LAB EXPERIMENT NO. 05

NAME: Parth Parekh **SAP ID:60004200006**

BRANCH:Computer Engg **BATCH:A1**

Aim:

Implementation of Clustering algorithm using

1. k-means
2. Hierarchical (single/complete/average)

Perform the experiment in Python.

Read any dataset from UCI dataset repository

Part A:

Program using inbuilt functions.

Plot the clusters.

Plot dendrogram (for hierarchical)

Part B:

Program the algorithm from scratch.

Plot the clusters.

Part C:

Find optimum no. of cluster using elbow method.

Theory:

Clustering

It is basically a type of *unsupervised learning method*. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labeled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

K means Clustering

We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize those items into groups. To achieve this, we will use the kMeans algorithm; an unsupervised learning algorithm.
(It will help if you think of items as points in an n-dimensional space). The algorithm will categorize the items into k groups of similarity.

The algorithm works as follows:

1. First, we initialize k points, called means, randomly.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

Elbow Method

A fundamental step for any unsupervised algorithm is to determine the optimal number of clusters into which the data may be clustered. The Elbow Method is one of the most popular methods to

determine this optimal value of k.

We now demonstrate the given method using the K-Means clustering technique using the Sklearn library of python.

Hierarchical Clustering in Machine Learning

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as hierarchical cluster analysis or HCA. In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the dendrogram. Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

Working of Dendrogram in Hierarchical clustering

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

Code:

Part A, B & C

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.cluster.hierarchy as sch
import math
from sklearn.cluster import KMeans, AgglomerativeClustering
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import silhouette_score
import random

data = pd.read_csv('clustering.csv')
data = data.loc[:, ['ApplicantIncome','LoanAmount']]
X = data.values
sns.scatterplot(X[:,0], X[:, 1])

plt.xlabel('Income')
plt.ylabel('Loan')
plt.show()

wcss = []
score=[]
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
    if i!=1:
      silHoutteScore = silhouette_score(X, kmeans.labels_, metric='euclide
an')
      score.append(silHoutteScore)
print("WCSS")
print(wcss)
print("Silhouette Score from 2 to 10")
print(score)

sns.lineplot(x=range(1,11), y=wcss, marker='o')
plt.xlabel('Number of Clusters(k)')
plt.ylabel('WCSS')
```

```
plt.show()

def kMeansUsingInbuilt(k,X):
    # training the K-means model on a dataset
    kmeans = KMeans(n_clusters = k, init = 'k-means++', random_state = 42)
    # dependent variable y_predict to train the model.to know which cluste
r it belongs to
    y_kmeans = kmeans.fit_predict(X)
    # Calculate Silhoutte Score
    score = silhouette_score(X, kmeans.labels_, metric='euclidean')
    # Print the score
    print('Silhouette Score for n = 2: %.3f' % score)
    sns.scatterplot(X[:, 0], X[:, 1], c=kmeans.labels_, cmap="rainbow",s=1
00, marker=".")
    sns.scatterplot(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_
[:, 1], color="black",s=100, label="Centroids")
    plt.title("K-means inbuilt")
    plt.xlabel('Income')
    plt.ylabel('Loan')

kMeansUsingInbuilt(2,X)

def calculate_cost(X, centroids, cluster):
  sum = 0
  for i, val in enumerate(X):
      sum += np.sqrt((centroids[int(cluster[i]), 0]-
val[0])**2 + (centroids[int(cluster[i]), 1]-val[1])**2)
  return sum

def kMeansFromScratch(X,k):
  diff = 1
  cluster = np.zeros(X.shape[0])
  # select k random centroids
  random_indices = np.random.choice(len(X), size=k, replace=False)
  centroids = X[random_indices, :]
  while diff:
    # for each observation
    for i, row in enumerate(X):
      mn_dist = float('inf')
      # dist of the point from all centroids
      for idx, centroid in enumerate(centroids):
        d = np.sqrt((centroid[0]-row[0])**2 + (centroid[1]-row[1])**2)
        # store closest centroid
        if mn_dist > d:
          mn_dist = d
```

```
            cluster[i] = idx
        new_centroids = pd.DataFrame(X).groupby(by=cluster).mean().values
        # if centroids are same then leave
        if np.count_nonzero(centroids-new_centroids) == 0:
          diff = 0
        else:
          centroids = new_centroids
      return centroids, cluster

cost_list = []
for k in range(1, 11):
  centroids, cluster = kMeansFromScratch(X, k)
  # WCSS (Within cluster sum of square)
  cost = calculate_cost(X, centroids, cluster)
  cost_list.append(cost)

sns.lineplot(x=range(1,11), y=cost_list, marker='o')
print(cost_list)
plt.xlabel('k')
plt.ylabel('WCSS')
plt.show()

centroids, cluster = kMeansFromScratch(X, 2)
sns.scatterplot(X[:,0], X[:, 1],cmap='rainbow', hue=cluster)
sns.scatterplot(centroids[:,0], centroids[:, 1], s=100, color='black',labe
l="Centroids")
plt.title("K means Scratch")
plt.xlabel('Income')
plt.ylabel('Loan')
plt.show()

