Experiment 07 - Clustering

| Roll No. |  |
| --- | --- |
| Name |  |
| Class | D15A |
| Subject | DS using Python Lab |
| LO Mapped | LO2: Understand the concept of the Data Science process and associated terminologies to solve real-world problems.  LO4: Apply the different unsupervised machine learning algorithms like Clustering or Association to solve the problems. |
|  |  |

**Aim**:

To implement the Clustering algorithm using Python.

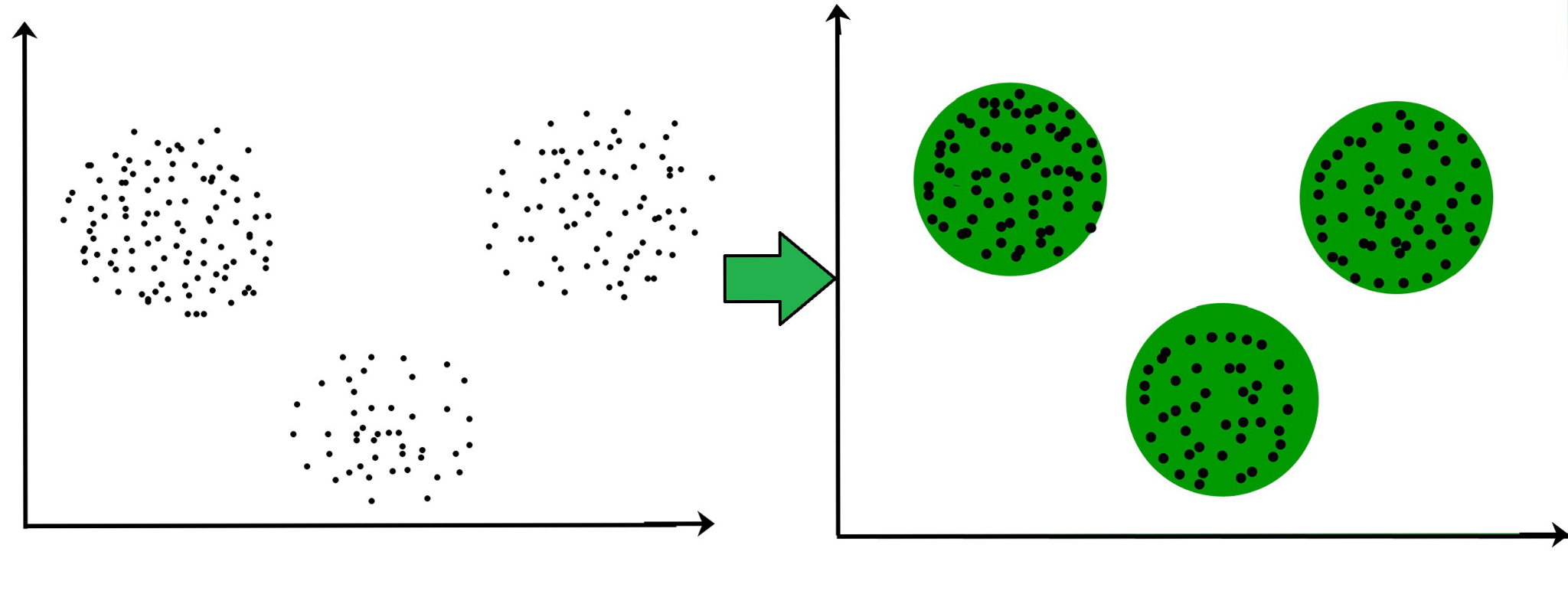
**Clustering Algorithm**:

**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 labelled 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 than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

For ex– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



**Why Clustering?**

Clustering is very much important as it determines the intrinsic grouping among the unlabelled data present. There are no criteria for good clustering. It depends on the user, what are the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding “natural clusters” and describe their unknown properties (“natural” data types), in finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions that constitute the similarity of points and each assumption make different and equally valid clusters.

**Applications of Clustering in different fields**

1. Marketing: It can be used to characterize & discover customer segments for marketing purposes.
2. Biology: It can be used for classification among different species of plants and animals.
3. Libraries: It is used in clustering different books on the basis of topics and information.
4. Insurance: It is used to acknowledge the customers, their policies and identifying the frauds.
5. City Planning: It is used to make groups of houses and to study their values based on their geographical locations and other factors present.
6. Earthquake studies: By learning the earthquake-affected areas we can determine the dangerous zones.

