Experiment : 7

Aim:

To implement the Clustering algorithm using Python.

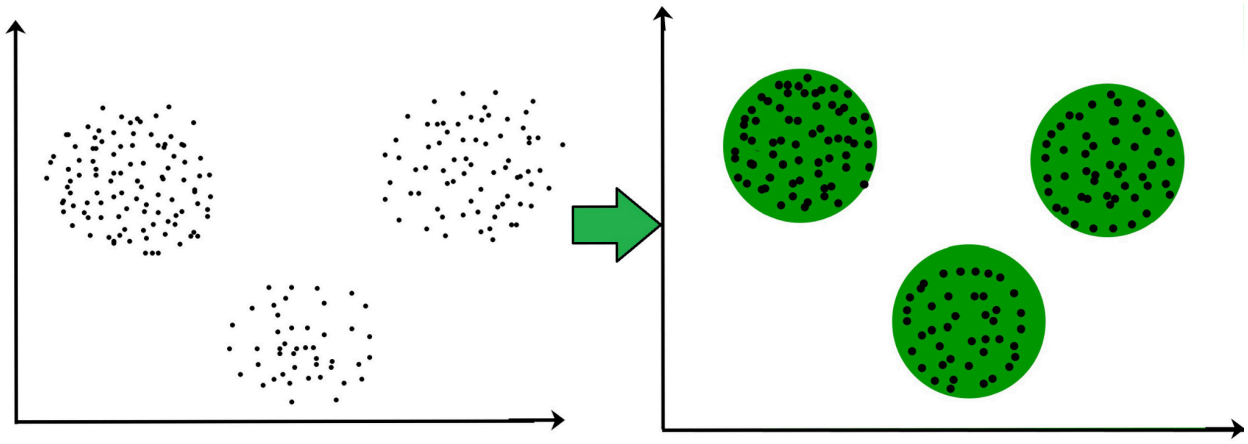
Clustering Algorithm:

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 $O(n^2)$ in complexity notation. $O(n^2)$ 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.

2. 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.

3. 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.

4. 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.

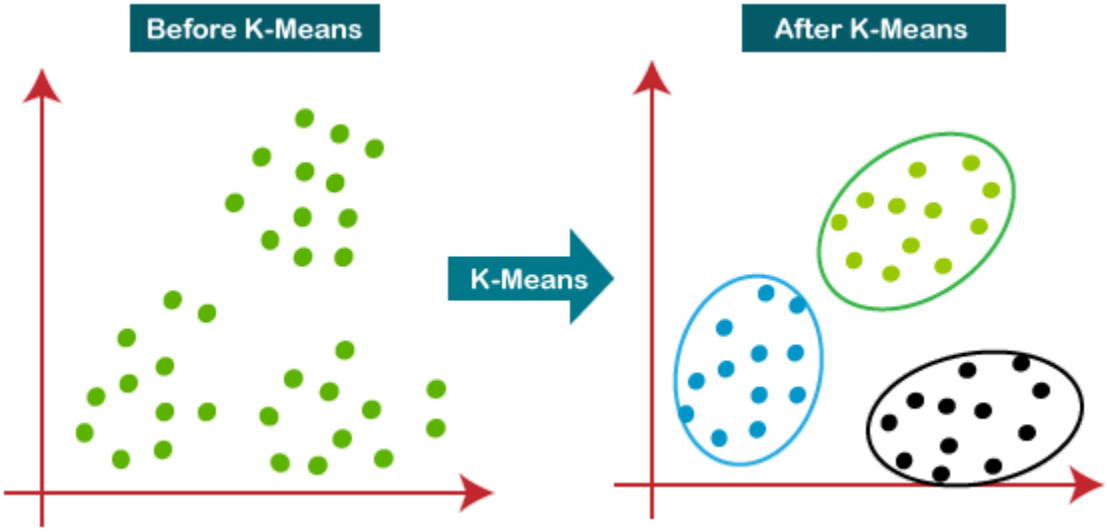
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters. The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, 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.

3. **max_iter: int, default=300**
   Maximum number of iterations of the k-means algorithm for a single run.

4. **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.

