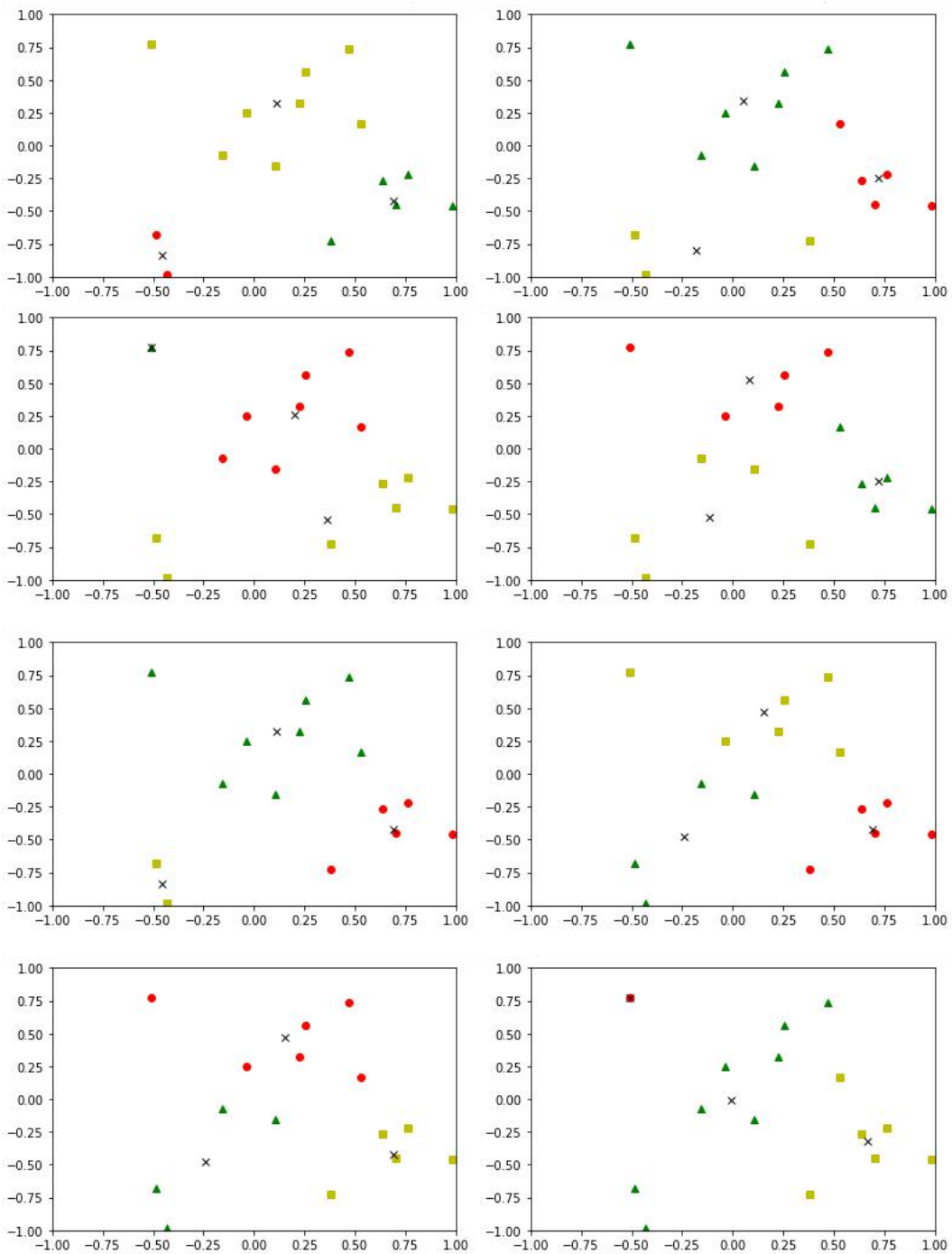
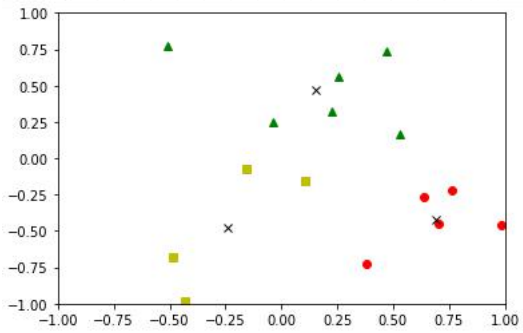
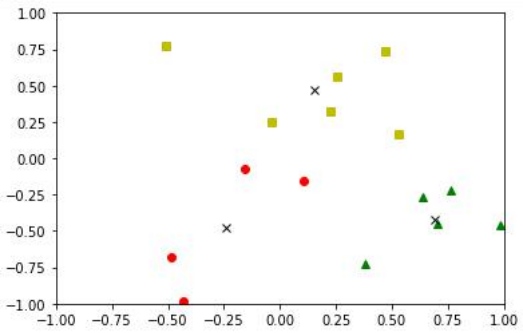
Have a look at the simple k-means program you have been given. Note, it can look for clusters of different ks; but for plotting purposes it fixed to look for k=3. The program can work from a randomly generated set of points or a defined set.