def hierarchicalInbuilt(k, method, X):
    hc = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean', l
inkage = method)
    y_hc = hc.fit_predict(X)
    # print(y_hc)
    plt.title('Hierarchichal inbuilt')
    plt.xlabel('Income ')
    plt.ylabel('Loan')
    sns.scatterplot(X[:, 0], X[:, 1],c=hc.labels_ ,s=100,cmap="rainbow", m
arker=".")
    plt.legend()
    plt.show()
    # Calculate Silhoutte Score
    score = silhouette_score(X, hc.labels_, metric='euclidean')
```

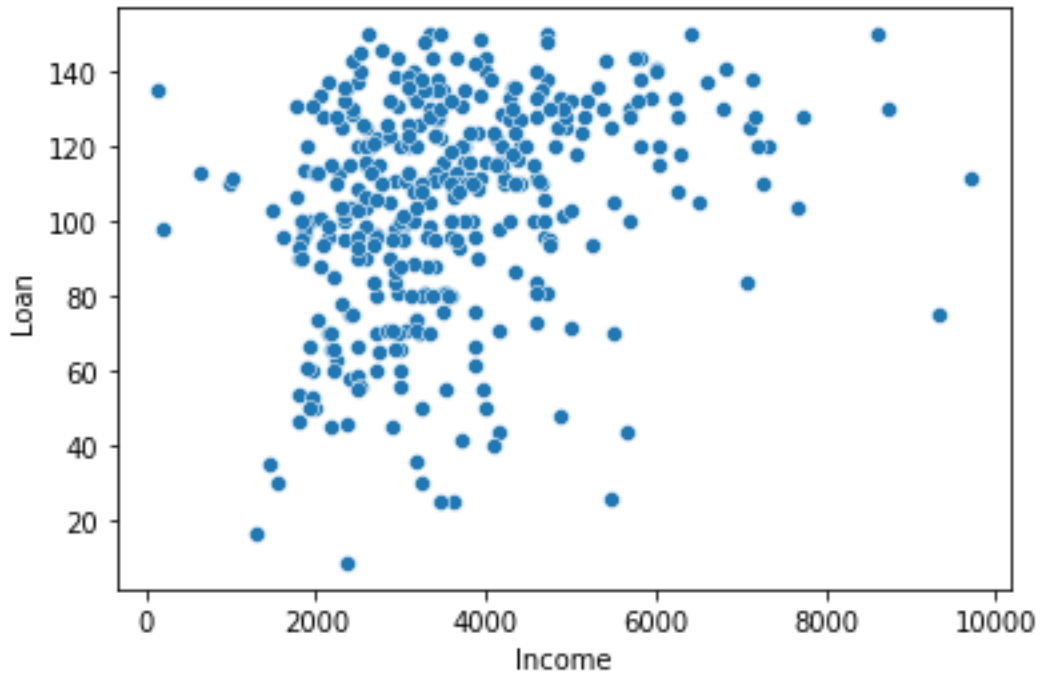
```
    # Print the score
    print('Silhouette Score: %.3f' % score)

def dendogram(link,X):
  dendrogram = sch.dendrogram(sch.linkage(X, method = link))
  plt.title('Dendrogram ')
  plt.xlabel('Customers')
  plt.ylabel('Euclidean distances')
  plt.show()

def hierarchicalFromScratch(k,linkage,X):
    clusters = [[i] for i in range(len(X))]
    matrix = [
        [math.sqrt(np.sum(np.square(np.subtract(X[i], X[j])))) for j in ra
nge(len(X))]
        for i in range(len(X))
    ]
    while len(clusters) != k:
        a = -1
        b = -1
        min_dist = 0
        if linkage == "ward":
            centers = [[0 for i in range(len(X[0]))] for j in range(len(cl
usters))]
            for i in range(len(clusters)):
                for point in clusters[i]:
                    for j in range(len(X[point])):
                        centers[i][j] += X[point][j]
                for j in range(len(X[0])):
                    centers[i][j] /= len(clusters[i])
            for i in range(len(clusters)):
                for j in range(i + 1, len(clusters)):
                    dist = (
                        len(clusters[i]) * len(clusters[j]) / (len(cluster
s[i]) + len(clusters[j]))
                    ) * (np.sum(np.square(np.subtract(centers[i], centers[
j]))))
                    if a == -1 or dist < min_dist:
                        min_dist = dist
                        a = i
                        b = j
        else:
            for i in range(len(clusters)):
                for j in range(i + 1, len(clusters)):
                    dist = 0
```

```
                    if linkage == "single":
                        dist = 999999999
                    for p1 in clusters[i]:
                        for p2 in clusters[j]:
                            if linkage == "complete":
                                dist = max(dist, matrix[p1][p2])
                            elif linkage == "single":
                                dist = min(dist, matrix[p1][p2])
                            else:
                                dist += matrix[p1][p2]
                    if linkage == "average":
                        dist /= len(clusters[i]) * len(clusters[j])
                    if a == -1 or dist < min_dist:
                        min_dist = dist
                        a = i
                        b = j

            clusters[a].extend(clusters.pop(b))
        labels = [-1 for _ in range(len(X))]
        for i in range(k):
            for p in clusters[i]:
                labels[p] = i
        plt.scatter(X[:, 0], X[:, 1], c=labels, cmap="rainbow", marker=".")
        plt.title("Hierarchichal from scratch")

print("Ward")
hierarchicalInbuilt(2, "ward", X)
# hierarchicalFromScratch(2,"ward",X)
dendogram("ward",X)

print("Single")
hierarchicalInbuilt(2, "single", X)
# hierarchicalFromScratch(2, "single",X)
dendogram("single",X)

print("Complete")
hierarchicalInbuilt(2, "complete", X)
# hierarchicalFromScratch(2, "complete",X)
dendogram("complete",X)

