In [50]:
```
## Import

from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import pandas as pd
```

Task 1

(a) Data Generation

In [2]:
```
# 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,
                    n_features=2, cluster_std = sd, center_box = (-10, 10),
                    random_state=0)
```

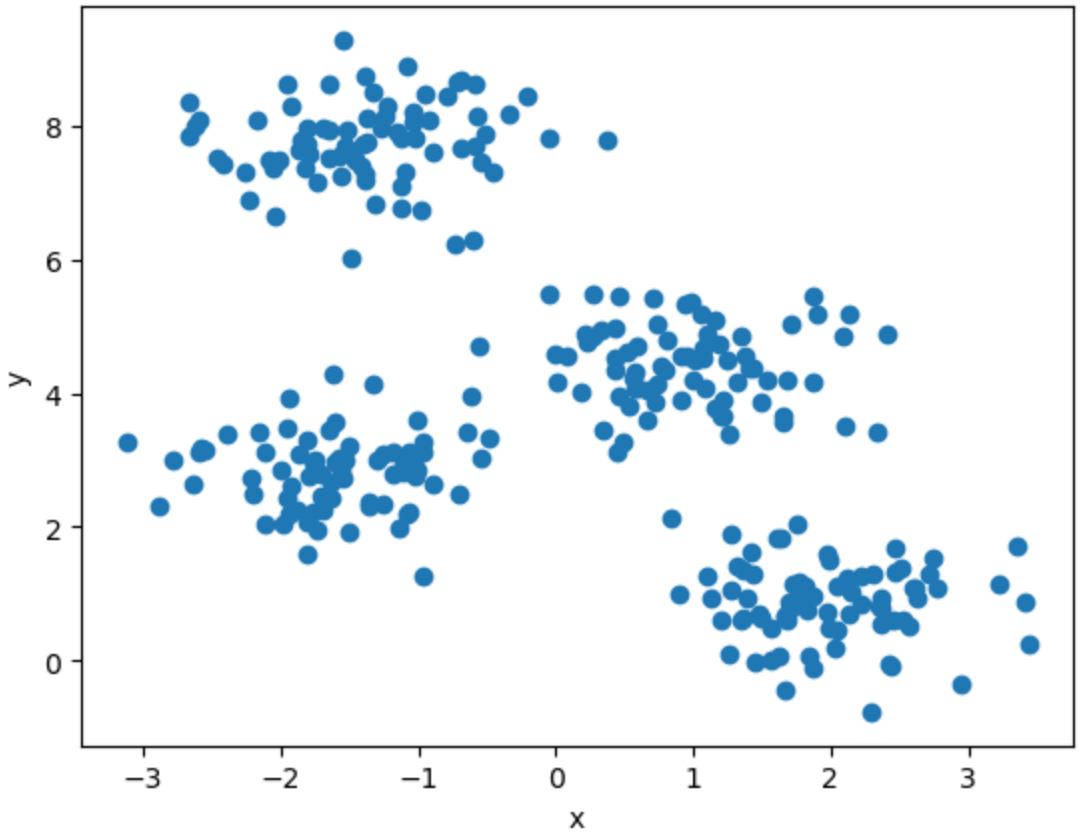
In [3]:
```
## Transform the generated information into dataframe
df_x_y = pd.DataFrame(x_y, columns = ['x','y'])
df_x_y['True group'] = group

df_x_y.head()
```

Out[3]:

	x	y	True group
0	0.836857	2.136359	1
1	-1.413658	7.409623	3
2	1.155213	5.099619	0
3	-1.018616	7.814915	3
4	1.271351	1.892542	1

In [4]:
```
plt.scatter(df_x_y['x'], df_x_y['y'])
plt.xlabel('x')
plt.ylabel('y')
```

Out[4]:
```
Text(0, 0.5, 'y')
```



(b) Apply k-means clustering

In [5]:
```
from sklearn.cluster import KMeans
import seaborn as sns
from sklearn.metrics.cluster import contingency_matrix
```

Run k-means clustering algorithm on the data obtained

In [26]:
```
## Run k-means clustering algorithm on the data obtained

center_record = {} # record the center
sse_record = {'k':[], 'sse':[]}

for k in range(1, 11):
    # fit the points with kmeans
    kmeans = KMeans(n_clusters=k, random_state=None).fit(x_y)

    # record results
    sse_record['k'].append(k) # record k
    sse_record['sse'].append(kmeans.inertia_) # record SSE
    # add the label into the df
    df_x_y['k='+str(k)] = kmeans.labels_ # record labels
    center_record[k] = kmeans.cluster_centers_ # record the center
```

```
F:\ana\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known t
o have a memory leak on Windows with MKL, when there are less chunks than available thre
ads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```

In [7]:
```
df_x_y.head()
```

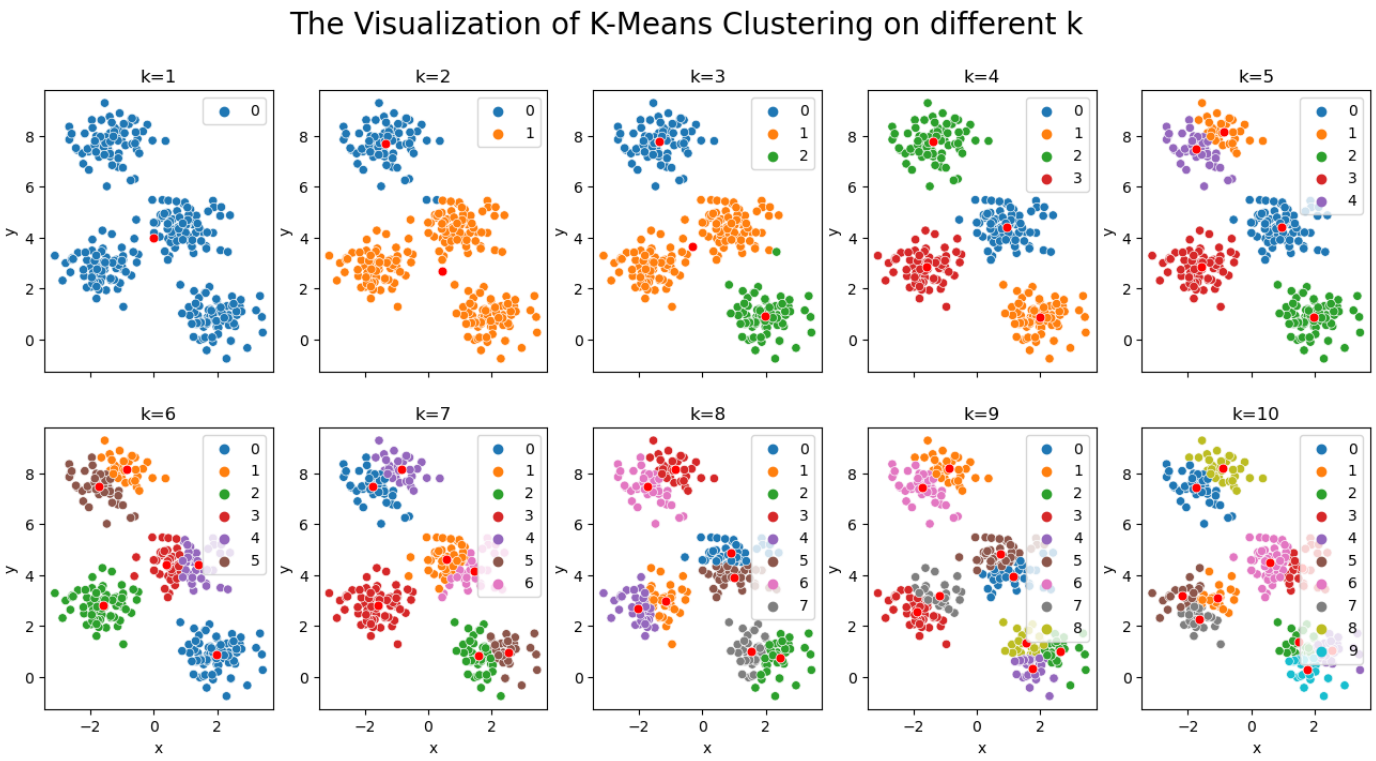
Out[7]:

	x	y	True group	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
0	0.836857	2.136359		1	0	0	0	1	1	5	1	7	7

1	-1.413658	7.409623	3	0	1	1	2	0	2	3	2	6	1
2	1.155213	5.099619	0	0	0	2	3	2	5	4	0	3	9
3	-1.018616	7.814915	3	0	1	1	2	4	4	2	7	0	8
4	1.271351	1.892542	1	0	0	0	0	1	1	5	1	7	7

visualize the clusters and their centers for different k

In [48]:
```
## visualize the clusters and their centers for different k

figure, axes = plt.subplots(2, 5, sharex=True, figsize=(16,7.5))
figure.suptitle('The Visualization of K-Means Clustering on different k', fontsize=20)

for i in range(1, 11):
    if i <= 5:
        num_row = 0
        num_col = i - 1
    else:
        num_row = 1
        num_col = i - 6
    con = 'k='+str(i)
    sns.scatterplot(ax=axes[num_row, num_col], data=df_x_y,
                    x='x', y='y', hue=con, palette="tab10")
    # create a df about the center points when k=i
    df_center = pd.DataFrame(center_record[i], columns = ['x','y'])
    sub_tit = 'k='+str(i)
    sns.scatterplot(ax=axes[num_row, num_col], data=df_center,
                    x='x', y='y', palette="deep", color="r", marker="o", s=40).set(title
```



Print the contingency tables of the clustering solutions.

In [164…
```
## Print the contingency tables of the clustering solutions.

for i in range(1, 11):
    con = 'k='+str(i)
    print(contingency_matrix(labels_true = df_x_y['True group'],
```

```
                    labels_pred = df_x_y[con], eps=None, sparse=False))