5. **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()
```

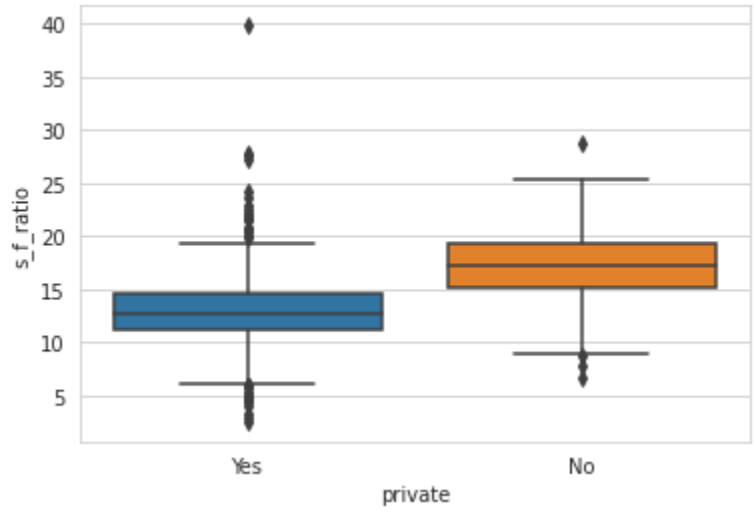
```
private        0
apps           0
accept         0
enroll         0
top10perc      0
top25perc      0
f_undergrad    0
p_undergrad    0
outstate       0
room_board     0
books          0
personal       0
phd            0
terminal       0
s_f_ratio      0
perc_alumni    0
expend         0
grad_rate      0
dtype: int64
```

EDA:

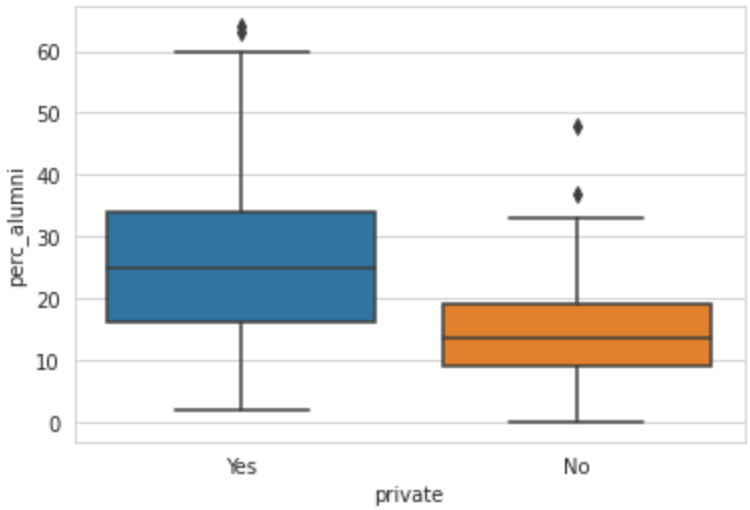
```
# Grad Rate vs Room Board where the points are colored by private
Column
sns.set_style('whitegrid')
sns.lmplot('room_board','grad_rate',data=df, hue='private',
           palette='coolwarm',height=6,aspect=1,fit_reg=True)
```

```
# 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)
```

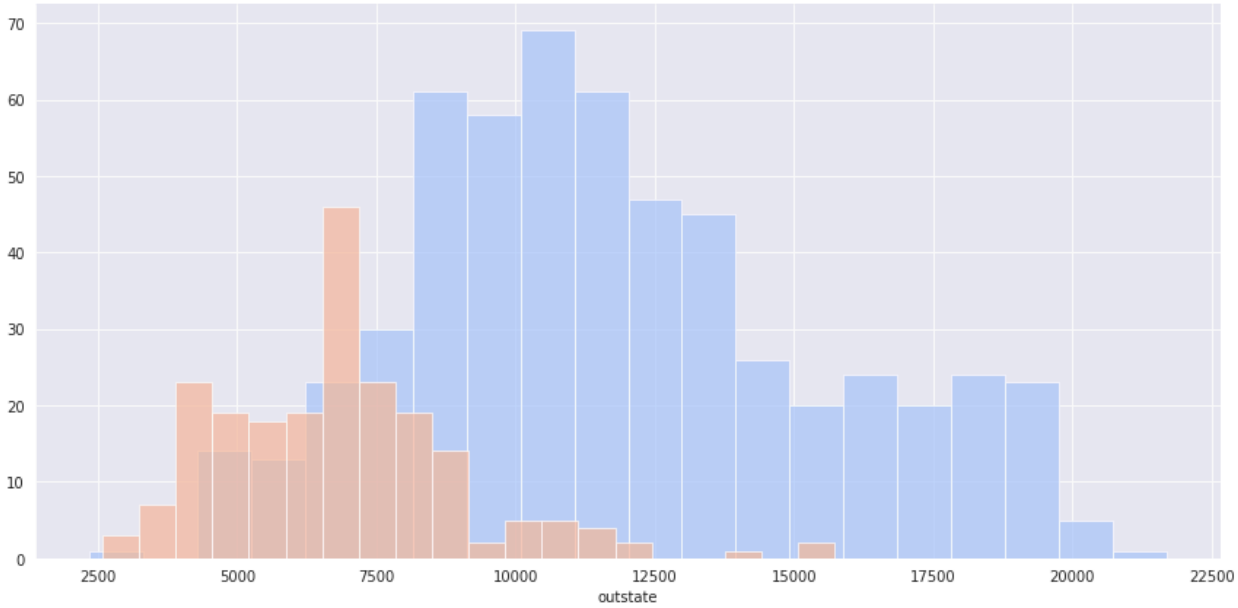

