Лабораторна робота №6. Кластеризація

Виконав студент групи КМ-91мп

Галета М.С.

Завдання на лабораторну роботу

1. Використовуються набори даних з лабораторної роботи №1.
2. Обравши всі стовпчики, крім останнього:
   a) Провести розбиття методом K-means на два та три кластери. Проілюструвати результати розбиття графічно.
   b) Провести розбиття на аналогічну кількість кластерів за допомогою інших двох методів (один з яких використовує аналогічну метрику, а другий – іншу). Результати розбиття проілюструвати графічно, виконати порівняльний аналіз отриманих результатів із результатами попереднього пункту.
   c) Збудувати дендрограму для перших 100 записів.
3. Проаналізувати отримані результати, враховуючи останній стовпчик початкових даних.
4. Для отримання результатів можна використовувати бібліотеки для мови Python (наприклад, scikit-learn або аналогічні).

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import plotly.figure_factory as ff
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
```

Зчитування датасету

In [2]:

```
df = pd.read_csv('MP-04-Galeta.csv', delimiter=';', names=['x1', 'x2', 'x3', 'x4', 'x5

X = df.drop(columns='y')
y = df['y']
```

Центрування і стандартизація даних

$$X_{new} = \frac{X - \mu}{\sigma}$$

In [3]:

```
class Scaler:
    def __init__(self):
        self.mean = None
        self.std = None

    def fit(self, X):
        self.mean = np.mean(X, axis=0, keepdims=True)
        self.std = np.std(X, axis=0, keepdims=True)

    def transform(self, X):
        X_new = (X - self.mean)/self.std
        return X_new

    def fit_transform(self, X):
        self.mean = np.mean(X, axis=0, keepdims=True)
        self.std = np.std(X, axis=0, keepdims=True)
        X_new = (X - self.mean)/self.std
        return X_new
```

In [4]:

```
sc = Scaler()
X = pd.DataFrame(columns=['x1', 'x2', 'x3', 'x4', 'x5', 'x6'], data=sc.fit_transform(X
```

KMeans

In [5]:

```
km2 = KMeans(2, random_state=42)
km3 = KMeans(3, random_state=42)

km2.fit(X)
km3.fit(X)
```

Out[5]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

In [6]:

```
km2_res = km2.predict(X)
km3_res = km3.predict(X)

pca_embeddings = PCA(2).fit_transform(X)
```

In [7]:

```
f, axes = plt.subplots(1, 2, figsize=(12, 6), sharex=True)
sns.scatterplot(x=pca_embeddings[:, 0], y=pca_embeddings[:, 1], hue=km2_res, ax=axes[0
sns.scatterplot(x=pca_embeddings[:, 0], y=pca_embeddings[:, 1], hue=km3_res, ax=axes[1
axes[0].set_title('KMeans 2 clusters')
axes[1].set_title('KMeans 3 clusters')
plt.show()
```

AgglomerativeClustering & GaussianMixture

In [8]:

