EXPERIMENT NO. 4 A

- **Aim**: Implement K-Means clustering on Iris.csv dataset. Determine the number of clusters using the elbow method.Dataset Link: https://www.kaggle.com/datasets/uciml/iris

- **Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

- **Software Requirement:**
  Jupyter Nootbook/Ubuntu

- **Theory:**

K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called flat clustering algorithm. The number of clusters identified from data by algorithm is represented by ‘K’ in K-means.

In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

Working of K-Means Algorithm

We can understand the working of K-Means clustering algorithm with the help of following steps −

Step 1 − First, we need to specify the number of clusters, K, need to be generated by this algorithm.

Step 2 − Next, randomly select K data points and assign each data point to a cluster. In simple words, classify the data based on the number of data points.

Step 3 − Now it will compute the cluster centroids.

Step 4 − Next, keep iterating the following until we find optimal centroid which is the assignment of data points to the clusters that are not changing any more −

4.1 − First, the sum of squared distance between data points and centroids would be computed.

4.2 − Now, we have to assign each data point to the cluster that is closer than other cluster (centroid).

4.3 − At last compute the centroids for the clusters by taking the average of all data points of that cluster.

K-means follows Expectation-Maximization approach to solve the problem. The Expectation-step is used for assigning the data points to the closest cluster and the Maximization-step is used for computing the centroid of each cluster.

While working with K-means algorithm we need to take care of the following things −

While working with clustering algorithms including K-Means, it is recommended to standardize the data because such algorithms use distance-based measurement to determine the similarity between data points.

Due to the iterative nature of K-Means and random initialization of centroids, K-Means may stick in a local optimum and may not converge to global optimum. That is why it is recommended to use different initializations of centroids

Implementation:

Importing the libraries and the data

```
import pandas as pd # Pandas (version : 1.1.5)
 import numpy as np # Numpy (version : 1.19.2)
 import matplotlib.pyplot as plt # Matplotlib (version :  3.3.2)
 from sklearn.cluster import KMeans # Scikit Learn (version : 0.23.2)
 import seaborn as sns # Seaborn (version : 0.11.1)
 plt.style.use('seaborn')
```

Importing the data from .csv file

First we read the data from the dataset using read_csv from the pandas library.

```
data = pd.read_csv('data\iris.csv')
```

Viewing the data that we imported to pandas dataframe object

```
data
```

```
        Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0        1            5.1           3.5            1.4           0.2
1        2            4.9           3.0            1.4           0.2
2        3            4.7           3.2            1.3           0.2
3        4            4.6           3.1            1.5           0.2
4        5            5.0           3.6            1.4           0.2
..     ...            ...           ...            ...           ...
145    146            6.7           3.0            5.2           2.3
146    147            6.3           2.5            5.0           1.9
147    148            6.5           3.0            5.2           2.0
148    149            6.2           3.4            5.4           2.3
149    150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

Viewing and Describing the data

Now we view the Head and Tail of the data using head() and tail() respectively.

data.head()

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
```

0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

data.tail()

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
145	146	6.7	3.0	5.2	2.3	
146	147	6.3	2.5	5.0	1.9	
147	148	6.5	3.0	5.2	2.0	
148	149	6.2	3.4	5.4	2.3	
149	150	5.9	3.0	5.1	1.8	

	Species
145	Iris-virginica
146	Iris-virginica
147	Iris-virginica
148	Iris-virginica
149	Iris-virginica

Checking the sample size of data - how many samples are there in the dataset using len().

```
len(data)
```

150

Checking the dimensions/shape of the dataset using shape.

```
data.shape
```

(150, 6)

Viewing Column names of the dataset using columns

```
data.columns
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
for i,col in enumerate(data.columns):
```

```
    print(f'Column number {1+i} is {col}')
```

```
Column number 1 is Id
 Column number 2 is SepalLengthCm
 Column number 3 is SepalWidthCm
 Column number 4 is PetalLengthCm
 Column number 5 is PetalWidthCm
 Column number 6 is Species
```

So, our dataset has 5 columns named:

- Id
- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm
- Species.

