ELG 5255[EG] Applied Machine Learning Summer

Assignment -Three (unsupervised learning)

(Group 34)

Ahmed Hallaba Omar Sorour

Kamel El-Sehly

Numerical Questions

Q1 (K-means)

1) A.

1- From the given table we created the distance matrix between the initial chosen centers A1, A4, and A7 and all the points to label them.

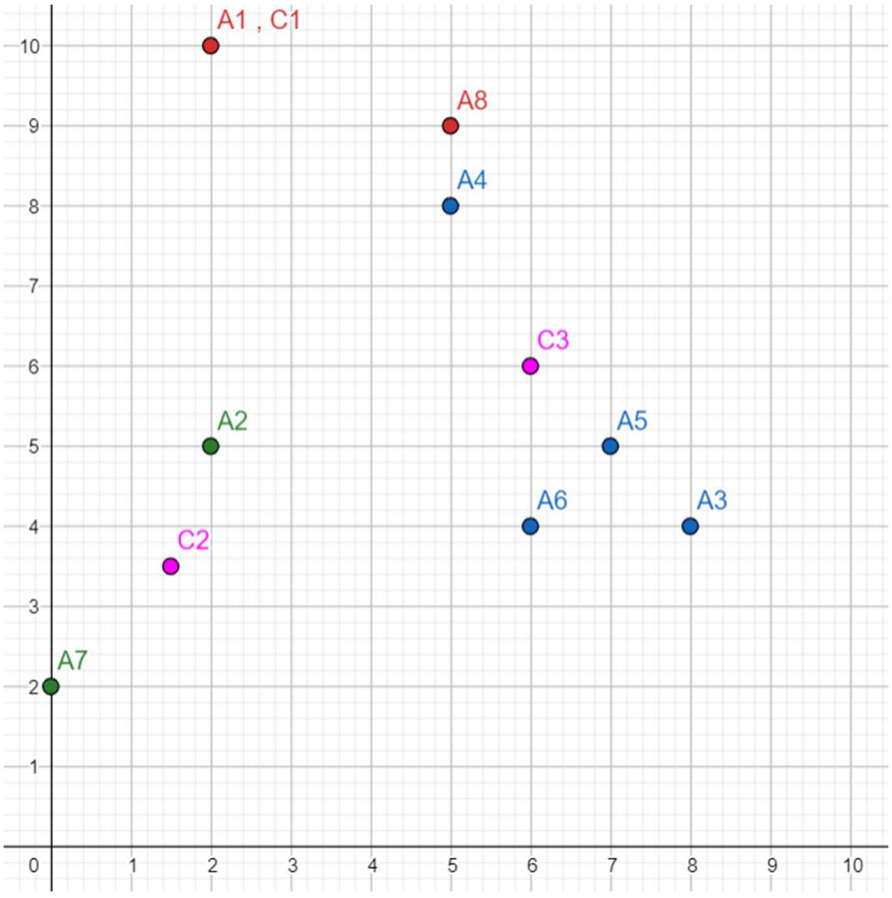
Point	C1	C2	C3	Label
A1	0	√13	√65	1
A2	√25	√18	√10	3
A3	√72	√25	√53	2
A4	√13	0	√52	2
A5	√50	√13	√45	2
A6	√52	√17	√29	2
A7	√65	√52	0	3
A8	√5	√2	√29	2

2- Then we calculated the new three centroids for all the points with the same label.
(A1) ➔ C1(2,10)
(A3, A4, A5, A6, A8) ➔ C2(6,6)
(A2, A7) ➔ C3(1.5, 3.5)

3- After that we calculated the new distance matrix between the points and the new centroids.

Point	C1	C2	C3	Label
A1	0	√32	√42.5	1
A2	√25	√17	√2.5	3
A3	√72	√8	√42.5	2
A4	√13	√5	√32.5	2
A5	√50	√2	√32.5	2
A6	√52	√4	√20.5	2
A7	√65	√41	√2.5	3
A8	√5	√13	√36.5	1

1) B. Draw a 10 by 10 space with all the clustered 8 points and the coordinates of the new centroids.



2) How many more iterations are needed to converge? Draw the result for each iteration (similar to 1.b)

Given the same initial centroids A1, A4, and A7, it will take the model 3 iterations to converge. The third and the fourth iteration had the same centroids. So only three points to reach the centroids of all clusters.

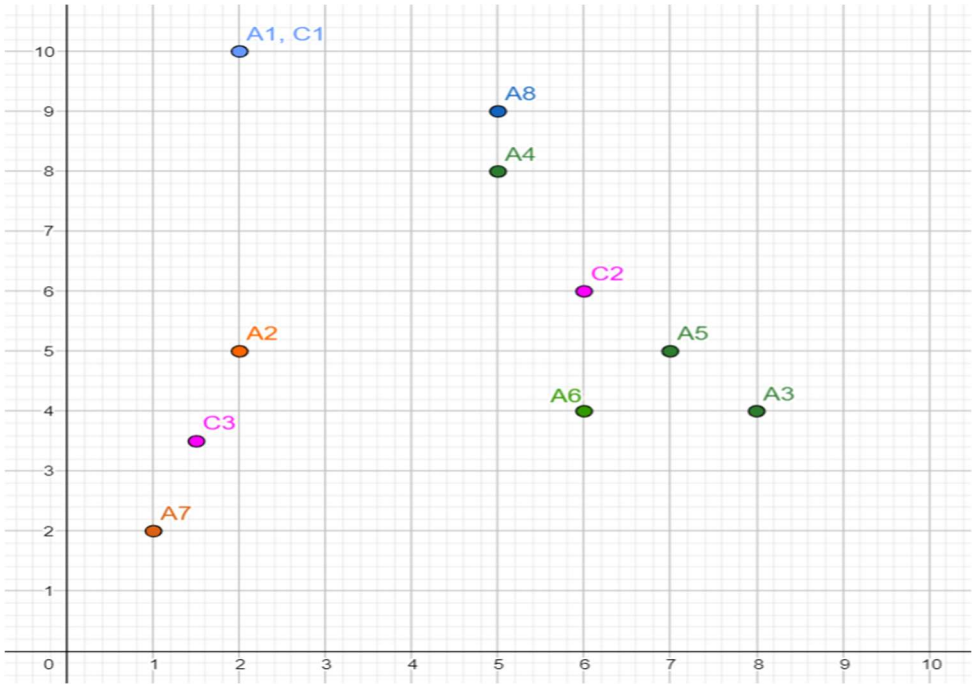
```
from sklearn.cluster import KMeans
my_centroids=np.array([[2,10], [5,8], [1,2]])
kmeans = KMeans(n_clusters=3, random_state=3,init=my_centroids).fit(Points)
kmeans.labels_
kmeans.cluster_centers_
kmeans.labels_
kmeans.n_iter_
```

```
C:\Users\SaWa\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1035: R
sed: performing only one init in KMeans instead of n_init=10.
  self._check_params(X)
```

