Analytical Report

Yulin Zhou
i6336858

October 16, 2023

1 Task 1

1.1 Part a: Data Generation

Firstly, I used the following codes to generate 4 Gaussian clusters in two-dimensional space in the range of [-10, +10] and with different means but the same standard deviation(i.e., 0.6). The visualization of the generated data is shown in Fig.1. As we can see, there are four clusters from the naked eye.

```
# define the sd
sd = 0.6

# define the total generated instances
total_num = 300

# generate the samples (4 centers with 2 features)
x_y, group = make_blobs(n_samples=total_num, centers=4, random_state=0,
                        n_features=2, cluster_std = sd, center_box = (-10, 10) )

## Transform the generated information into dataframe
df_x_y = pd.DataFrame(x_y, columns = ['x','y'])
df_x_y['True group'] = group
```

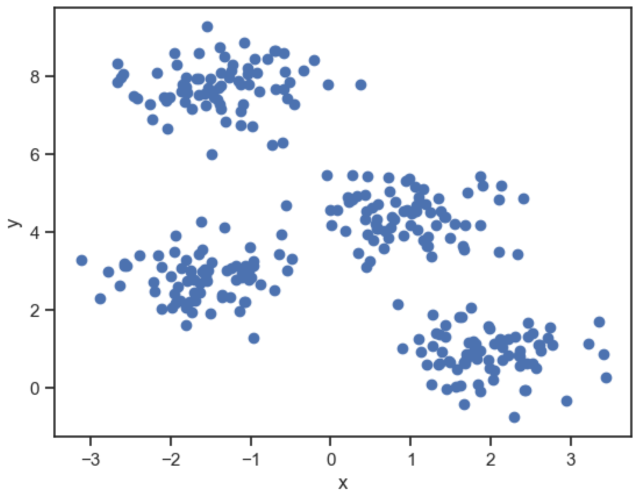


Figure 1: Scatter plot of the generated data

1.2 Part b: Run k-means clustering algorithm

For k in {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, I used the 'for' loop to run 'KMeans' function on the generated data. The visualization of the outcome is shown in Fig.2. The center of each cluster is represented by red points.

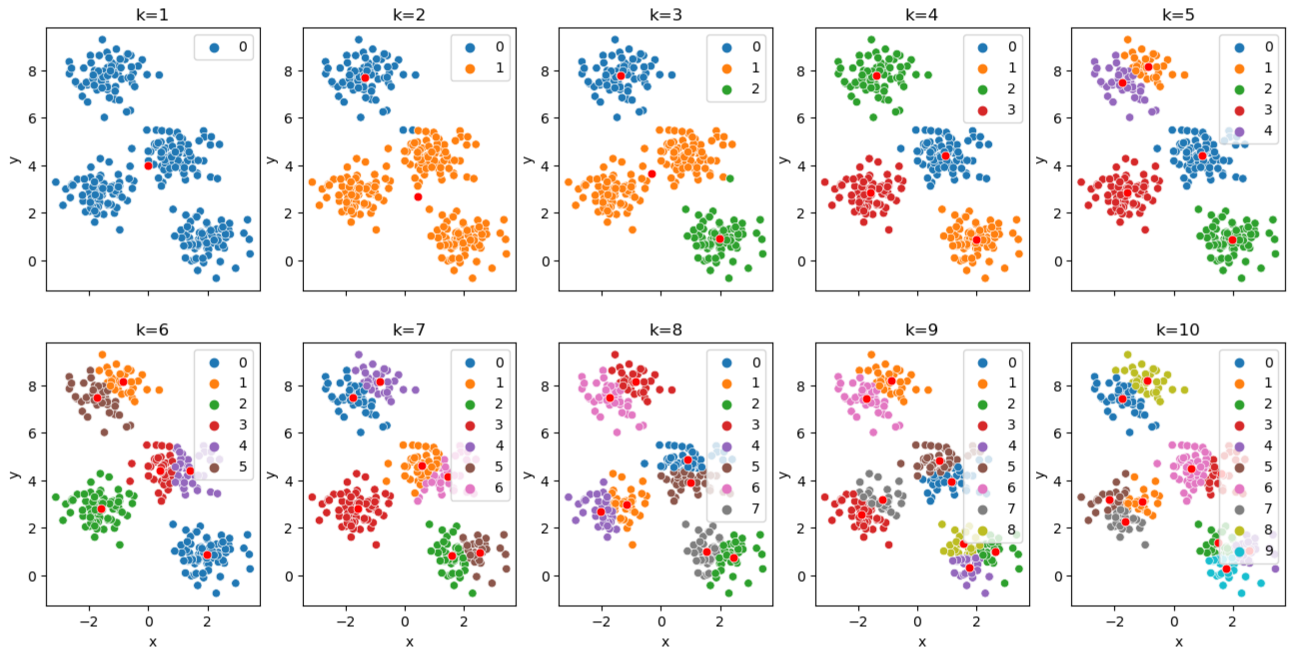


Figure 2: The visualization of K-Means Clustering in different k (sd=0.6)

Besides, the contingency tables of the clustering solutions for various k are summarized in Table 1, where the first column is the true group, and the columns under different k are the clustered results from K-Means. The data in the table is not arranged according to the diagonal of the matrix, because the 'KMeans' function only groups the points but does not control the groups. As we can see, when k=4, the result of clustering is perfect, which is the same as the generating rule.

