KMEANS

1. Import the necessary packages we plan to use.
2. Load the given CSV file by Pandas.
3. Use column operation in Pandas to select necessary attributes. The values of these attributes are then extracted to a Numpy array for clustering.
4. Initialize the Kmeans function in Sklearn by specifying the number of clusters. If we want the results to be reproducible, we also need to specify the random state.
5. Fit the function by feeding the Numpy array.
6. Allocate the data to the specific cluster.
7. Use Matplotlib to visualize the clustering result. kind of factors/settings:

Parameter selection. (e.g. number of clusters and random state. The value of the random_state determines initial centroids, different random_state value results in different initial centroids.

DBSCAN: (Density Based Clustering)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from pandas import DataFrame
```

1. Import the necessary packages we use above
2. Load the data from csv using **pd.read_csv('.')**
3. Normalise the data using `x = data[['x', 'y']].values`
4. Predict using DBSCAN with Eps = 0.2 and minPts = 10 using:

```
y_pred = DBSCAN(eps=0.2, min_samples=10).fit_predict(X)
print(y_pred)
```

5. Plot the results using:

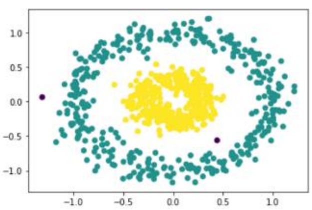
```
plt.scatter(X[:,0], X[:,1], c=y_pred)
print('Number of clusters:
{}'.format(len(set(y_pred[np.where(y_pred !=
-1)]))))
```

Note:

<u>When print(y_pred) is called:</u>

- -1 denotes a noisy point
- 0 and 1 specify two clusters the points belong to.

Number of clusters: 2



General Process:

1. Label all points as noise, core or border points
2. Eliminate all the noise points
3. Initialise a cluster_index which is used to differentiate different clusters
4. Iterate through all core points, check if a point hasn't been assigned a cluster_index, assign it if it hasn't
5. Cluster_index = cluster_index + 1
6. For all the points within the esp circle of the core point, the point hasn't been assigned a cluster_index, then we increment the cluster index and assign the point the cluster_index.
   (Check if dist(P_J, P_i) < EPS, then the point (P_j) will be with the circle, centered around P_i with readius EPS. (To know which are the core and border points)

AGNES (Hierarchical-based clustering)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
```

1. Import the necessary packages we plan to use.
2. Load the given CSV file by Pandas.
   `Data = pd.read_csv('..')`
3. Have a look at the first few lines of the data by using data.head(3)
4. Normalise the data so that the model does not become biased towards variables with high magnitude using the normalize algorithm from sklearn.preprocessing and

```
data_scaled = normalize(data)
data_scaled = pd.DataFrame(data_scaled,
columns=data.columns)
```

5. Can have a look at the first three lines again to check the normalise
6. Use the AgglomerativeClustering function from sklearn.cluster

```
cluster = AgglomerativeClustering(n_clusters=3,
affinity='euclidean', linkage='complete')
```

7. Then clustering result of data can be saved into a variable using and printed:

```
y_predict = cluster.fit_predict(data_scaled)

print(y_predict)
```

Note: This produces like an array/list of data points where 0, 1 and 2 denote which cluster the data points are in (points are assigned these clusters)

Then we can use a dendrogram to visualise the data:

```
dend_max = shc.dendrogram(shc.linkage(data_scaled,
method='complete', metric='euclidean'))
```

Key Points to use AgglomerativeClustering:

1. The distance function is used, which is determined by the attribute **affinity**. We use 'Euclidean' distance.
2. The way to update the distances between clusters is determined by the attribute **linkage**. We use 'complete' but 'single' and 'average' can also be used.
3. The number of clusters you need in the final output is determined by the attribute **n_clusters.**

```
cluster = AgglomerativeClustering(n_clusters=3,
affinity='euclidean', linkage='complete')
```

Minimum (single linkage)	• Is best at handling non-elliptical shapes • Is sensitive to noise
Maximum (complete linkage)	• Tends to form globular shapes • Tends to break large clusters
Average (Average linkage)	• For avg hierarchical clustering