# (i, j): (true i, predict j)
```

```
[[75]
 [75]
 [75]
 [75]]
[[73  2]
 [75  0]
 [75  0]
 [ 0 75]]
[[ 1  0 74]
 [75  0  0]
 [ 0  0 75]
 [ 0 75  0]]
[[75  0  0  0]
 [ 0  0  0 75]
 [ 0  0 75  0]
 [ 0 75  0  0]]
[[ 0 75  0  0  0]
 [ 0  0 75  0  0]
 [ 0  0  0 75  0]
 [31  0  0  0 44]]
[[ 0 36  0  0  0 39]
 [ 0  0 75  0  0  0]
 [ 0  1  0 74  0  0]
 [44  0  0  0 31  0]]
[[ 0  0  0 32 43  0  0]
 [ 0  0 39  0  0  0 36]
 [74  0  0  0  1  0  0]
 [ 0 31  0  0  0 44  0]]
[[ 0  0  0 27 48  0  0  0]
 [ 0 38  0  0  0 37  0  0]
 [ 0  0 33  0  0  0 42  0]
 [32  0  0  0  0  0  0 43]]
[[ 0  0  0 48  0 27  0  0  0]
 [ 0  0 34  0 15  0 26  0  0]
 [ 0 42  0  0  0  0  0  0 33]
 [43  0  0  0  0  0  0 32  0]]
[[41  0  0  0  0  0 34  0  0  0]
 [ 0 28  0  0  0  0  0 26 21  0]
 [ 0  0  0 31  0 19  0  0  0 25]
 [ 0  0 43  0 32  0  0  0  0  0]]
```

(c)Plot the sum of square errors (SSE)

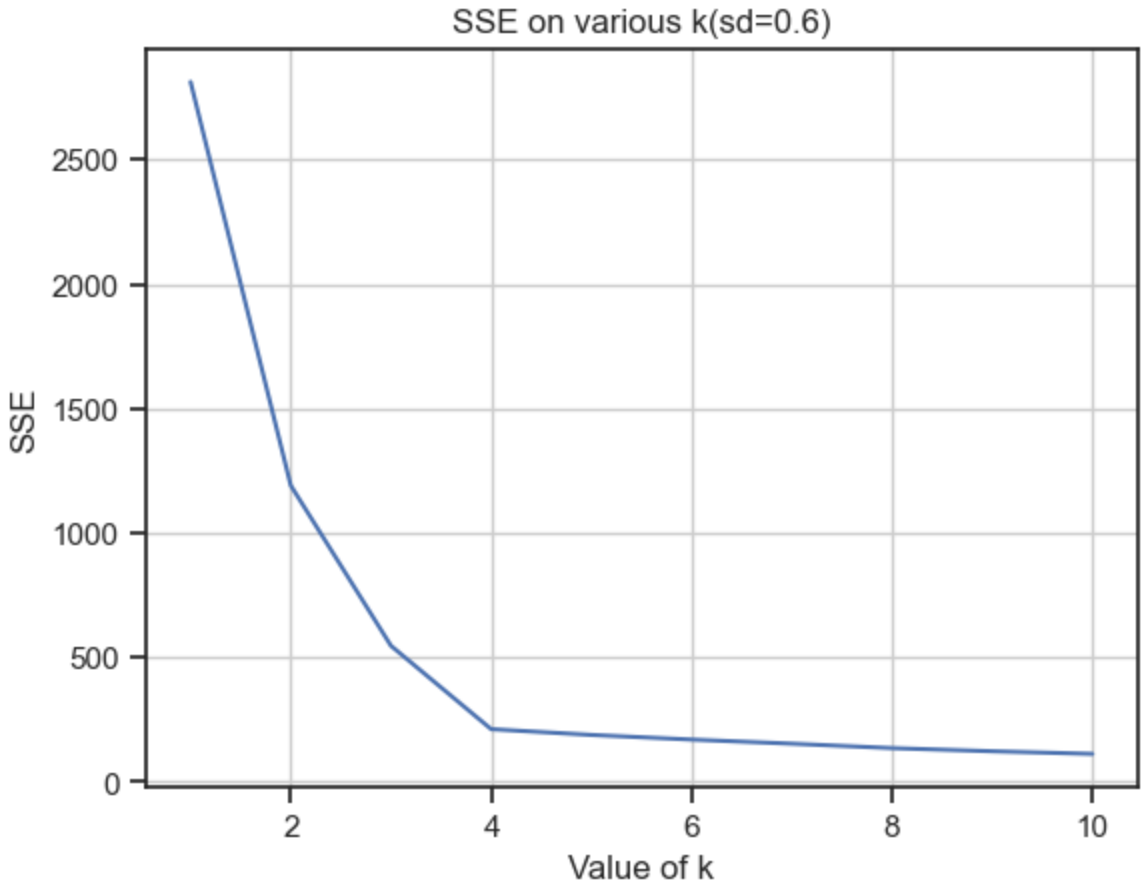
In [165…
```
df_sse = pd.DataFrame.from_dict(sse_record)
df_sse.head()
```

Out[165]:

	k	sse
0	1	2812.137595
1	2	1190.782359
2	3	546.891150
3	4	212.005996
4	5	188.829860

In [166…
```
sns.set(style="ticks")
ax = sns.lineplot(data = df_sse, x = 'k', y = 'sse')
```

```
ax.set(xlabel='Value of k', ylabel='SSE', title='SSE on various k(sd=0.6)')
plt.grid()  #just add this
```



(d) Repeat (a)-(c) with sd= 0.1 and 2.5.

In [181…

```
# define a function to make codes more efficient

def from_a2c(sd, total_num, seed_random):

    # generate the samples (4 centers with 2 features)
    x_y, group = make_blobs(n_samples=total_num, centers=4,
                    n_features=2, cluster_std = sd, center_box = (-10, 10),
                    random_state=0)
    df_x_y = pd.DataFrame(x_y, columns = ['x','y'])
    df_x_y['True group'] = group

    center_record = {} # record the center
    sse_record = {'k':[], 'sse':[]}

    for k in range(1, 11):
        # fit the points with k-means
        kmeans = KMeans(n_clusters=k, random_state=seed_random).fit(x_y)
        sse_record['k'].append(k)
        sse_record['sse'].append(kmeans.inertia_)
        # add the label into the df
        df_x_y['k='+str(k)] = kmeans.labels_

        # record the center
        center_record[k] = kmeans.cluster_centers_


    ## visualize the clusters and their centers for different k
    figure, axes = plt.subplots(2, 5, sharex=True, figsize=(16,7.5))
    fig_titile = 'The Visualization of K-Means Clustering on different k(sd='+str(sd)+')
```

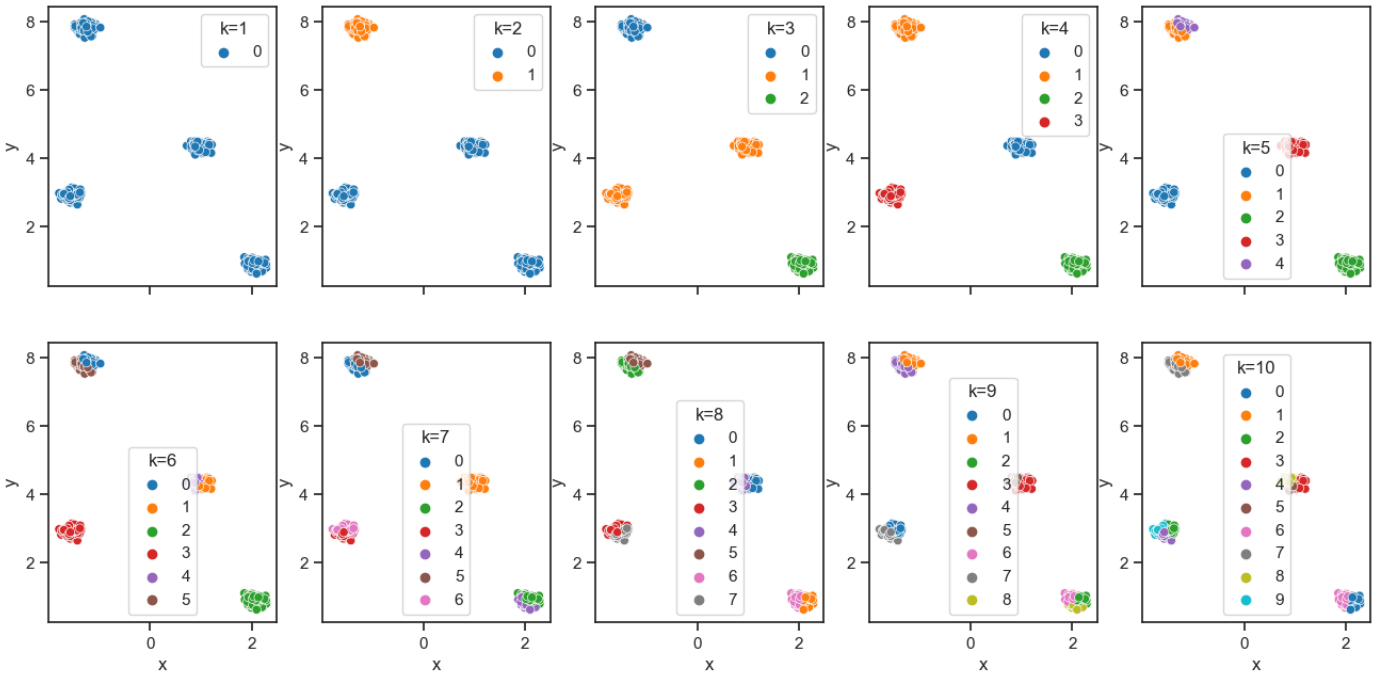
```
    figure.suptitle(fig_titile,
                    fontsize=20)

    for i in range(1, 11):
        if i <= 5:
            num_row = 0
            num_col = i - 1
        else:
            num_row = 1
            num_col = i - 6
        con = 'k='+str(i)
        sns.scatterplot(ax=axes[num_row, num_col], data=df_x_y,
                        x='x', y='y', hue=con, palette="tab10")
    plt.show(figure)