print("Average")
hierarchicalInbuilt(2, "average", X)
# hierarchicalFromScratch(2, "average",X)
dendogram("average",X)
```

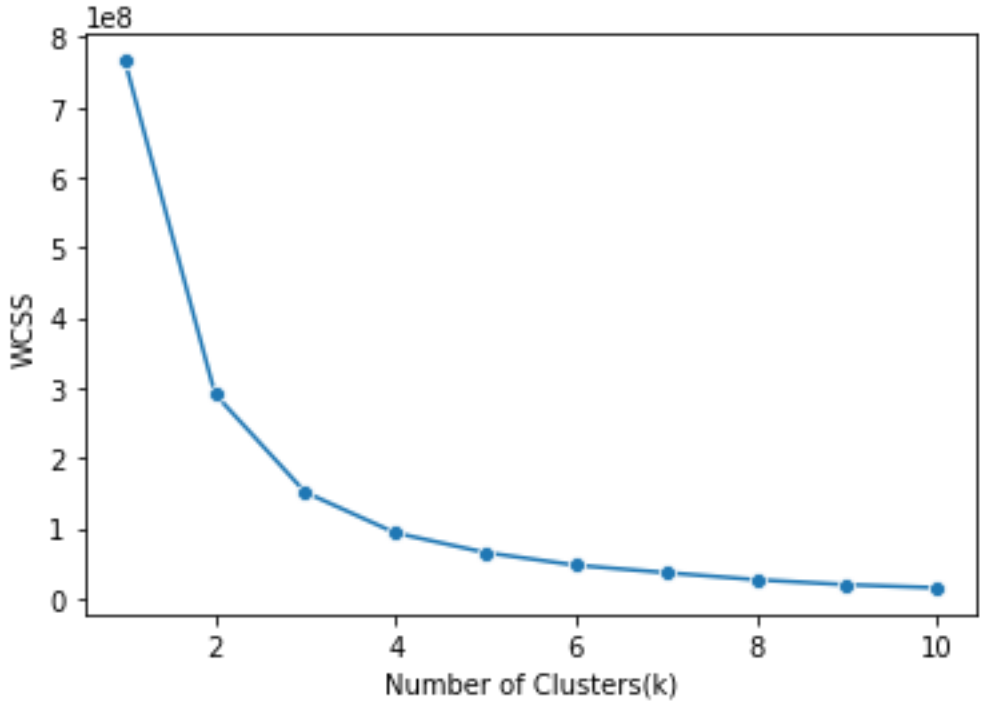
```
print("Average")
hierarchicalInbuilt(2, "average", X)
# hierarchicalFromScratch(2, "average",X)
dendogram("average",X)
```

Output

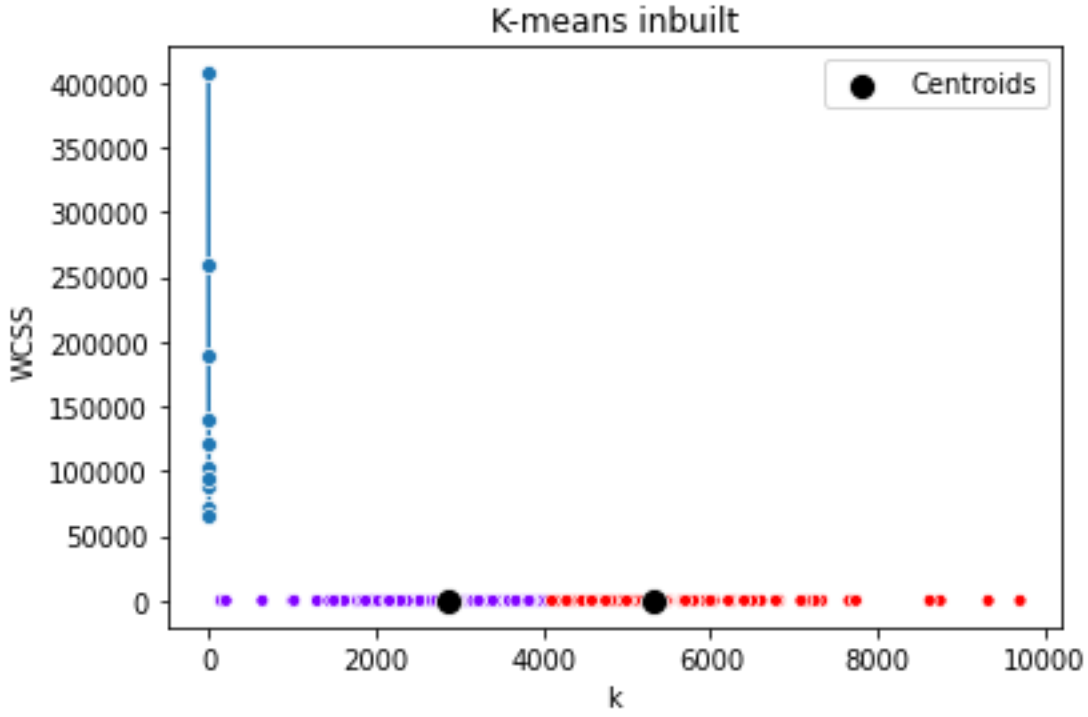
Part A, B & C

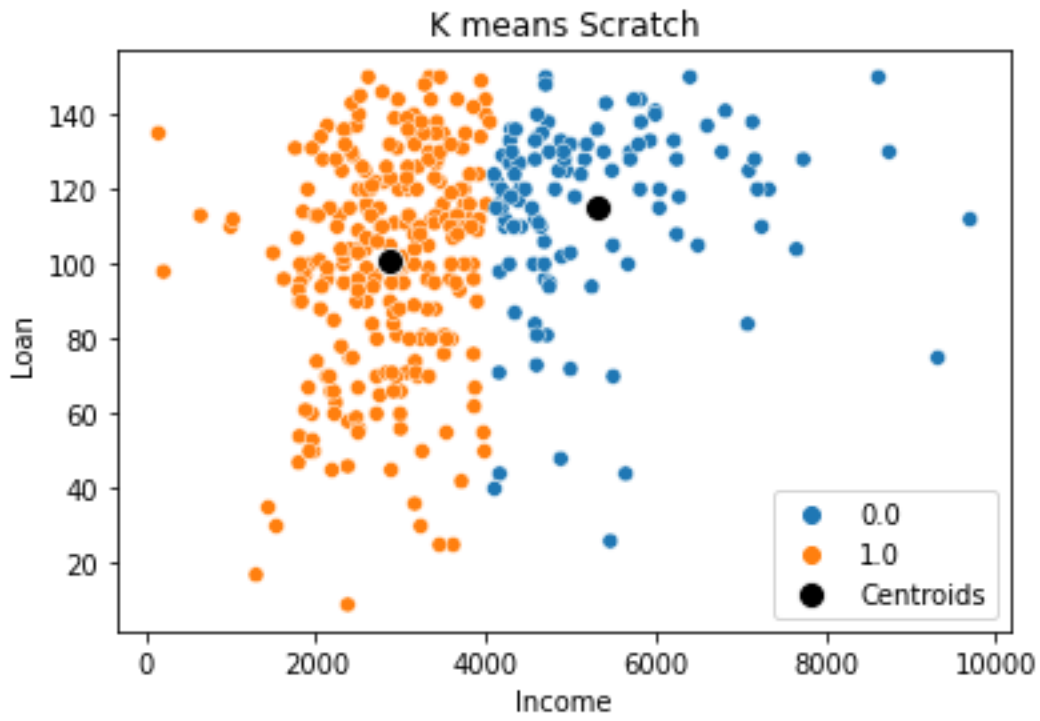