View datatypes of each column in the dataset using dtype.

```
data.dtypes
```

```
Id                int64
 SepalLengthCm    float64
 SepalWidthCm     float64
 PetalLengthCm    float64
 PetalWidthCm     float64
 Species          object
 dtype: object
```

Gathering Further information about the dataset using info()

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
 RangeIndex: 150 entries, 0 to 149
 Data columns (total 6 columns):
  #   Column            Non-Null Count  Dtype
 ---  ------        --------------  -----
  0   Id            150 non-null    int64
```

1	SepalLengthCm	150 non-null	float64
2	SepalWidthCm	150 non-null	float64
3	PetalLengthCm	150 non-null	float64
4	PetalWidthCm	150 non-null	float64
5	Species	150 non-null	object

dtypes: float64(4), int64(1), object(1)

memory usage: 7.2+ KB

Describing the data as basic statistics using describe()

data.describe()

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

Checking the data for inconsistencies and further cleaning the data if needed.

Checking data for missing values using isnull().

data.isnull()

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
..	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False

148	False	False	False	False	False	False
149	False	False	False	False	False	False

[150 rows x 6 columns]

Checking summary of missing values

data.isnull().sum()

Id 0

SepalLengthCm 0

SepalWidthCm 0

PetalLengthCm 0

PetalWidthCm 0

Species 0

dtype: int64

The 'Id' column has no relevence therefore deleting it would be better.

Deleting 'customer_id' columnn using drop().

data.drop('Id', axis=1, inplace=True)

data.head()

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Modelling

K - Means Clustering

K-means clustering is a clustering algorithm that aims to partition n observations into k clusters. Initialisation – K initial “means” (centroids) are generated at random Assignment – K clusters are created by associating each observation with the nearest centroid Update – The centroid of the clusters becomes the new mean, Assignment and Update are repeated iteratively until convergence The end result is that the

sum of squared errors is minimised between points and their respective centroids. We will use KMeans Clustering. At first we will find the optimal clusters based on inertia and using elbow method. The distance between the centroids and the data points should be less.

First we need to check the data for any missing values as it can ruin our model.

```
data.isna().sum()
```

```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

We conclude that we don't have any missing values therefore we can go forward and start the clustering procedure.

We will now view and select the data that we need for clustering.

```
data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Checking the value count of the target column i.e. 'Species' using value_counts()

```
data['Species'].value_counts()
```

```
Iris-setosa      50
Iris-versicolor  50
```

```
Iris-virginica     50
Name: Species, dtype: int64
```

Splitting into Training and Target data

Target Data

```
target_data = data.iloc[:,4]
target_data.head()
```

```
0        Iris-setosa
1        Iris-setosa
2        Iris-setosa
3        Iris-setosa
4        Iris-setosa
Name: Species, dtype: object
```

Training data

```
clustering_data = data.iloc[:,[0,1,2,3]]
clustering_data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Now, we need to visualize the data which we are going to use for the clustering. This will give us a fair idea about the data we're working on.

```
fig, ax = plt.subplots(figsize=(15,7))
sns.set(font_scale=1.5)
ax = sns.scatterplot(x=data['SepalLengthCm'],y=data['SepalWidthCm'], s=70, color='#f73434',
```

```
edgecolor='#f73434', linewidth=0.3)
 ax.set_ylabel('Sepal Width (in cm)')
 ax.set_xlabel('Sepal Length (in cm)')
 plt.title('Sepal Length vs Width', fontsize = 20)
 plt.show()
```

This gives us a fair Idea and patterns about some of the data.

Determining No. of Clusters Required

The Elbow Method

The Elbow method runs k-means clustering on the dataset for a range of values for k (say from 1-10) and then for each value of k computes an average score for all clusters. By default, the distortion score is computed, the sum of square distances from each point to its assigned center.

When these overall metrics for each model are plotted, it is possible to visually determine the best value for k. If the line chart looks like an arm, then the “elbow” (the point of inflection on the curve) is the best value of k. The “arm” can be either up or down, but if there is a strong inflection point, it is a good indication that the underlying model fits best at that point.