Table 1: The contingency tables of the clustering solutions

	The value of k										
	1	10									
Group	A	A	B	C	D	E	F	G	H	I	J
A	75	41	0	0	0	0	0	34	0	0	0
B	75	0	28	0	0	0	0	0	26	21	0
C	75	0	0	0	31	0	19	0	0	0	25
D	75	0	0	43	0	32	0	0	0	0	0
	2		9								
	A	B	A	B	C	D	E	F	G	H	I
A	73	2	0	0	0	48	0	27	0	0	0
B	75	0	0	0	34	0	15	0	26	0	0
C	75	0	0	42	0	0	0	0	0	0	33
D	0	75	43	0	0	0	0	0	0	32	0
	3			8							
	A	B	C	A	B	C	D	E	F	G	H
A	1	0	74	0	0	0	27	48	0	0	0
B	75	0	0	0	38	0	0	0	37	0	0
C	0	0	75	0	0	33	0	0	0	42	0
D	0	75	0	32	0	0	0	0	0	0	43
	4				7						
	A	B	C	D	A	B	C	D	E	F	G
A	75	0	0	0	0	0	0	32	43	0	0
B	0	0	0	75	0	0	39	0	0	0	36
C	0	0	75	0	74	0	0	0	1	0	0
D	0	75	0	0	0	31	0	0	0	44	0
	5					6					
	A	B	C	D	E	A	B	C	D	E	F
A	0	75	0	0	0	0	36	0	0	0	39
B	0	0	75	0	0	0	0	75	0	0	0
C	0	0	0	75	0	0	1	0	74	0	0
D	31	0	0	0	44	44	0	0	0	31	0

1.3 Part c

The sum of square errors (SSE) for k in {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} is shown in Fig.3, which indicates that the natural number of clusters is 4. Because when k is smaller than 4, SSE experiences a fast drop, and SSE nearly keeps the same when k is larger than 4.

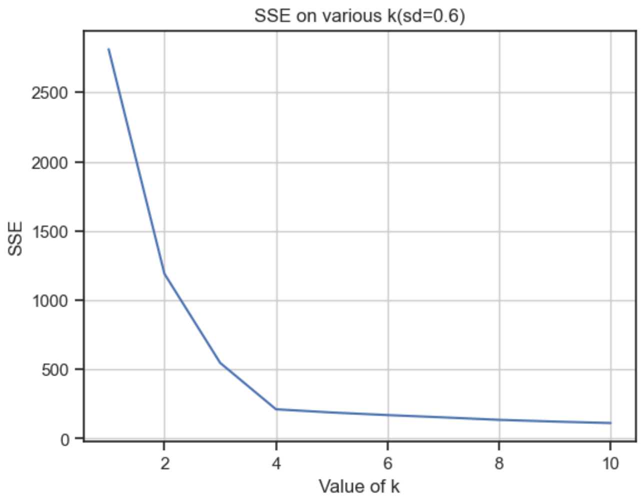


Figure 3: The plot of SSE on various k(sd=0.6)

1.4 Part d

I repeated the steps during parts (a) and (c), When the standard deviation equals 0.1, the corresponding results are shown in Fig.4 and 5. The outcome of the condition on standard deviation equals 2.5 is shown in Fig.6 and 7.

As we can see, when the standard deviation is small(for example, 0.1 here), the figure of SSE on various k clearly indicates that the number of clusters is 4. However, when the standard deviation is large(for example, 2.5), then we cannot tell the natural number of clusters. Because when the value of SSE keeps decreasing apparently both when k is smaller than 4 and larger than 4.

