VIVIAN JOSEPH

(20BCE0777)

Machine Learning

Lab DA-4

Clustering Algorithms

K-means Clustering:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
%matplotlib inline
import numpy as np
df = pd.read_csv('Mall_Customers.csv')
X = df.iloc[:, [3,4]].values
```

Make a distance function and a function to calculate WCSS

```
def distance(point1,point2):
    return np.sqrt(np.sum((point1-point2)**2))
```

```
def calculate_wcss(centroids, clusters, data):
    wcss = 0
    for i in range(centroids.shape[0]):
        cluster_points = data[clusters == i]
        centroid = centroids[i]
        distances = distance(cluster_points,centroid)**2
        wcss += np.sum(distances)
    return wcss
```

Make a function to calculate K Means

```
def kmeans(k):
    n=X.shape[0]
    # centroids = np.random.randn(k, 2)
    centroids = X[:k, :]
    # print(centroids)
    clusters=np.zeros(n)
    epochs=100
    for e in range(epochs):
        # print(centroids)
        for i in range(n):
            distances=[distance(X[i],centroid) for centroid in centroids]
            clusters[i]=np.argmin(distances)

        new_centroids=np.zeros((k,2))
        for i in range(k):
            points=[X[j] for j in range(n) if clusters[j]==i]
            # print(len(points))
            if len(points)>0:
                new_centroids[i]=np.mean(points,axis=0)
```

```
        if np.all(centroids==new_centroids):
            # print(e,"epochs")
            break
        else:
            centroids=new_centroids

    return clusters, centroids
```

Check WCSS for a range of values

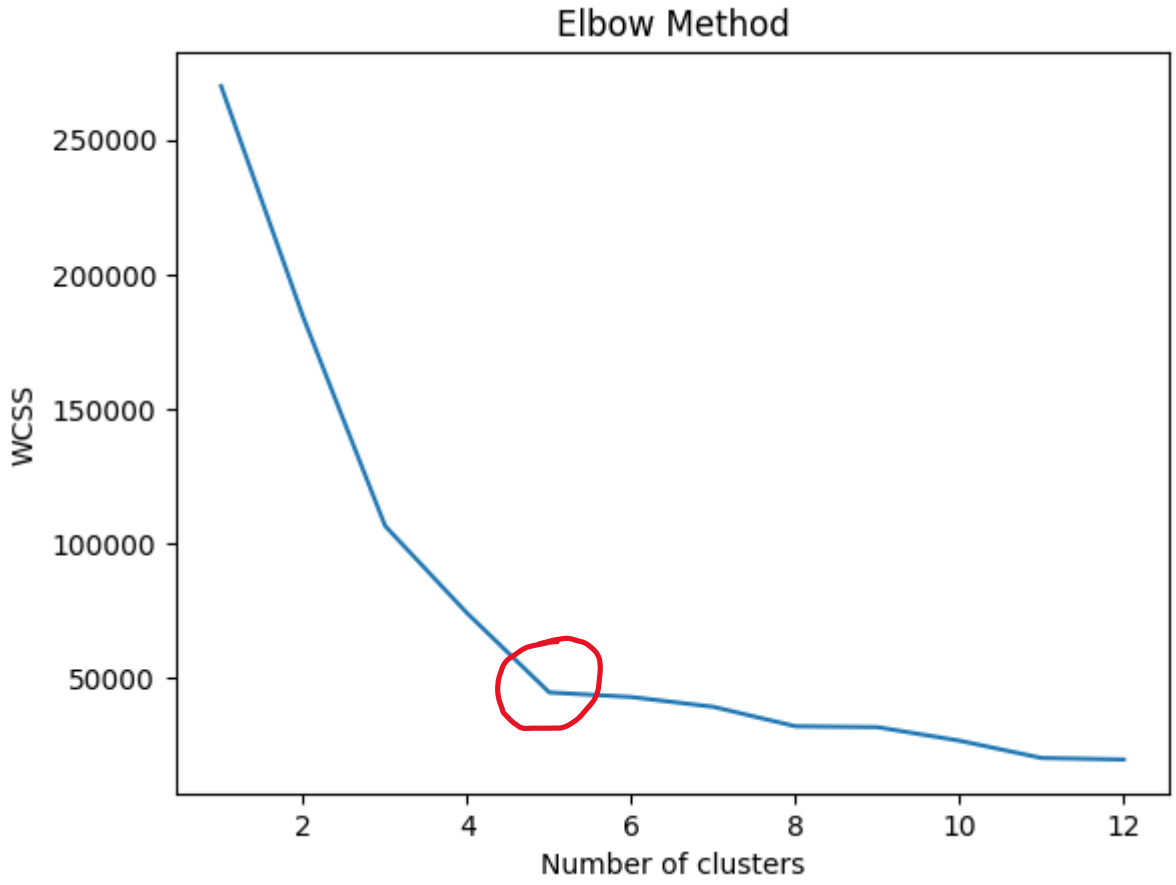
```
wcss=[]
for k in range(1,13):
    clusters, centroids=kmeans(k)
    print(k,end=": ")
    for i in range(k):
            points=[X[j] for j in range(200) if clusters[j]==i]
            print(len(points),end=" ")
    print()
    wcss.append(calculate_wcss(centroids,clusters,X))
```

Checking what is the distribution of points in each cluster for each k