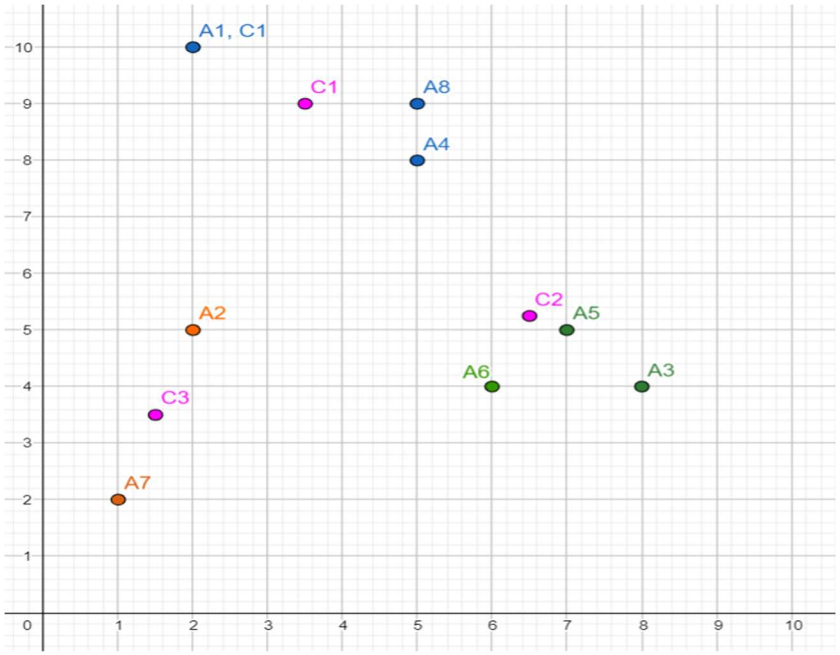
```
4
```

The centroids will change with each iteration as in the following figures:

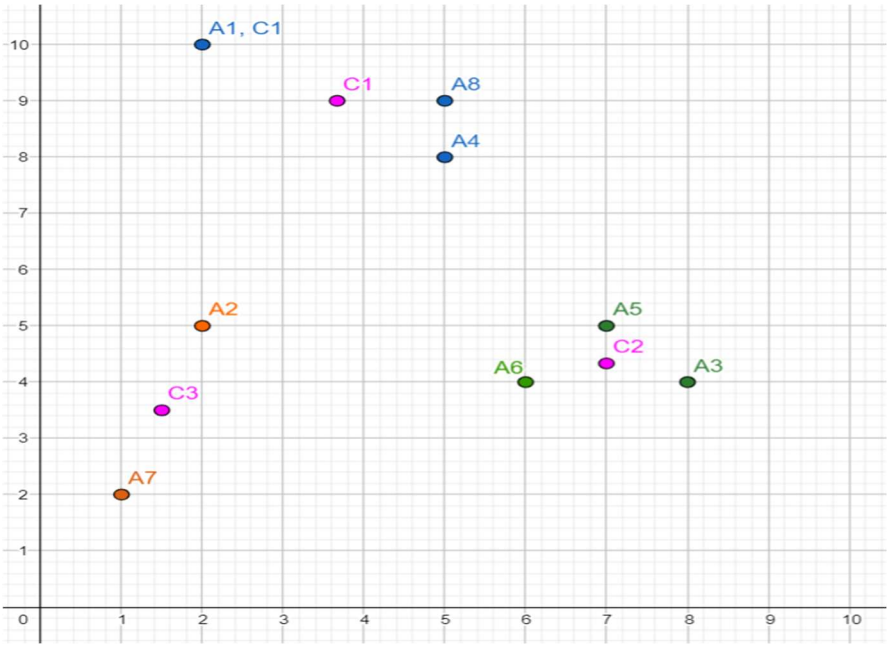
After the first iteration



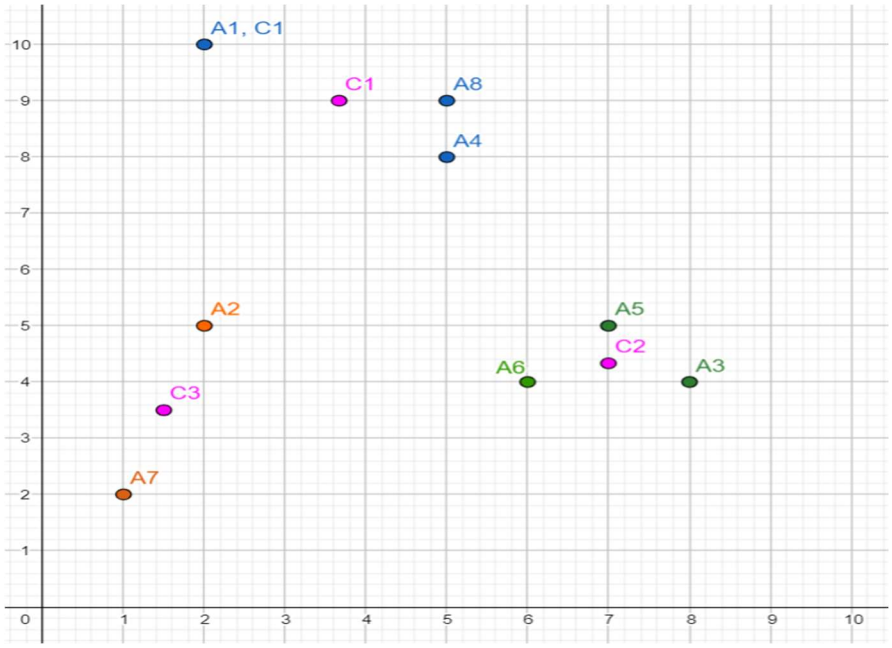
After the second iteration



After the third iteration



The fourth iteration had the same results as in the third one



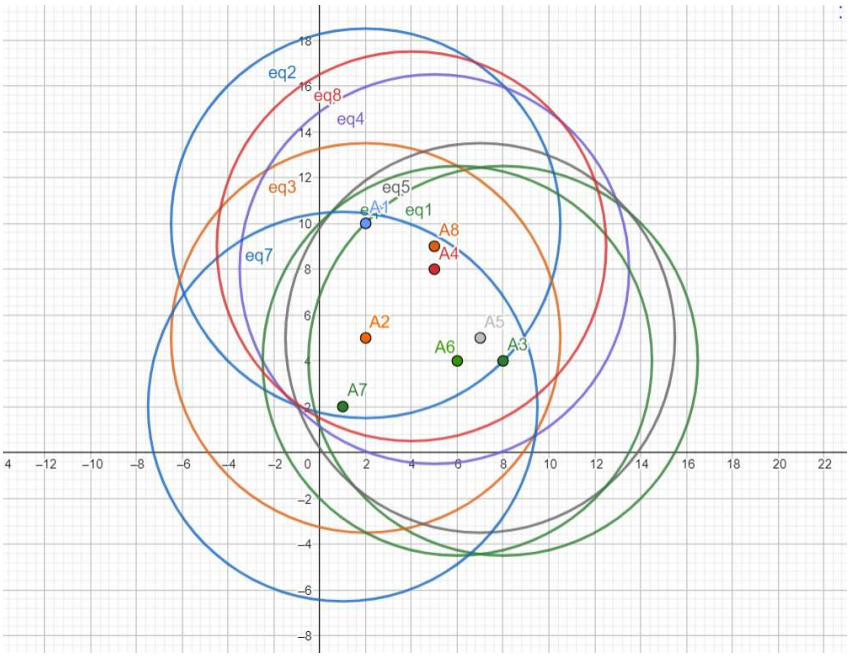
Q2 DBSCAN

1) Our group number is 34 therefore our epsilon is going to be $\varepsilon = \frac{34}{3+1} = \mathbf{8.5}$

This is a very big epsilon with regards to the dataset so even if we set the minimum number of the points to 8 which is the maximum number that we have in the dataset all points will be core points hence we will have only one cluster.

(We should have enrolled ourselves in a group of those who have number from 4 to 12 to be able to create better epsilon for this problem 😊)

2) We chose the 8 to be the minpoints per each epsilon.
3) Referring back to the distance matrix ALL of the distances between all points are less than 8.5 therefore ALL points are core points and we have only one cluster.



Because of the value of the epsilon that we have chosen every circle contains all of the other points in the dataset (i.e. all points belong to ONLY ONE CLUSTER)

2) If we choose minpoints =2 and Epsilon= √10, compare the results with Q1.2 (k-means). Draw the 10 by 10 space and illustrate the discovered clusters.

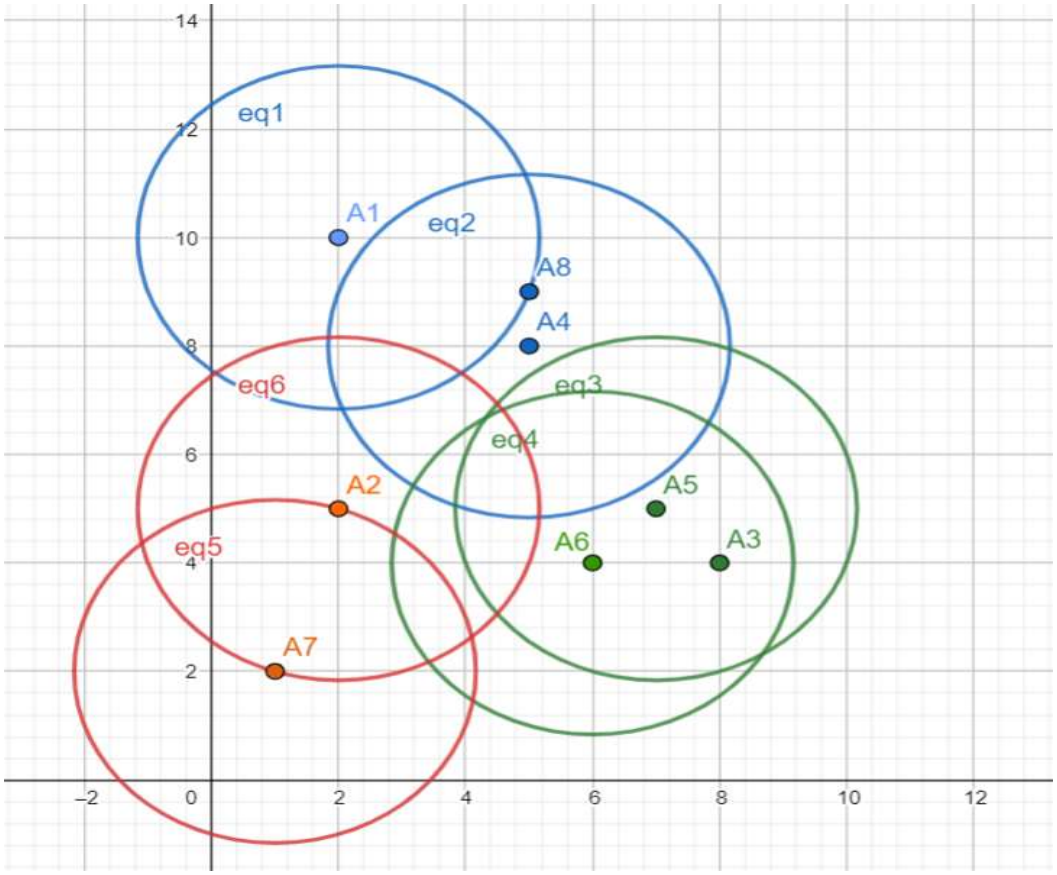
The first step was to determine the core, the borders, and the noise points. We did this via the following algorithm:

- Referring back to distance matrix we check the distance between each point and all of the other points in the dataset.
- Around each point we drew a circle with epsilon radius.
- If the circle around that point contains number of points more than or equal to the minpoints then this point is considered as a core point.
- if not, it's considered border or noise point.
- In our case all of the points were core points.

The second step was to identify the clusters we have in our dataset through the following algorithm.

- We chose any one of the core points that we label in the previous step and give it a number that represents its cluster.
- We checked the closest neighborhood point if it was a core as well, we assigned it to the same cluster as the first point.
- If the last core point did not have any neighbors without label (cluster number), then there were no more points belonged to that cluster.
- We did this for all points till we did not have any point without a label.

Here were the corresponding clusters:



Using DBSCAN clustering with $\varepsilon = \sqrt{10},$ resulted in the SAME clusters generated using K-means algorithm.

Programming Questions

Q4 choosing the best of k for k-means algorithm

1- We first imported the Pokemon data and after this we applied KElbowVisualizer function provided by YellowBrick to get the best number of clusters.

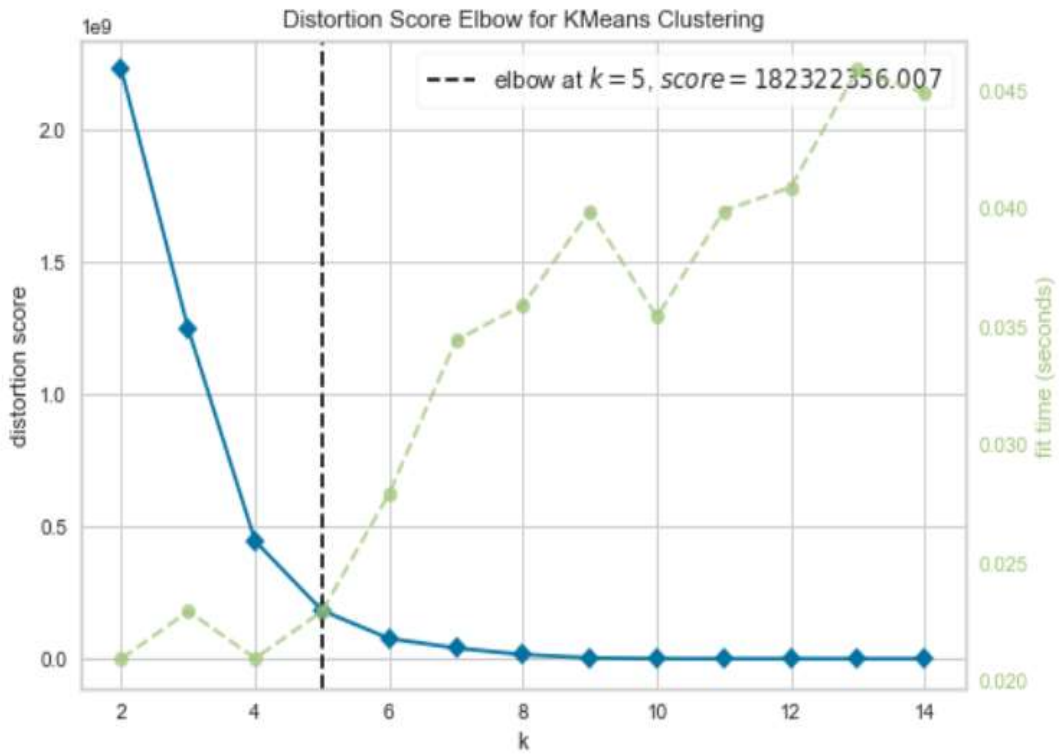
Importing Pokemon Data

```
Data=pd.read_csv('pokemon.csv')
Data=pd.DataFrame(Data)
Data= Data.drop(["type1"], axis=1)
```

choosing the best number of k for k-means algorithm using elbow method

```
model = KMeans()
visualizer = KElbowVisualizer(model, k=(2,15))

visualizer.fit(Data)        # Fit the data to the visualizer
visualizer.show()
```



