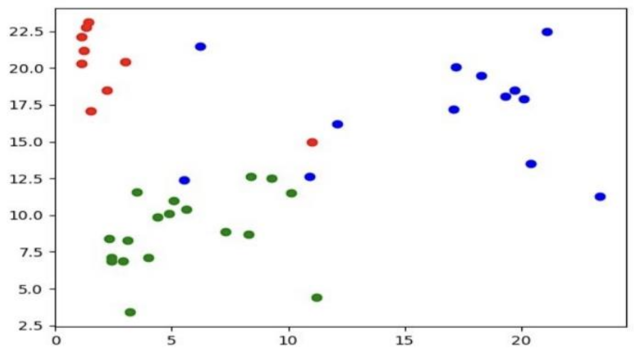
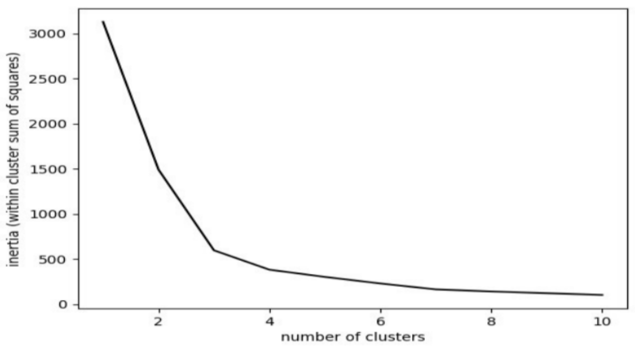
Question 1: K-Means Clustering

1. Visualise the original data points with different colours for their original cluster labels in a scatter plot.



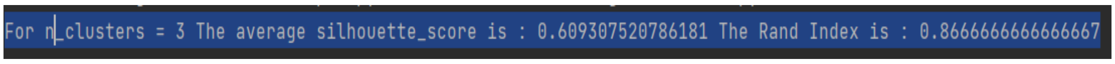
colors = ‘r’,‘g’,‘b’

2. Using x and y, use the k-means algorithm to cluster the dataset into x (from 1 to 10) number of clusters. Plot inertia (within cluster sum of squares) against the number of clusters. What is the best number of clusters for this data?



As can be seen from the figure, as the number of clusters increases, inertia decreases. The best number of clusters for this data is 10.

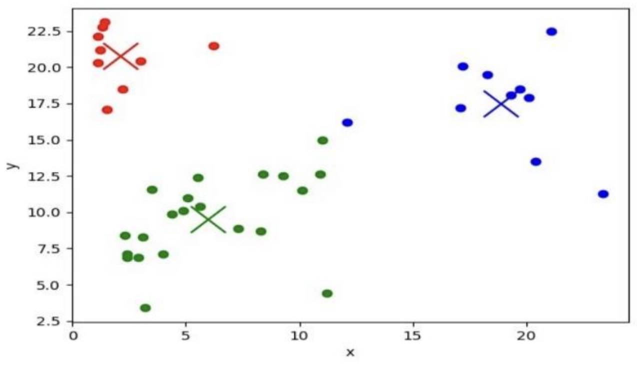
3. Calculate the Rand Index as an extrinsic measure (i.e. when are know the original/groud-truth clusters), and Silhouette Score as an intrinsic measure (unsupervised) for the given dataset with 3 clusters.



```
For n_clusters = 3 The average silhouette_score is : 0.609307520786181 The Rand Index is : 0.8666666666666667
```

For n_clusters = 3 The average silhouette_score is : 0.609307520786181 The Rand Index is : 0.8666666666666667.
Use silhouette_score, rand_score classes in sklearn.metrics.

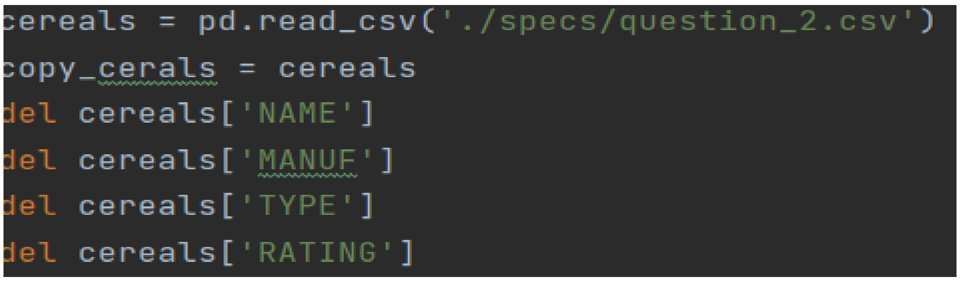
5. Plots the clustering results (including the centroids) as in Q1.1



The new clustering result is shown in the figure, where ‘X’ represents the centroids.

