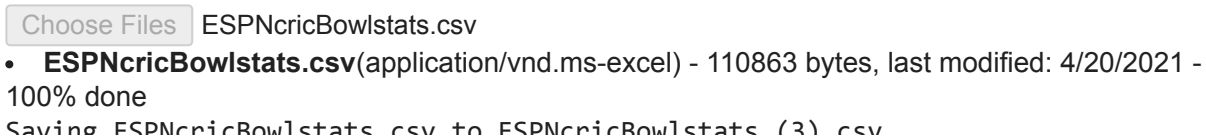
Import Header file

```
import io
from numpy import unique
from numpy import where
from sklearn.cluster import KMeans
from matplotlib import pyplot
import pandas as pd
from google.colab import files
```

Import Data set

```
uploaded = files.upload()
data = pd.read_csv(io.BytesIO(uploaded['ESPNcricBowlstats.csv']))
```



Clustering Algorithms - ODI Bowler Categorization

K-means clustering with cluster size 5 To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids It halts creating and optimizing clusters when either:

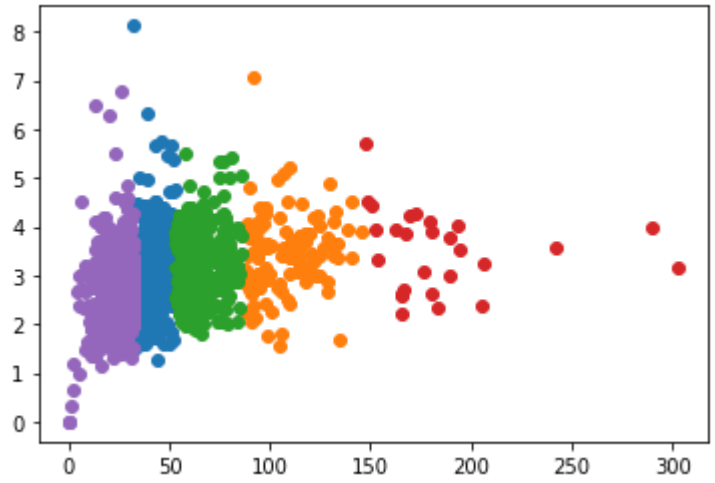
1. The centroids have stabilized — there is no change in their values because the clustering has been successful.

2. The defined number of iterations has been achieved.

```
# k-means clustering

X=data[["Ave","Econ"]]
X=(X.to_numpy())
model = KMeans(n_clusters=5)
model.fit(X)
yhat = model.predict(X)
clusters = unique(yhat)

for cluster in clusters:
  row_ix = where(yhat == cluster)
  pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()
```

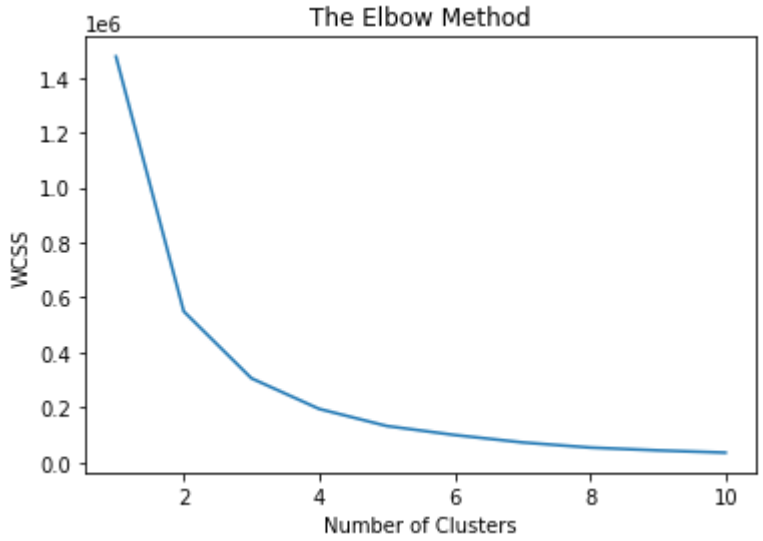


Elbow Method In cluster analysis, the elbow method is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use.