Q1： Use the init_board fn to randomly generate 15 points; store this output and set the data variable to it.

Now run this set 10 times and note the clusters found by k-means.

Report the results of these runs and the extent to which the same clusters are found.



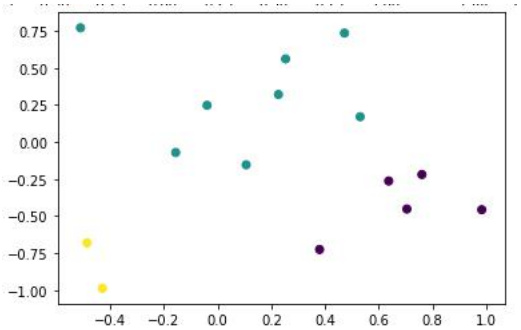



The coordinates of the 15 points are:

```
[[-0.48655663 -0.68089407]
 [-0.03957789  0.2470585 ]
 [ 0.47216899  0.73433396]
 [ 0.70440037 -0.452346  ]
 [ 0.98336695 -0.45695733]
 [-0.42963869 -0.98722538]
 [ 0.63681214 -0.26343467]
 [-0.15636626 -0.07109884]
 [ 0.53049172  0.16995934]
 [ 0.37938582 -0.72532696]
 [ 0.10593223 -0.15411577]
 [ 0.76036982 -0.21952132]
 [ 0.22618451  0.32020366]
 [-0.51082014  0.76997145]
 [ 0.25288438  0.5602143 ]]
```

In this experiment , the k was set to three, therefore there are three clusters in each of these plots. From these results, I found that k-means method would produces different answers on different runs, specifically, in terms of the different results, the cluster center would be different points and some points would be set into different clusters in different run. After looking at the principle of the k-means program I think the probable reason to this problem could be that when kmeans starts it picks up the random initial points to cluster the data. Moreover, I found that the figure 1 and figure 5 have the same result. And the figure 6,7,9,10 has the same result.

As shown in the following figure, is the result generated by KMeans function in sklearn.cluster,