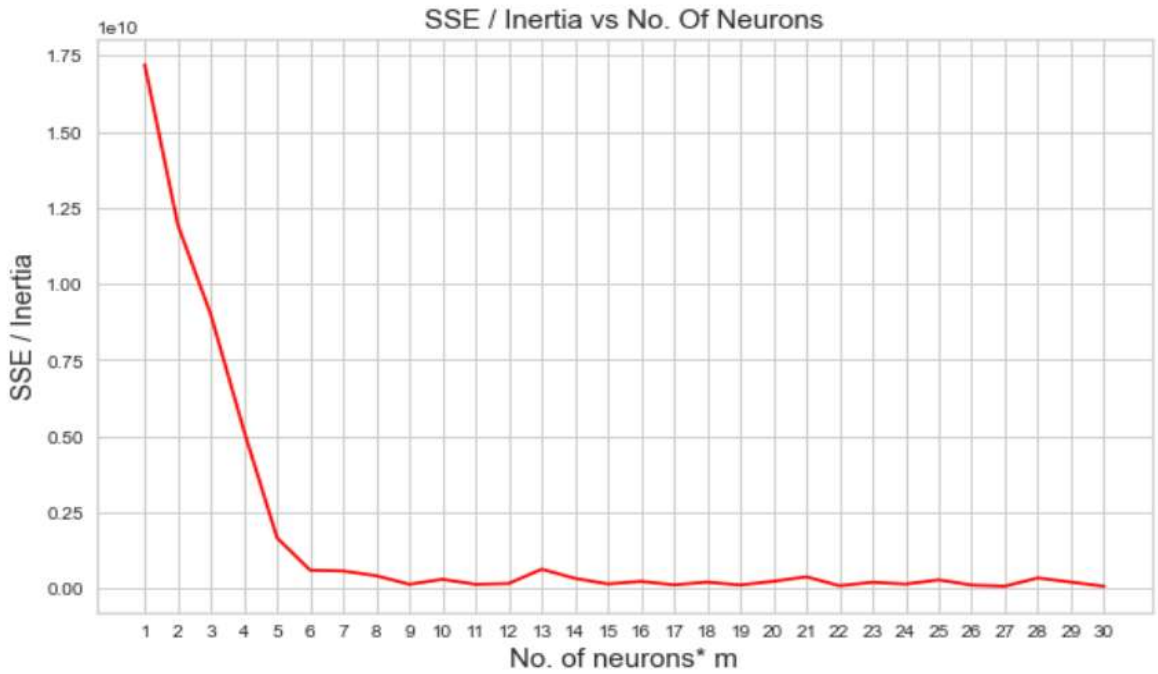
2- From the previous figure we can conclude that the optimal number of clusters is at **K=5**

Q5 Choosing the best number of neurons for SOM algorithm

1- We first created a function to which we pass the data and the dimension n of the SOM and it automatically creates a range for the other dimension of the SOM m so that the maximum number of neurons does NOT exceed the maximum number of neurons that we also pass to the function but it has a default of 30 maximum neurons.

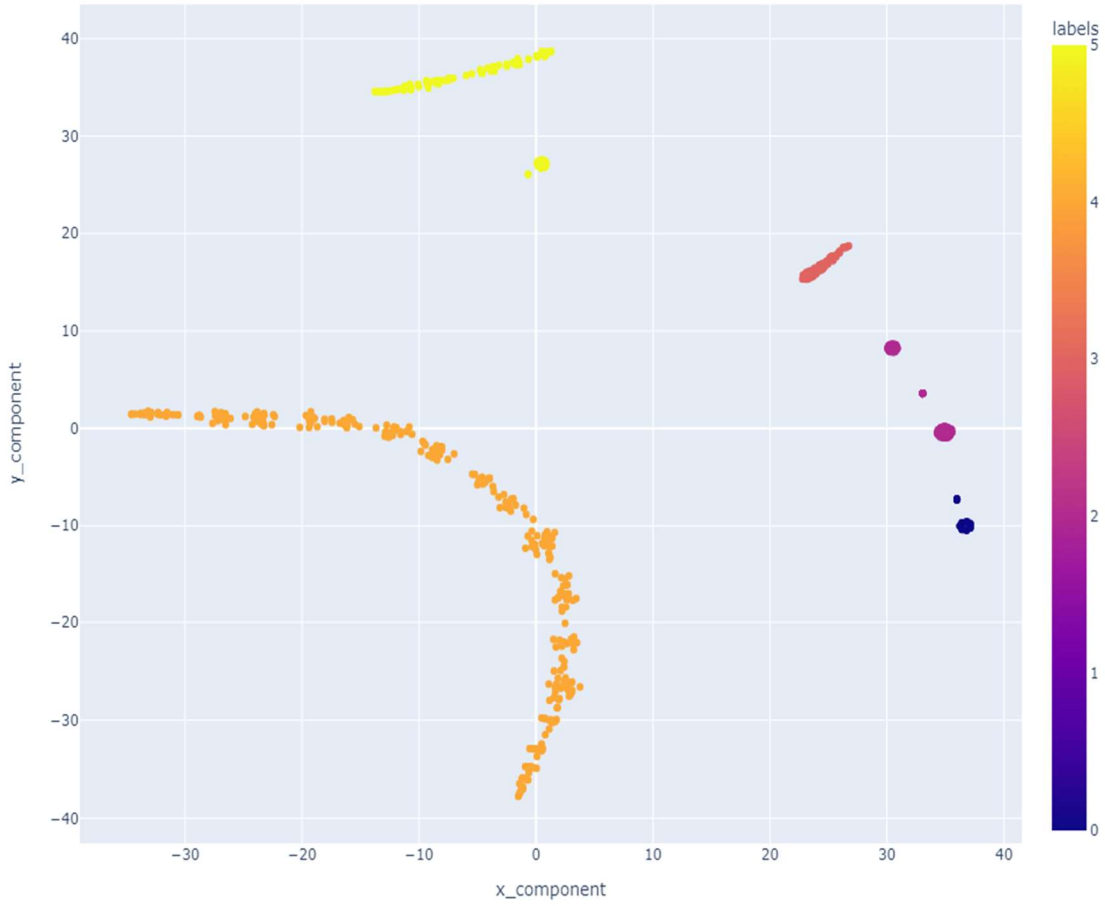
```
def draw_inertia_som(df,n, max_neurons=30 ):
    import random
    import matplotlib.pyplot as plt
    np.random.seed(555)
    df=df.to_numpy()
    inertia_values = []
    dim= df.shape[1]
    for i in range(1,int(max_neurons/n)+1):
        som = SOM(m=i, n=n, dim=dim)
        som.fit_predict(df)
        inertia_values.append(som.inertia_)
    fig, ax = plt.subplots(figsize=(10, 6))
    plt.plot(range(1,int(max_neurons/n)+1), inertia_values, color='red')
    plt.xlabel('No. of neurons* m', fontsize=15)
    plt.xticks(range(1,31))
    plt.ylabel('SSE / Inertia', fontsize=15)
    plt.title('SSE / Inertia vs No. Of Neurons', fontsize=15)
    plt.grid(True)
    plt.show()
```

```
draw_inertia_som(Data,1)
```