    ## Print the contingency tables of the clustering solutions.

    for i in range(1, 11):
        con = 'k='+str(i)
        print(contingency_matrix(labels_true = df_x_y['True group'],
                        labels_pred = df_x_y[con], eps=None, sparse=False))

    df_sse = pd.DataFrame.from_dict(sse_record)

    sns.set(style="ticks")
    ax = sns.lineplot(data = df_sse, x = 'k', y = 'sse')
    ax.set(xlabel='Value of k', ylabel='SSE', title='SSE on various k(sd='+str(sd)+')')
    plt.grid()  #just add this
    plt.show(ax)
```

In [182…

```
from_a2c(sd=0.1, total_num=300, seed_random=None)
```

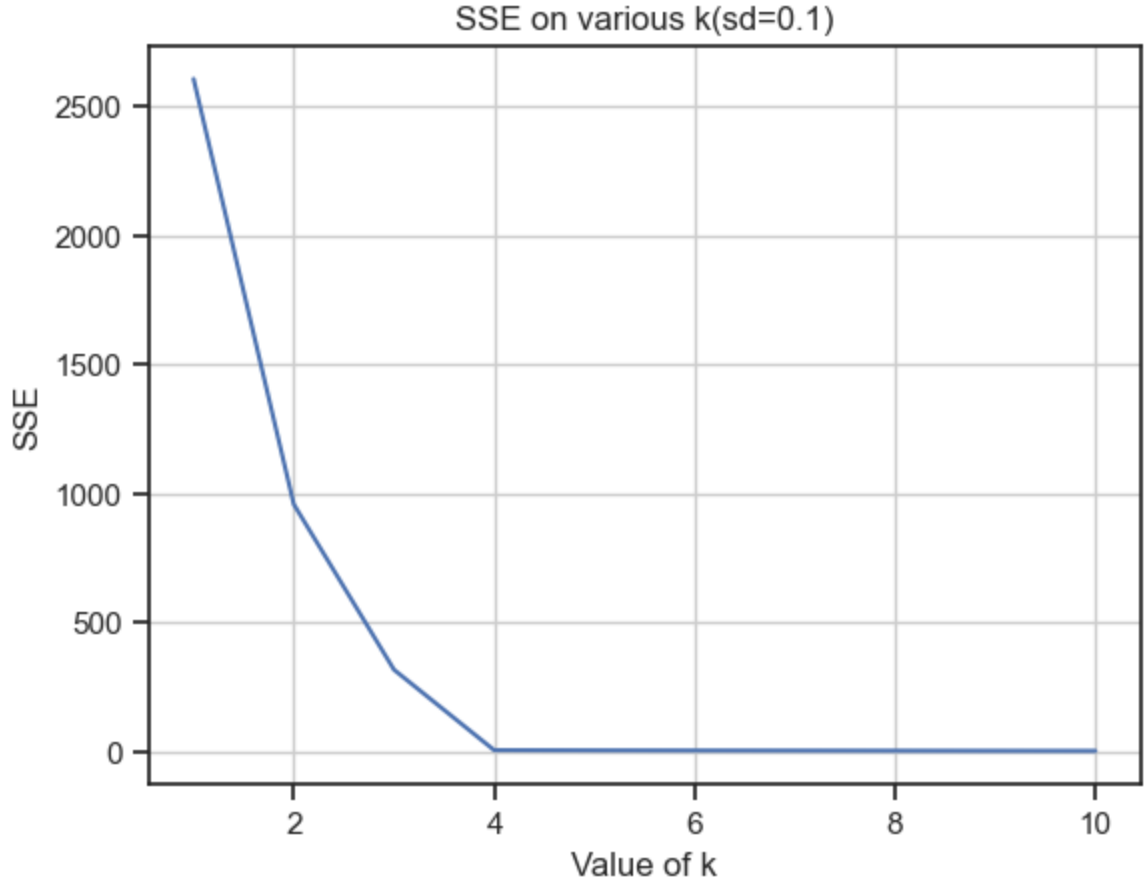
```
F:\ana\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known t
o have a memory leak on Windows with MKL, when there are less chunks than available thre
ads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```

The Visualization of K-Means Clustering on different k(sd=0.1)




```
[[75]
 [75]
 [75]
 [75]]
[[75  0]
```

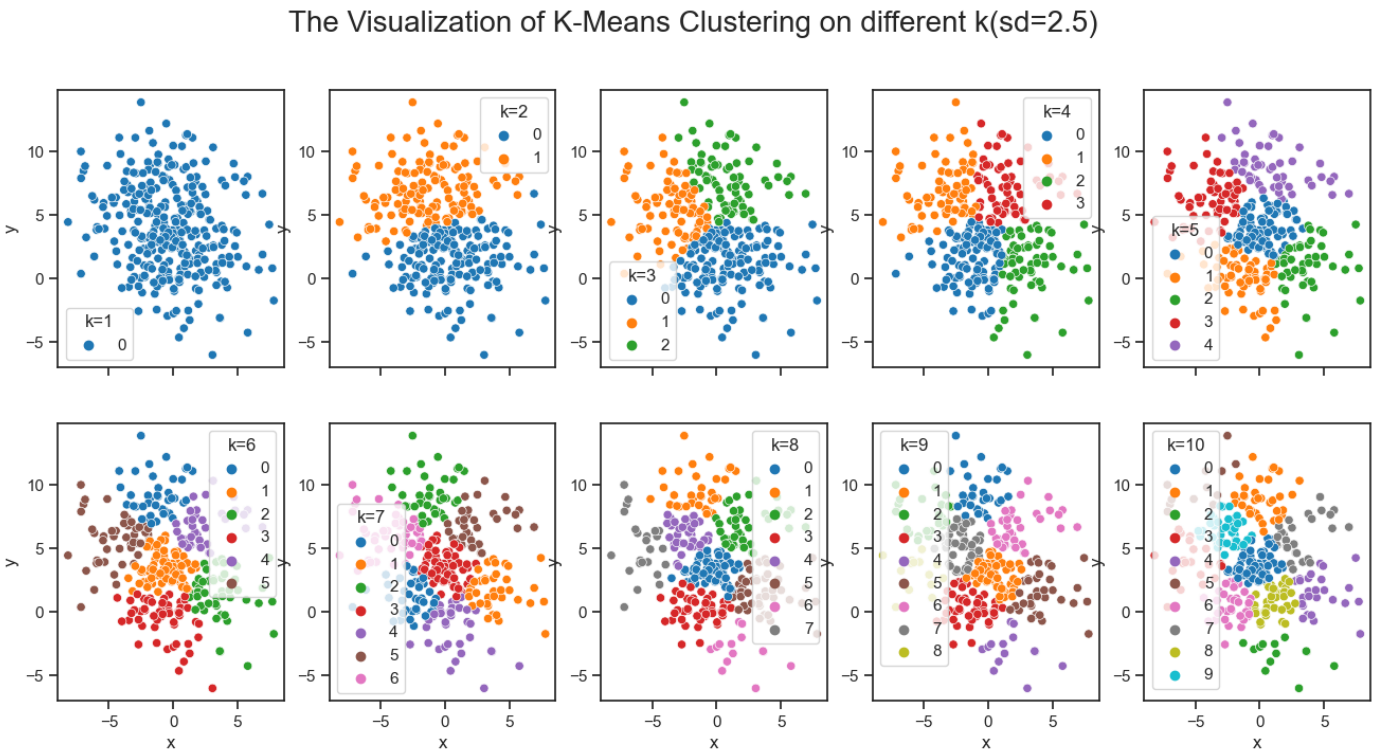
```
 [75  0]
 [75  0]
 [ 0 75]]
[[ 0 75  0]
 [ 0  0 75]
 [ 0 75  0]
 [75  0  0]]
[[75  0  0  0]
 [ 0  0 75  0]
 [ 0  0  0 75]
 [ 0 75  0  0]]
[[ 0  0  0 75  0]
 [ 0  0 75  0  0]
 [75  0  0  0  0]
 [ 0 44  0  0 31]]
[[ 0 35  0  0 40  0]
 [ 0  0 75  0  0  0]
 [ 0  0  0 75  0  0]
 [31  0  0  0  0 44]]
[[ 0 75  0  0  0  0  0]
 [ 0  0 39  0 36  0  0]
 [ 0  0  0 28  0  0 47]
 [43  0  0  0  0 32  0]]
[[35  0  0  0 40  0  0  0]
 [ 0 34  0  0  0  0 41  0]
 [ 0  0  0 37  0  0  0 38]
 [ 0  0 43  0  0 32  0  0]]
[[ 0  0  0 48  0 27  0  0  0]
 [ 0  0 27  0  0  0 34  0 14]
 [37  0  0  0  0  0  0 38  0]
 [ 0 31  0  0 44  0  0  0  0]]
[[ 0  0  0 25  0 26  0  0 24  0]
 [30  0  0  0  0  0 45  0  0  0]
 [ 0  0 28  0 26  0  0  0  0 21]
 [ 0 32  0  0  0  0  0 43  0  0]]
```