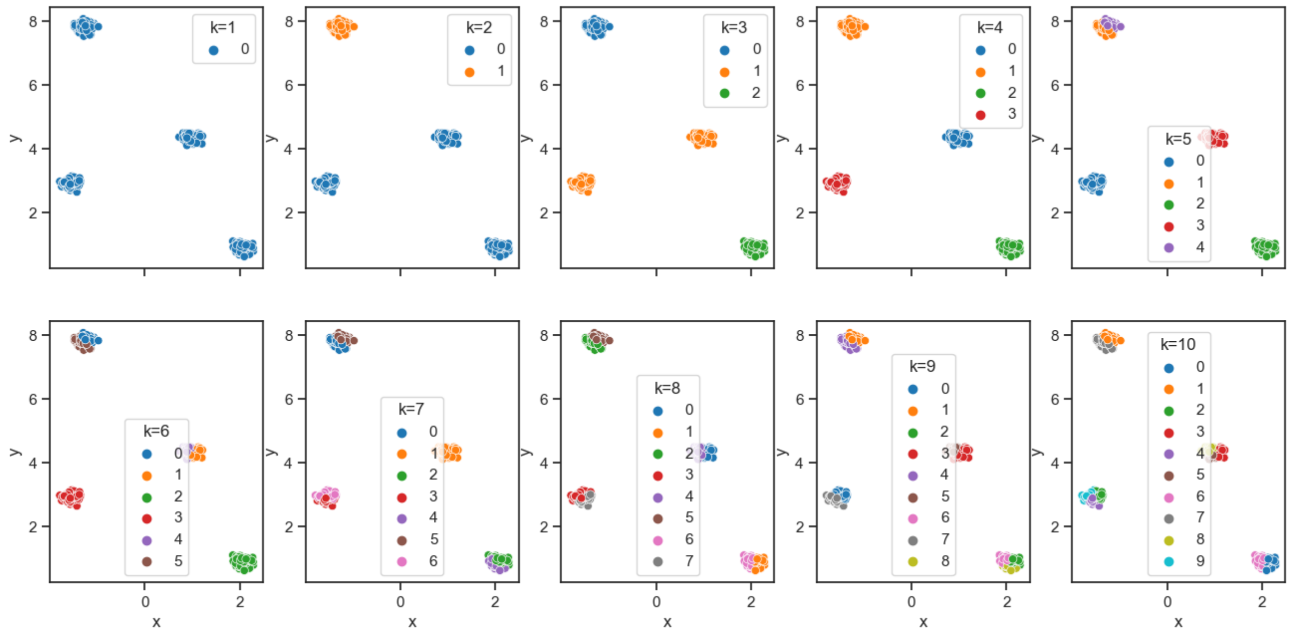


Figure 4: The visualization of K-Means Clustering in different k (sd=0.1)

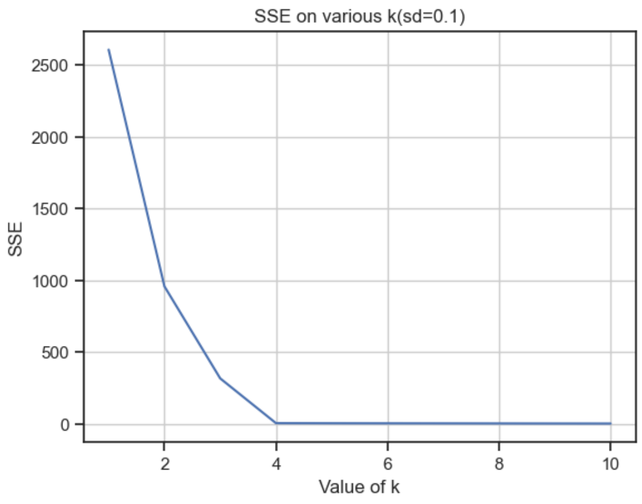


Figure 5: The plot of SSE on various k(sd=0.1)

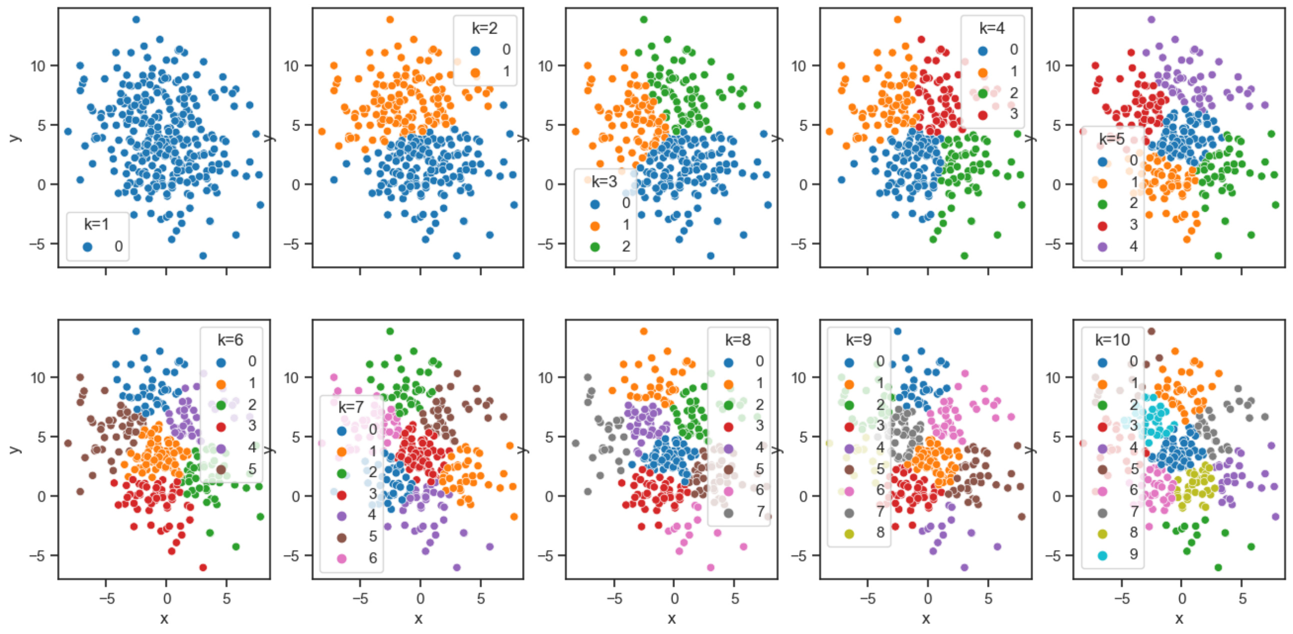


Figure 6: The visualization of K-Means Clustering in different k (sd=2.5)

