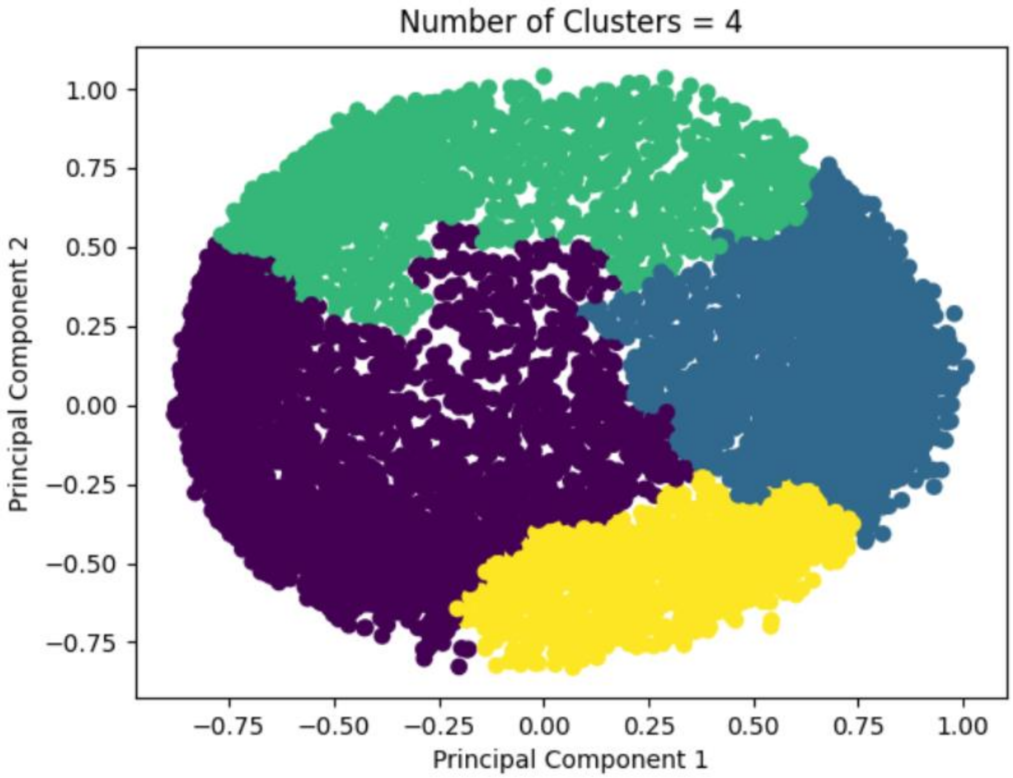
Dropping TENURE column from final_data and storing as X_pca
Considering the class column and storing as y_pca
Import AgglomerativeClustering from sklearn.cluster

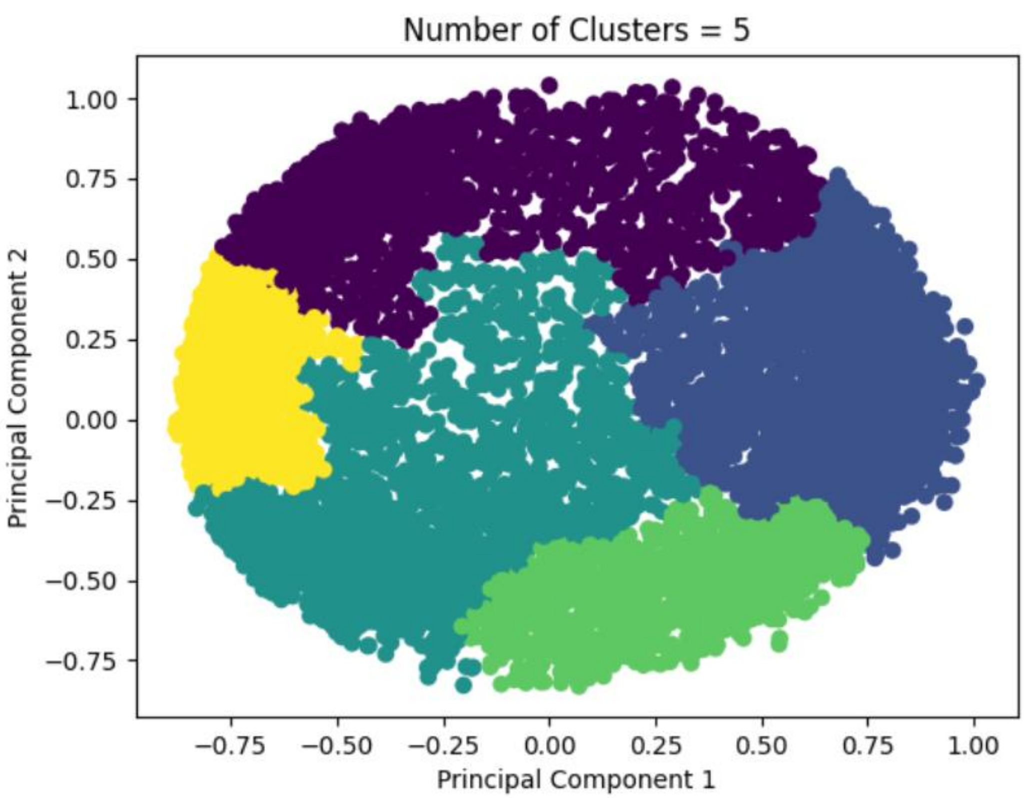
**Predicting the target column for each value of k:**
Create a list k with the cluster values i.e; 2,3,4,5 to iterate through
Create an empty list to store the result of for each cluster value
Iterate through the values of the k
Create an object of AgglomerativeClustering for each value of k and feed the X_pca, y_pca to it
Now predict the target column for each value of k and store it in result list