**Types of Clustering**

Broadly speaking, clustering can be divided into two subgroups :

1. Hard Clustering: In hard clustering, each data point either belongs to a cluster completely or not. For example, in the above example each customer is put into one group out of the 10 groups.
2. Soft Clustering: In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned. For example, from the above scenario each customer is assigned a probability to be in either of 10 clusters of the retail store.

**Clustering Algorithms**

When choosing a clustering algorithm, you should consider whether the algorithm scales to your dataset. Datasets in machine learning can have millions of examples, but not all clustering algorithms scale efficiently. Many clustering algorithms work by computing the similarity between all pairs of examples. This means their runtime increases as the square of the number of examples n, denoted as O(n2) in complexity notation. O(n2) algorithms are not practical when the number of examples are in millions.

1. Density-Based Methods:

These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), etc.

1. Hierarchical Based Methods:

The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category

1. Agglomerative (bottom-up approach)
2. Divisive (top-down approach)

examples CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies), etc.

1. Partitioning Methods:

These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example K-means, CLARANS (Clustering Large Applications based upon Randomized Search), etc.

1. Grid-based Methods:

In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects, for example STING (Statistical Information Grid), wave cluster, CLIQUE (CLustering In Quest), etc.

**K-Means Clustering Algorithm**

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. It is an iterative algorithm that divides the unlabeled dataset into K different clusters in such a way that each dataset belongs to only one group that has similar properties. Here K defines the number of predefined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

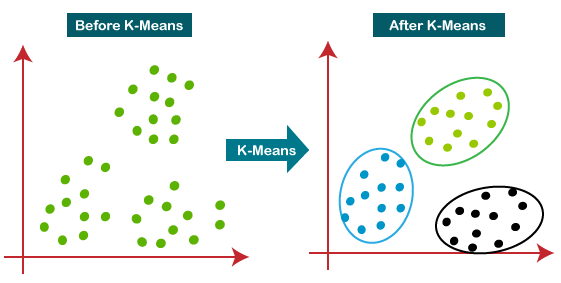
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters. The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

1. Determines the best value for K center points or centroids by an iterative process.
2. Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has data points with some commonalities, and it is away from other clusters.



**How does the K-Means Algorithm Work?**

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be different from the input dataset).

Step-3: Assign each data point to its closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid in each cluster.

Step-5: Repeat the third step, which means assigning each data point to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

**Python Library Function Used**:

Python library used for classification is scikit-learn

**sklearn.cluster.KMeans**

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified. It scales well to a large number of samples and has been used across a large range of application areas in many different fields.

The k-means algorithm divides a set of samples into disjoint clusters, each described by the mean

of the samples in the cluster. The means are commonly called the cluster “centroids”; note that they are not, in general, points from, although they live in the same space.

Parameters:

1. **n\_clusters: int, default=8**The number of clusters to form as well as the number of centroids to generate.
2. **n\_init: int, default=10**

Number of times the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n\_init consecutive runs in terms of inertia.

1. **max\_iter: int, default=300**

Maximum number of iterations of the k-means algorithm for a single run.

1. **tol: float, default=1e-4**

Relative tolerance with regards to Frobenius norm of the difference in the cluster centres of two consecutive iterations to declare convergence.

1. **verbose: int, default=0**

Verbosity mode.

Attributes:

1. **cluster\_centers\_: ndarray of shape (n\_clusters, n\_features)**  
     
   Coordinates of cluster centres. If the algorithm stops before fully converging (see tol and max\_iter), these will not be consistent with labels\_.
2. **labels\_: ndarray of shape (n\_samples)**  
     
   Labels of each point

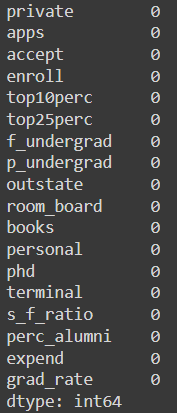
**Data Modeling and Analysis**

**Dataset**: College data (Private and Public universities acceptance dataset)

**Preprocessing:**

# There are no null values in the dataset

df.isna().sum()