```
1: 200
2: 136 64
3: 123 39 38
4: 101 22 38 39
5: 23 22 35 39 81
6: 23 8 35 15 80 39
7: 15 8 12 15 76 39 35
8: 15 6 12 17 76 28 35 11
9: 14 6 8 17 76 28 5 11 35
10: 10 6 8 15 43 42 5 11 32 28
11: 11 6 3 15 47 36 9 11 24 28 10
12: 11 7 3 9 47 35 9 6 24 28 10 11
```

Plot the elbow graph

```
plt.plot(range(1, 13), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



Calculate kmeans clusters for **k=5**

```
clusters, centroids=kmeans(5)
print(centroids)
```

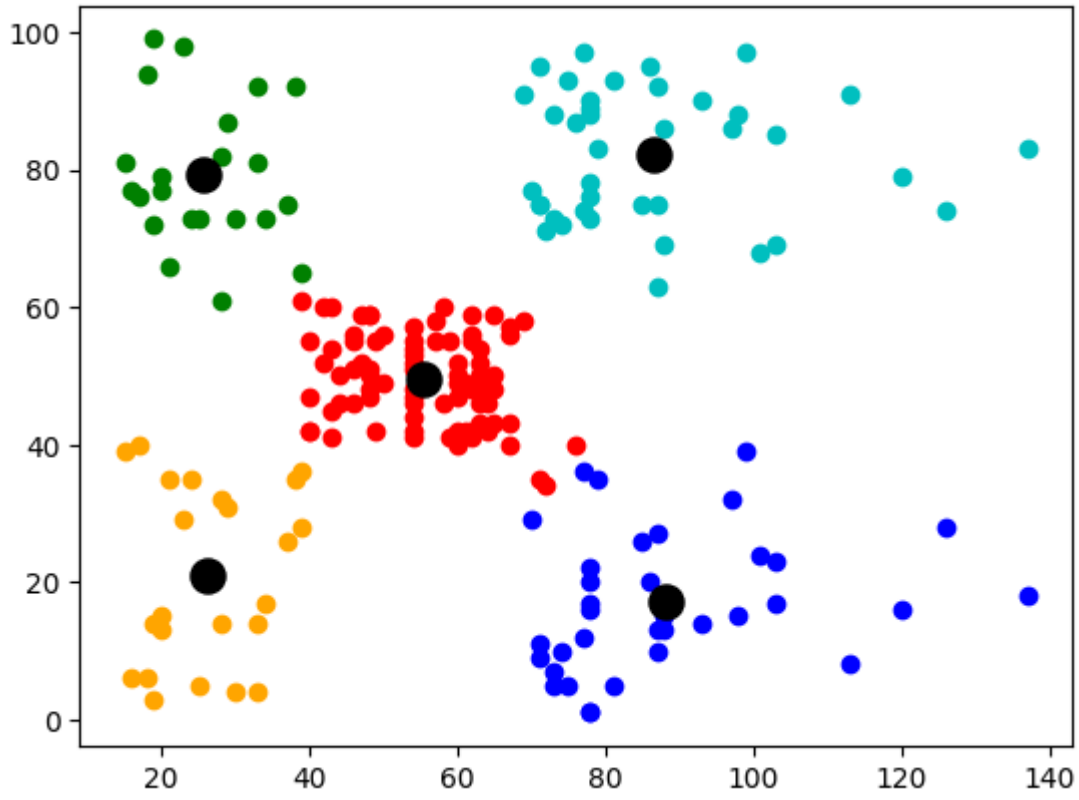
```
[[26.30434783 20.91304348]
 [25.72727273 79.36363636]
 [88.2        17.11428571]
 [86.53846154 82.12820513]
 [55.2962963  49.51851852]]
```

```
print("WCSS:",calculate_wcss(centroids,clusters,X))
```

```
WCSS: 44448.45544793371
```