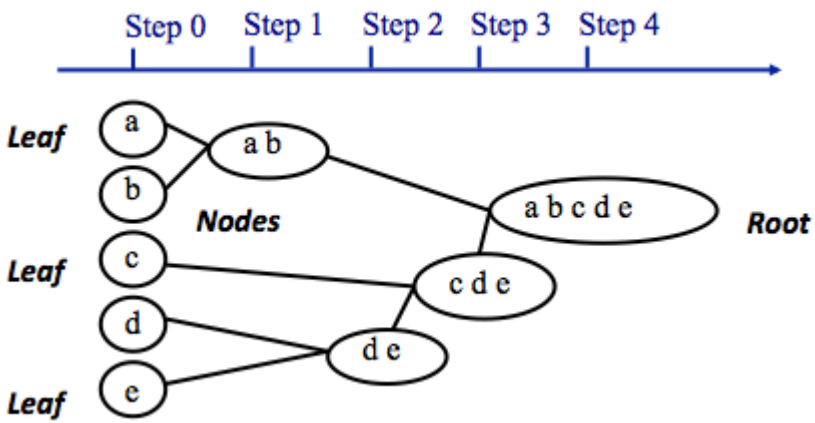
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

```
wcss=[]
for i in range(1,11):
    kmeans=KMeans(n_clusters=i, init='k-means++',random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS') #Within Cluster Sum of Squares
plt.show()
```



Agglomerative clustering Agglomerative clustering works in a “bottom-up” manner. That is, each object is initially considered as a single-element cluster (leaf). At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (nodes). This procedure is iterated until all points are member of just one single big cluster (root).

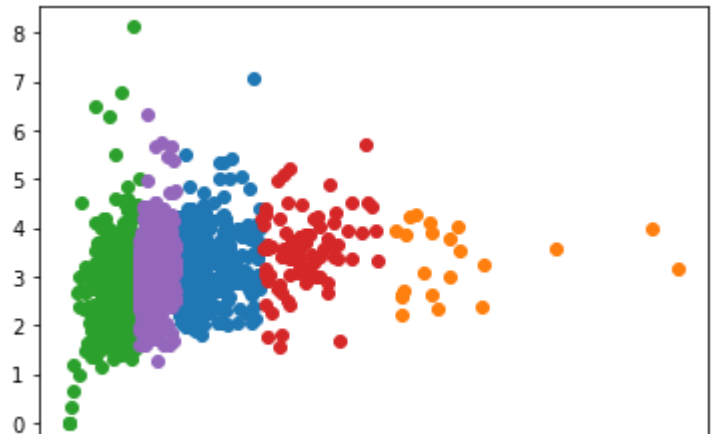



```
# agglomerative clustering
from numpy import unique
from numpy import where
from sklearn.cluster import AgglomerativeClustering
from matplotlib import pyplot
import pandas as pd

data = pd.read_csv("ESPNcricBowlstats.csv")
X=data[["Ave","Econ"]]
X=(X.to_numpy())
model = AgglomerativeClustering(n_clusters=5)

yhat = model.fit_predict(X)
clusters = unique(yhat)

for cluster in clusters:
  row_ix = where(yhat == cluster)
  pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()
```



dbscan clustering Density-Based Clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.

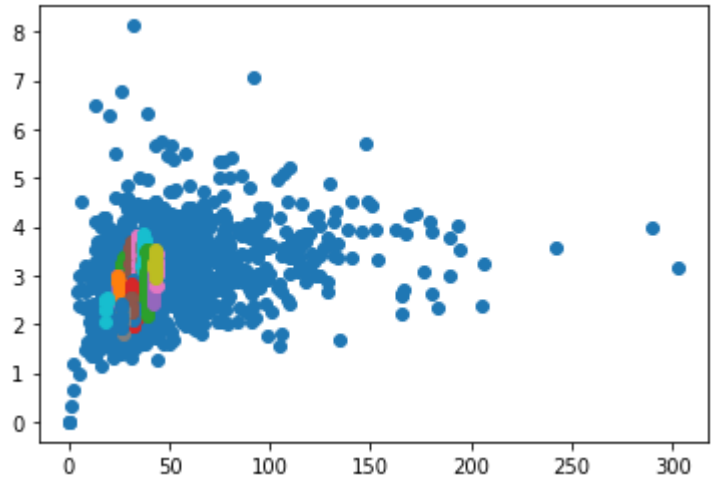
Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

The DBSCAN algorithm uses two parameters:

1. minPts: The minimum number of points (a threshold) clustered together for a region to be considered dense.
2. eps (ε): A distance measure that will be used to locate the points in the neighborhood of any point.

```
# dbscan clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import DBSCAN
from matplotlib import pyplot
data = pd.read_csv("ESPNcricBowlstats.csv")
X=data[["Ave","Econ"]]
X=(X.to_numpy())
model = DBSCAN(eps=0.30, min_samples=9)
yhat = model.fit_predict(X)
clusters = unique(yhat)
```

```
for cluster in clusters:
  row_ix = where(yhat == cluster)
  pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()
```



✓ 0s completed at 08:26