```
agg2 = AgglomerativeClustering(2)
agg3 = AgglomerativeClustering(3)
gaus2 = GaussianMixture(2, random_state=42)
gaus3 = GaussianMixture(3, random_state=42)
```

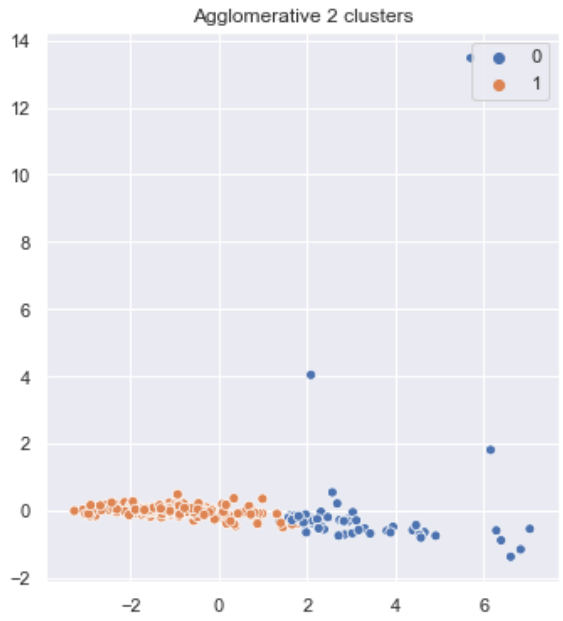
In [9]:

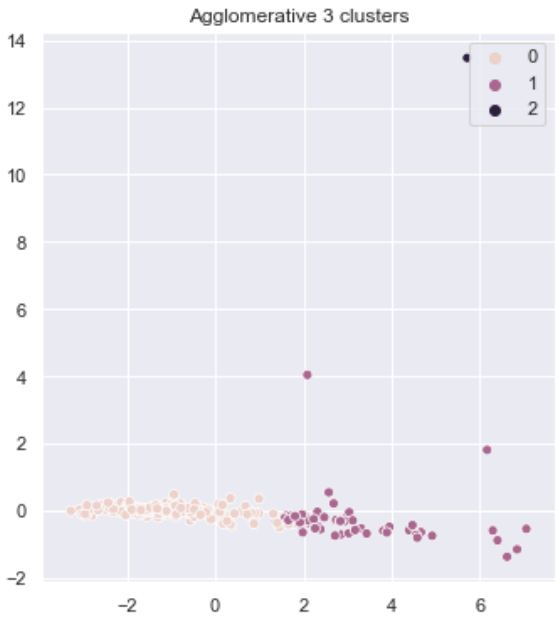
```
agg2_res = agg2.fit_predict(X)
agg3_res = agg3.fit_predict(X)
gaus2_res = gaus2.fit_predict(X)
gaus3_res = gaus3.fit_predict(X)
```

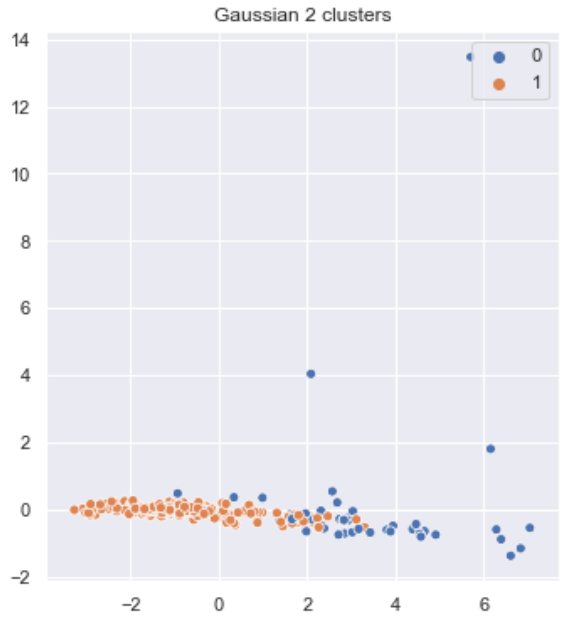
In [10]:

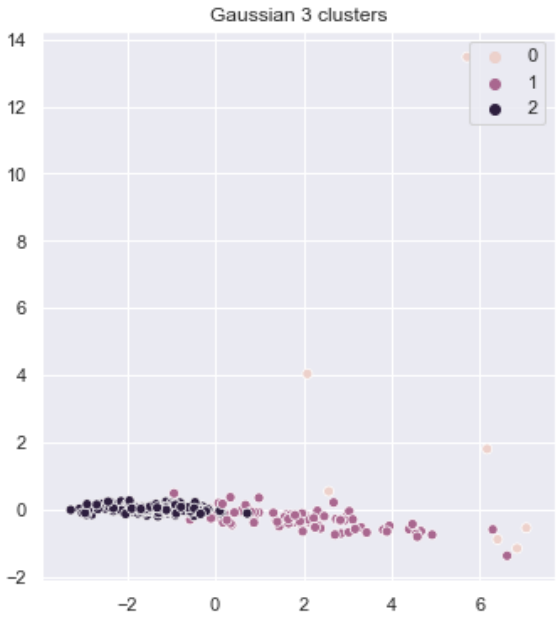
```
f, axes = plt.subplots(1, 2, figsize=(12, 6), sharex=True)
sns.scatterplot(x=pca_embeddings[:, 0], y=pca_embeddings[:, 1], hue=agg2_res, ax=axes[
sns.scatterplot(x=pca_embeddings[:, 0], y=pca_embeddings[:, 1], hue=agg3_res, ax=axes[
axes[0].set_title('Agglomerative 2 clusters')
axes[1].set_title('Agglomerative 3 clusters')

f, axes = plt.subplots(1, 2, figsize=(12, 6), sharex=True)
sns.scatterplot(x=pca_embeddings[:, 0], y=pca_embeddings[:, 1], hue=gaus2_res, ax=axes
sns.scatterplot(x=pca_embeddings[:, 0], y=pca_embeddings[:, 1], hue=gaus3_res, ax=axes
axes[0].set_title('Gaussian 2 clusters')
axes[1].set_title('Gaussian 3 clusters')
plt.show()
```









In [11]:

```
res_list = [
    km2_res, agg2_res, gaus2_res,
    km3_res, agg3_res, gaus3_res
]
```

In [12]:

```
mean_dist = []
dist_matrix = np.mean(np.square(X.values - X.values[:, np.newaxis]), axis=2)
for res in res_list:
    df_res = pd.concat((X, pd.DataFrame({'res':res})), axis=1)
    dist_mean_dict = {}
    for cluster in np.unique(df_res['res']):
        df_cluster_index = df_res[df_res['res'] == cluster].drop('res', axis=1).index
        dist_mean = []
        for inx in df_cluster_index:
            dist_mean.append(dist_matrix[df_cluster_index, inx].mean())
        dist_mean_dict[cluster] = np.mean(dist_mean)
    mean_dist.append(dist_mean_dict)
```

In [13]:

```
for inx, val in enumerate(mean_dist):
    print(f'Метод {inx}, кількість кластерів {len(val)}, середня внутрішньокласова відс
```

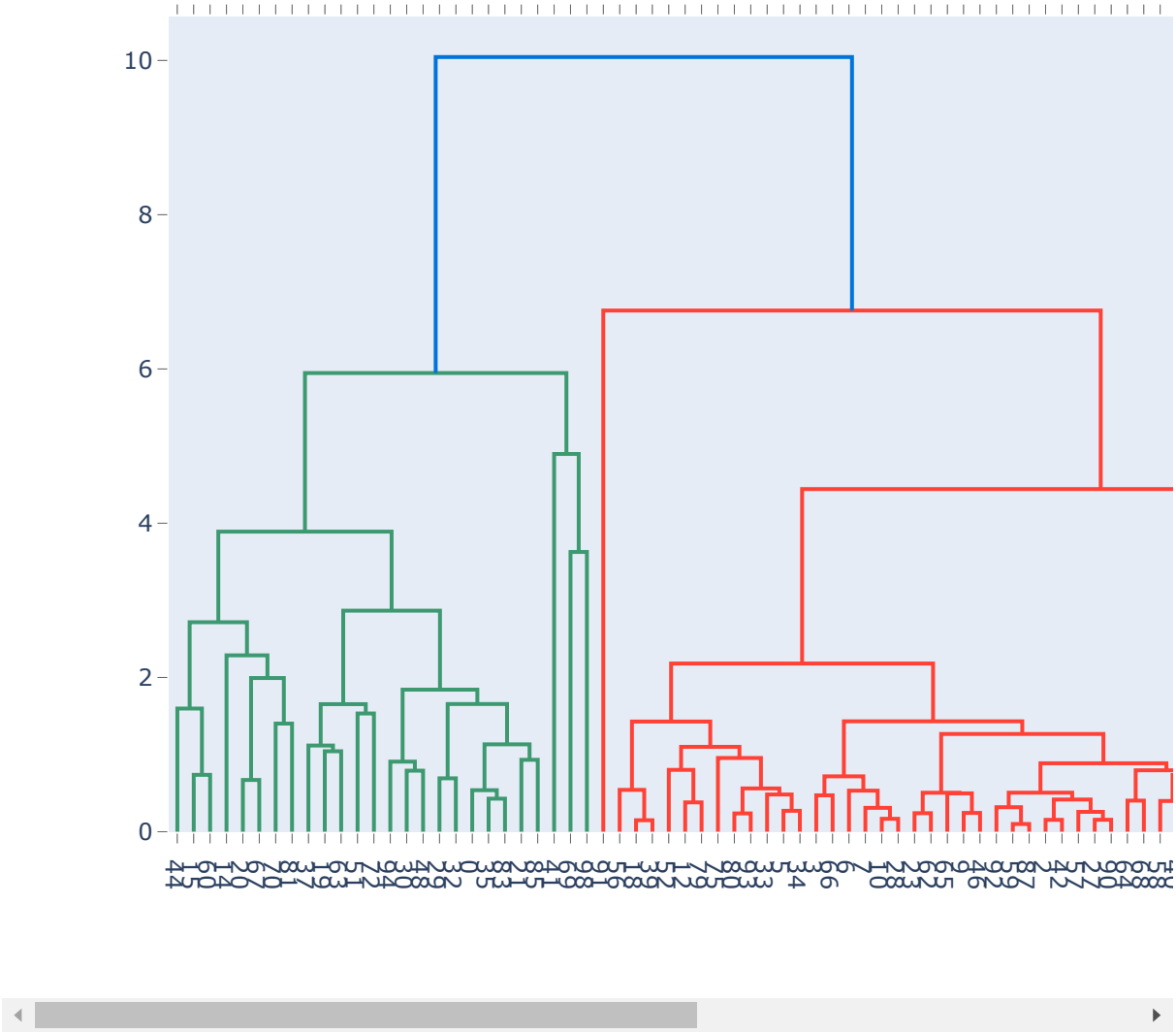
```
Метод 0, кількість кластерів 2, середня внутрішньокласова відстань {0: 0.395
06906746054954, 1: 2.329417115263443}, середня внутрішньокласова відстань по
методу 1.3622430913619963
Метод 1, кількість кластерів 2, середня внутрішньокласова відстань {0: 2.474
927845553782, 1: 0.48936205043248404}, середня внутрішньокласова відстань по
методу 1.4821449479931332
Метод 2, кількість кластерів 2, середня внутрішньокласова відстань {0: 3.481
1839979965566, 1: 0.7773732184660165}, середня внутрішньокласова відстань по
методу 2.1292786082312865
Метод 3, кількість кластерів 3, середня внутрішньокласова відстань {0: 0.395
06906746054954, 1: 1.4215683411463262, 2: 0.0}, середня внутрішньокласова ві
дстань по методу 0.6055458028689585
Метод 4, кількість кластерів 3, середня внутрішньокласова відстань {0: 0.489
36205043248404, 1: 1.426270699623292, 2: 0.0}, середня внутрішньокласова від
стань по методу 0.6385442500185919
Метод 5, кількість кластерів 3, середня внутрішньокласова відстань {0: 10.04
5770369590437, 1: 1.247823123317706, 2: 0.30382355147152684}, середня внутрі
шньокласова відстань по методу 3.8658056814598907
```

Найкращим виявився метод KMeans, як для двох, так і для трьох кластерів

Дендрограма для перших 100 записів

In [14]:

```
fig = ff.create_dendrogram(X.head(100))
fig.update_layout(width=1000, height=600)
fig.show()
```



3. Проаналізувати отримані результати, враховуючи останній стовпчик початкових даних

In [15]:

```
res_dict = {
    'km2_res': km2_res,
    'km3_res': km3_res,
    'agg2_res': agg2_res,
    'agg3_res': agg3_res,
    'gaus2_res': gaus2_res,
    'gaus3_res': gaus3_res,
    'y': y
}
res_df = pd.DataFrame(res_dict)
```

In [16]:

```
res_df.groupby('y')['km2_res', 'agg2_res', 'gaus2_res'].agg([np.mean, np.std])
```

Out[16]:

	km2_res		agg2_res		gaus2_res	
	mean	std	mean	std	mean	std
y						
0.000	0.0	0.0	1	0.0	1	0.0
0.525	0.0	NaN	1	NaN	1	NaN
0.556	0.0	NaN	1	NaN	1	NaN
0.825	1.0	NaN	0	NaN	0	NaN
0.842	1.0	NaN	1	NaN	1	NaN
0.860	1.0	NaN	1	NaN	1	NaN
0.884	0.0	NaN	1	NaN	1	NaN
1.009	0.0	NaN	1	NaN	1	NaN

In [17]:

```
res_df.groupby('y')['km3_res', 'agg3_res', 'gaus3_res'].agg(
    lambda x: x.value_counts().index[0]
)
```

Out[17]:

y	km3_res	agg3_res	gaus3_res
0.000	0	0	2
0.525	0	0	2
0.556	0	0	2
0.825	1	1	0
0.842	1	0	1
0.860	1	0	1
0.884	0	0	2
1.009	0	0	2
1.112	1	1	0