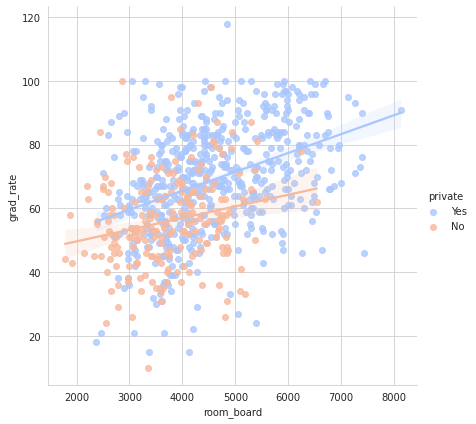
**EDA:**

# Grad Rate vs Room Board where the points are colored by private Column

sns.set\_style('whitegrid')

sns.lmplot('room\_board','grad\_rate',data=df, hue='private',

palette='coolwarm',height=6,aspect=1,fit\_reg=True)

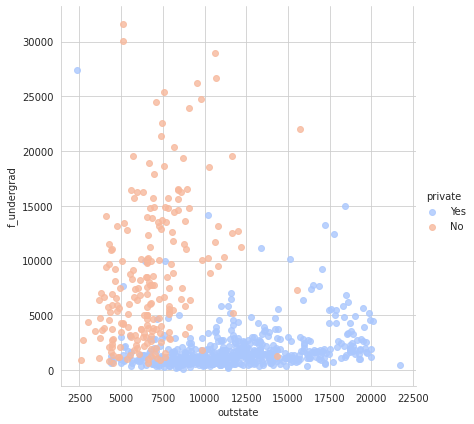


# f\_undergrad vs Outstate

sns.set\_style('whitegrid')

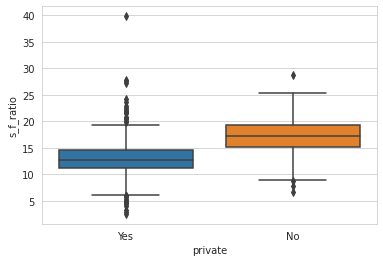
sns.lmplot('outstate','f\_undergrad',data=df, hue='private',

palette='coolwarm',height=6,aspect=1,fit\_reg=False)



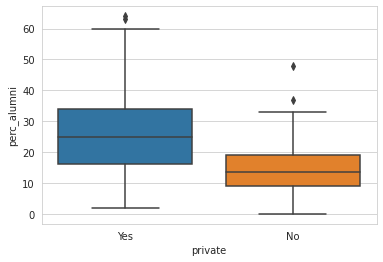
# Boxplot of student-faculty ratio based on type of college

sns.boxplot(x='private',y='s\_f\_ratio',data=df)



#Boxplot of Percentage of Alumni who donate based on type of college

sns.boxplot(x='private',y='perc\_alumni',data=df)

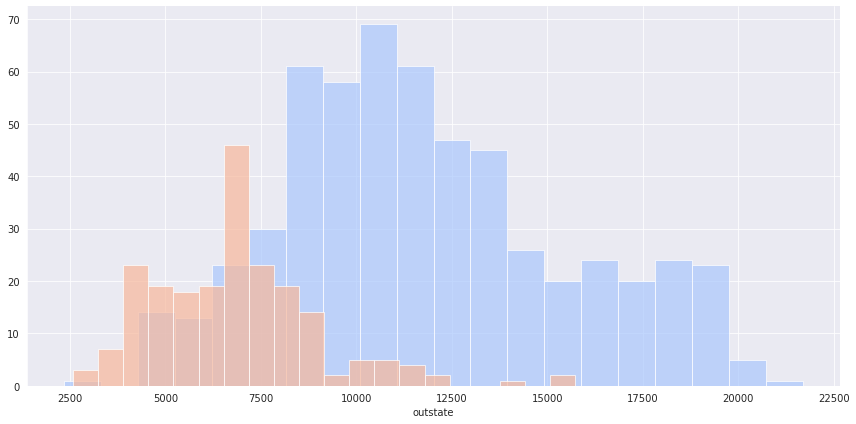


#Histogram for Out of Station Students' Tuition based on Private Column

sns.set\_style('darkgrid')

g = sns.FacetGrid(df,hue="private",palette='coolwarm',height=6,aspect=2)

g = g.map(plt.hist,'outstate',bins=20,alpha=0.7)