Plot the clusters and respective points

```
colors=['orange','g','b','c','r']
for i in range(5):
    for j in range(200):
        if clusters[j]==i:
            plt.scatter(X[j,0], X[j,1], c=colors[i], cmap='rainbow')
plt.scatter(centroids[:,0], centroids[:, 1], s=150, c='k')
plt.show()
```



K-modes Clustering:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
%matplotlib inline
import numpy as np
```

To get classes for k-modes, convert numerical data to categorial data

```
df = pd.read_csv('Mall_Customers.csv')

age_bins = [0, 15, 30, 45,60,80, 100]
age_labels = [1, 2, 3,4,5,6]
df['Age Class'] = pd.cut(df['Age'], bins=age_bins, labels=age_labels)

income_bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,200]
income_labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11]
df['Income Class'] = pd.cut(df['Annual Income (k$)'], bins=income_bins,
labels=income_labels)

X = df.iloc[:, [5,6]].values
```

Make a function to calculate k-mode clusters for any k

```
def kmode(k):
    epochs=100

    rng = np.random.default_rng()
    centroids = rng.choice(X, size=k, replace=False)
    n_samples, n_features = X.shape
    distances = np.zeros((n_samples, k))

    for e in range(epochs):
        for i in range(n_samples):
            data=X[i]
            for j in range(k):
                cen=centroids[j]
                dis=0
                for d in range(n_features):
                    if data[d]!=cen[d]:
                        dis+=1
                distances[i, j] = dis

        labels = np.argmin(distances, axis=1)