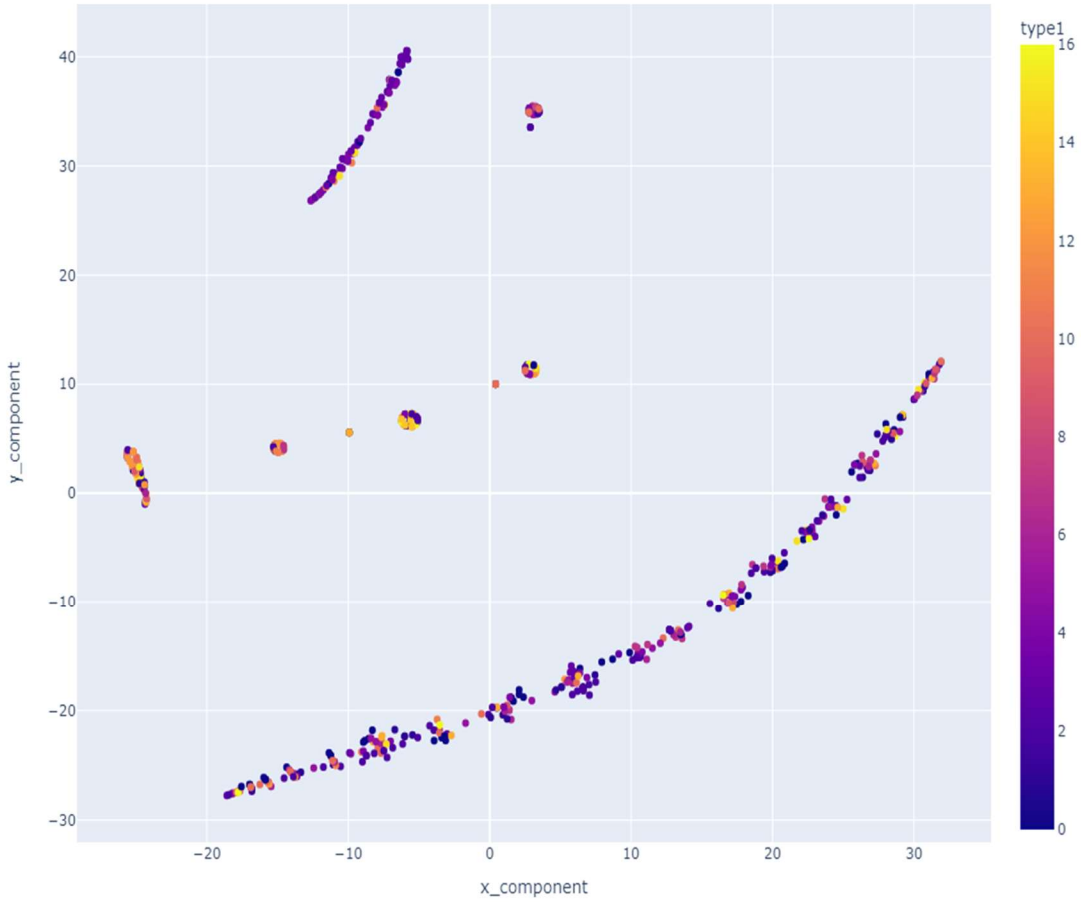


2- As we can see from the previous figure the best number of m when n is set to 1 is at **m=6** yet, we should try different numbers of n and for each n we should produce the optimal number of m and from all combination of n and m we have we should choose the one that has the lowest inertia.

3- Visualizing the result of SOM cluster using t-sne.



4- Before applying the SOM



As we can see that the SOM was verry effective in separating most of the clusters

Code:

Comparison between the data after and before appying SOM

Before applying SOM

```
]:
df=pd.DataFrame(pd.read_csv('pokemon.csv'))

df_without_label=df.drop(["type1"], axis=1)
tsne = TSNE(n_components=2, random_state=123)
X_embedded = TSNE(n_components=2).fit_transform(df_without_label)
df_without_label["type1"]=df["type1"].factorize()[0]
df_without_label["x_component"]=X_embedded[:,0]
df_without_label["y_component"]=X_embedded[:,1]
```

```
]: import plotly.express as px

fig = px.scatter(df_without_label, x="x_component", y="y_component", color = "type1", size_max=60)
fig.update_layout(
     height=800)
fig.show()
```

After applying SOM

```
df_som=Data

df_som=df_som.to_numpy()
np.random.seed(555)
som = SOM(m=6, n=1, dim=20)
labels=som.fit_predict(df_som)


df_som=pd.DataFrame(df_som)

df_som["labels"]=labels
```

```
tsne = TSNE(n_components=2, random_state=123)
X_embedded_som = TSNE(n_components=2).fit_transform(df_som)

df_som["x_component"]=X_embedded_som[:,0]
df_som["y_component"]=X_embedded_som[:,1]
```

```
fig = px.scatter(df_som, x="x_component", y="y_component", color = "labels", size_max=60)
fig.update_layout(
     height=800)
fig.show()
```

Q6 (Tunning DBSCAN)

At first, we set the value of the min_samples to (10) and iterate on the value of the epsilon from (0.1) to (0.9) we found out that the best value closer to 10 clusters is 0.7 giving 11 clusters

