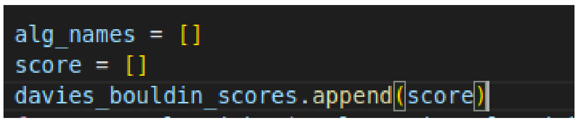
Lab 5 – Unsupervised Learning

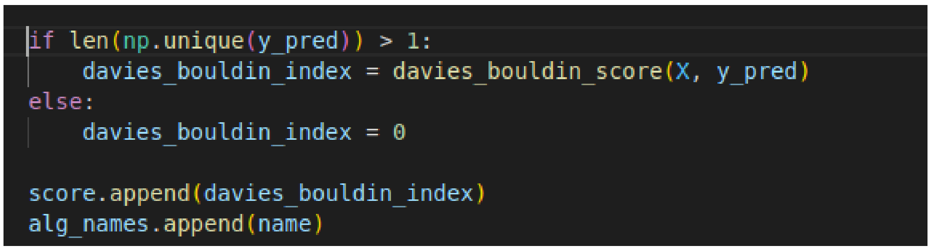
Author: Gonçalo Aguiar 904475

Task 1: Go to the website https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html and run a program that tests different clustering methods. Analyze the features of these methods - their parameters, typical applications, and metrics they use - summarized in a table in section 2.3.1 under the link: https://scikit-learn.org/stable/modules/clustering.html. Finally, try to add to this code the calculation of clustering quality using the Davies-Bouldin index, described in section 2.3.10.7 under the above link. Display the results in the form of a bar chart. Which method gave the best result according to this measure for different types of clusters? Do visual inspection of clustering correctness and Davies-Bouldin index give consistent results?

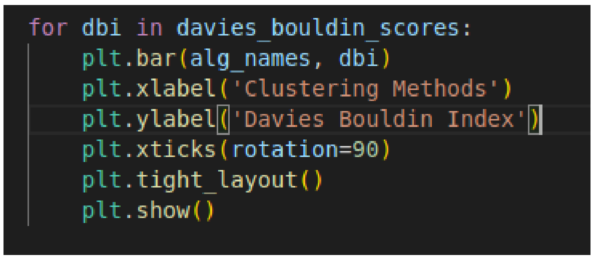
In order to complete the given code the following parts were added:



```
alg_names = []
score = []
davies_bouldin_scores.append(score)
```

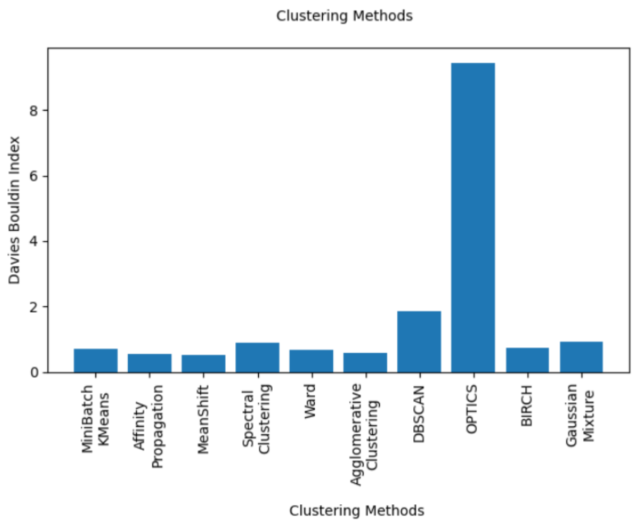


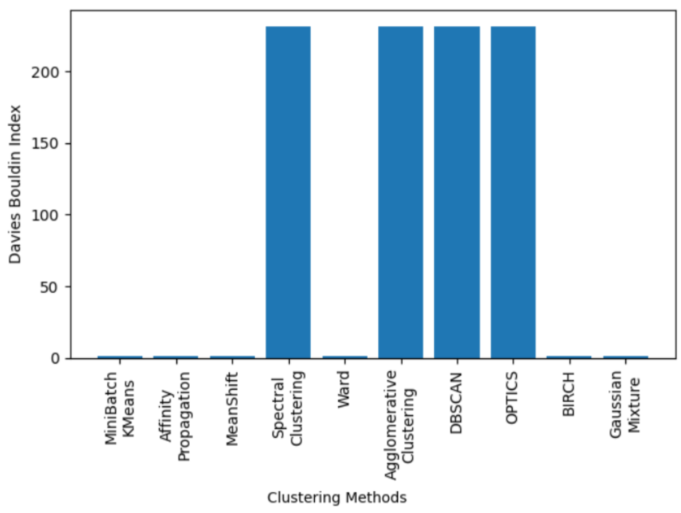
```
if len(np.unique(y_pred)) > 1:
    davies_bouldin_index = davies_bouldin_score(X, y_pred)
else:
    davies_bouldin_index = 0

score.append(davies_bouldin_index)
alg_names.append(name)
```