**Code and Observation**:

**Clustering Using Python Libraries:**

kmeans = KMeans(n\_clusters=2,verbose=0,tol=1e-3,max\_iter=300,n\_init=20)

kmeans.fit(df[['room\_board', 'expend']])

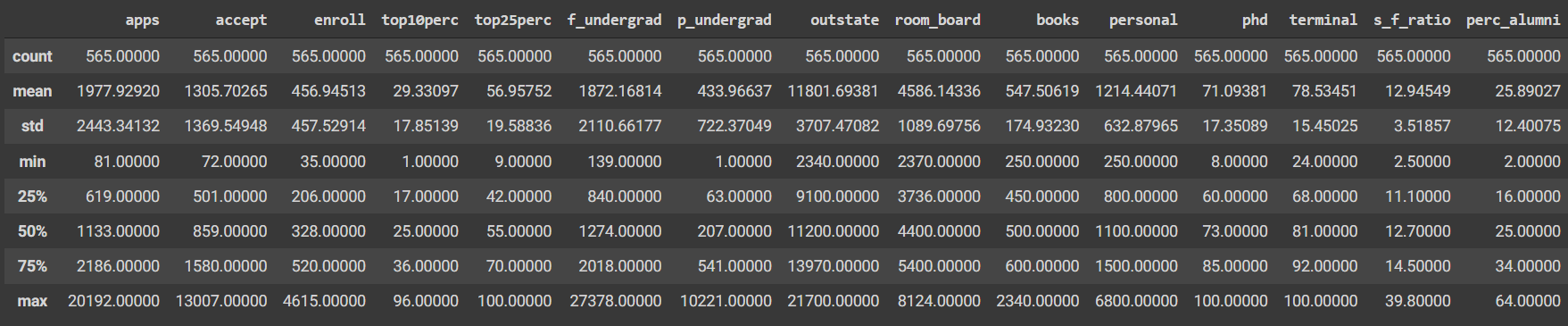
clus\_cent=kmeans.cluster\_centers\_

clus\_cent



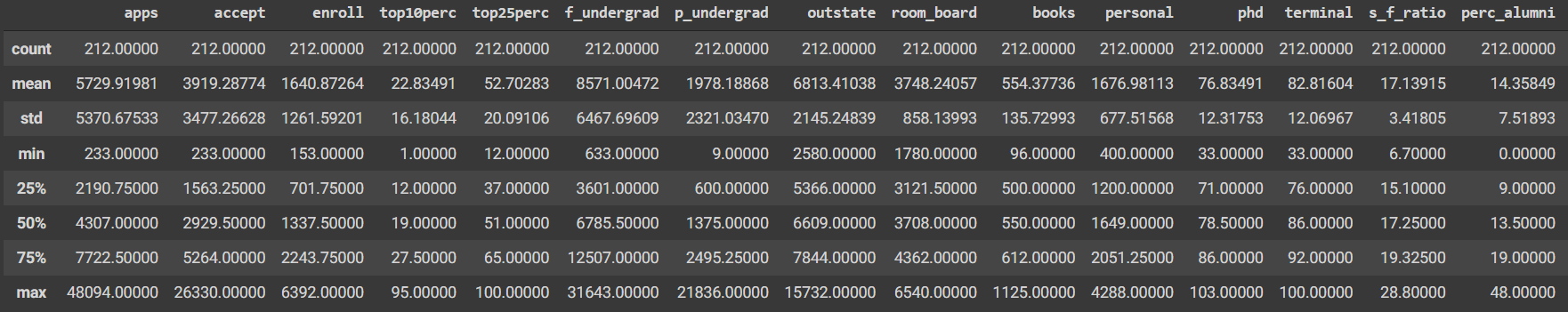
# Stats for Private Colleges

df[df["private"] == 'Yes'].describe()