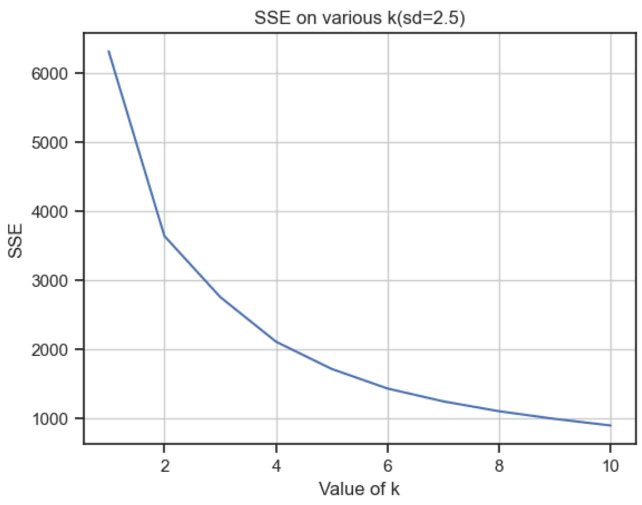


Figure 7: The plot of SSE on various k(sd=2.5)

1.5 Part e

By setting the parameter 'random_state' of KMeans to an integer number(I used 100 here), I got another cluster-center initialization case. Then, I repeated the part (d). The results of clustering are shown in Fig.8 and 9.

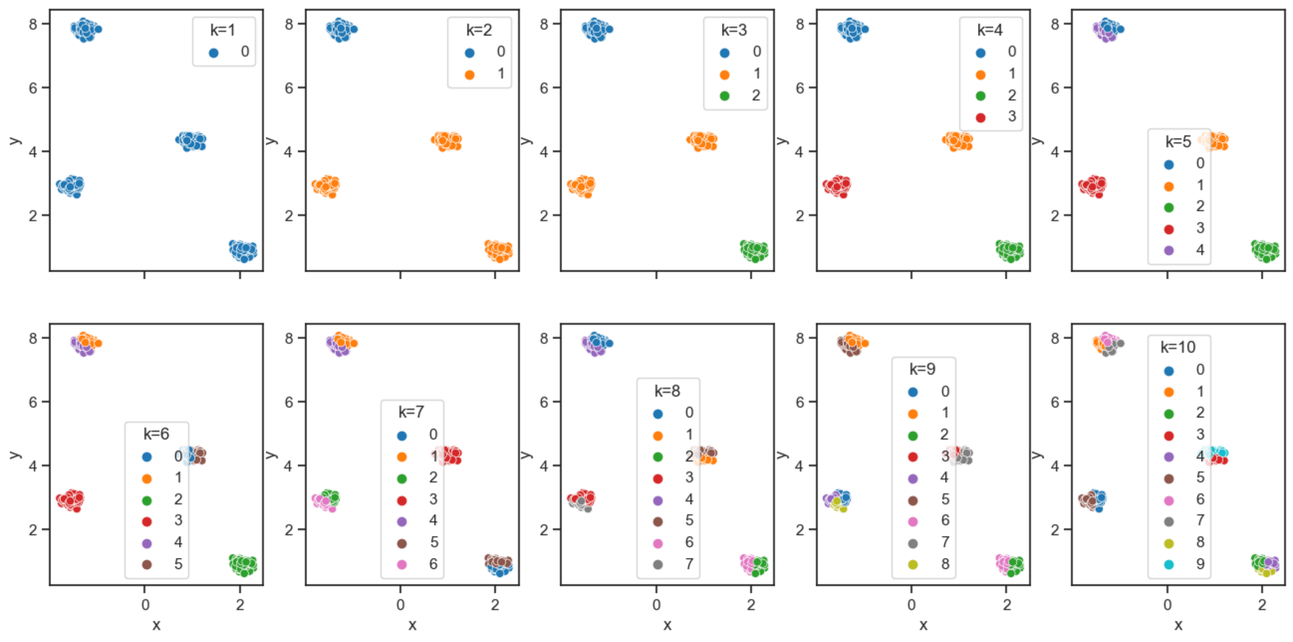


Figure 8: The visualization of K-Means Clustering in different k (random_state=100, sd=0.1)