In [183…

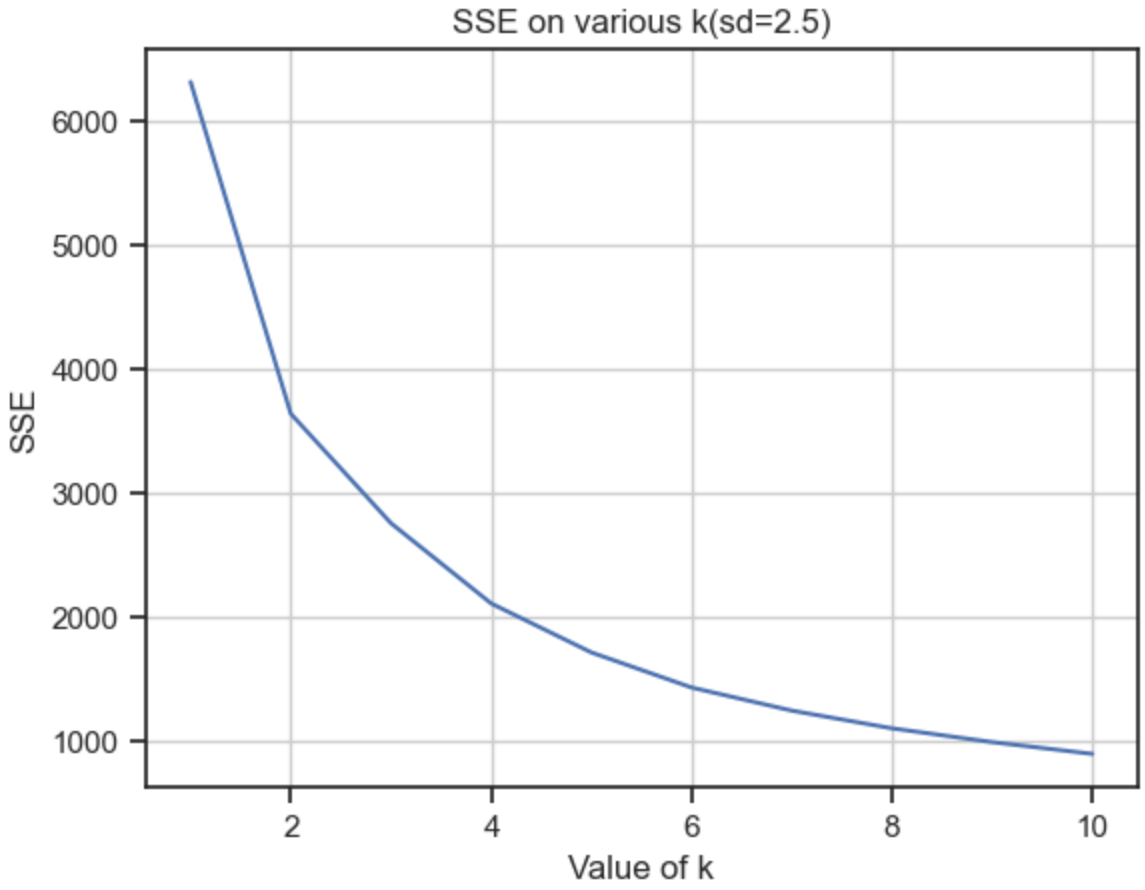
```
from_a2c(sd=2.5, total_num=300, seed_random=None)
```

```
F:\ana\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known t
o have a memory leak on Windows with MKL, when there are less chunks than available thre
ads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```




```
[[75]
 [75]
 [75]
 [75]]
[[33 42]
 [70  5]
 [55 20]
 [ 6 69]]
[[28 15 32]
 [70  2  3]
 [41 28  6]
 [ 5 37 33]]
[[18 13 15 29]
 [28  2 42  3]
 [51 14  4  6]
 [ 6 43  1 25]]
[[35  4 10 10 16]
 [13 25 36  1  0]
 [27 30  1 14  3]
 [ 6  2  1 36 30]]
[[ 9 19 14  4 22  7]
 [ 0 14 32 27  1  1]
 [ 2 31  2 21  4 15]
 [29  7  1  2 11 25]]
[[ 3 11  9 24  1 20  7]
 [ 6 26  0 12 29  1  1]
 [27  2  2 20  7  4 13]
 [ 2  0 27  8  0 11 27]]
[[16  4 25  4 11 14  0  1]
 [12  0  1 18  2 29 13  0]
 [26  2  4 23  6  2  1 11]
 [ 5 27 11  1 18  1  0 12]]
[[ 6 18  2  4  0  9 22 14  0]
 [ 0 15  0 18 13 26  1  2  0]
 [ 2 19  1 24  1  0  4 12 12]
 [23  4 20  1  0  0 10 15  2]]
[[19  9  0  2  6  1  3 19  6 10]
```

```
[10  0 12  0 21  0  8  1 22  1]
[22  2  1 13  0  0 21  4  7  5]
[ 5 24  0  5  0 11  1  4  2 23]]
```



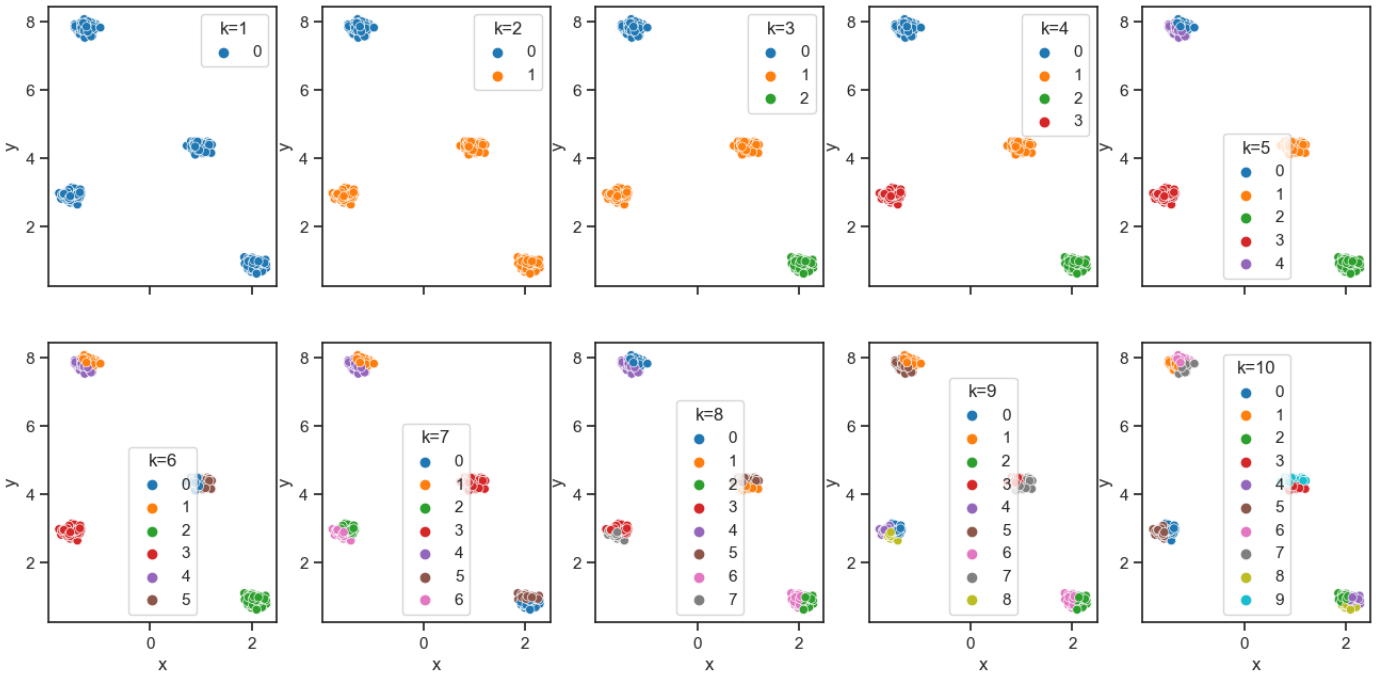
(e) Repeat (d) by setting the parameter random_state of K-Means to an integer number.

In [184…

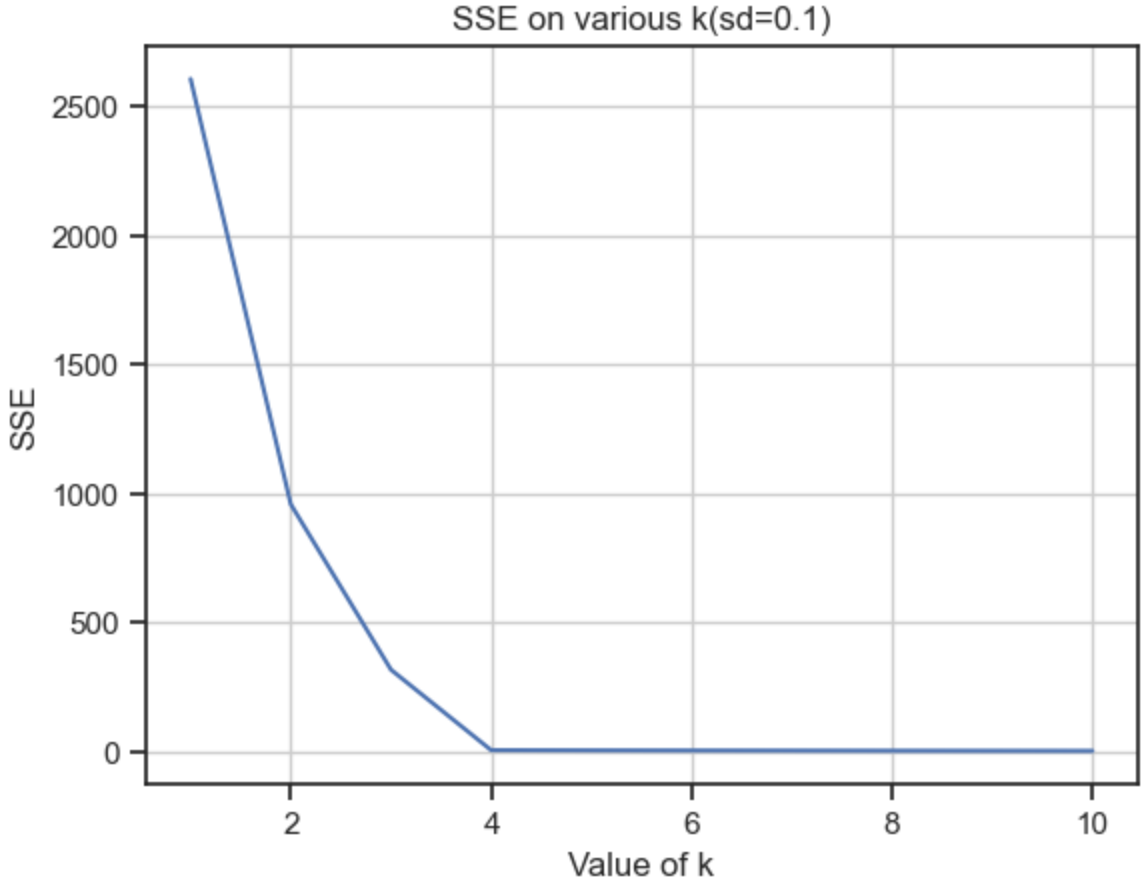
```
from_a2c(sd=0.1, total_num=300, seed_random=100)
```