# Stats for Non-Private Colleges

df[df["private"] == 'No'].describe()



converter = lambda cluster : 1 if cluster == 'Yes' else 0

df1 = df

df1['Cluster'] = df['private'].apply(converter)

kmeans = KMeans(n\_clusters=2,verbose=0,tol=1e-3,max\_iter=50,n\_init=10)

kmeans.fit(df[['room\_board', 'expend']])

clus\_cent=kmeans.cluster\_centers\_

df\_desc=pd.DataFrame(df[['room\_board', 'expend']].describe())

feat = list(df\_desc.columns)

kmclus = pd.DataFrame(clus\_cent,columns=feat)

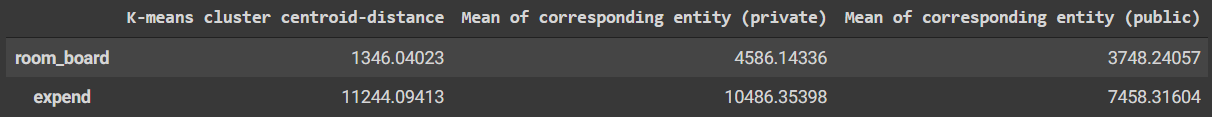
a=np.array(kmclus.diff().iloc[1])

centroid\_diff = pd.DataFrame(abs(a),columns=['K-means cluster centroid-distance'],index=df\_desc.columns)

centroid\_diff['Mean of corresponding entity (private)']=np.array(df[['room\_board', 'expend']][df['private']== 'Yes'].mean())

centroid\_diff['Mean of corresponding entity (public)']=np.array(df[['room\_board', 'expend']][df['private']== 'No'].mean())

centroid\_diff



**Clustering Using User-defined function**

# Considering Two Columns, Expenditure, Room\_Board costs

df2 = df[["room\_board", "expend"]]

df2['Cluster'] = df["private"].apply(converter)

df2.head()

from math import sqrt

def distance(p1, p2):

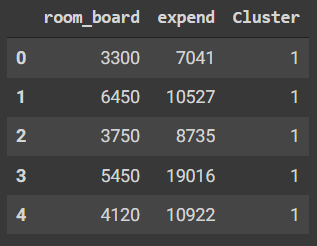
x1 = p1[0]

y1 = p1[1]

x2 = p2[0]

y2 = p2[1]

return sqrt((x2-x1) \*\* 2 + (y2-y1) \*\* 2)



cluster\_choice = lambda d0, d1: 0 if d0 < d1 else 1

def calculate\_centroids(point):

arr1 = point['x']

arr2 = point['y']

return (np.mean(arr1), np.mean(arr2))

# Two Clusters, Private and Public

centroids = [(df2['room\_board'][0], df2['expend'][0]), (df2['room\_board'][1], df2['expend'][1])]

clusters = {

0: {

'x': [],

'y': []

},

1: {

'x': [],

'y': []

}

}

for k in range(20):

labels = []

for row in df2.iterrows():

r = dict(row[1])

point = (r['room\_board'], r['expend'])

if len(centroids) == 0:

centroids = [(df2['room\_board'][0], df2['expend'][0]), (df2['room\_board'][1], df2['expend'][1])]

d1 = distance(point, centroids[0])

d2 = distance(point, centroids[1])

cluster = cluster\_choice(d1, d2)

clusters[cluster]['x'].append(r['room\_board'])

clusters[cluster]['y'].append(r['room\_board'])

labels.append(cluster)

centroids = [calculate\_centroids(clusters[0]), calculate\_centroids(clusters[1])]

clusters = {

0: {

'x': [],

'y': []

},

1: {

'x': [],

'y': []

}

}

centroids = []

**Comparing Metrics of both the implementations:**

Using python libraries:

from sklearn.metrics import davies\_bouldin\_score

db\_index = davies\_bouldin\_score(df1.drop(['private'], axis=1), kmeans.labels\_)

db\_index



Using custom function:

db\_index\_custom = davies\_bouldin\_score(df2, kmeans.labels\_)

db\_index\_custom



Thus, the implementation of clustering using python libraries gives better results as the Davies Bouldin Score of the same is lower than that of the custom function.

**Conclusion**:

Thus, we have learnt about clustering, various ways to implement clustering in a dataset and compared clustering using our function with the python libraries