```
WCSS
[766336682.7979002, 291148680.62682676, 151285948.9451614, 93198687.75918041,
65202494.16480887, 47190441.552793756, 36603536.441917, 26565650.85894323,
19449449.28237724, 15047832.840574719]
Silhouette Score from 2 to 10
[0.5957289038117295, 0.5414122623500681, 0.5438061099276809,
0.5435435186572795, 0.5225239222596462, 0.5228633945949646,
0.5349821352011659, 0.5406718078411041, 0.5463055485024382]
```

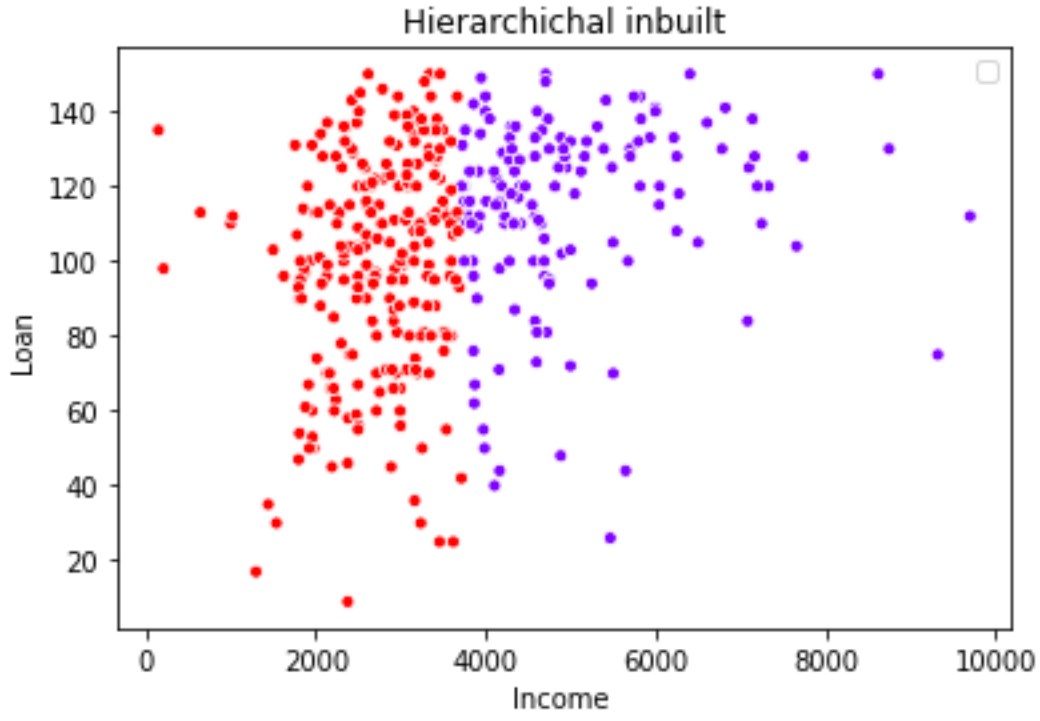