```
F:\ana\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known t
o have a memory leak on Windows with MKL, when there are less chunks than available thre
ads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```

The Visualization of K-Means Clustering on different k(sd=0.1)

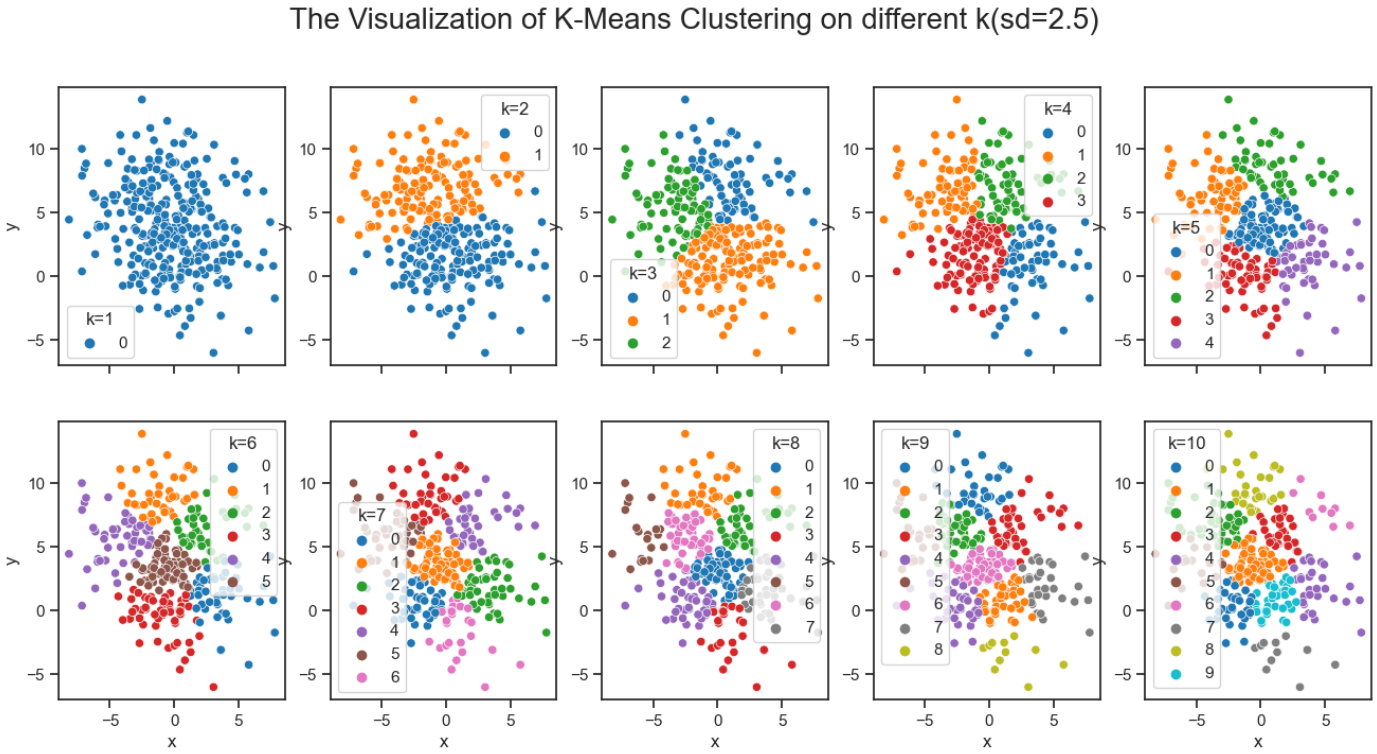



```
[[75]
 [75]
 [75]
 [75]]
[[ 0 75]
 [ 0 75]
 [ 0 75]
 [75  0]]
[[ 0 75  0]
 [ 0  0 75]
 [ 0 75  0]
 [75  0  0]]
[[ 0 75  0  0]
 [ 0  0 75  0]
 [ 0  0  0 75]
 [75  0  0  0]]
[[ 0 75  0  0  0]
 [ 0  0 75  0  0]
 [ 0  0  0 75  0]
 [31  0  0  0 44]]
[[38  0  0  0  0 37]
 [ 0  0 75  0  0  0]
 [ 0  0  0 75  0  0]
 [ 0 32  0  0 43  0]]
[[ 0  0  0 75  0  0  0]
 [37  0  0  0  0 38  0]
 [ 0  0 37  0  0  0 38]
 [ 0 31  0  0 44  0  0]]
[[ 0 37  0  0  0 38  0  0]
 [ 0  0 30  0  0  0 45  0]
 [ 0  0  0 47  0  0  0 28]
 [31  0  0  0 44  0  0  0]]
[[ 0  0  0 44  0  0  0 31  0]
 [ 0  0 32  0  0  0 43  0  0]
 [34  0  0  0 16  0  0  0 25]
 [ 0 31  0  0  0 44  0  0  0]]
[[ 0  0  0 36  0  0  0  0  0 39]
 [ 0  0 34  0 26  0  0  0 15  0]
 [39  0  0  0  0 36  0  0  0  0]
 [ 0 33  0  0  0  0 25 17  0  0]]
```



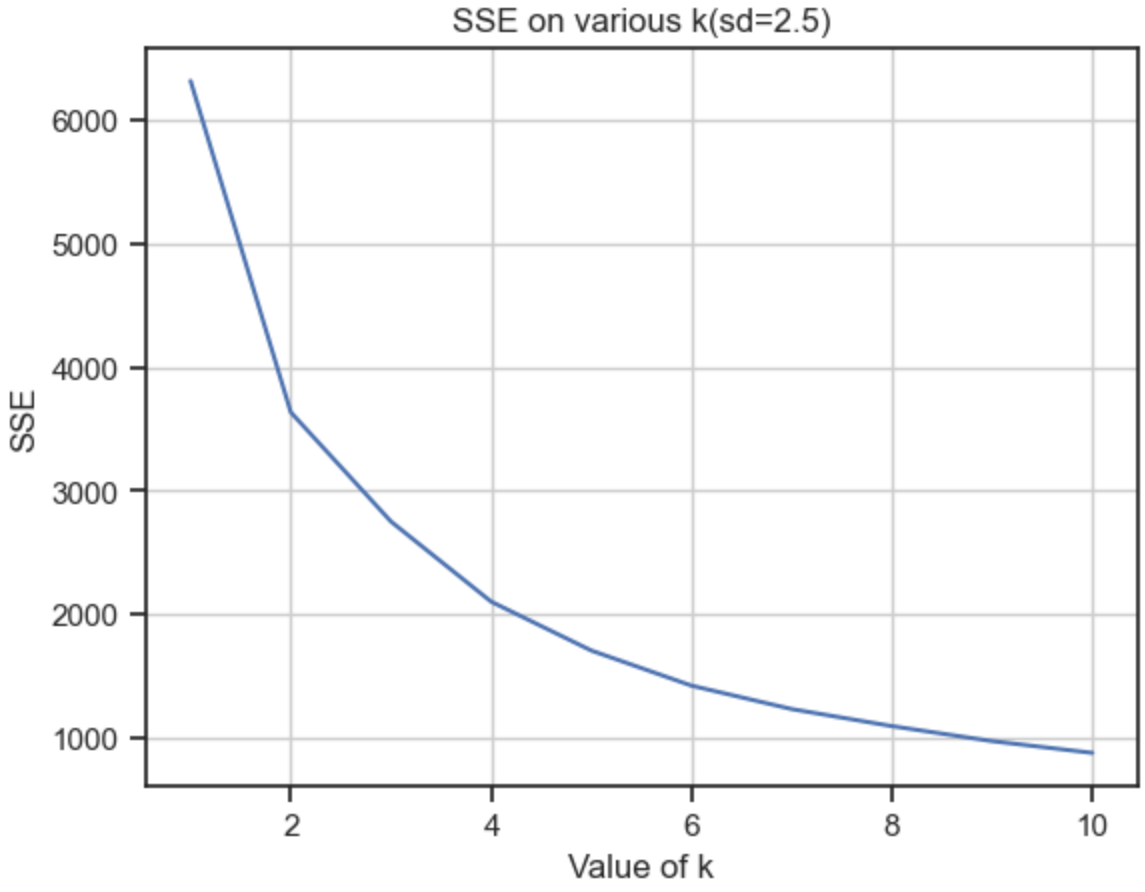
In [185…
```
from_a2c(sd=2.5, total_num=300, seed_random=100)
```

```
F:\ana\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known t
o have a memory leak on Windows with MKL, when there are less chunks than available thre
ads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```