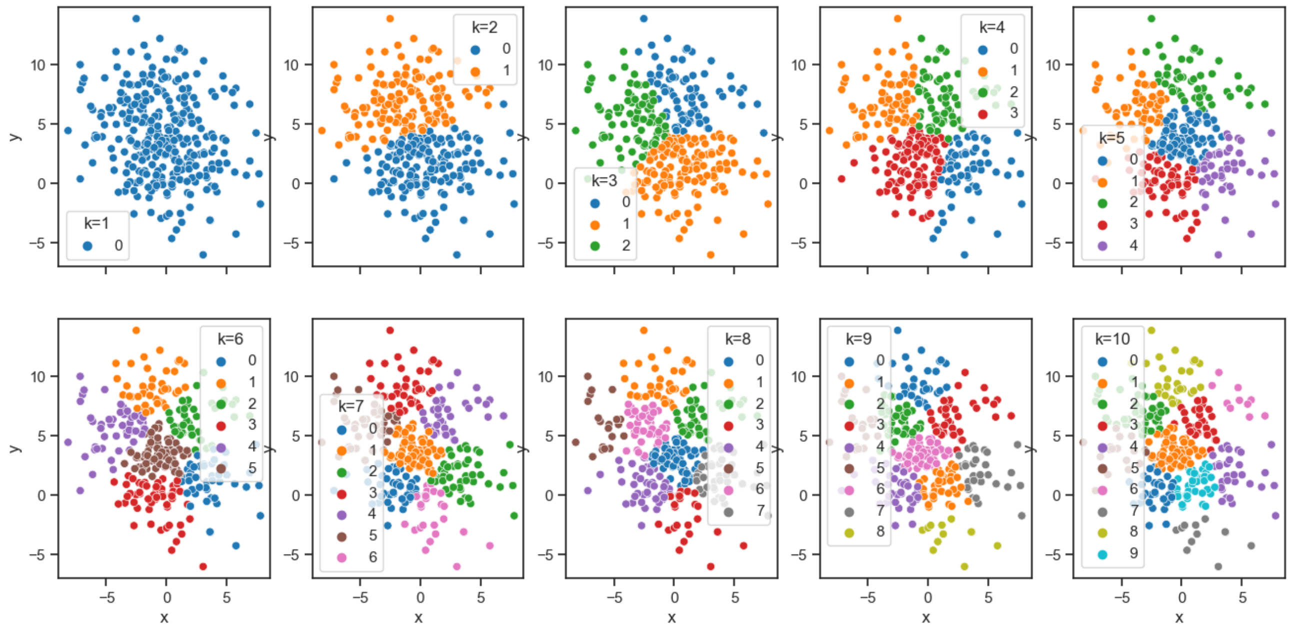


Figure 9: The visualization of K-Means Clustering in different k (random_state=100, sd=2.5)

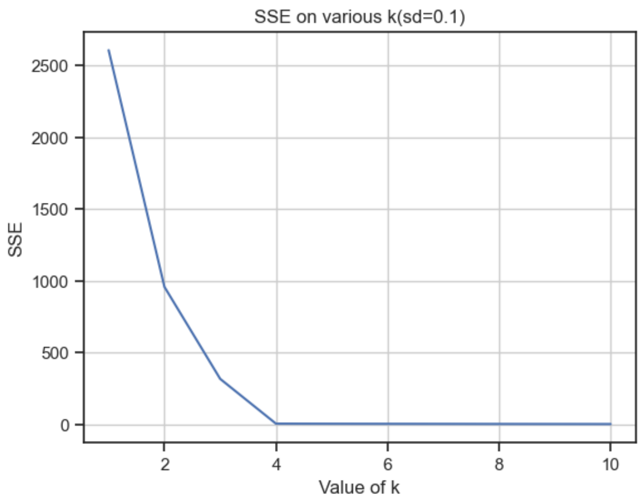


Figure 10: The plot of SSE on various k(random_state=100, sd=0.1)

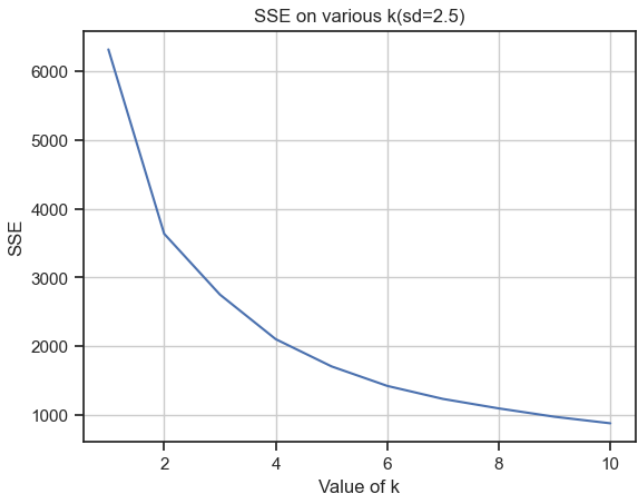


Figure 11: The plot of SSE on various k(random_state=100, sd=2.5)

As we can see, the received clustering solutions are similar to those obtained for 'random_state=None', especially when k is small. The outcome of clustering becomes slightly different when k is large enough. When it comes to the trends of SSE, the curves obtained in this part are almost the same as in the previous part.

The parameter 'random_state=None' in the function 'KMeans' determines random number generation for centroid initialization. Thus, when k is small, the final outcome still would converge, since the algorithm of K-Means seeks for lower SSE and won't stop until gets the same clustering result. However, things become different when k becomes larger than the natural number of clustering. The final clustering groups will be different, because of the different choices of the initial center points. The outcome is also reasonable because the points stop changing groups and the SSE converges. For example, when k=10, the clustering result in Fig.4 and 8 are different.

One possible way to make k-means initialization less dependent on 'random_state' is to set the parameter 'init'='k-means++'. It works because it ensures that the initial cluster centers are well spread out across all the points. Thus it can lead to more consistent clustering results.

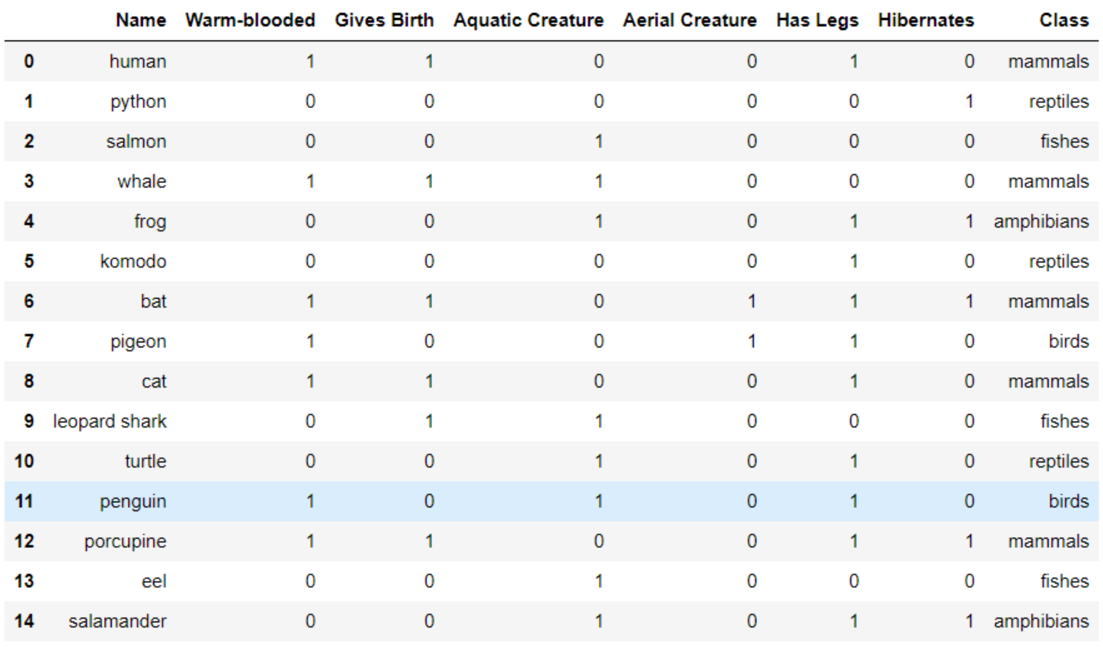
2 Task 2

2.1 Part a

According to the following codes, I loaded the vertebrate.csv data and printed it, which is shown in Fig.12.

```
## import csv file
vertebrate_data = pd.read_csv('F://UM//Data Mining//Assign_3//vertebrate.csv')

vertebrate_data
```



	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	human	1	1	0	0	1	0	mammals
1	python	0	0	0	0	0	1	reptiles
2	salmon	0	0	1	0	0	0	fishes
3	whale	1	1	1	0	0	0	mammals
4	frog	0	0	1	0	1	1	amphibians
5	komodo	0	0	0	0	1	0	reptiles
6	bat	1	1	0	1	1	1	mammals
7	pigeon	1	0	0	1	1	0	birds
8	cat	1	1	0	0	1	0	mammals
9	leopard shark	0	1	1	0	0	0	fishes
10	turtle	0	0	1	0	1	0	reptiles
11	penguin	1	0	1	0	1	0	birds
12	porcupine	1	1	0	0	1	1	mammals
13	eel	0	0	1	0	0	0	fishes
14	salamander	0	0	1	0	1	1	amphibians

Figure 12: The printed vertebrate data

2.2 Part b

After dropping the non-numeric data, I set the parameter 'method' of the function 'linkage' as 'single', 'complete', and 'average' respectively. Then I got the results which are shown in Fig.13, 14, and 15.

In general, the outcome given by the max-link is the most natural one to me.

It seems that single-link is unable to identify birds because it mixes birds with other classes, and it cannot identify whales belonging to mammals.

As for the result of max-link, it seems reasonable, although it's not perfect. Because it identifies that birds and mammals are much closer, and other classes are more relevant. Also, it almost correctly combines the animals belonging to the same class into the same group.

When it comes to the result of the average-link, It seems nice that it roughly splits all the animals into three groups, mammals, fishes, and combined reptiles and amphibians. However, it cannot correctly classify birds. Also, it's unable to identify whales as mammals, python is not fish.