```
Silhouette Score for n = 2: 0.596
[407191.41309939686, 259651.61686873352, 188962.42849695045,
140863.66839342698, 120878.32615053724, 102261.33269541785,
89384.89778934214, 94960.8404972458, 71931.67062695007, 65706.44825331408]
```

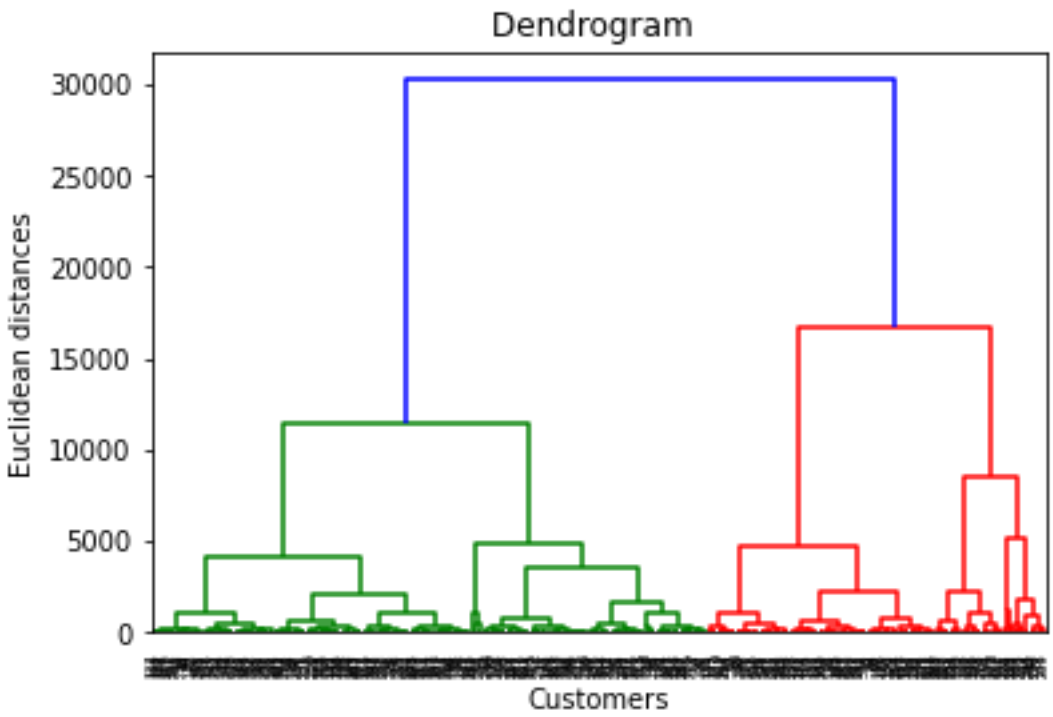




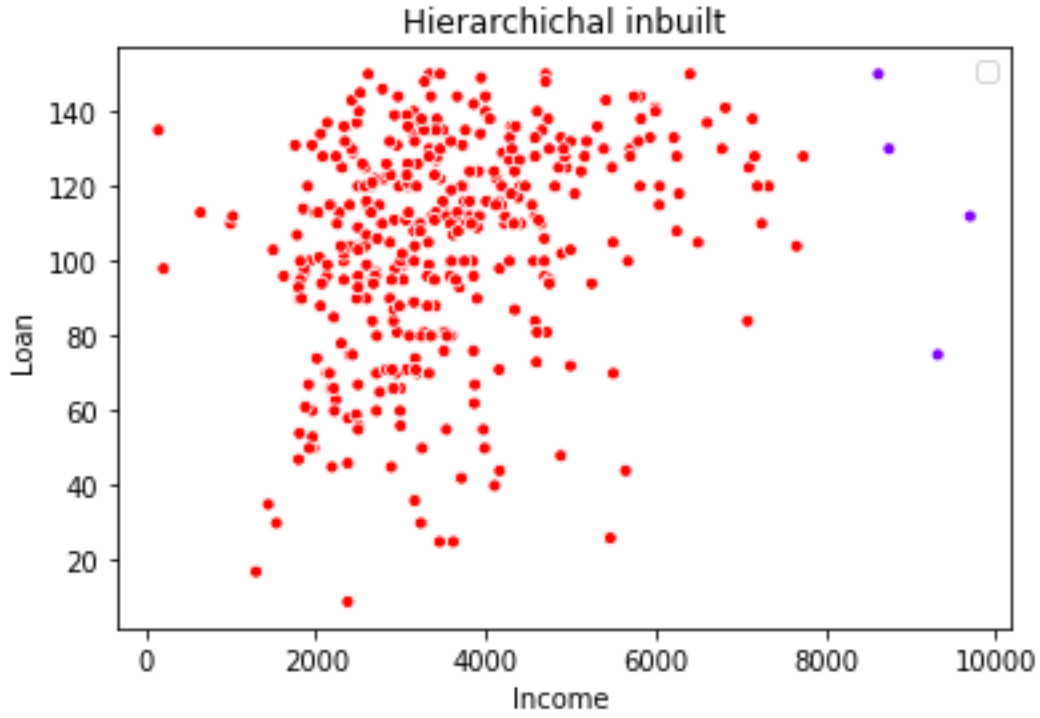