        for j in range(k):
            cluster_j = X[labels == j]
```

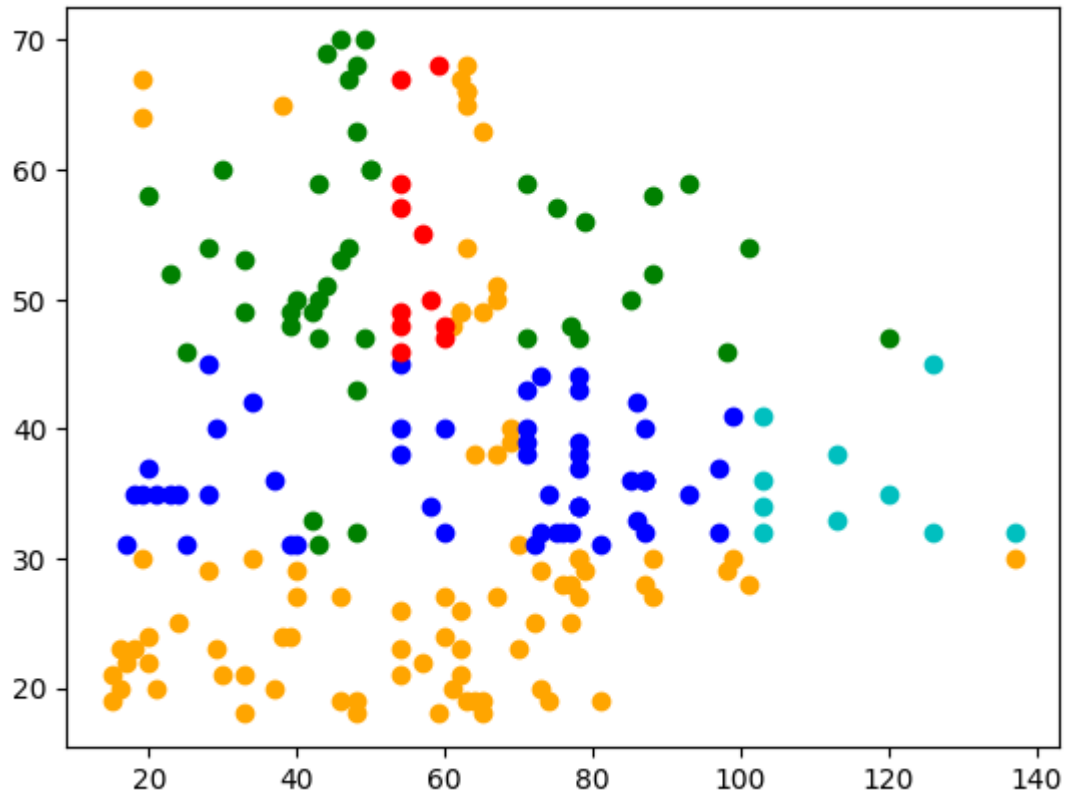
```
            centroids[j]  = np.apply_along_axis(lambda x: np.bincount(x).argmax(), axis=0,
arr=cluster_j)

    return labels, centroids
```

We got **K=5,** calculate the clusters for these

```
clusters, centroids=kmode(5)

colors=['orange','g','b','c','r']
for i in range(5):
    for j in range(200):
        if clusters[j]==i:
            plt.scatter(df.iloc[j, 3], df.iloc[j, 2], c=colors[i], cmap='rainbow')
# plt.scatter(centroids[:,0], centroids[:, 1], s=150, c='k')
plt.show()
```



This data aren't look as impressive on a graph as the it is calculated on mode and not Euclidian distances

Hierarchial Clustering:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
%matplotlib inline
import numpy as np
```

```
df = pd.read_csv('Mall_Customers.csv')
X = df.iloc[:, [3,4]].values
```

Function to calculate Euclidian distance

```
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))
```

Initialize distance vector

```
dist_matrix = np.zeros((len(X), len(X)))
for i in range(len(X)):
    for j in range(i+1, len(X)):
        dist_matrix[i][j] = euclidean_distance(X.iloc[i, :], X.iloc[j, :])
        dist_matrix[j][i] = dist_matrix[i][j]
```

Finf=d the closest clusters

```
def find_closest_clusters(dist_matrix):
    min_dist = np.inf
    closest_clusters = ()
    for i in range(len(dist_matrix)):
        for j in range(i+1, len(dist_matrix)):
            if dist_matrix[i][j] < min_dist:
                min_dist = dist_matrix[i][j]
                closest_clusters = (i, j)
    return closest_clusters
```

Merge the found clusters

```
def merge_clusters(data, dist_matrix, cluster1, cluster2):
    new_cluster = np.mean(np.vstack((data.iloc[cluster1, :], data.iloc[cluster2, :])),
axis=0)
    new_dist_row = np.zeros((1, len(dist_matrix)))
    for i in range(len(dist_matrix)):
        if i != cluster1 and i != cluster2:
            new_dist_row[0][i] = np.mean([dist_matrix[cluster1][i],
dist_matrix[cluster2][i]])
    new_dist_row[0][cluster1] = np.inf
```

```
    new_dist_row[0][cluster2] = np.inf
    new_dist_col = np.concatenate((dist_matrix, np.zeros((1, len(dist_matrix)))), axis=0)
    new_dist_col = np.concatenate((new_dist_col, np.zeros((len(dist_matrix)+1, 1))),
axis=1)
    new_dist_col[-1][:-1] = new_dist_row
    new_dist_col[:-1, -1] = new_dist_row.T
    new_dist_col[-1][-1] = np.inf
    new_data = data.drop([cluster1, cluster2], axis=0)
    new_data.loc[len(new_data)] = new_cluster
    return new_data, new_dist_col
```