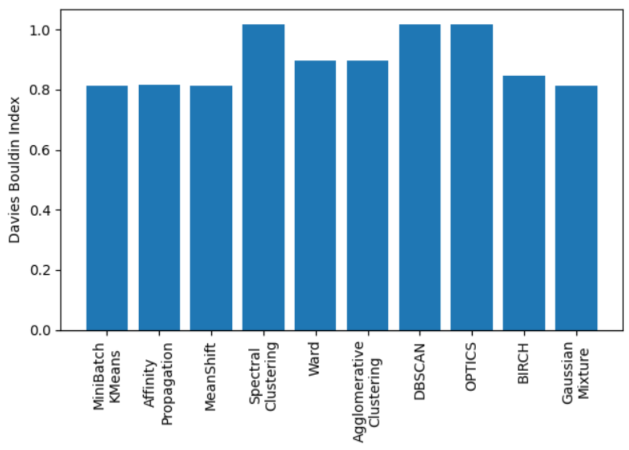


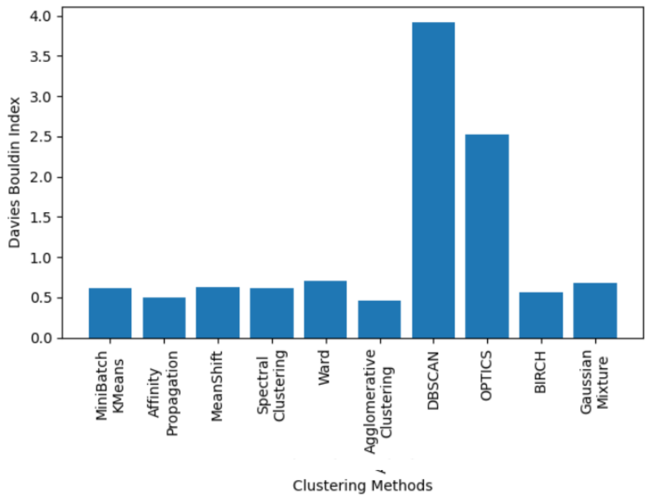
```
for dbi in davies_bouldin_scores:
    plt.bar(alg_names, dbi)
    plt.xlabel('Clustering Methods')
    plt.ylabel('Davies Bouldin Index')
    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()
```

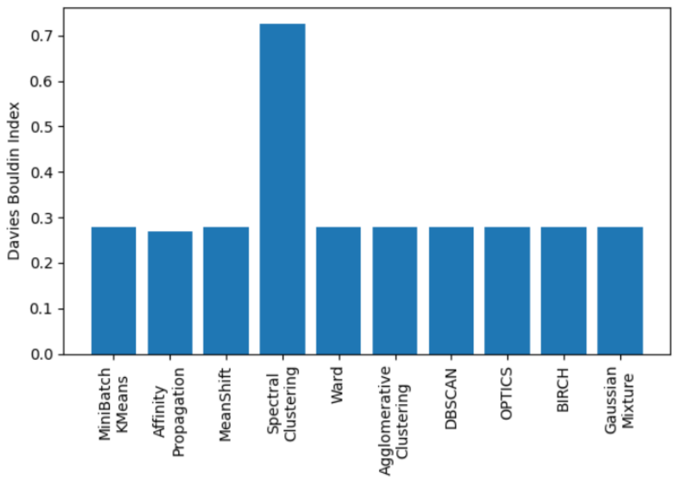
The result was the following plots(for each of the given datasets):