```
No handles with labels found to put in legend.
Ward
```



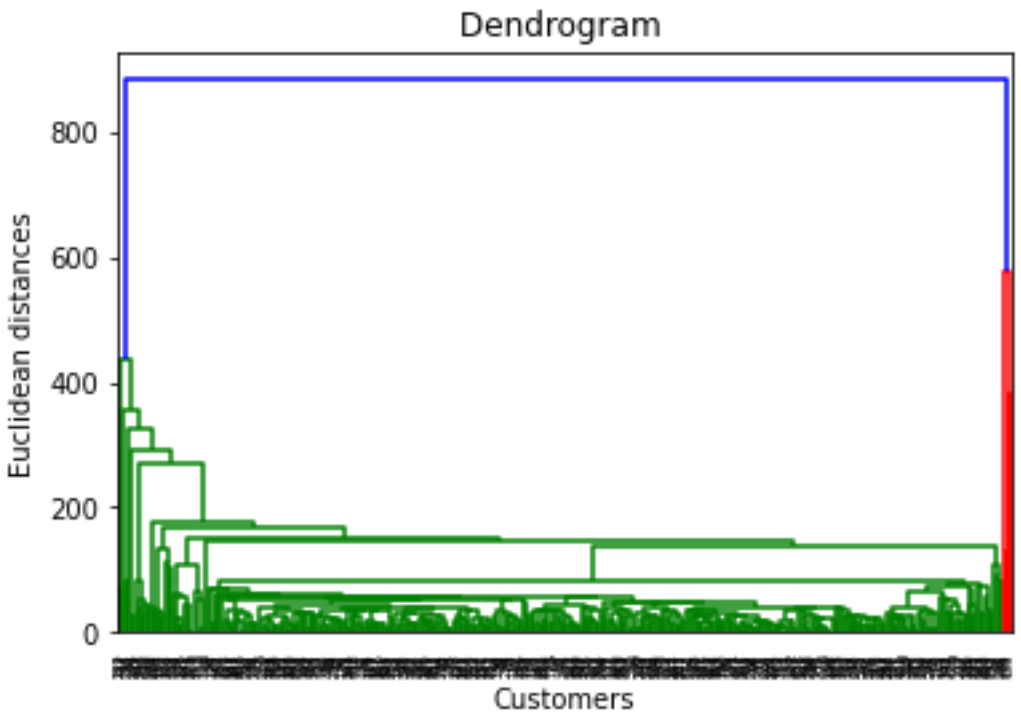

```
Silhouette Score: 0.557
```



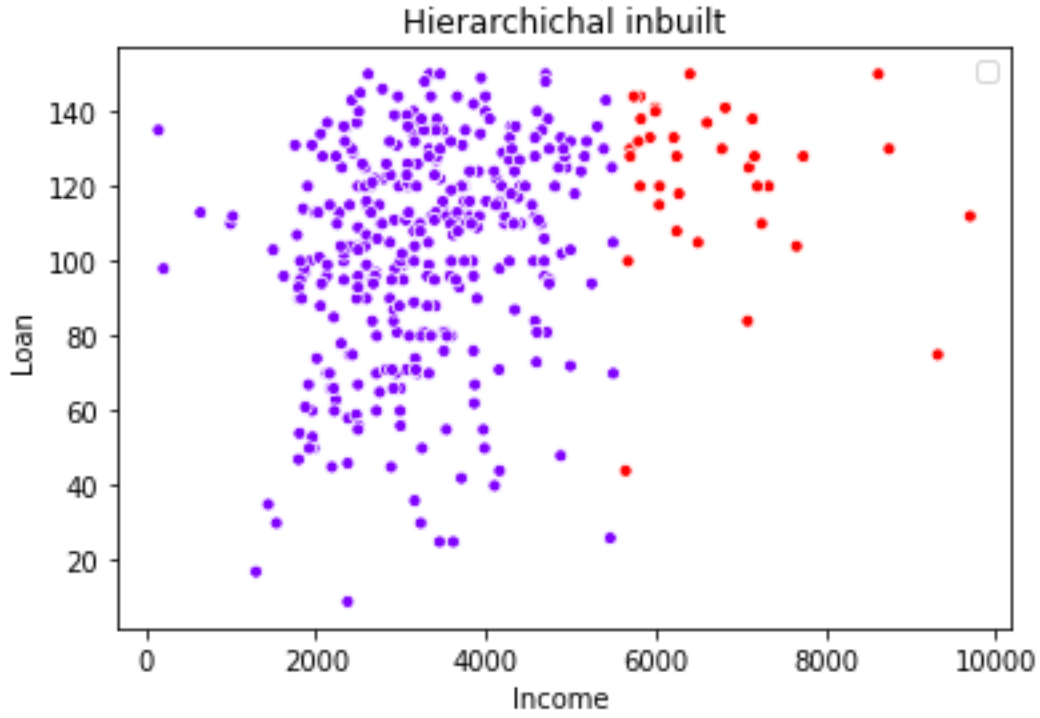

```
No handles with labels found to put in legend.
Single
```



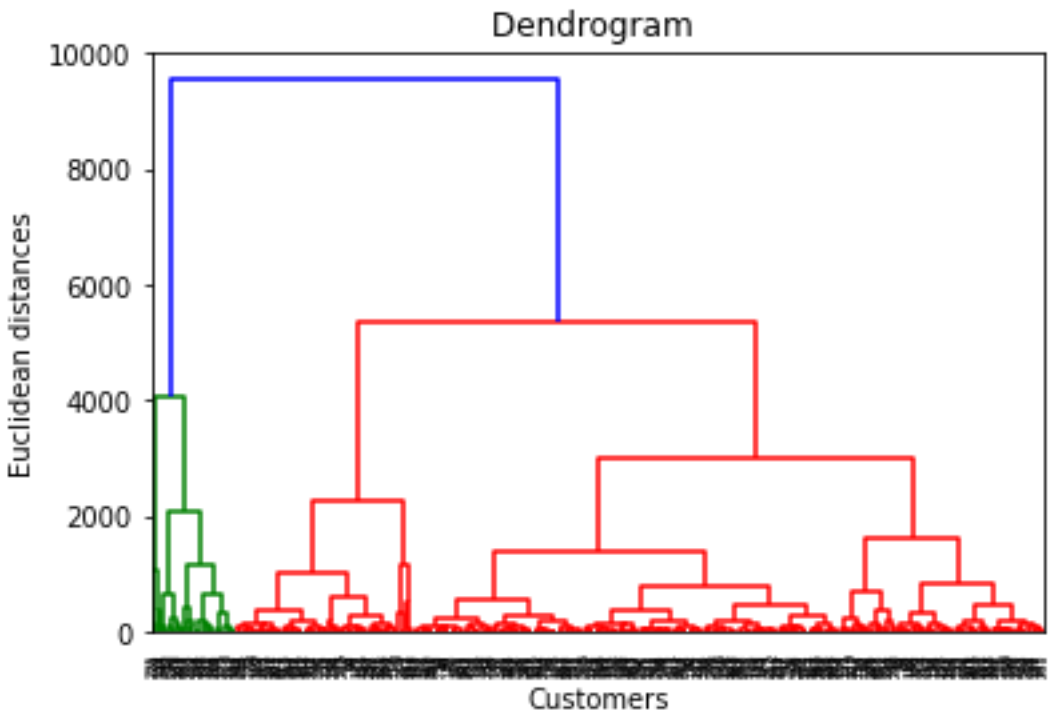