We use the Elbow Method which uses Within Cluster Sum Of Squares (WCSS) against the the number of clusters (K Value) to figure out the optimal number of clusters value. WCSS measures sum of distances of observations from their cluster centroids which is given by the below formula.

formula

where Yi is centroid for observation Xi. The main goal is to maximize number of clusters and in limiting case each data point becomes its own cluster centroid.

With this simple line of code we get all the inertia value or the within the cluster sum of square.

```
from sklearn.cluster import KMeans
 wcss=[]
 for i in range(1,11):
```

```
        km = KMeans(i)
    km.fit(clustering_data)
    wcss.append(km.inertia_)
np.array(wcss)
```

```
array([680.8244        , 152.36870648,  78.94084143,  57.31787321,
       46.53558205,  38.93096305,  34.29998554,  30.21678683,
       28.23999745,  25.95204113])
```

Inertia can be recognized as a measure of how internally coherent clusters are.

Now, we visualize the Elbow Method so that we can determine the number of optimal clusters for our dataset.

```
fig, ax = plt.subplots(figsize=(15,7))
 ax = plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
 plt.axvline(x=3, ls='--')
 plt.ylabel('WCSS')
 plt.xlabel('No. of Clusters (k)')
 plt.title('The Elbow Method', fontsize = 20)
 plt.show()
```

It is clear, that the optimal number of clusters for our data are 3, as the slope of the curve is not steep enough after it. When we observe this curve, we see that last elbow comes at k = 3, it would be difficult to visualize the elbow if we choose the higher range.

Clustering

Now we will build the model for creating clusters from the dataset. We will use n_clusters = 3 i.e. 3 clusters as we have determined by the elbow method, which would be optimal for our dataset.

Our data set is for unsupervised learning therefore we will use fit_predict() Suppose we were working with supervised learning data set we would use fit_tranform()

```
from sklearn.cluster import KMeans
```

```
kms = KMeans(n_clusters=3, init='k-means++')
kms.fit(clustering_data)
```

```
KMeans(n_clusters=3)
```

Now that we have the clusters created, we will enter them into a different column

```
clusters = clustering_data.copy()
clusters['Cluster_Prediction'] = kms.fit_predict(clustering_data)
clusters.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	
3	4.6	3.1	1.5	0.2	
4	5.0	3.6	1.4	0.2	

	Cluster_Prediction
0	1
1	1
2	1
3	1
4	1

We can also get the centroids of the clusters by the cluster_centers_ attribute of KMeans algorithm.

```
kms.cluster_centers_
```

```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.418     , 1.464     , 0.244     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

Now we have all the data we need, we just need to plot the data. We will plot the data using scatterplot which will allow us to observe different clusters in different colours.

```
fig, ax = plt.subplots(figsize=(15,7))
 plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 0]['SepalLengthCm'],
        y=clusters[clusters['Cluster_Prediction'] == 0]['SepalWidthCm'],
        s=70,edgecolor='teal', linewidth=0.3, c='teal', label='Iris-versicolor')


 plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 1]['SepalLengthCm'],
        y=clusters[clusters['Cluster_Prediction'] == 1]['SepalWidthCm'],
        s=70,edgecolor='lime', linewidth=0.3, c='lime', label='Iris-setosa')


 plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 2]['SepalLengthCm'],
        y=clusters[clusters['Cluster_Prediction'] == 2]['SepalWidthCm'],
        s=70,edgecolor='magenta', linewidth=0.3, c='magenta', label='Iris-virginica')

 plt.scatter(x=kms.cluster_centers_[:, 0], y=kms.cluster_centers_[:, 1], s = 170, c = 'yellow', label =
'Centroids',edgecolor='black', linewidth=0.3)
 plt.legend(loc='upper right')
 plt.xlim(4,8)
 plt.ylim(1.8,4.5)
 ax.set_ylabel('Sepal Width (in cm)')
 ax.set_xlabel('Sepal Length (in cm)')
 plt.title('Clusters', fontsize = 20)
 plt.show()
```