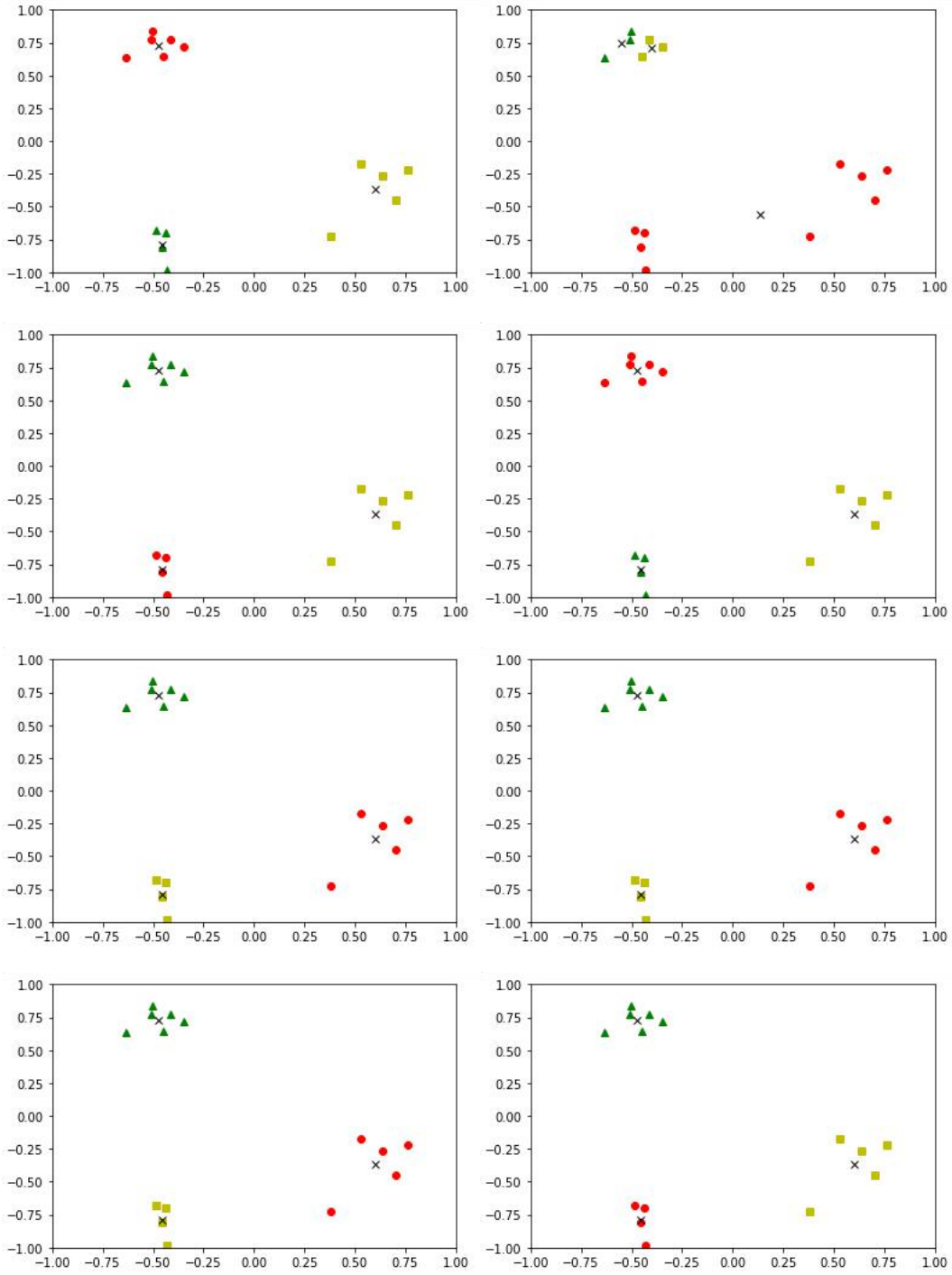


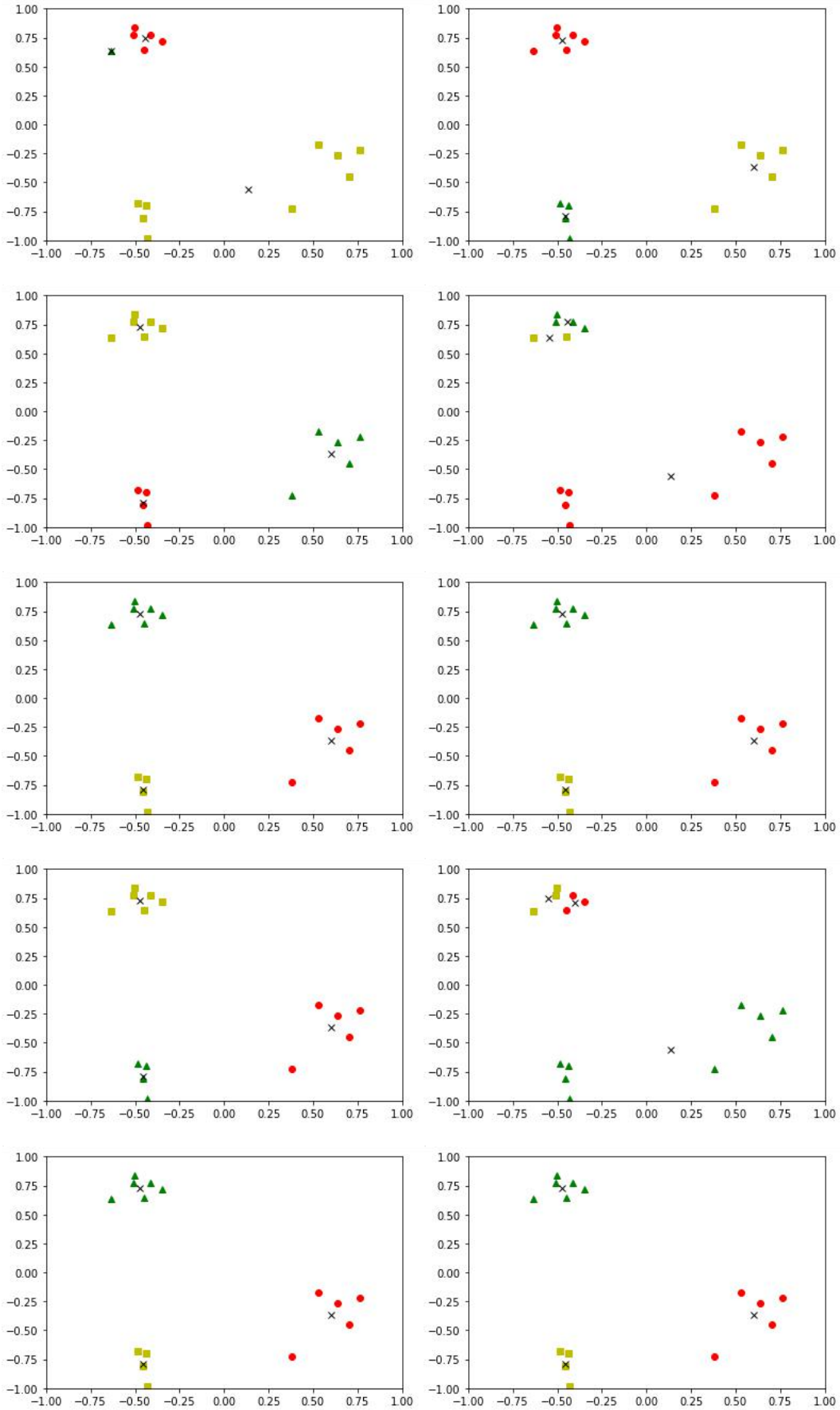
We can see that it has the same cluster with the figure 1 and figure5, so the figure 1 and figure 5 should be the better cluster result.