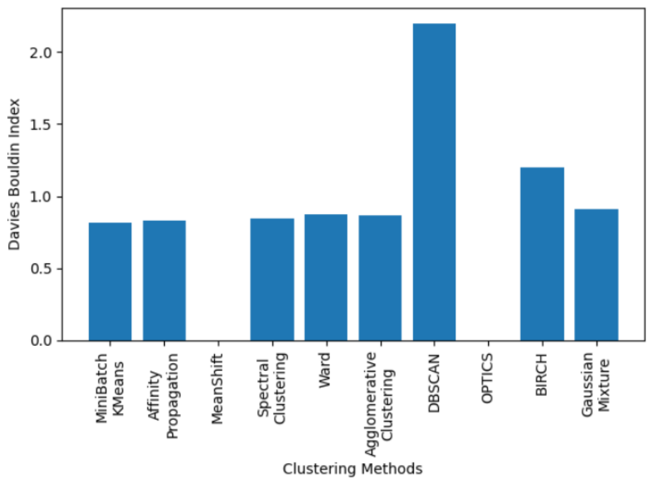










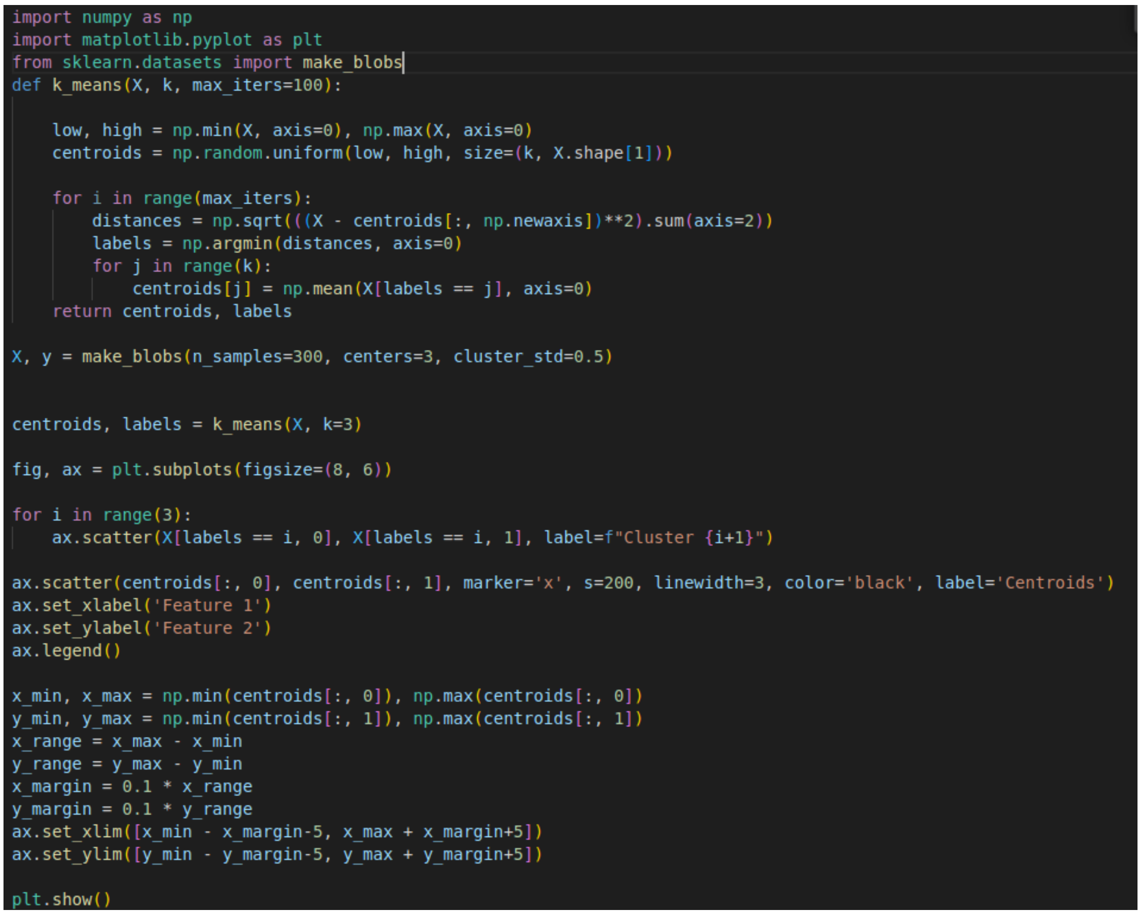


The method with the best result is the MeanShift because it is the one that as the lower average values if we consider all the cases.

Regarding if visual inspection of clustering correctness and Davies-Bouldin index give consistent results, we can see that sometimes the results that both give aren't consistent. For example in the last dataset the MeanShift got a DBI score of 0 but if we look at the plot we can see that the result is wrong. So it's important to use both methods to evaluate the clustering result.

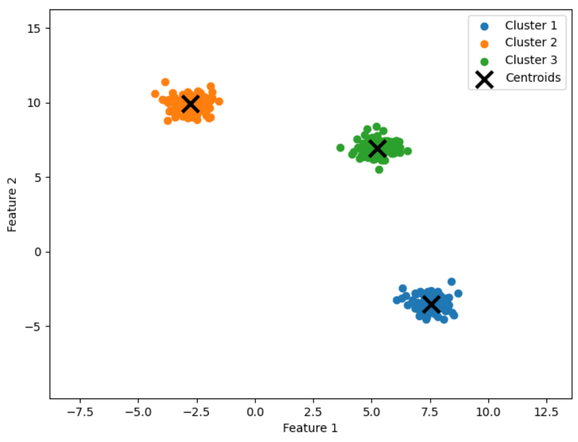
Task 3: Implement the k-means method on your own (without using ready-made implementations from the internet). To do this, generate artificially clustered data using the sklearn.datasets.make_blobs function. You can decide on the number, variance, and location of the generated clusters, as well as the number of samples in each of them. The implementation should allow you to change the target number of clusters k and work for any number of features - however, for convenient visualization, check the correctness of the method's operation for two or three features. You can generate initial cluster centroids using the np.random.uniform function, taking into account the low and high parameters of this function according to the range of variability of features in the data set (the goal is to generate centroids that will be located somewhere within the data points).