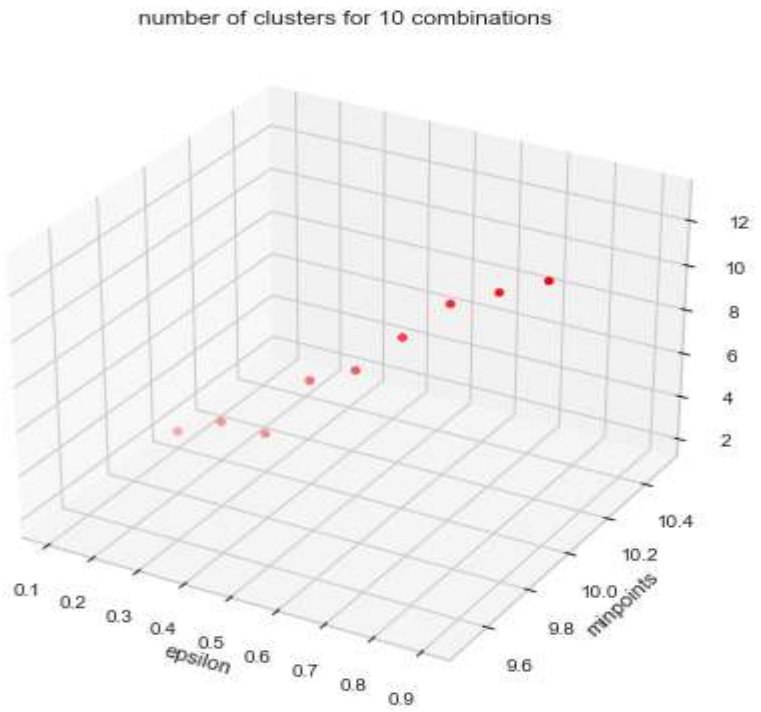
```
X = StandardScaler().fit_transform(Data)
```

```
n_clusters=[]
for i in range(1,10):
    db = DBSCAN(eps=i/10, min_samples=10).fit(X)
    uniqueValues, indicesList=np.unique(db.labels_, return_counts=True)
    n = len(uniqueValues)
    n_clusters.append(n)
    print(n)


# Creating figure
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")

# Creating plot
x_axis=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
ax.scatter3D(x_axis ,10, n_clusters, color = "red")
plt.title("number of clusters for 10 combinations")
plt.xlabel('epsilon')
plt.ylabel('minpoints')


# show plot
plt.show()
```



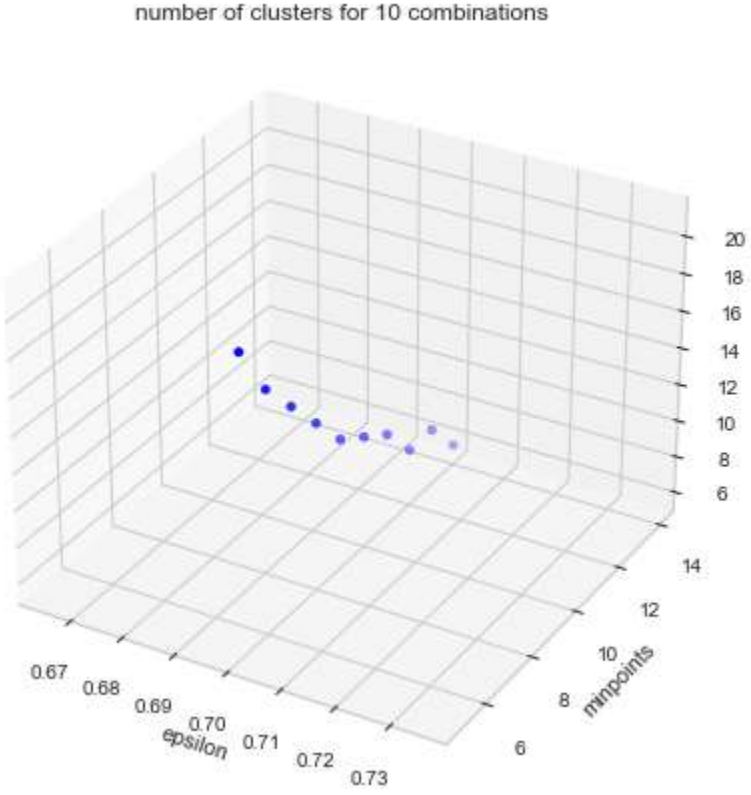
then we set the value of the epsilon and iterated on the value of the min_samples from (5) to (15) we found out that the value of 11 gives us exactly 10 clusters

```
n_clusters=[]
for i in range(5,15):
  db = DBSCAN(eps=0.7, min_samples=i).fit(X)
  uniqueValues, indicesList=np.unique(db.labels_, return_counts=True)
  n = len(uniqueValues)
  n_clusters.append(n)
  print(n)


# Creating figure
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")

# Creating plot
ax.scatter3D(0.7 ,range(5,15), n_clusters, color = "blue")
plt.title("number of clusters for 10 combinations")
plt.xlabel('epsilon')
plt.ylabel('minpoints')


# show plot
plt.show()
```