```
[[75]
 [75]
 [75]
 [75]]
[[33 42]
 [70  5]
```

```
 [55 20]
 [ 6 69]]
[[32 28 15]
 [ 4 69  2]
 [ 7 40 28]
 [32  5 38]]
[[14 13 29 19]
 [41  2  4 28]
 [ 3 14  6 52]
 [ 1 43 25  6]]
[[35 10 16  4 10]
 [13  1  0 25 36]
 [27 15  3 29  1]
 [ 6 36 30  2  1]]
[[12  9 24  4  7 19]
 [32  0  2 26  1 14]
 [ 2  2  4 22 16 29]
 [ 1 30 10  2 26  6]]
[[ 4 20 14  9 21  7  0]
 [ 9 12 28  0  1  1 24]
 [26 25  3  2  4 11  4]
 [ 1  7  2 29 11 25  0]]
[[18  7 22  0  4  1 12 11]
 [14  0  1 22  9  0  2 27]
 [20  2  4  4 27  7  9  2]
 [ 5 26 11  0  2 12 19  0]]
[[ 7  6 11 23  3  1 16  8  0]
 [ 0 24  2  1  7  0  9 21 11]
 [ 2  6  6  4 24  7 24  1  1]
 [24  2 21 10  2 12  4  0  0]]
[[ 3 18  9 19  8  1  6  0  5  6]
 [ 8 10  1  1 22  0  0 11  0 22]
 [20 24  3  4  1 13  0  1  2  7]
 [ 1  6 31  6  0  3  4  0 22  2]]
```



Task 2

(a) Load and print the vertebrate.csv data

In [49]:
```
## import csv file
vertebrate_data = pd.read_csv('F://UM//Data Mining//Assign_3//vertebrate.csv')

vertebrate_data
```

Out[49]:

	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

In [51]:
```
name_col  = list(vertebrate_data['Name'])
class_col = vertebrate_data['Class']
num_cols = vertebrate_data.drop(columns=['Name', 'Class'])
```

In [73]:
```
# combine the name and class
name_calss_col = [name_col[i]+'_'+class_col[i] for i in range(len(name_col))]
```

Out[73]:
```
'python_reptiles'
```

In [52]:
```
num_cols.head()
```

Out[52]:

	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates
0	1	1	0	0	1	0
1	0	0	0	0	0	1
2	0	0	1	0	0	0
3	1	1	1	0	0	0
4	0	0	1	0	1	1

(b) Run single-link, max-link and average-link hierarchical clustering

Run single-link, max-link and average-link hierarchical clustering on this data. Visualize the hierarchies. Choose the hierarchy that in your view is most natural given the data and explain why.

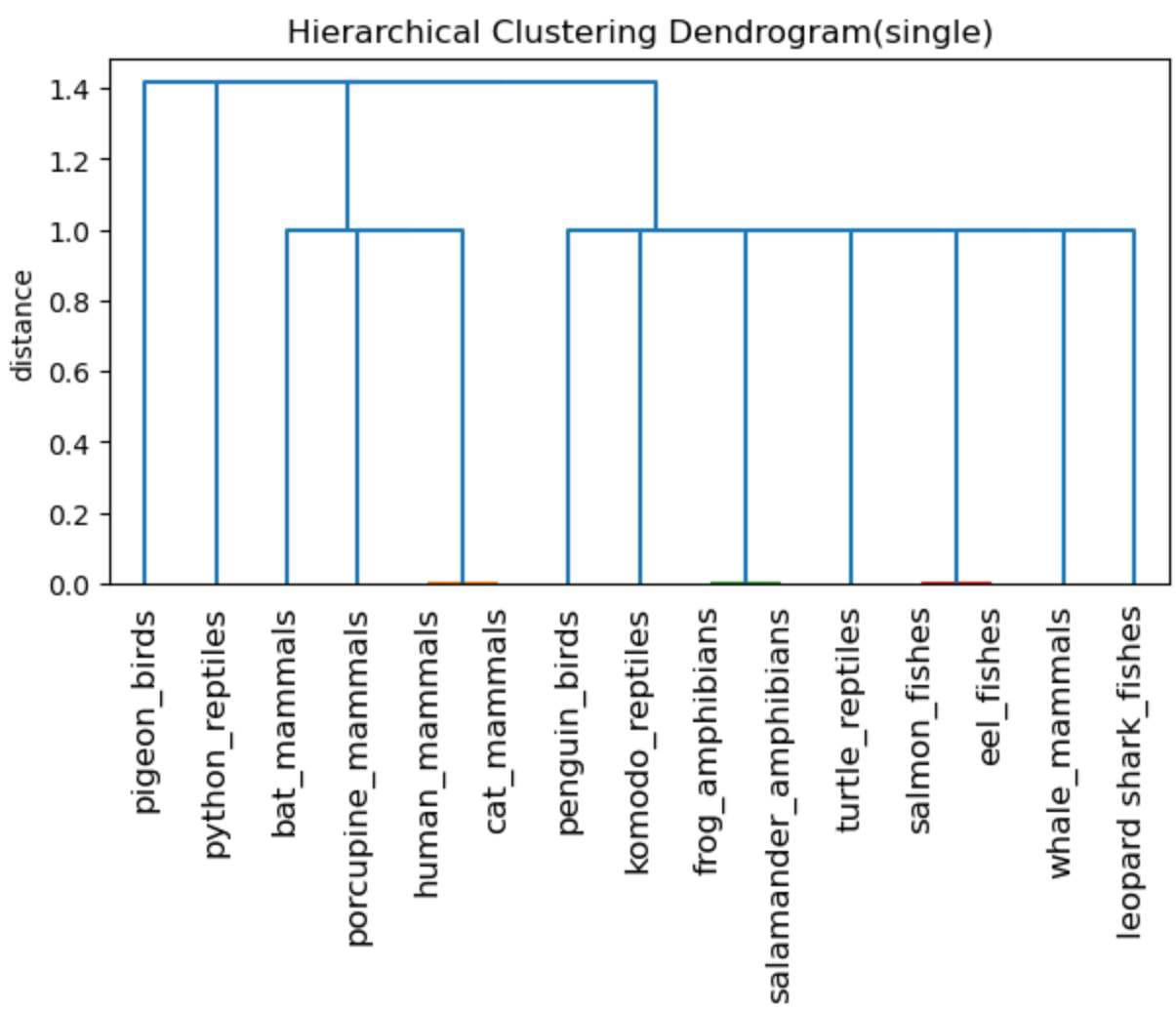
In [53]:
```
from scipy.cluster.hierarchy import linkage, dendrogram
```

In [54]:
```
np_data = num_cols.to_numpy()
```

In [74]:
```
# define function to apply single-link, max-link and average-link
def cluster_sig_max_ave(med):
    # med: the method choosen
    cluster = linkage(np_data, method=med)

    plt.figure(figsize=(7, 3.5))
    plt.title('Hierarchical Clustering Dendrogram('+str(med)+')')
    # plt.xlabel('sample index')
    plt.ylabel('distance')
    dendrogram(
        cluster,
        labels=name_calss_col,
        leaf_rotation=90.,  # rotates the x axis labels
        leaf_font_size=12,  # font size for the x axis labels
        color_threshold= .6
    )
    plt.show()
```

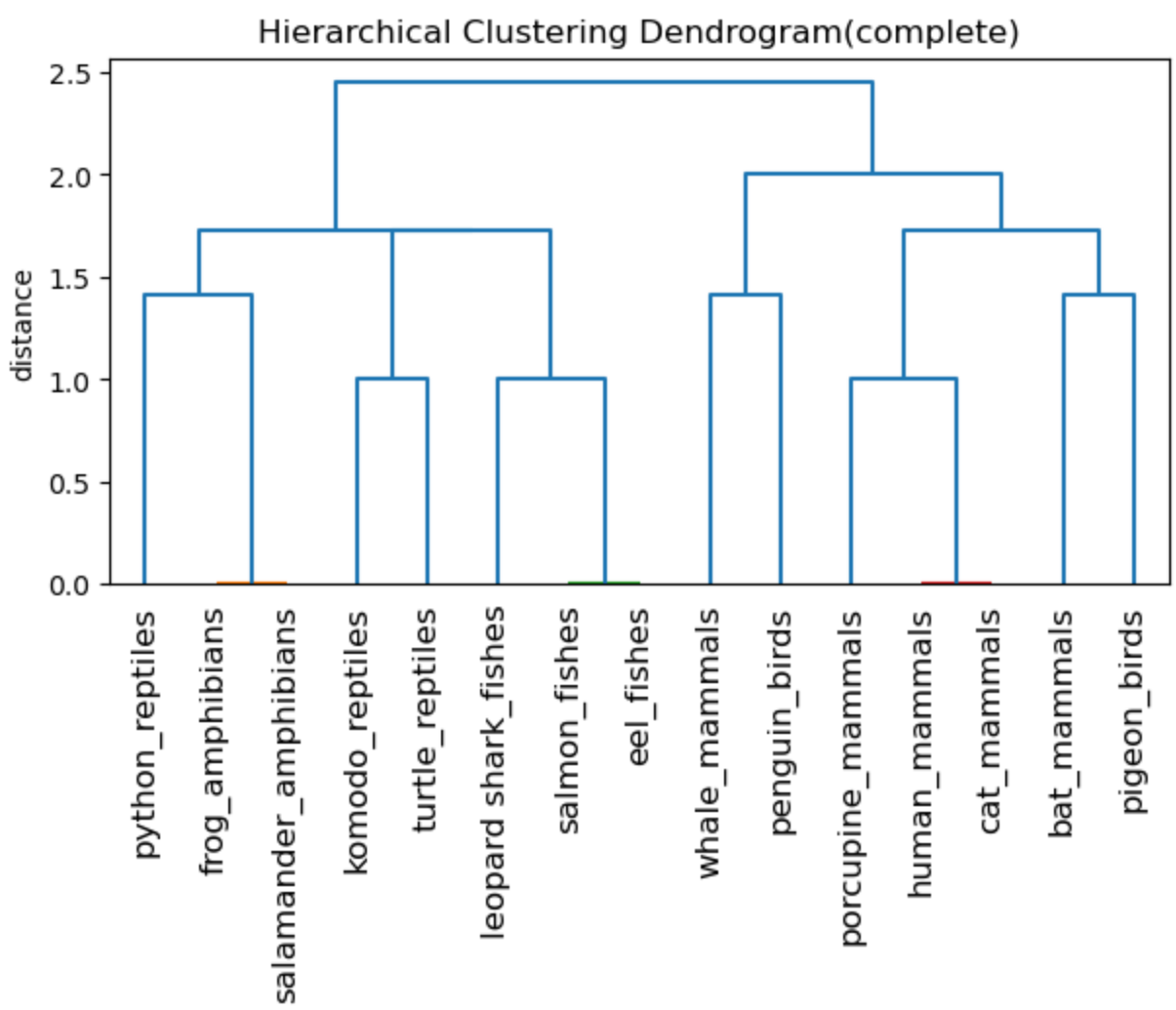
single-link

In [75]:
```
cluster_sig_max_ave('single')
```



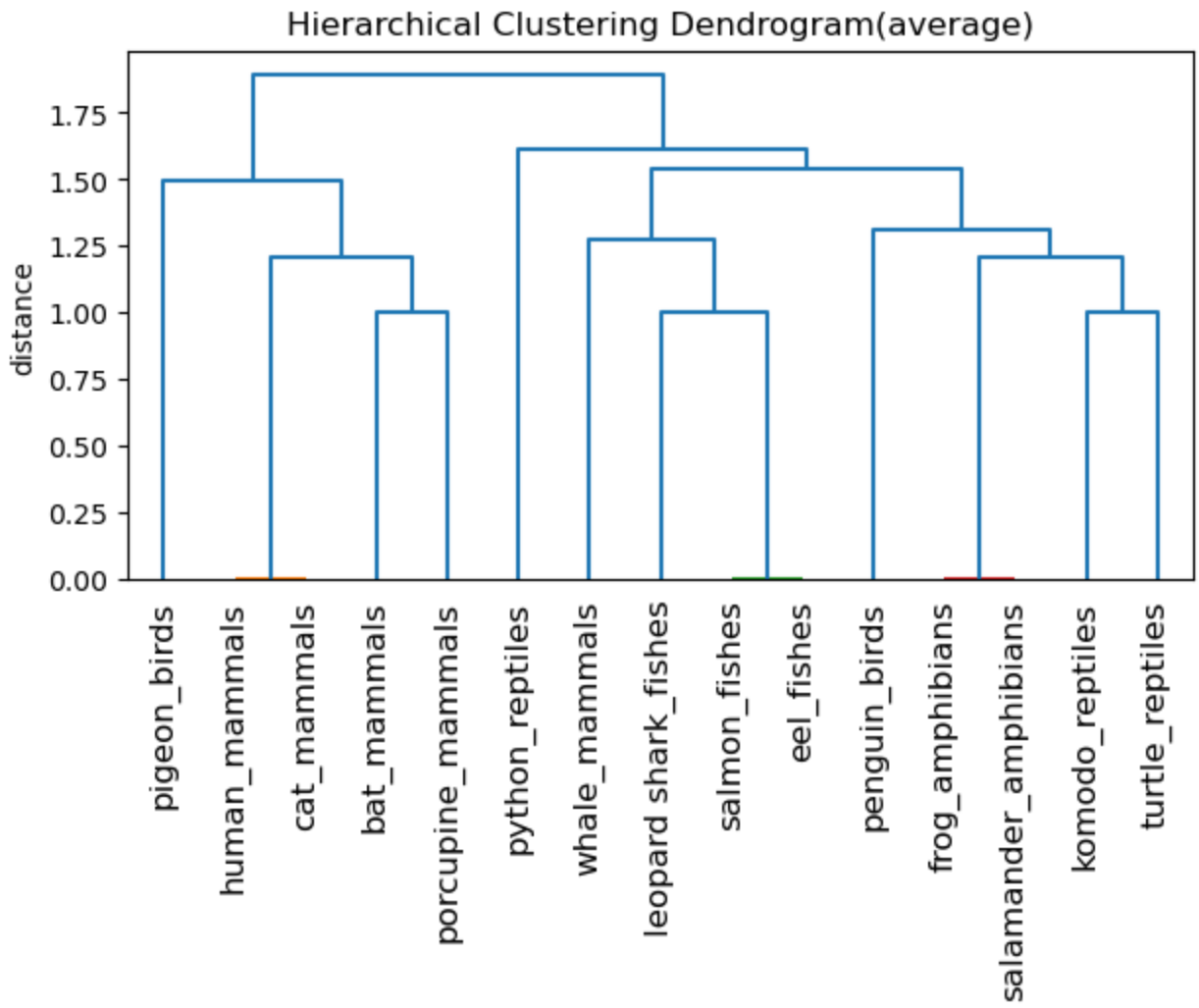
Max-link (Complete Link)

In [76]:
```
cluster_sig_max_ave('complete')
```



average-link

In [77]:
```
cluster_sig_max_ave('average')
```



Task 3

(a) Load and visualize the chameleon.csv data.

In [299…
```
## import csv file
data_chameleon = pd.read_csv('F://UM//Data Mining//Assign_3//chameleon.csv')