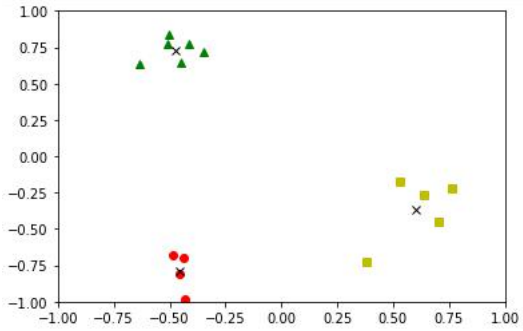
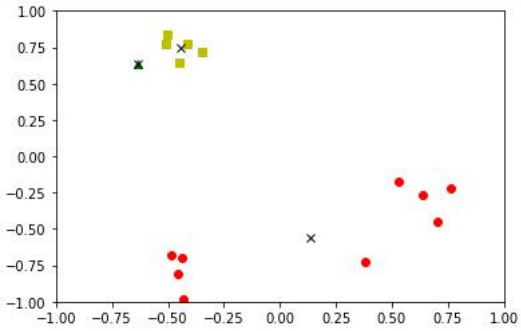
Q2： Now, create you own set of data, again with 15 points. You should construct this data-set to have very clear clusters (a bit like the simple 6-point example shown).