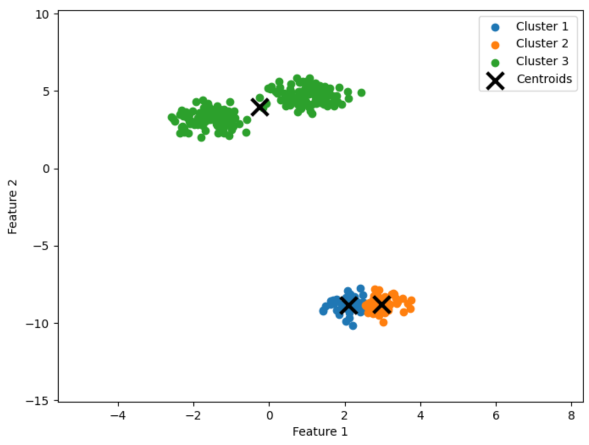
In order to complete the task the following code was written:

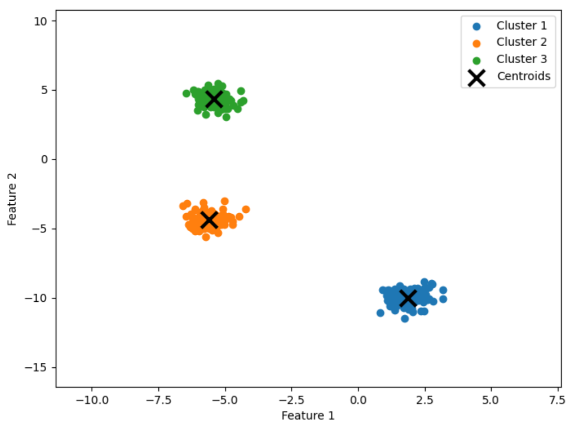


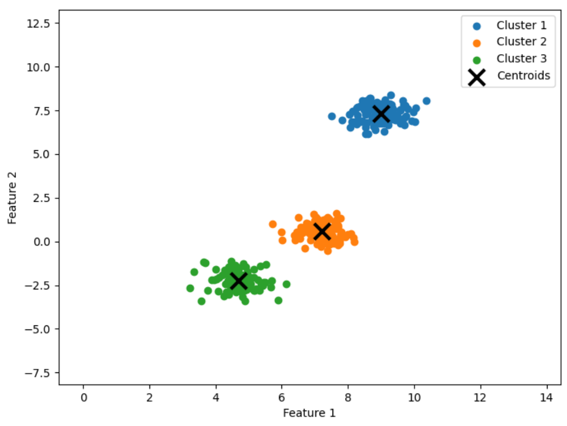
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
def k_means(X, k, max_iters=100):

    low, high = np.min(X, axis=0), np.max(X, axis=0)
    centroids = np.random.uniform(low, high, size=(k, X.shape[1]))

    for i in range(max_iters):
        distances = np.sqrt(((X - centroids[:, np.newaxis])**2).sum(axis=2))
        labels = np.argmin(distances, axis=0)
        for j in range(k):
            centroids[j] = np.mean(X[labels == j], axis=0)
    return centroids, labels

X, y = make_blobs(n_samples=300, centers=3, cluster_std=0.5)


centroids, labels = k_means(X, k=3)

fig, ax = plt.subplots(figsize=(8, 6))

for i in range(3):
    ax.scatter(X[labels == i, 0], X[labels == i, 1], label=f"Cluster {i+1}")

ax.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200, linewidth=3, color='black', label='Centroids')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.legend()

x_min, x_max = np.min(centroids[:, 0]), np.max(centroids[:, 0])
y_min, y_max = np.min(centroids[:, 1]), np.max(centroids[:, 1])
x_range = x_max - x_min
y_range = y_max - y_min
x_margin = 0.1 * x_range
y_margin = 0.1 * y_range
ax.set_xlim([x_min - x_margin-5, x_max + x_margin+5])
ax.set_ylim([y_min - y_margin-5, y_max + y_margin+5])

plt.show()
```

Some of the results were:


Cluster 1
Cluster 2
Cluster 3
Centroids
Feature 1
Feature 2


Cluster 1
Cluster 2
Cluster 3
Centroids
Feature 1
Feature 2


Cluster 1
Cluster 2
Cluster 3
Centroids
Feature 1
Feature 2


Cluster 1
Cluster 2
Cluster 3
Centroids
Feature 1
Feature 2