data_chameleon.head()
```

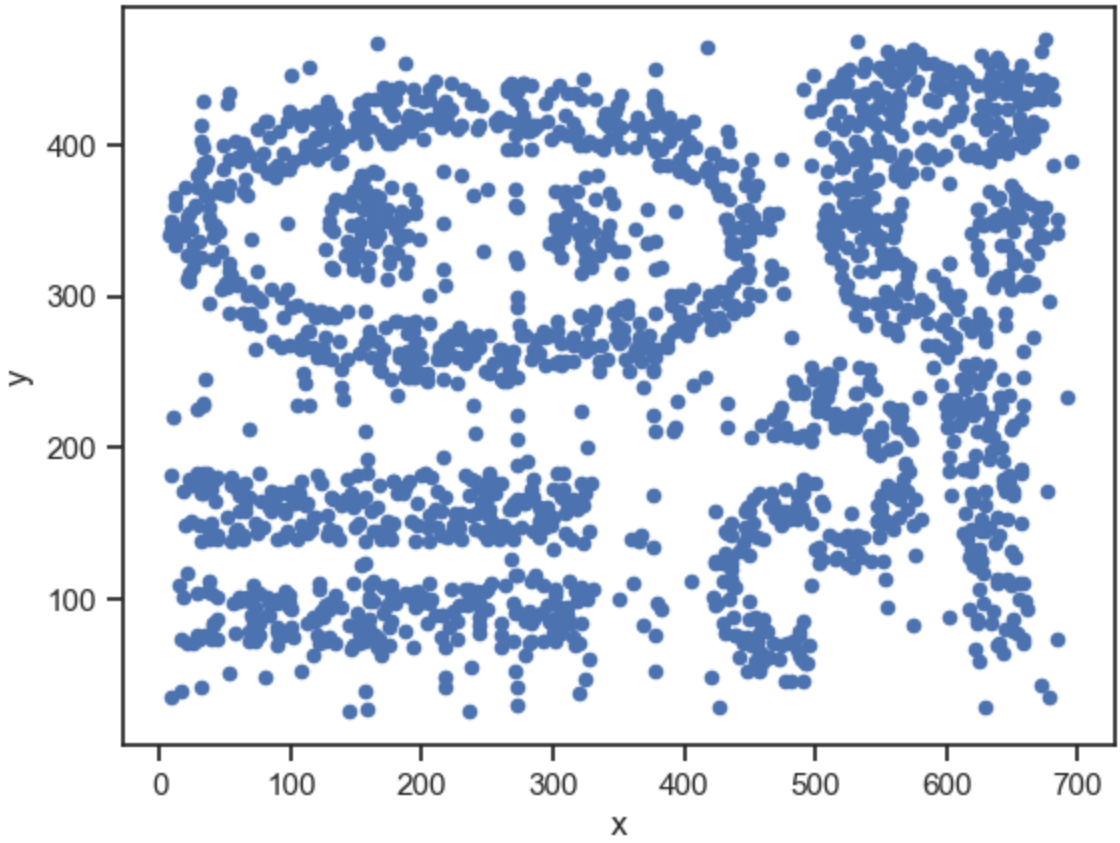
Out[299]:

	x	y
0	650.914	214.888
1	41.767	179.408
2	509.126	233.749
3	486.403	152.427
4	46.883	367.904

In [300…
```
data_chameleon.plot.scatter(x='x',y='y')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
<AxesSubplot:xlabel='x', ylabel='y'>
```

Out[300]:



(b) Run the DBSCAN method

Run the DBSCAN method on this data for eps=15.5 and min_samples=5. Vizualize the clustering solutions.

In [271…

```
from sklearn.cluster import DBSCAN
```

In [324…

```
db = DBSCAN(eps=15.5, min_samples=5).fit(data_chameleon)

core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = pd.DataFrame(db.labels_)

data_chameleon['group'] = labels
data_chameleon.head()
```

Out[324]:

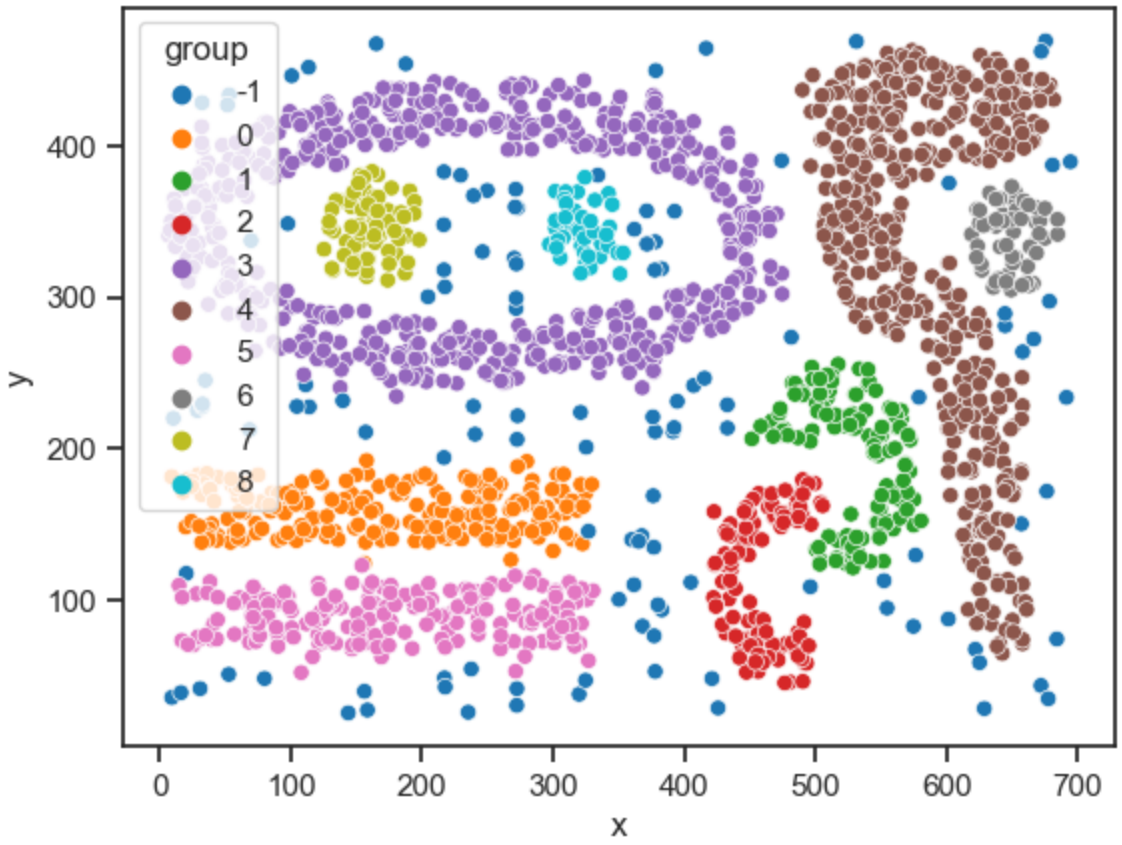
	x	y	group
0	650.914	214.888	4
1	41.767	179.408	0
2	509.126	233.749	1
3	486.403	152.427	2
4	46.883	367.904	3

In [325…