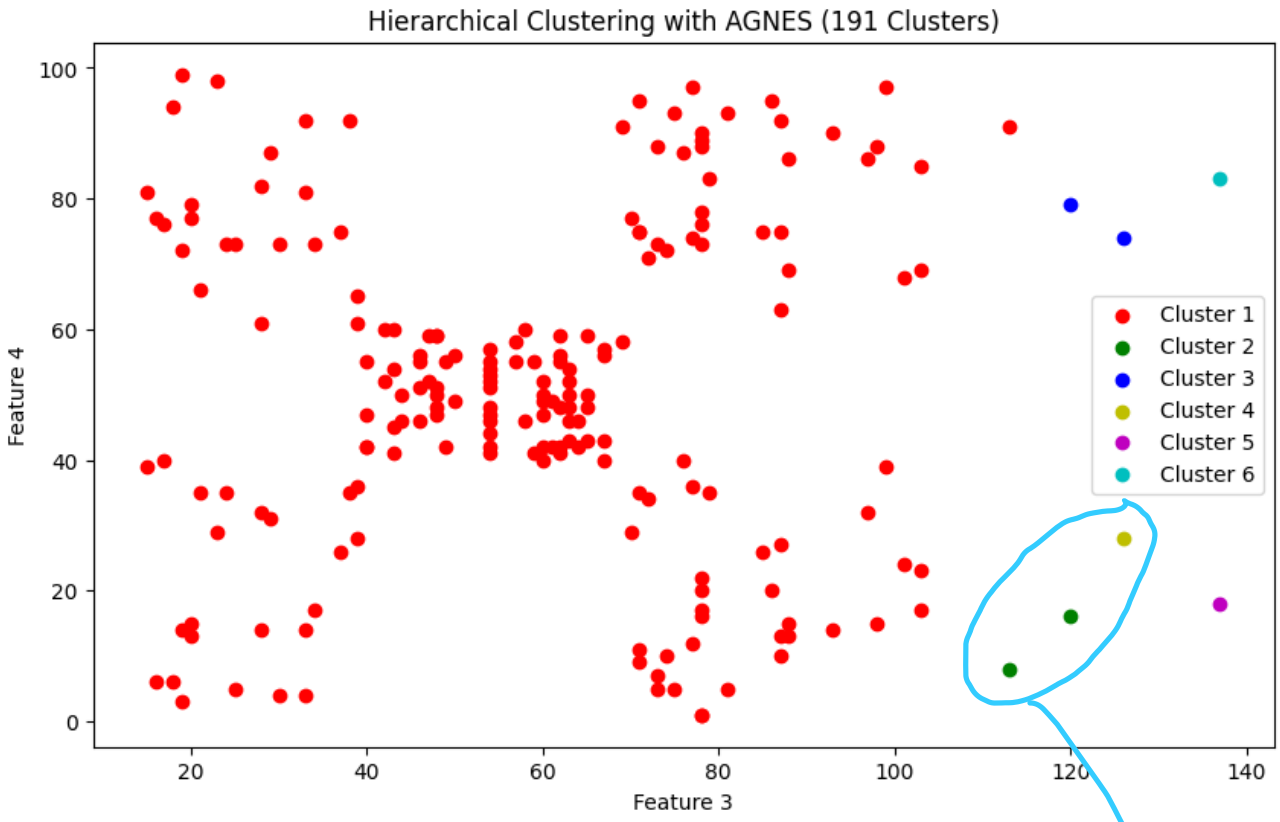
make a list of all clusters at every given point

```
clusters = list(range(len(X)))
while len(clusters) > 1:
    closest_clusters = find_closest_clusters(dist_matrix)
    cluster1, cluster2 = closest_clusters[0], closest_clusters[1]
    X, dist_matrix = merge_clusters(X, dist_matrix, cluster1, cluster2)
    clusters.remove(cluster1)
    clusters.remove(cluster2)
    clusters.append(len(X)-1)
```