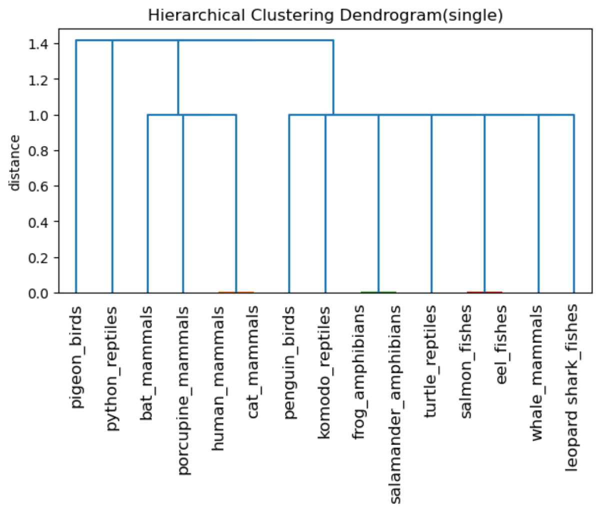


Figure 13: The hierarchical clustering result of single-link

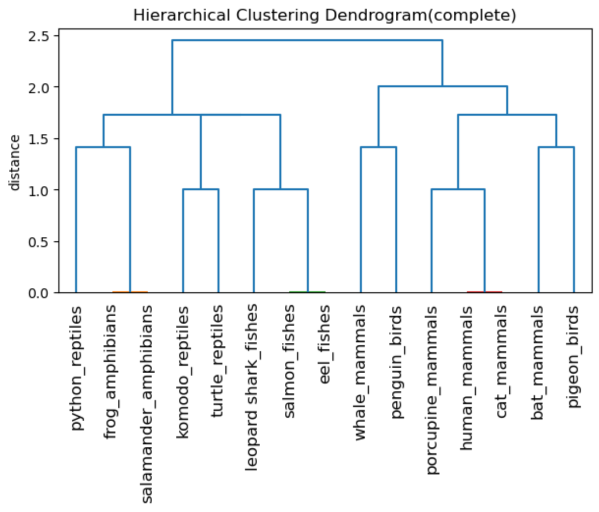


Figure 14: The hierarchical clustering result of max-link

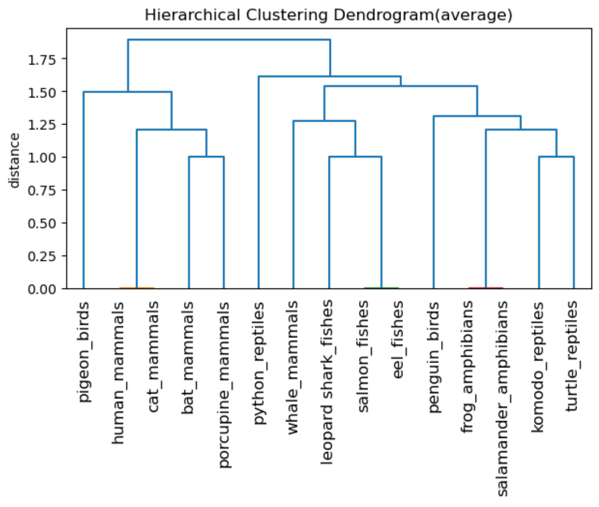


Figure 15: The hierarchical clustering result of average-link

3 Task 3

3.1 Part a

By the following codes, I import the CSV file. The visualization of the data is shown in Fig.16.

```
## import csv file
data_chameleon = pd.read_csv('F://UM//Data Mining//Assign_3//chameleon.csv')

data_chameleon.head()
```

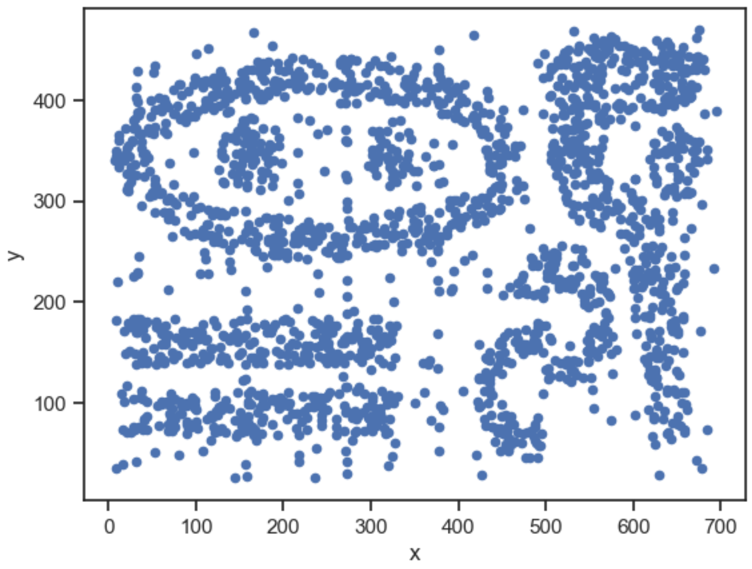


Figure 16: The visualization of chameleon.csv data

3.2 Part b

By setting the parameter 'eps'=15.5 and 'min_samples'=5, I used function DBSCAN on the given data, the clustering result is shown in Fig.17. All the points are split into 8 groups, which seems reasonable.