```
Silhouette Score: 0.702
```



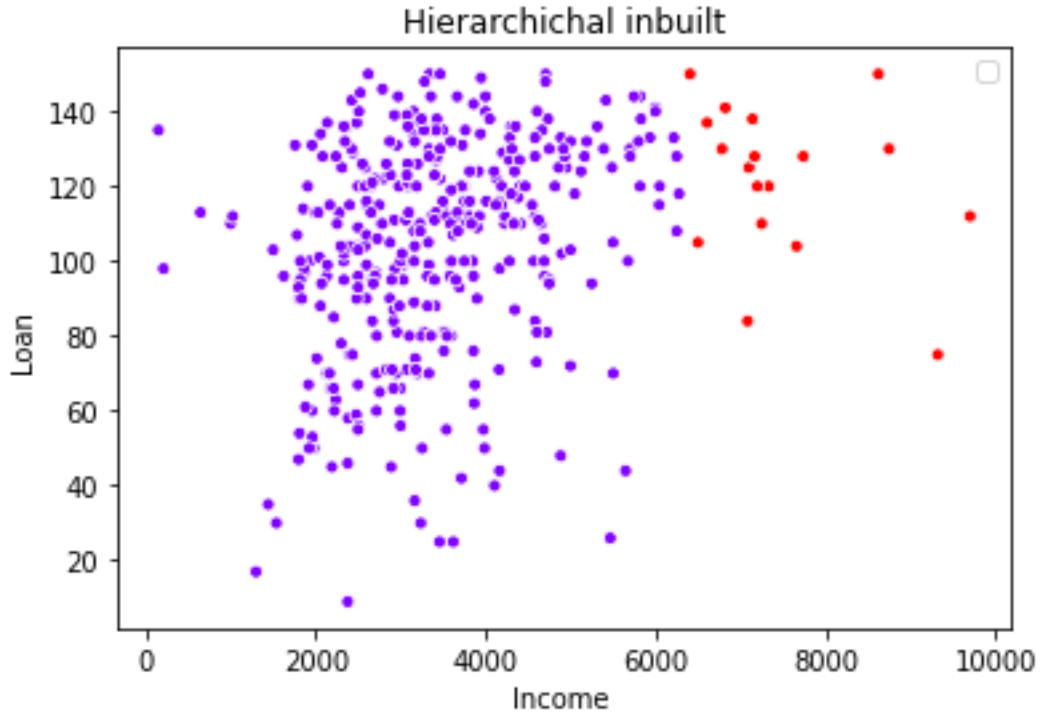

```
No handles with labels found to put in legend.
Complete
```



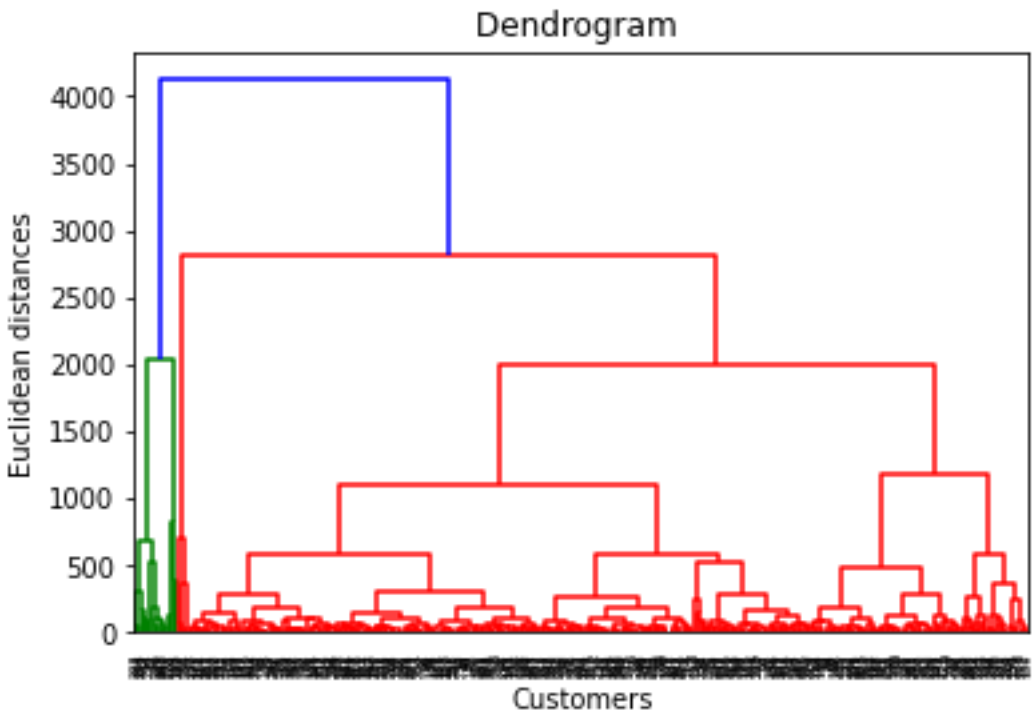

```
Silhouette Score: 0.634
```



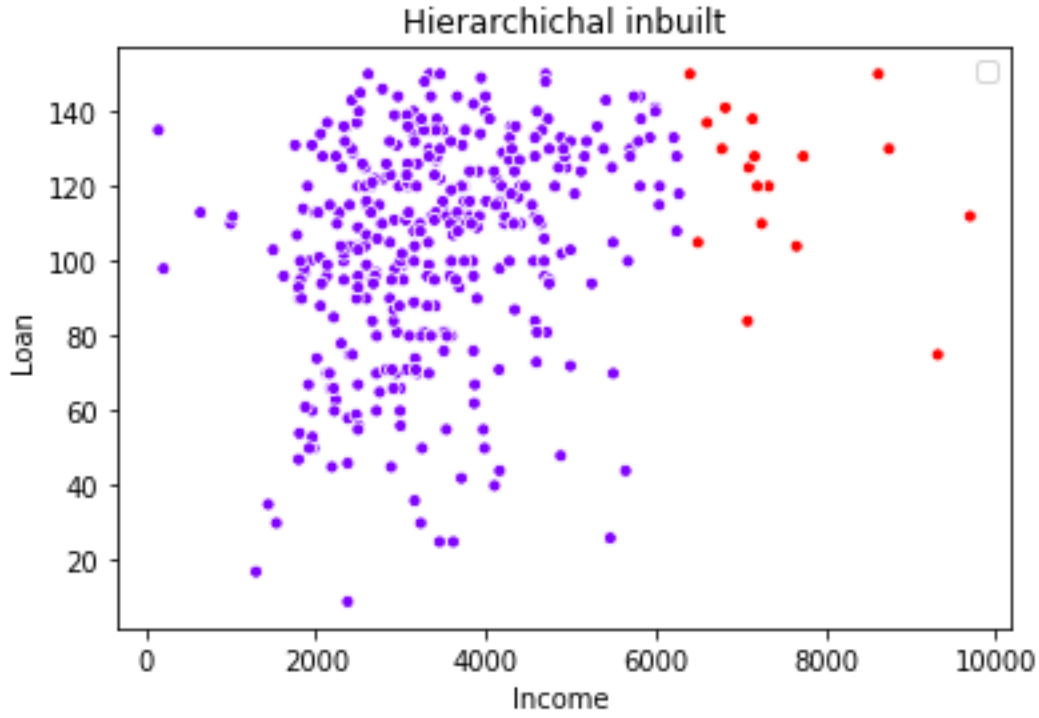

```
No handles with labels found to put in legend.
Average
```



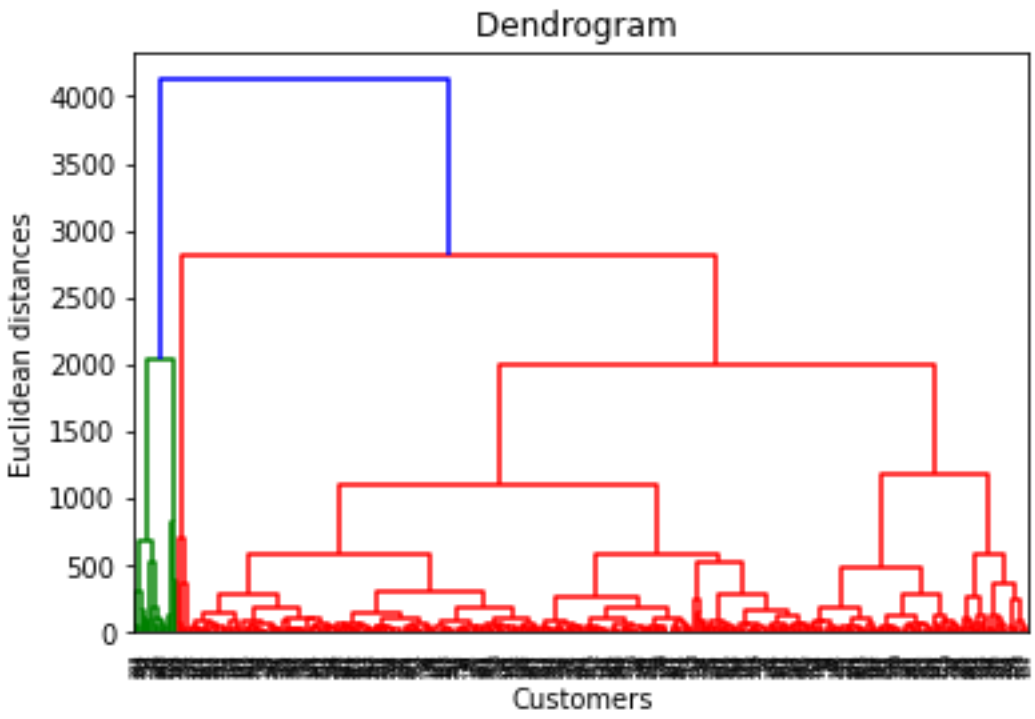

```
Silhouette Score: 0.647
```




```
No handles with labels found to put in legend.
Average
```




```
Silhouette Score: 0.647
```



Conclusion:

Thus, we have successfully implemented the k-means clustering and hierarchical clustering algorithm using python inbuilt functions and from scratch.We found Single Hierarchal silhouette score to be 0.702 which is the highest showing that clusters are dense and well-separated than other clusters.