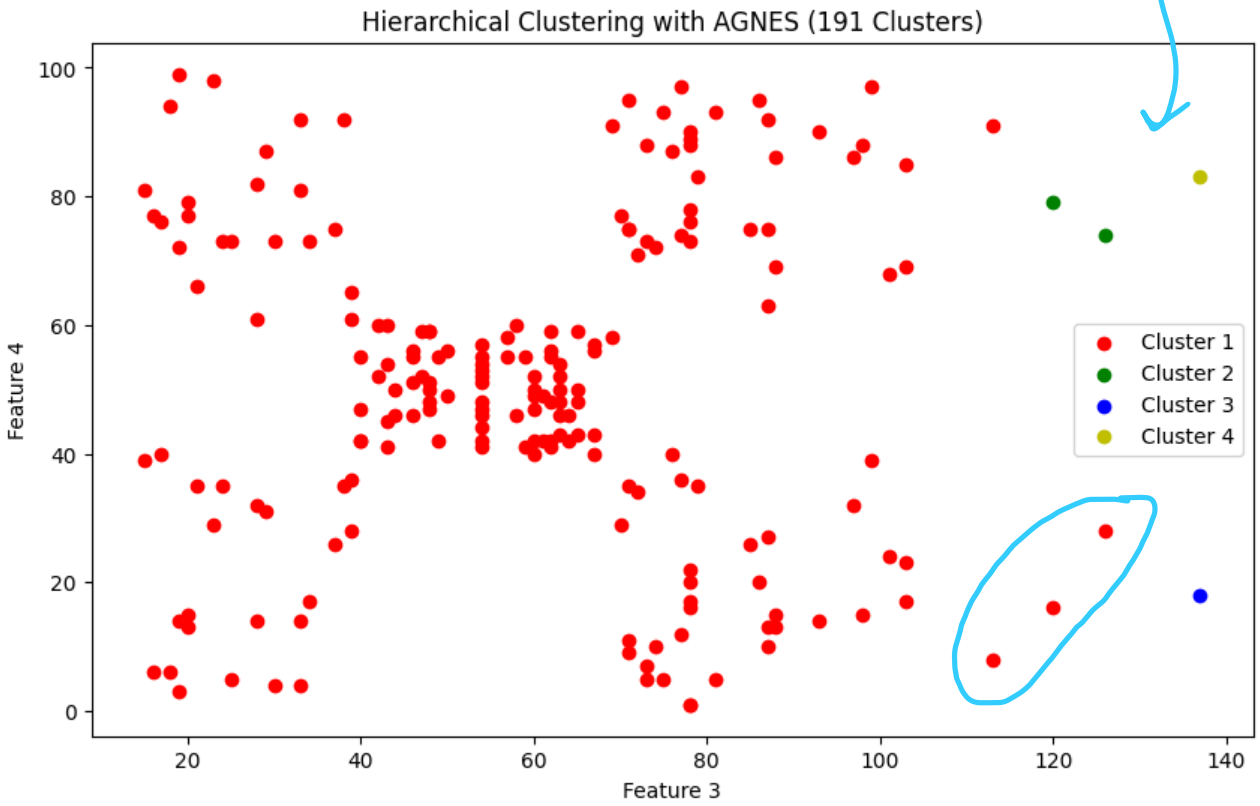
Let’s look at clusters formed at any given point

```
clusters = agnes(X, 6)
plt.figure(figsize=(10, 6))
colors = ['r', 'g', 'b', 'y', 'm', 'c']
for i, cluster in enumerate(clusters):
    plt.scatter(X[cluster, 0], X[cluster, 1], color=colors[i%6], label=f'Cluster {i+1}')
plt.legend()
plt.title(f'Hierarchical Clustering with AGNES ({k} Clusters)')
plt.xlabel('Feature 3')
plt.ylabel('Feature 4')
plt.show()
```

For 6 clusters formed out of total data points



Cluster 2 and cluster 4 joined with cluster 1



For 4 clusters formed out of total data points