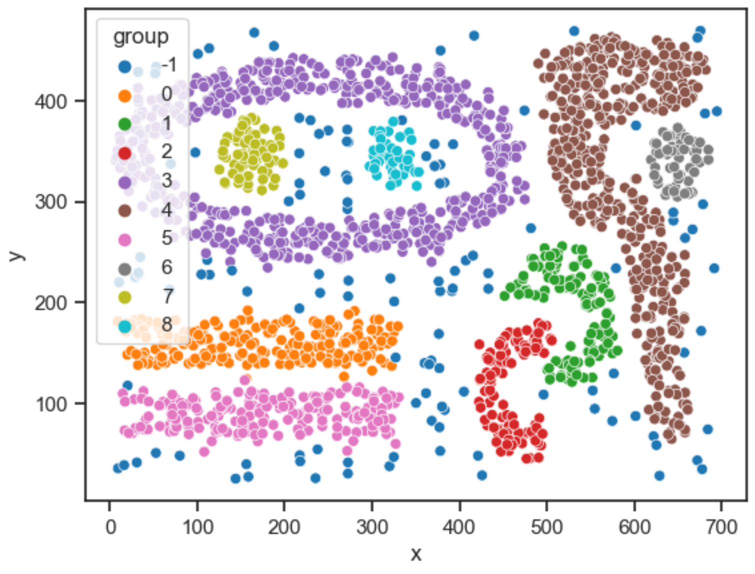


Figure 17: The clustering result of chameleon.csv data

3.3 Part c

By changing the values of the parameters 'eps' and 'min_samples' from 1 to 21 with step 5, I got different clustering results, which are summarized in Fig.18, whose sub-title shows the setting of parameters and the number of clustered groups. To make the graph more clear, I removed the legends for the cases with too many groups.

To understand the reason why we get such results, the meaning of parameters 'eps' and 'min_sample' should be clarified. The value of 'eps' is the maximum distance between two samples for one to be considered as in the neighborhood of the other. The value of 'min_sample' is the number of samples (or total weight) in a neighborhood for a point to be considered as a core point.

Thus, when 'eps' is too small, the smaller the 'min_sample' is, the more obtained groups will be, since we would get more corn points. Besides, it's hard to find appropriate results in this condition(when eps is too small). When 'min_sample' is too large, it would be difficult for us to get a reasonable clustering result.

However, if 'min_sample' is too small or 'eps' is too large, we still can obtain good clustering results by tuning another parameter. Also, small 'eps' and large 'min_sample' is a terrible parameter setting.

All in all, the parameter settings of the DBSCAN method are really important, and it will directly determine the final outcome. In other words, the results of the DBSCAN method are very sensitive to the parameter settings. If a reasonable clustering result is wanted, then an appropriate parameter setting should be given.

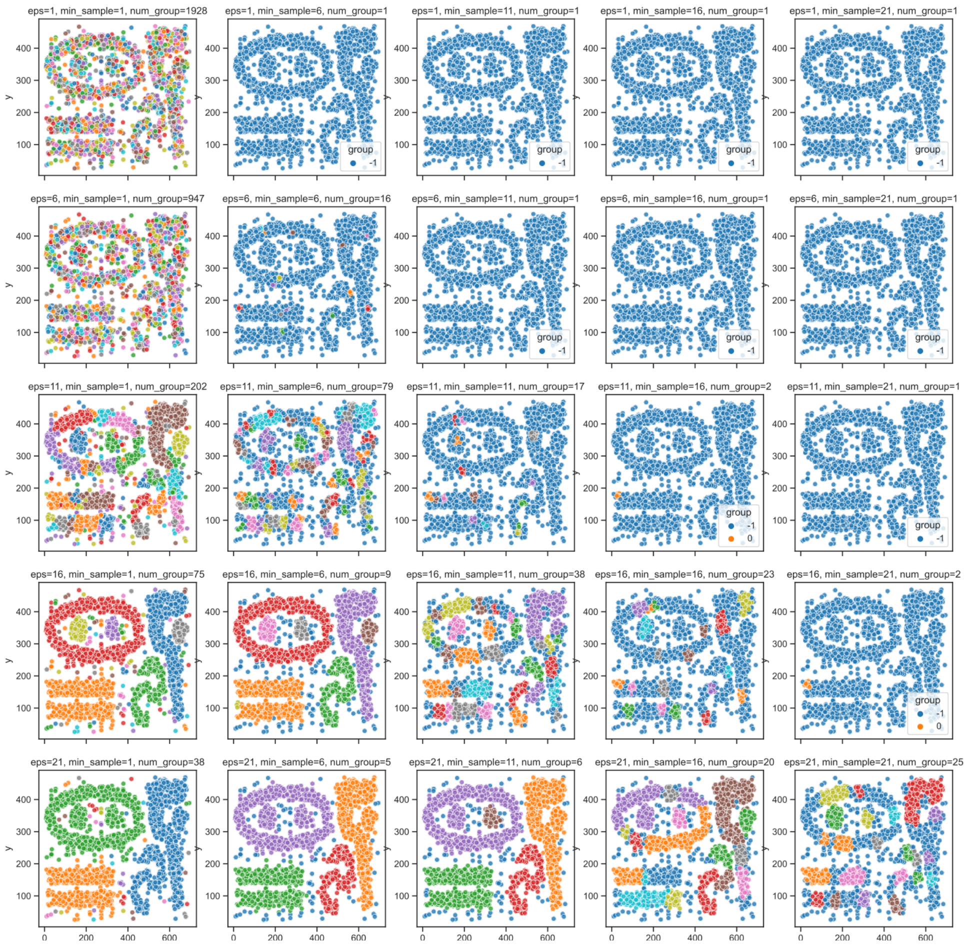


Figure 18: Clustering results of the experiments with the DBSCAN method

Academic Integrity Declaration

I, Yulin Zhou, hereby declare that I have not used large language models or any automated tools for generating the written answers and interpretations in this Analytical Report.