Now run this set 20 times and note the clusters found by k-means.

Report the results of these runs and the extent to which the same clusters are found.




1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

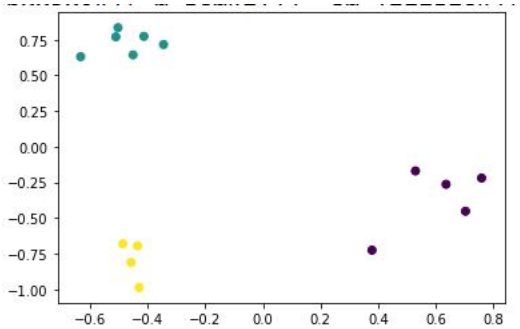
The coordinates of the 15 points are:

```
[-0.48655663, -0.68089407]
[-0.42963869, -0.98722538]
[-0.43563213, -0.69444201]
[-0.45777777, -0.81134551]
[-0.51082014,  0.76997145]
[-0.50341145,  0.83457819]
[-0.34513122,  0.71636418]
[-0.63311345,  0.63123311]
[-0.41343444,  0.77337741]
[-0.45111111,  0.64355411]
[ 0.37938582, -0.72532696]
[ 0.70440037, -0.452346  ]
[ 0.63681214, -0.26343467]
[ 0.76036982, -0.21952132]
[ 0.53049172, -0.16995934]
```

In this experiment,I set my data set to those have clear clusters.

From the results generated by k-means, I found that the extent to which the same clusters are found is much higher than the former experiment. The figure 1,4,5,6,7,8,10,11,13,15,17,18,19, they all have the same cluster result which divides the whole data set into three distinct clusters. The possible reason to this is that the because the tighter the data, the greater the probability of random sampling will set initial center point to the point located in the interior, the classifier will naturally divide the clusters distinctly. I also think that the reason that the rest of the result was different from the result above is that there is a outlier [0.37938582 -0.72532696] in this data set, and when the initial center point was randomly set to this point, then the method will perform poorly due to this outlier.

I also performed the k-means function in sklearn.cluster. It can be seen that it is the same as the figure I firstly mentioned. So the result 1,4,5,6,7,8,10,11,13,15,17,18,19 should be better cluster result.



Q3： Do some research on the problem that k-means produces different answers on different runs.

Describe two typical solutions to this problem with references to the literature you read to answer his question.

The K-mean algorithm has a problem that the result varies depending on the initial value, but the initial center point was chosen randomly, so it leads to the problem that k-means would produce different answers on different runs. It is a disadvantage of the k-Mean algorithm that different results can be obtained each time, even if the k-Mean algorithm is applied to the same data with the same number of point.

There are two solutions to solve the weakness in k-means.

The first one is using The k-means library provided by Python scikit learn. We can overcome the drawback of random initial value by setting the n_init parameter. Because that it is the number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia. Therefore we can overcome this by iterating the initial values and iterating through the algorithm.

The second solution is using k-means++. It specifies a procedure to initialize the cluster centers before moving forward with the standard k-means clustering algorithm. The basic idea of k-means++ algorithm to select initial seeds is that the distance between the initial cluster centers should be as far as possible. The first step in k-means++ is to randomly pick a data point as a cluster centroid , and then The next centroid will be the one whose squared distance is the farthest from the current centroid. Then calculate the squared distance of current centroids, and set the next centroid to the point which is the farthest in these points. The rest procedure will be done in the same manner.

We now can continue with the K-Means algorithm after initializing the centroids. Using K-Means++ to initialize the centroids tends to improve the clusters.

References:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

https://www.geeksforgeeks.org/ml-k-means-algorithm/