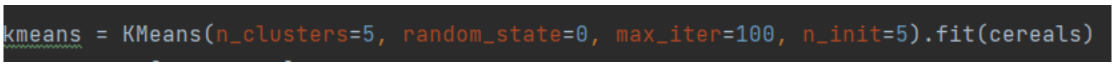
Question 2: K-Means Clustering

1. Discard the columns NAME, MANUF, TYPE, and RATING.



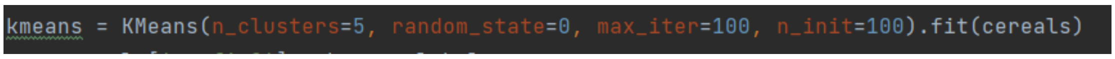
```
cereals = pd.read_csv('./specs/question_2.csv')
copy_cerals = cereals
del cereals['NAME']
del cereals['MANUF']
del cereals['TYPE']
del cereals['RATING']
```

2. Run the k-means algorithm using 5 clusters as a target, 5 maximum runs, and 100 maximum optimization steps. Keep the random state to 0. Save the cluster labels in a new column called config1.



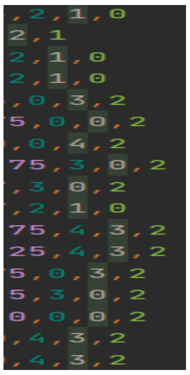
```
kmeans = KMeans(n_clusters=5, random_state=0, max_iter=100, n_init=5).fit(cereals)
```

3. Run k-means again, but this time use 100 maximum runs and 100 maximum optimization steps. Again, use a random state of 0. Save the cluster labels in a new column called config2.



```
kmeans = KMeans(n_clusters=5, random_state=0, max_iter=100, n_init=100).fit(cereals)
```

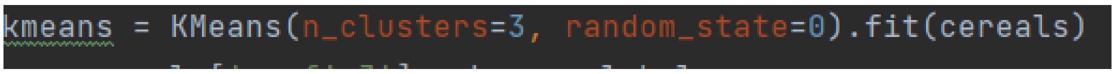
4. Are the clustering results obtained with the first configuration different from the results obtained with the second configuration? Explain your answer in your report.



```
,2,1,0
2,1
2,1,0
2,1,0
,0,3,2
5,0,0,2
,0,4,2
75,3,0,2
,3,0,2
,2,1,0
75,4,3,2
25,4,3,2
5,0,3,2
5,3,0,2
0,0,0,2
,4,3,2
,4,3,2
```

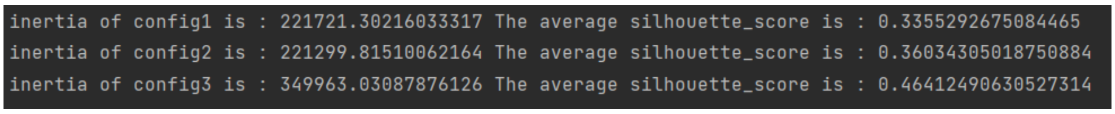
The clustering results obtained with different configuration are different (deep green and pink). They use different numbers of time that the k-means algorithm will be run with different centroid seeds, The more times, the more likely it is to get the best output.

5. Run the clustering algorithm again, but this time use only 3 clusters. Save the generated cluster labels in a new column called config3.



```
kmeans = KMeans(n_clusters=3, random_state=0).fit(cereals)
```