**Plotting the scatter plot for each value of k:**
Iterate through the result list which contains the target columns for each value of k
Plot the scatter plot between ‘Principal component 1’ and ‘Principal component 2’ and pass the target column as parameter to the ‘c’ to differentiate between clusters
Label the x-axis with ‘Principal Component 1’
Label the y-axis with ‘Principal Component 2’
Give the title to the graph as ‘Number of Clusters’
Show the graph


Number of Clusters = 2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Principal Component 2
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Principal Component 1


Number of Clusters = 3
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Principal Component 2
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Principal Component 1


Number of Clusters = 4
Principal Component 2
Principal Component 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


Number of Clusters = 5
Principal Component 2
Principal Component 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

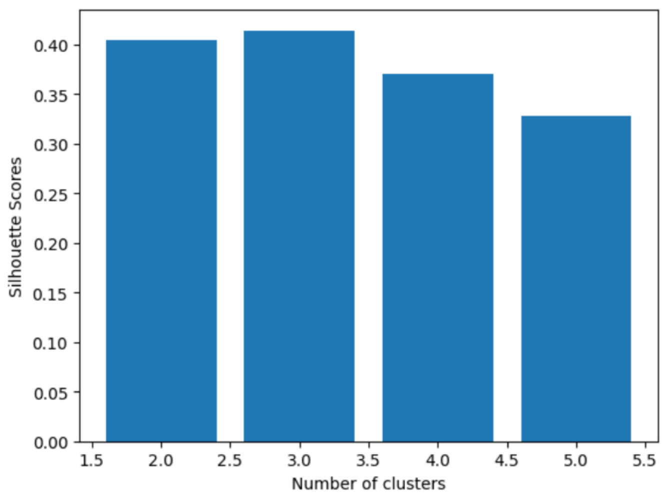
**e. Evaluate different variations using Silhouette Scores and Visualize results with a bar chart.**

```
In [8]:  silhouette_scores = [] # creating empty list
         for i in result: # looping through the result list
             score = silhouette_score(X_pca, i) # finding silhouette score for each value in result
             silhouette_scores.append(score) # appending the silhouette score to list
             print(f'Silhouette score for Number of Clusters = {len(np.unique(i))} is {score}' ) # printing the silhouette scores

         Silhouette score for Number of Clusters = 2 is 0.40418006820211444
         Silhouette score for Number of Clusters = 3 is 0.4142053214287074
         Silhouette score for Number of Clusters = 4 is 0.3698250853495397
         Silhouette score for Number of Clusters = 5 is 0.32839641176826617
```

Creating an empty list silhouette_scores to store the silhouette_score for each value of k
Iterate through the target columns which are stored in result list
Calculate the silhouette score for each target column in the result and store in silhouette_scores
Print all the silhouette_scores

```
In [9]:  plt.bar([2,3,4,5],silhouette_scores) # plotting bar chart between Number of clusters and silhouette scores
         plt.xlabel('Number of clusters') # labelling x-axis
         plt.ylabel('Silhouette Scores') # labelling y-axis
         plt.show() # displaying the graph
```




Plot the bar chart for the values of k and their respective silhouette scores
Label the x-axis with Number of Clusters
Label the y-axis with silhouette scores
Display the bar graph

---

**GIT Link:** [https://github.com/sunkavallisowjanya/MachineLearning_Assignment6](https://github.com/sunkavallisowjanya/MachineLearning_Assignment6)

**Video Link:** [https://youtu.be/RBfpz7r3-5U](https://youtu.be/RBfpz7r3-5U)