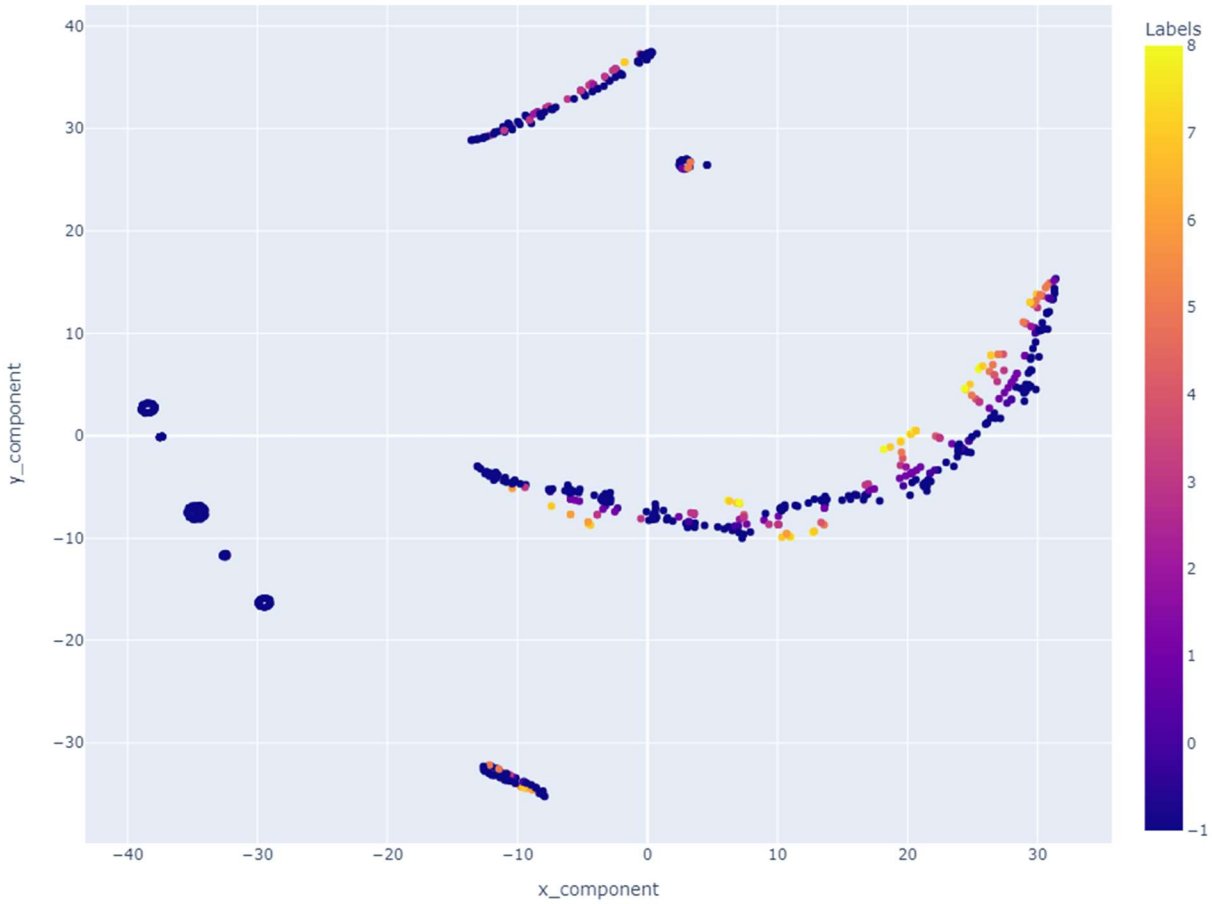

number of clusters for 10 combinations
20
18
16
14
12
10
8
6
14
12
10
8
6
minpoints
0.67
0.68
0.69
0.70
0.71
0.72
0.73
epsilon

now after assigning the parameters (eps=0.7, min_samples=11) we will plot the clusters

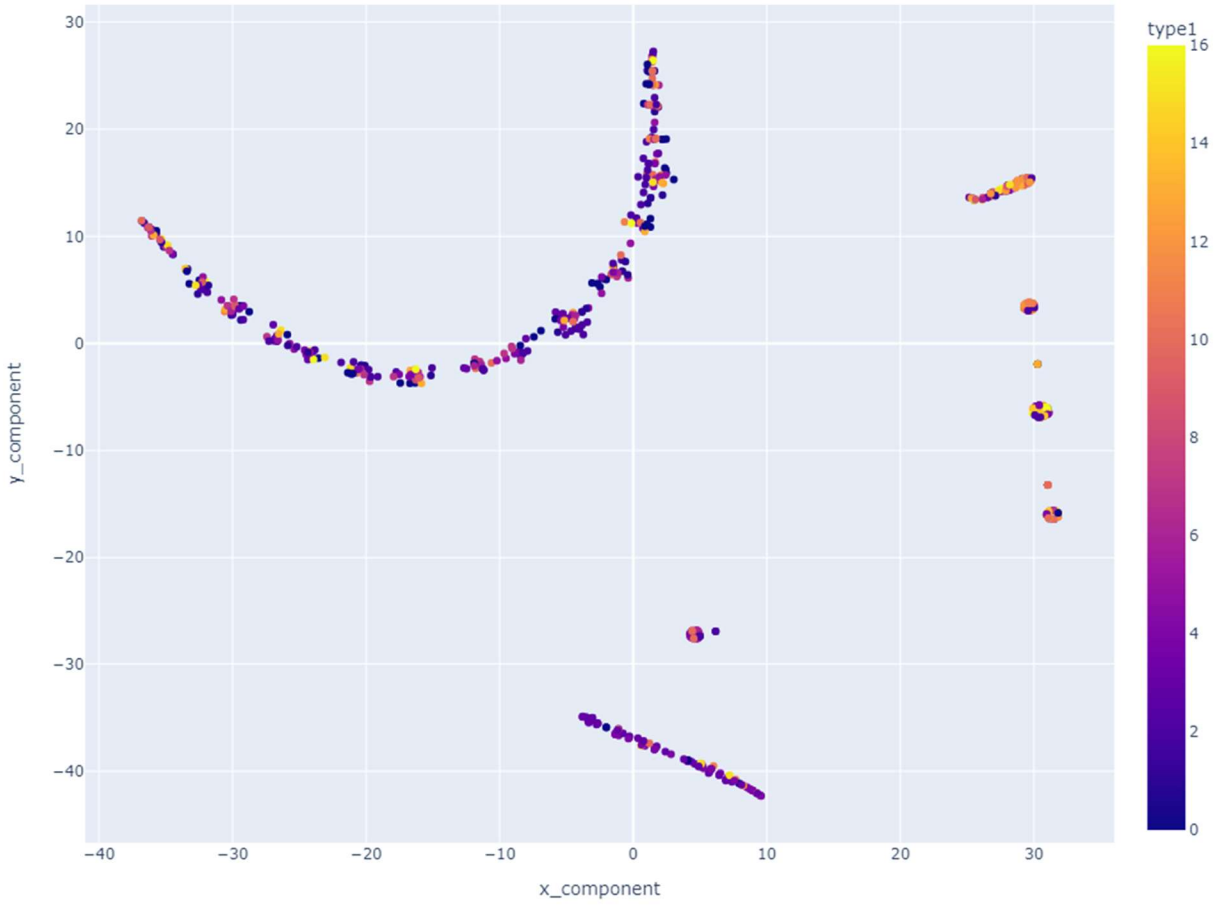
in order to do that we will need to reduce the dimensionality of the features into 2 components

using the T-SNE feature reduction algorithm it's all set.

This was the figure of resulted clusters



This was be fore applying the DBSCAN clustering



As we can see that the clusters were slightly separated from each other.