6. Which clustering solution is better? Discuss it in your report.

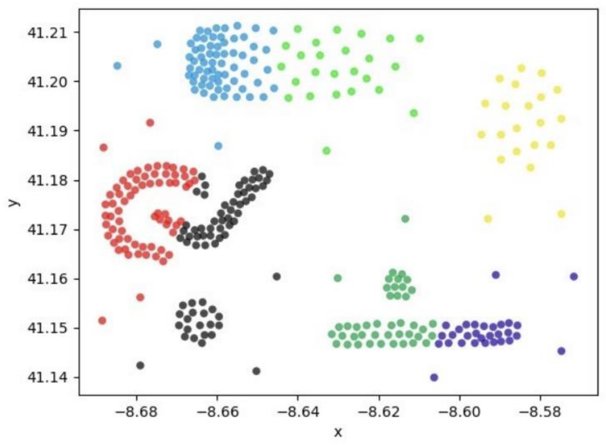


```
inertia of config1 is : 221721.30216033317 The average silhouette_score is : 0.3355292675084465
inertia of config2 is : 221299.81510062164 The average silhouette_score is : 0.36034305018750884
inertia of config3 is : 349963.03087876126 The average silhouette_score is : 0.46412490630527314
```

Inertia of config2 is the best, while the silhouette_score of config3 is better.

Question 3: DBSCAN Clustering Algorithm

1. Discard the ID column, the use the X and Y coordinates as data input to the K-Means algorithm to cluster it into 7 clusters. Perform 5 maximum runs, and 100 maximum optimization steps. Keep a random state to 0.



As can be seen from the figure, the clustering results are not good. Different clusters are very close.

3. Normalize the X and Y columns in a range between 0 and 1, then use the DBSCAN algorithm to cluster the points again. Use a value of 0.4 for epsilon, and set the 3 minimum points equals to 4.

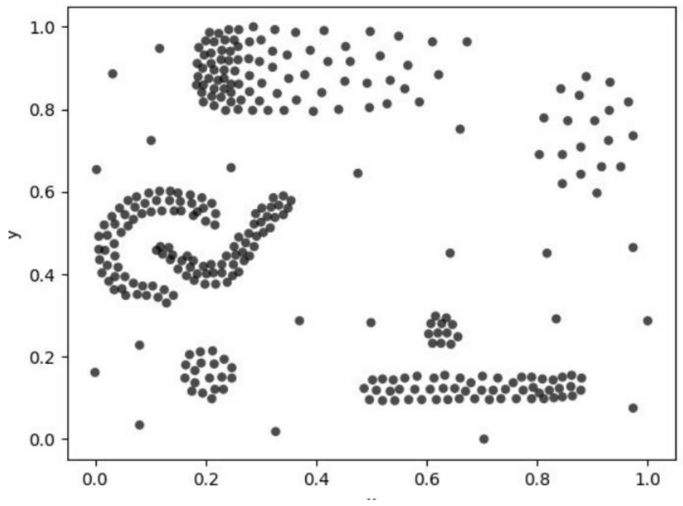
(1) 0-1 Normalize

```
data_norm = (data - data.min()) / (data.max() - data.min())
```

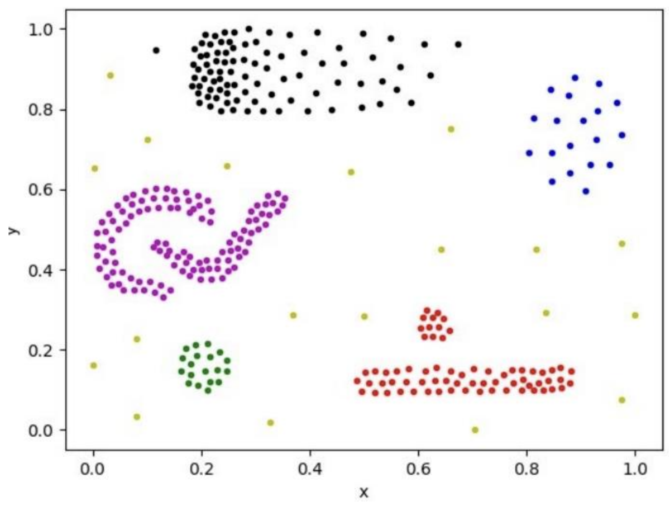
(2) Train and predict

```
dbscan = DBSCAN(eps=0.4, min_samples=4).fit(data_norm[['x', 'y']])
```

(3) plot the figure



4. Execute DBSCAN again, but this time use a value of 0.08 for epsilon.



6. Discuss the different clustering solutions in your report. Which solution is the best? What is the reason behind the dereferences in the results?

Obviously the second result is better. The first configuration did not work at all. Probably because after 0-1 normalization, the distance between data points becomes smaller. Therefore, setting a value of 0.4 for epsilon is relatively large. For this algorithm, most samples are considered to be near another sample, so almost all samples belong to the same one cluster.