```
# Boxplot of student-faculty ratio based on type of college
sns.boxplot(x='private',y='s_f_ratio',data=df)
```




```
#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
```

```
array([[ 4187.68175,  8233.97372],
       [ 5622.13043, 20279.1413 ]])
```

```
# Stats for Private Colleges
df[df["private"] == 'Yes'].describe()
```

	apps	accept	enroll	top10perc	top25perc	f_undergrad	p_undergrad	outstate	room_board	books	personal	phd	terminal	s_f_ratio	perc_alumni
count	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000
mean	1977.92920	1305.70265	456.94513	29.33097	56.95752	1872.16814	433.96637	11801.69381	4586.14336	547.50619	1214.44071	71.09381	78.53451	12.94549	25.89027
std	2443.34132	1369.54948	457.52914	17.85139	19.58836	2110.66177	722.37049	3707.47082	1089.69756	174.93230	632.87965	17.35089	15.45025	3.51857	12.40075
min	81.00000	72.00000	35.00000	1.00000	9.00000	139.00000	1.00000	2340.00000	2370.00000	250.00000	250.00000	8.00000	24.00000	2.50000	2.00000
25%	619.00000	501.00000	206.00000	17.00000	42.00000	840.00000	63.00000	9100.00000	3736.00000	450.00000	800.00000	60.00000	68.00000	11.10000	16.00000
50%	1133.00000	859.00000	328.00000	25.00000	55.00000	1274.00000	207.00000	11200.00000	4400.00000	500.00000	1100.00000	73.00000	81.00000	12.70000	25.00000
75%	2186.00000	1580.00000	520.00000	36.00000	70.00000	2018.00000	541.00000	13970.00000	5400.00000	600.00000	1500.00000	85.00000	92.00000	14.50000	34.00000
max	20192.00000	13007.00000	4615.00000	96.00000	100.00000	27378.00000	10221.00000	21700.00000	8124.00000	2340.00000	6800.00000	100.00000	100.00000	39.80000	64.00000

```
# Stats for Non-Private Colleges
df[df["private"] == 'No'].describe()
```

	apps	accept	enroll	top10perc	top25perc	f_undergrad	p_undergrad	outstate	room_board	books	personal	phd	terminal	s_f_ratio	perc_alumni
count	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000
mean	5729.91981	3919.28774	1640.87264	22.83491	52.70283	8571.00472	1978.18868	6813.41038	3748.24057	554.37736	1676.98113	76.83491	82.81604	17.13915	14.35849
std	5370.67533	3477.26628	1261.59201	16.18044	20.09106	6467.69609	2321.03470	2145.24839	858.13993	135.72993	677.51568	12.31753	12.06967	3.41805	7.51893
min	233.00000	233.00000	153.00000	1.00000	12.00000	633.00000	9.00000	2580.00000	1780.00000	96.00000	400.00000	33.00000	33.00000	6.70000	0.00000
25%	2190.75000	1563.25000	701.75000	12.00000	37.00000	3601.00000	600.00000	5366.00000	3121.50000	500.00000	1200.00000	71.00000	76.00000	15.10000	9.00000
50%	4307.00000	2929.50000	1337.50000	19.00000	51.00000	6785.50000	1375.00000	6609.00000	3708.00000	550.00000	1649.00000	78.50000	86.00000	17.25000	13.50000
75%	7722.50000	5264.00000	2243.75000	27.50000	65.00000	12507.00000	2495.25000	7844.00000	4362.00000	612.00000	2051.25000	86.00000	92.00000	19.32500	19.00000
max	48094.00000	26330.00000	6392.00000	95.00000	100.00000	31643.00000	21836.00000	15732.00000	6540.00000	1125.00000	4288.00000	103.00000	100.00000	28.80000	48.00000

```
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
```

	K-means cluster centroid-distance	Mean of corresponding entity (private)	Mean of corresponding entity (public)
room_board	1346.04023	4586.14336	3748.24057
expend	11244.09413	10486.35398	7458.31604

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)
```

	room_board	expend	Cluster
0	3300	7041	1
1	6450	10527	1
2	3750	8735	1
3	5450	19016	1
4	4120	10922	1

```
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
```

```
1.1333615669636636
```

Using custom function:

```
db_index_custom = davies_bouldin_score(df2, kmeans.labels_)
db_index_custom
```

```
1.9457780421226276
```

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