```
sns.scatterplot(data = data_chameleon, x='x',y='y',hue='group', palette="tab10")
```

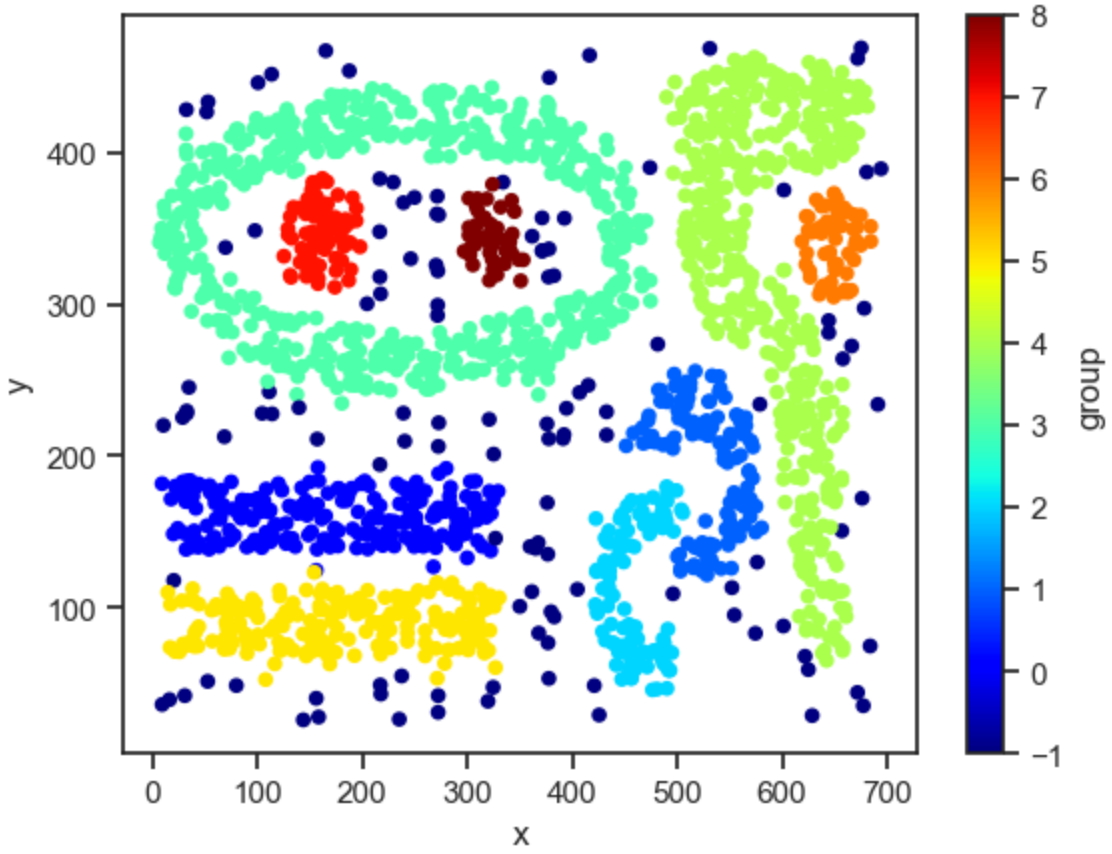
Out[325]:

```
<AxesSubplot:xlabel='x', ylabel='y'>
```



In [326…
```
data_chameleon.plot.scatter(x='x',y='y',c='group', colormap='jet')
```

Out[326]:
```
<AxesSubplot:xlabel='x', ylabel='y'>
```

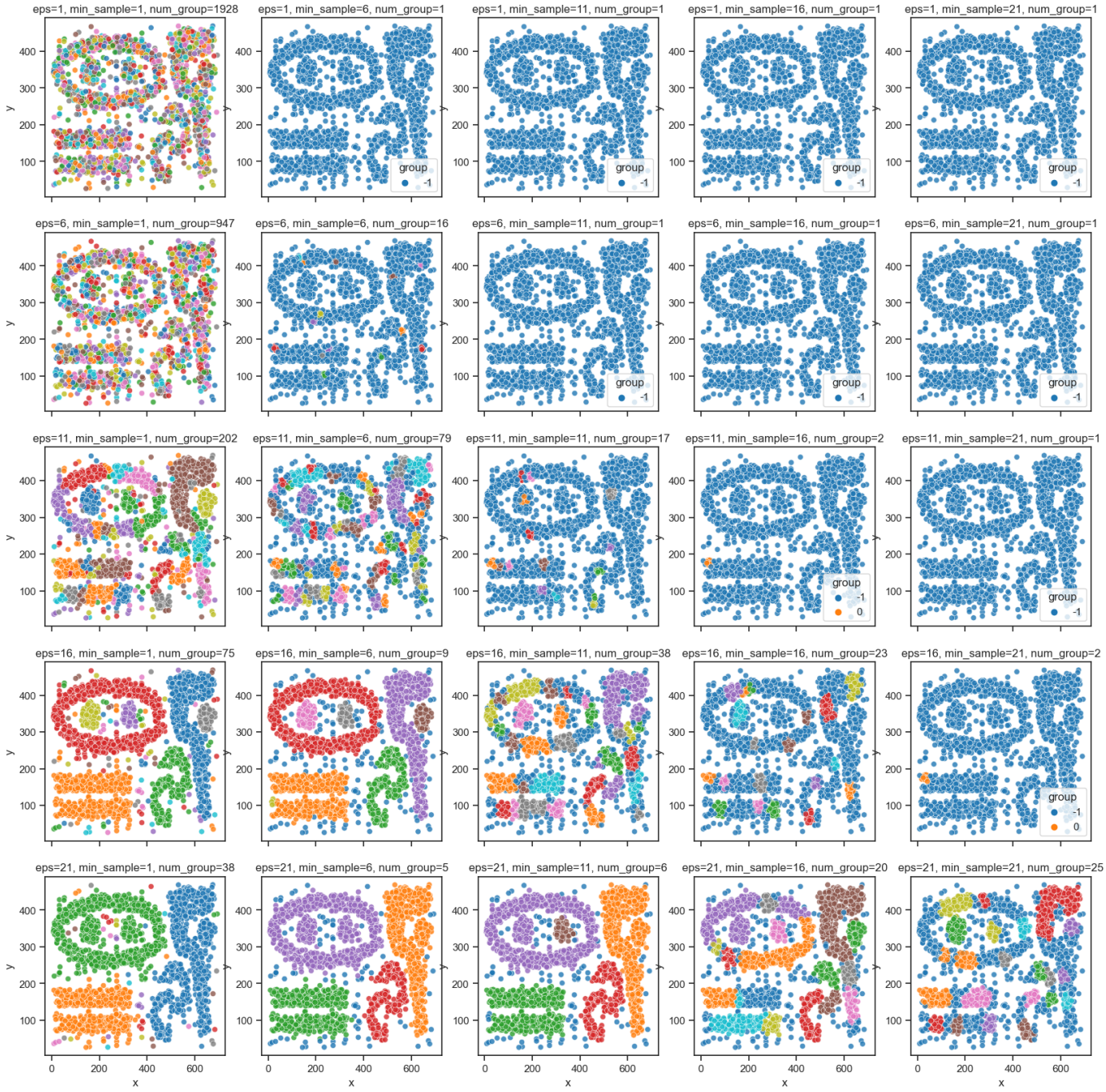


(c) Experiment with the DBSCAN method

for eps in [1, 21] with step 5 and min_samples in [1, 21] with step 5. Comment on the clusters for different settings.

In [344…

```
%matplotlib inline

figure, axes = plt.subplots(5, 5, sharex=True, figsize=(20,20))
figure.suptitle('Experiment with the DBSCAN method with different eps and min_sample', f

for ep in range(1, 21+1, 5):
    for min_sam in range(1, 21+1, 5):

        db = DBSCAN(eps=ep, min_samples=min_sam).fit(data_chameleon)

        core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
        core_samples_mask[db.core_sample_indices_] = True
        labels = pd.DataFrame(db.labels_)

        data_chameleon['group'] = labels

        num_row = ep // 5
        num_col = min_sam // 5

        num_legend = len(set(list(data_chameleon['group'])))

        sub_tit = 'eps='+str(ep)+', min_sample='+str(min_sam)+', num_group='+str(num_leg
        if num_legend <=4 :
            show_leg = True
        else:
            show_leg = False
        sns.scatterplot(ax=axes[num_row, num_col], data=data_chameleon,
                        x='x', y='y', hue='group', alpha=0.8,
                        palette="tab10", legend=show_leg).set(title=sub_tit)
```



In [345…]
```
figure.savefig("Experiment with the DBSCAN.pdf")
```

In [ ]: