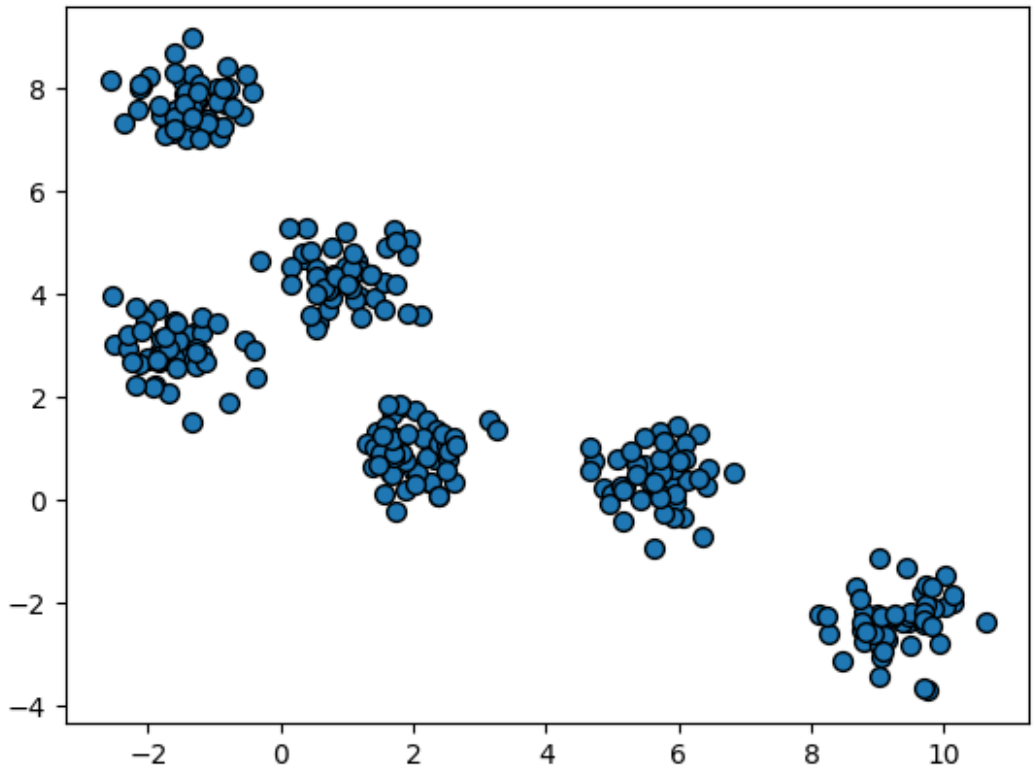
Домашнее задание 6 - Шмаков Владимир Б04-105

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import make_circles, make_blobs
from sklearn import cluster
from matplotlib import cm
from sklearn.metrics import silhouette_score, calinski_harabasz_score,
davies_bouldin_score, mean_squared_error
from sklearn.cluster import AgglomerativeClustering
import scipy.stats as sts
from scipy import  optimize
from matplotlib.animation import  FuncAnimation
import hdbscan
from sklearn.metrics import adjusted_rand_score as ari
```

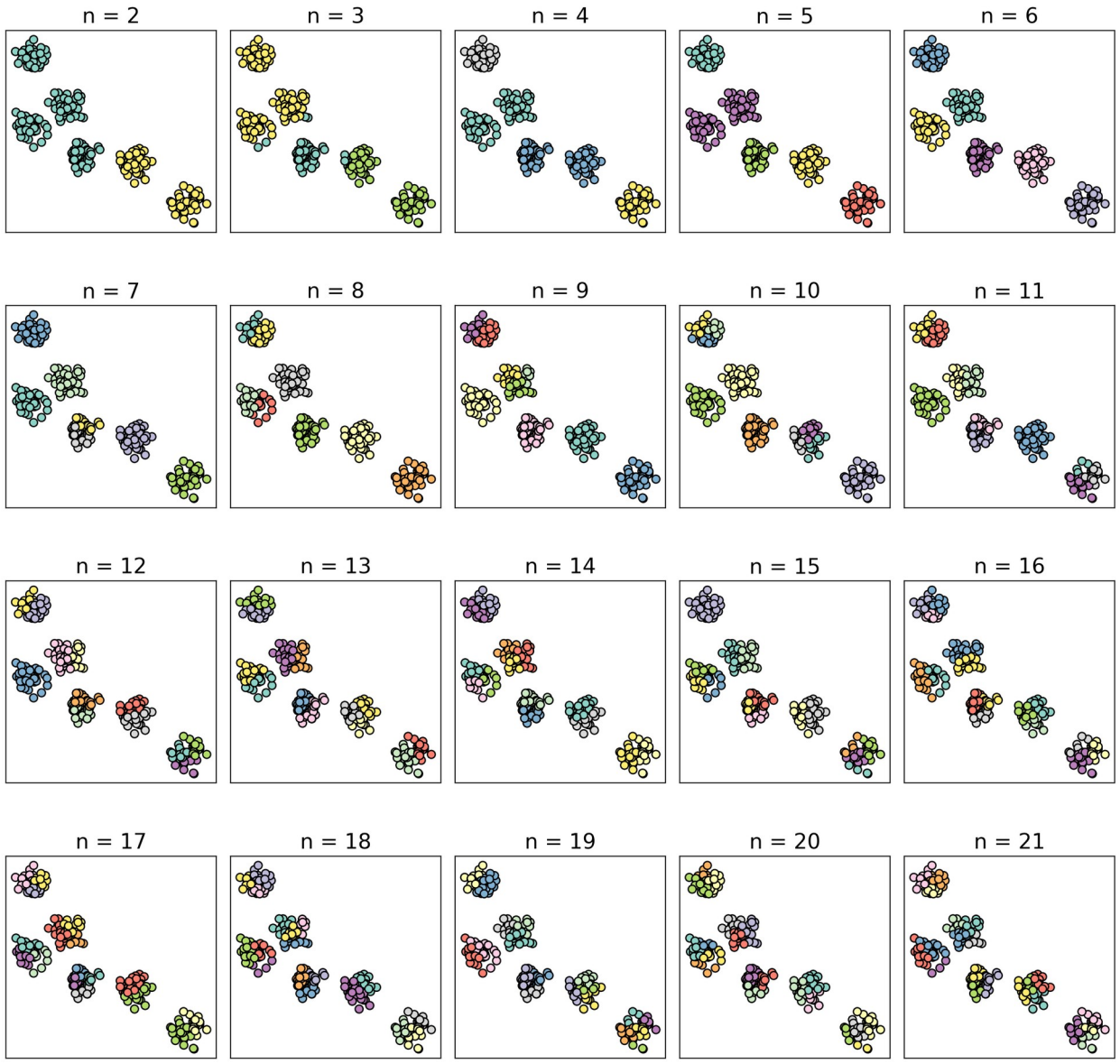
Задача 1

Кластеризуйте данный датасет с помощью метода `k-means`. На основе трёх различных **внутренних** метрик оценки кластеризации, подберите наилучшее количество кластеров k.

```
X, y = make_blobs(n_samples = 300, centers = 6, cluster_std = 0.5,
random_state = 0)
plt.scatter(X[:, 0], X[:, 1], s=50, edgecolor='k')
plt.show()
```

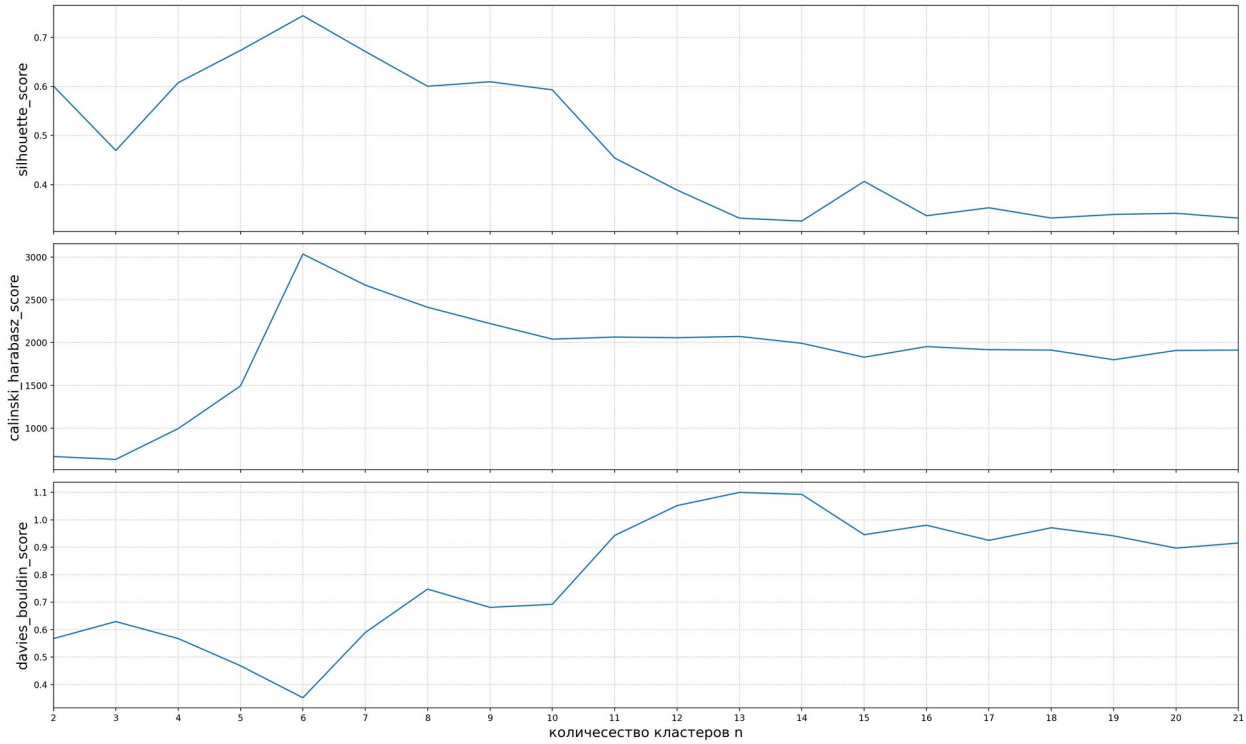



```
n_clusters_values = np.arange(20, dtype = np.int64) + 2
metric_names = ['silhouette_score', 'calinski_harabasz_score',
'davies_bouldin_score']
metric_callable = [silhouette_score, calinski_harabasz_score,
davies_bouldin_score]
metric_values = {n: [] for n in metric_names}
fig, ax = plt.subplots(4, len(n_clusters_values) // 4, figsize = (12,
12), dpi = 200)

for plotable, n in zip(ax.flatten(), n_clusters_values):
    k_means_model = cluster.KMeans(n_clusters = n, n_init =
'auto').fit(X)
    y_prediction = k_means_model.predict(X)
    plotable.scatter(X[:, 0], X[:, 1], c = y_prediction,
edgecolor='k', cmap = 'Set3')
    plotable.set_title(f'n = {n}', fontsize = 16)
    plotable.set_xticks([])
    plotable.set_yticks([])
    plotable.set_aspect('equal')
    for name, metric in zip(metric_names, metric_callable):
        metric_values[name].append(metric(X, y_prediction))

fig.tight_layout()
```




```
fig, ax = plt.subplots(len(metric_names), 1, figsize = (20, 12), dpi =
200, sharex = True)


for name, metric, plotable in zip(metric_names, metric_callable, ax):
        plotable.plot(n_clusters_values, metric_values[name], label =
name)
        plotable.set_xticks(n_clusters_values)
        plotable.grid(ls = ':')
        plotable.set_ylabel(name, fontsize = 16)
        plotable.set_xlim(np.min(n_clusters_values),
np.max(n_clusters_values))

ax[-1].set_xlabel("количесество кластеров n", fontsize = 16)
```

```
fig.tight_layout()
```



Вывод

Как видим, максимум метрик *silhoutte* и *calinski harabaz* достигается при $n=6$. При $n=6$ также достигается минимум метрики *davies bouldin*.

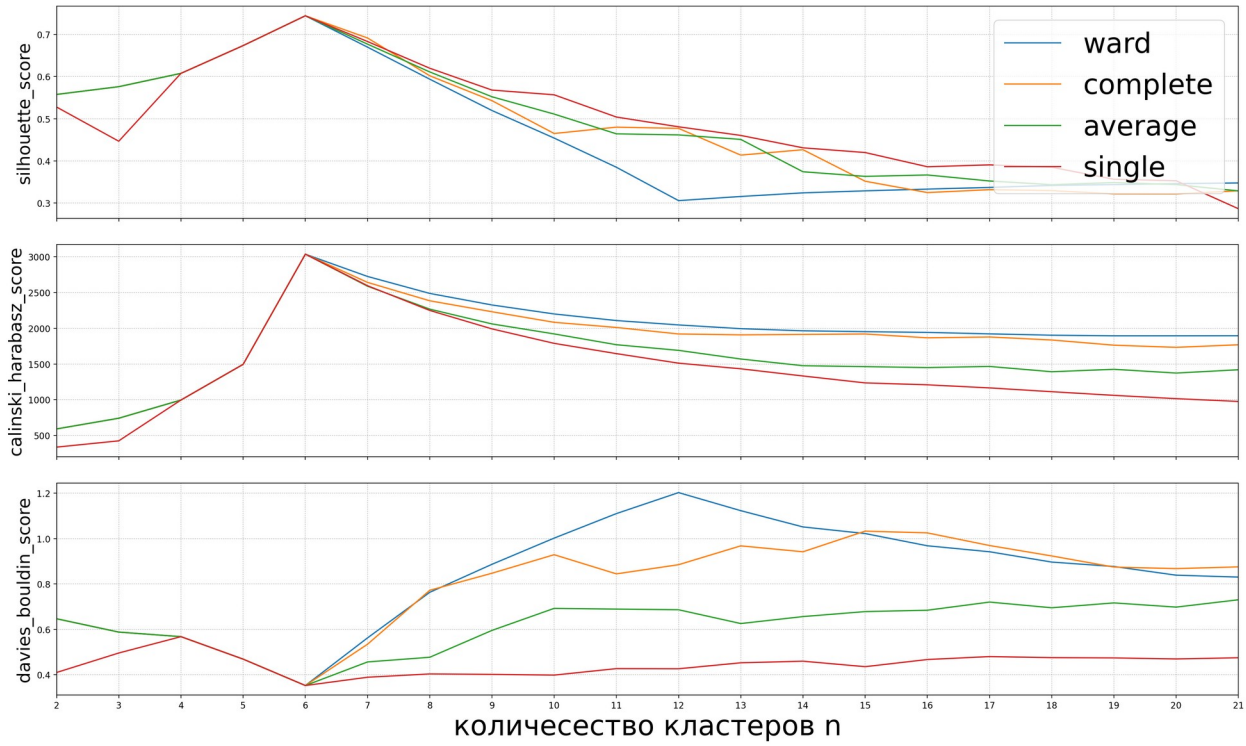
Согласно документации `sklearn` кластеризация считается наилучшей, если метрика *davies bouldin* минимальна.

Задача 2

Для предыдущего датасета подберите наилучшие гиперпараметры (мера несходства) с использованием агломеративного метода. Постройте дендрограмму для наилучшей модели.

```
param_linkage = ["ward", "complete", "average", "single"]
metric_names = ['silhouette_score', 'calinski_harabasz_score',
'davies_bouldin_score']
metric_callable = [silhouette_score, calinski_harabasz_score,
davies_bouldin_score]
metric_values = {linkage: {metr: [] for metr in metric_names} for
linkage in param_linkage}

for linkage in param_linkage:
    for n in n_clusters_values:
```

```
        model = AgglomerativeClustering(n_clusters = n, linkage =
linkage)
        y_prediction = model.fit_predict(X)
        for name, func in zip(metric_names, metric_callable):
            metric_values[linkage][name].append(func(X, y_prediction))

fig, ax = plt.subplots(len(metric_names), 1, figsize = (20, 12), dpi =
200, sharex = True)


for name, metric, plotable in zip(metric_names, metric_callable, ax):
    for linkage in param_linkage:
        plotable.plot(n_clusters_values, metric_values[linkage][name],
label = linkage)
        plotable.set_xticks(n_clusters_values)
        plotable.grid(ls = ':')
        plotable.set_ylabel(name, fontsize = 20)
        plotable.set_xlim(np.min(n_clusters_values),
np.max(n_clusters_values))


ax[-1].set_xlabel("количесество кластеров n", fontsize = 32)
ax[0].legend(fontsize = 32)
fig.tight_layout()
```



Как видим, все метрики достигают своих наилучших значений при параметрах `n = 6`.
Предположу что модель `single`(красная кривая) является наилучшей.

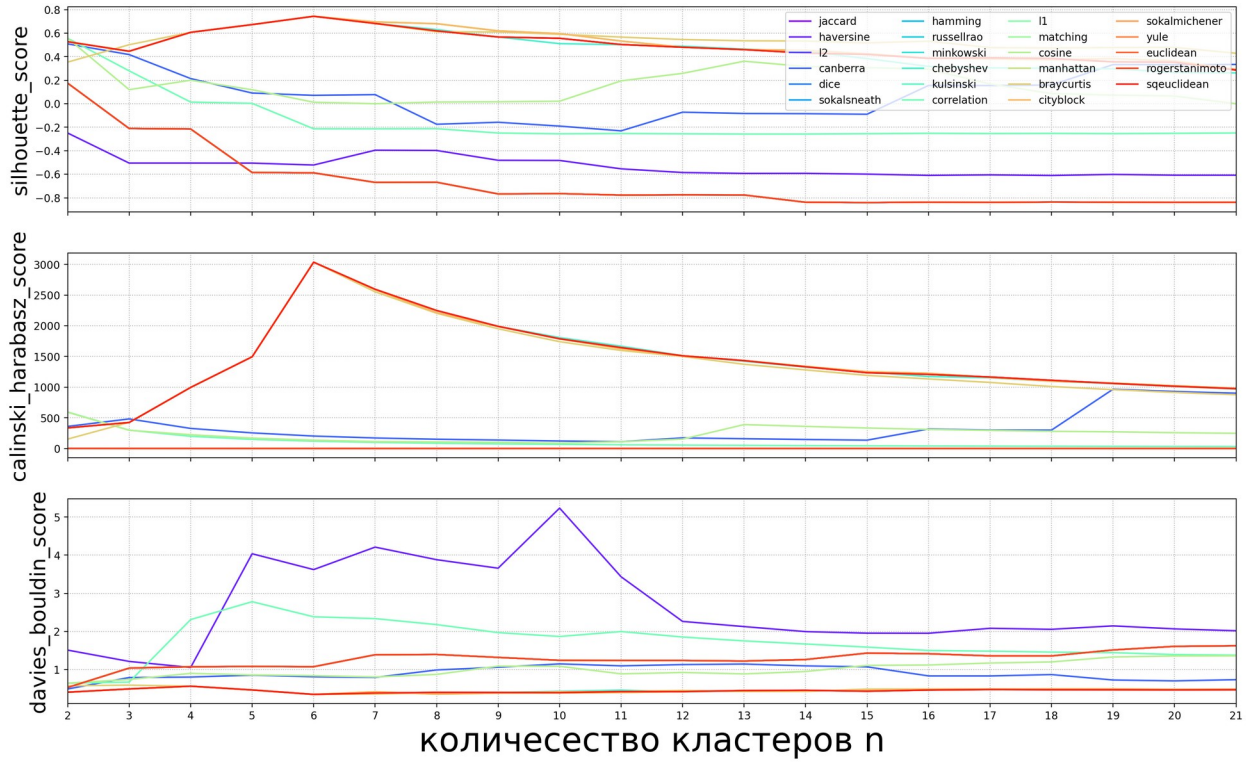
На рисунке выше видно, что красная кривая метрики `davies bouldin` лежит ниже остальных. Также красная кривая лежит заметно выше остальных при использовании метрики `sillhoutte`.

Посмотрим на кривые оценки качества моделей которые используют различные меры расстояния между точками

```
param_norm = ['jaccard', 'haversine', 'l2', 'canberra', 'dice',
'sokalsneath', 'hamming', 'russellrao', 'minkowski', 'chebyshev',
'kulsinski', 'correlation', 'l1', 'matching', 'cosine', 'manhattan',
'braycurtis', 'cityblock', 'sokalmichener', 'yule', 'euclidean',
'rogerstanimoto', 'sqeuclidean']
metric_names = ['silhouette_score', 'calinski_harabasz_score',
'davies_bouldin_score']
metric_callable = [silhouette_score, calinski_harabasz_score,
davies_bouldin_score]
metric_values = {norm: {metr: [] for metr in metric_names} for norm in
param_norm}

for norm in param_norm:
    print(norm)
    for n in n_clusters_values:
        model = AgglomerativeClustering(n_clusters = n, linkage =
'single', metric = norm)
        y_prediction = model.fit_predict(X)
        for name, func in zip(metric_names, metric_callable):
            metric_values[norm][name].append(func(X, y_prediction))
```

```
jaccard

haversine
l2
canberra
dice
sokalsneath
hamming
russellrao
minkowski
chebyshev
kulsinski
correlation
l1
matching
cosine
manhattan
braycurtis
cityblock
sokalmichener
yule
euclidean
```

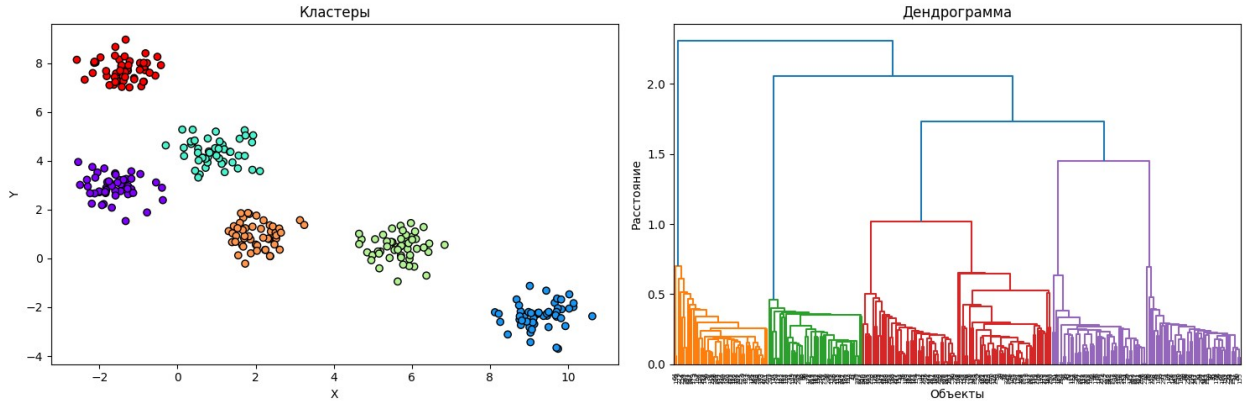
```
rogerstanimoto
sqeuclidean

fig, ax = plt.subplots(len(metric_names), 1, figsize = (20, 12), dpi =
200, sharex = True)


for name, metric, plotable in zip(metric_names, metric_callable, ax):
    for ind, norm in enumerate(param_norm):
        plotable.plot(n_clusters_values, metric_values[norm][name],
label = norm, color = cm.rainbow(ind / len(param_norm)))
        plotable.set_xticks(n_clusters_values)
        plotable.grid(ls = ':')
        plotable.set_ylabel(name, fontsize = 20)
        plotable.set_xlim(np.min(n_clusters_values),
np.max(n_clusters_values))


ax[-1].set_xlabel("количесество кластеров n", fontsize = 32)
ax[0].legend(fontsize = 10, loc = 'upper right', ncol = 4);
```



Согласно графикам выше, норма `cosine` оказалась наихучшей. Остальные метрики ведут себя практически одинаково. И достигают наилучших значений при `n = 6`.

```
from scipy.cluster.hierarchy import dendrogram, linkage
```

```
best_model = AgglomerativeClustering(n_clusters = 6,
                                     linkage = 'single',
                                      metric = 'manhattan')
clusters = best_model.fit_predict(X)

# Ссылка
Z = linkage(X, method = 'single')

# Создание субплотов
fig, axs = plt.subplots(1, 2, figsize=(15, 5))

# Левый субплот с кластерами
axs[0].scatter(X[:, 0], X[:, 1], c = clusters, cmap='rainbow',
edgecolor='k')
axs[0].set_title('Кластеры')
axs[0].set_xlabel('X')
axs[0].set_ylabel('Y')

# Правый субплот с дендрограммой
dendrogram(Z, ax=axs[1])
axs[1].set_title('Дендрограмма')
axs[1].set_xlabel('Объекты')
axs[1].set_ylabel('Расстояние')

plt.tight_layout()
plt.show()
```



Задача 3

3. Матрица расстояний (смежности) и матрица сходств

Расстояния между парами векторов из двух множеств $d\left(X_l, X_j\right)$ могут быть представлены в виде симметричной матрицы расстояний (матрица смежности):

$$D=\begin{pmatrix} 0 & d_{12} & \dots & d_{1n} \\ d_{21} & 0 & \dots & d_{2n} \\ \dots & \dots & \dots & \dots \\ d_{n1} & d_{n2} & \dots & 0 \end{pmatrix}$$

Понятием, противоположным расстоянию, является понятие сходства между объектами. Неотрицательная вещественная функция $S(x_i, x_j)=S_{ij}$ называется **мерой сходства**, если:

1) $0 \le S(x_i, x_j) < 1$ для $x_i \neq x_j$

2) $S(x_i, x_i)=1$

3) $S(x_i, x_j)=S(x_j, x_i)$

Пары значений мер сходства можно объединить в **матрицу сходства**:

$$S=\begin{pmatrix} 1 & s_{12} & \dots & s_{1n} \\ s_{21} & 1 & \dots & s_{2n} \\ s_{n1} & s_{n2} & \dots & 1 \end{pmatrix}$$

Величину S_{ij} называют **коэффициентом сходства**.

Постройте матрицу смежности по датасету первой задачи, взяв в качестве меры расстояния евклидову метрику.

Постройте на основе неё какую-нибудь матрицу сходства (вам надо самим придумать функцию сходства - просто удовлетвроите трём свойствам).

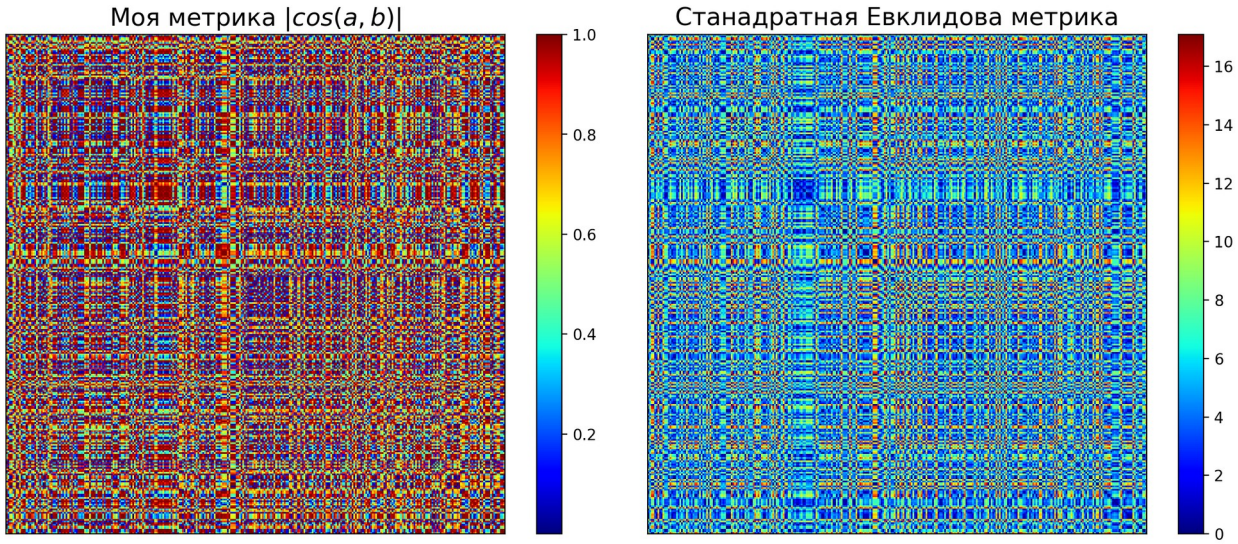
Визуализируйте обе матрицы в виде картинки. Используйте imshow() или pcolormesh() из Matplotlib.

Для получения матриц используйте функцию sklearn.metrics.pairwise_distances (тут можно в качестве метрики использовать свою собственную функцию - так можно сделать матрицу сходств) или какую-то другую отсюда https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.pairwise

Можно также посмотреть здесь https://scikit-learn.org/stable/modules/classes.html#module-sklearn.neighbors , например, функцию neighbors.kneighbors_graph

```
from sklearn.metrics import pairwise_distances

euclidian_matrix = pairwise_distances(X)
my_metric = lambda a,b: np.abs(np.dot(a, b) / (np.linalg.norm(a) *
np.linalg.norm(b)))
my_matrix = pairwise_distances(X, metric = my_metric)

fig, ax = plt.subplots(1, 2, figsize = (12, 5), dpi = 200)

im_my = ax[0].imshow(my_matrix, cmap = 'jet')
```

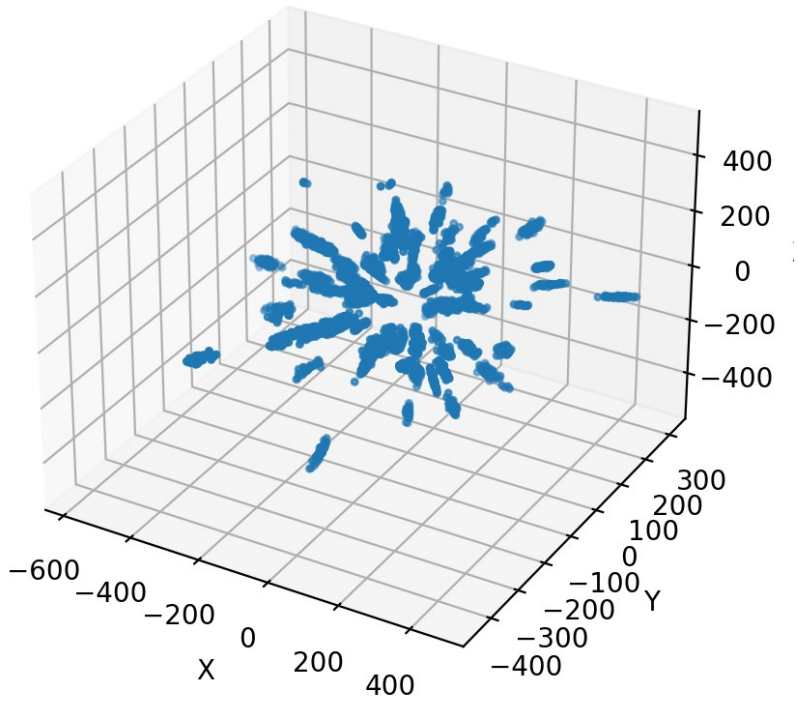
```
plt.colorbar(im_my)
im_euclidian = ax[1].imshow(euclidian_matrix, cmap = 'jet')
plt.colorbar(im_euclidian)

ax[0].set_title(r"Моя метрика $|cos(a, b)|$", fontsize = 16)
ax[1].set_title("Станадратная Евклидова метрика", fontsize = 16)
for a in ax:
    a.set_xticks([])
    a.set_yticks([])
fig.tight_layout()
```



Задача 4

```
data = pd.read_csv("toy_galaxies.csv") # Не забудьте скачать файл
display(data.head(3))

# Отобразим датасет
fig = plt.figure(figsize = (5, 5), dpi = 200)
ax = fig.add_subplot(111, projection='3d')
ax.scatter3D(data.x,data.y,data.z, marker='.')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

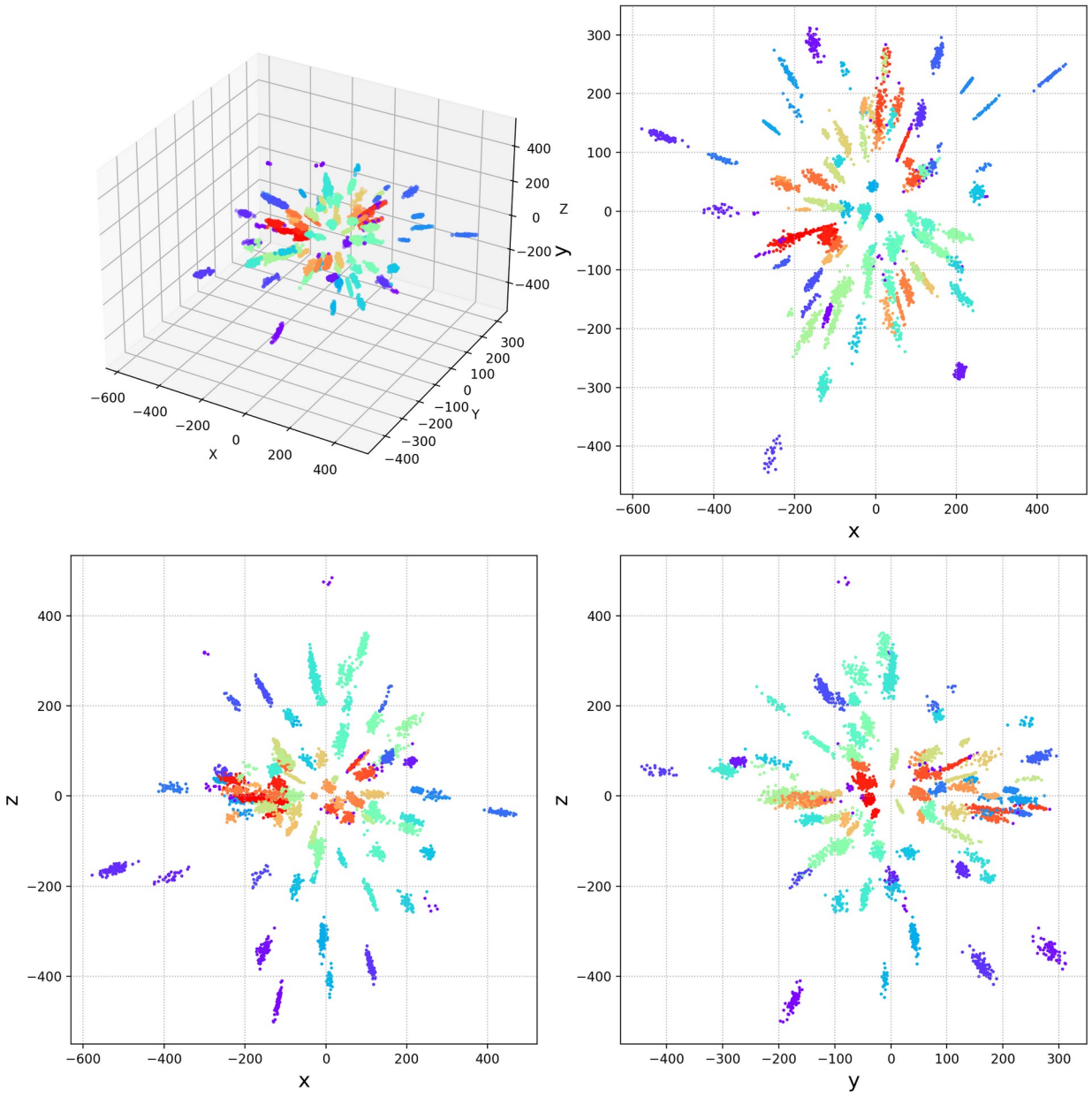
	x	y	z	label
0	-35.283431	-21.779673	-113.964124	1.0
1	-35.603636	-18.430420	-113.298009	1.0
2	-34.202934	-22.362906	-106.979709	1.0




```
def plot_galaxies(data,cluster):
    fig = plt.figure(figsize = (12 , 12), dpi = 200)

    ax = fig.add_subplot(221, projection='3d')
    ax_xy = fig.add_subplot(222)
    ax_xz = fig.add_subplot(223)
    ax_yz = fig.add_subplot(224)
    ax.scatter3D(data.x, data.y, data.z, marker = '.', c =
cluster.labels_, cmap = 'rainbow');
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    for plotable, keys in zip([ax_xy, ax_xz, ax_yz], [('x', 'y'),
('x', 'z'), ('y', 'z')]):
        plotable.scatter(data[keys[0]], data[keys[1]], c =
cluster.labels_, cmap = 'rainbow', s = 2)
        plotable.grid(ls = ':')
        plotable.set_xlabel(keys[0], fontsize = 16)
        plotable.set_ylabel(keys[1], fontsize = 16)
    ARI = round(ari(data.label,cluster.labels_),4) # считаем ARI -
оценка качества кластеризации
    fig.tight_layout()

    print('ARI = {}'.format(ARI))
```

```
# hdbscan работает следующим образом
cluster = hdbscan.HDBSCAN(metric="euclidean",
                          min_cluster_size=9,
                          algorithm="generic",
                          alpha=0.8,
                          cluster_selection_method='eom')

cluster.fit(data[['x','y','z']].to_numpy())

plot_galaxies(data,cluster)
```

ARI = 0.8682



```
min_cluster_size_values = np.arange(5, 50, 5, dtype = int)
min_samples_values = np.arange(1, 50, 5, dtype = int)
cluster_selection_epsilon_values = np.linspace(0.1, 3, 10)
alpha_values = np.linspace(0.001, 1, 10)
ari_values = []
A, B, C = np.meshgrid(min_cluster_size_values,
                      min_samples_values,
                      alpha_values,
                      indexing='ij')


param_combinations = np.column_stack((A.ravel(), B.ravel(),
C.ravel()))

for param in param_combinations:
    print(param)
    cluster = hdbscan.HDBSCAN(metric="euclidean",
                          min_cluster_size = int(param[0]),
                          algorithm="generic",
                          alpha = param[2],
                          cluster_selection_method='eom',
                          min_samples = int(param[1]))
    cluster.fit(data[['x','y','z']].to_numpy())
    ARI = ari(data.label,cluster.labels_) # считаем ARI - оценка
качества кластеризации
    ari_values.append(ARI)


[5.e+00 1.e+00 1.e-03]
[5.    1.    0.112]
[5.    1.    0.223]
[5.    1.    0.334]
[5.    1.    0.445]
[5.    1.    0.556]
[5.    1.    0.667]
[5.    1.    0.778]
[5.    1.    0.889]
[5. 1. 1.]
[5.e+00 6.e+00 1.e-03]
[5.    6.    0.112]
[5.    6.    0.223]
[5.    6.    0.334]
[5.    6.    0.445]
[5.    6.    0.556]
[5.    6.    0.667]
[5.    6.    0.778]
[5.    6.    0.889]
[5. 6. 1.]
[5.0e+00 1.1e+01 1.0e-03]
[ 5.    11.     0.112]
```

```
[ 5.    11.     0.223]
[ 5.    11.     0.334]
[ 5.    11.     0.445]
[ 5.    11.     0.556]
[ 5.    11.     0.667]
[ 5.    11.     0.778]
[ 5.    11.     0.889]
[ 5. 11.  1.]
[5.0e+00 1.6e+01 1.0e-03]
[ 5.    16.     0.112]
[ 5.    16.     0.223]
[ 5.    16.     0.334]
[ 5.    16.     0.445]
[ 5.    16.     0.556]
[ 5.    16.     0.667]
[ 5.    16.     0.778]
[ 5.    16.     0.889]
[ 5. 16.  1.]
[5.0e+00 2.1e+01 1.0e-03]
[ 5.    21.     0.112]
[ 5.    21.     0.223]
[ 5.    21.     0.334]
[ 5.    21.     0.445]
[ 5.    21.     0.556]
[ 5.    21.     0.667]
[ 5.    21.     0.778]
[ 5.    21.     0.889]
[ 5. 21.  1.]
[5.0e+00 2.6e+01 1.0e-03]
[ 5.    26.     0.112]
[ 5.    26.     0.223]
[ 5.    26.     0.334]
[ 5.    26.     0.445]
[ 5.    26.     0.556]
[ 5.    26.     0.667]
[ 5.    26.     0.778]
[ 5.    26.     0.889]
[ 5. 26.  1.]
[5.0e+00 3.1e+01 1.0e-03]
[ 5.    31.     0.112]
[ 5.    31.     0.223]
[ 5.    31.     0.334]
[ 5.    31.     0.445]
[ 5.    31.     0.556]
[ 5.    31.     0.667]
[ 5.    31.     0.778]
[ 5.    31.     0.889]
[ 5. 31.  1.]
[5.0e+00 3.6e+01 1.0e-03]
```

```
[ 5.    36.     0.112]
[ 5.    36.     0.223]
[ 5.    36.     0.334]
[ 5.    36.     0.445]
[ 5.    36.     0.556]
[ 5.    36.     0.667]
[ 5.    36.     0.778]
[ 5.    36.     0.889]
[ 5. 36.  1.]
[5.0e+00 4.1e+01 1.0e-03]
[ 5.    41.     0.112]
[ 5.    41.     0.223]
[ 5.    41.     0.334]
[ 5.    41.     0.445]
[ 5.    41.     0.556]
[ 5.    41.     0.667]
[ 5.    41.     0.778]
[ 5.    41.     0.889]
[ 5. 41.  1.]
[5.0e+00 4.6e+01 1.0e-03]
[ 5.    46.     0.112]
[ 5.    46.     0.223]
[ 5.    46.     0.334]
[ 5.    46.     0.445]
[ 5.    46.     0.556]
[ 5.    46.     0.667]
[ 5.    46.     0.778]
[ 5.    46.     0.889]
[ 5. 46.  1.]
[1.e+01 1.e+00 1.e-03]
[10.     1.     0.112]
[10.     1.     0.223]
[10.     1.     0.334]
[10.     1.     0.445]
[10.     1.     0.556]
[10.     1.     0.667]
[10.     1.     0.778]
[10.     1.     0.889]
[10.  1.  1.]
[1.e+01 6.e+00 1.e-03]
[10.     6.     0.112]
[10.     6.     0.223]
[10.     6.     0.334]
[10.     6.     0.445]
[10.     6.     0.556]
[10.     6.     0.667]
[10.     6.     0.778]
[10.     6.     0.889]
[10.  6.  1.]
```

```
[1.0e+01 1.1e+01 1.0e-03]
[10.    11.     0.112]
[10.    11.     0.223]
[10.    11.     0.334]
[10.    11.     0.445]
[10.    11.     0.556]
[10.    11.     0.667]
[10.    11.     0.778]
[10.    11.     0.889]
[10. 11.  1.]
[1.0e+01 1.6e+01 1.0e-03]
[10.    16.     0.112]
[10.    16.     0.223]
[10.    16.     0.334]
[10.    16.     0.445]
[10.    16.     0.556]
[10.    16.     0.667]
[10.    16.     0.778]
[10.    16.     0.889]
[10. 16.  1.]
[1.0e+01 2.1e+01 1.0e-03]
[10.    21.     0.112]
[10.    21.     0.223]
[10.    21.     0.334]
[10.    21.     0.445]
[10.    21.     0.556]
[10.    21.     0.667]
[10.    21.     0.778]
[10.    21.     0.889]
[10. 21.  1.]
[1.0e+01 2.6e+01 1.0e-03]
[10.    26.     0.112]
[10.    26.     0.223]
[10.    26.     0.334]
[10.    26.     0.445]
[10.    26.     0.556]
[10.    26.     0.667]
[10.    26.     0.778]
[10.    26.     0.889]
[10. 26.  1.]
[1.0e+01 3.1e+01 1.0e-03]
[10.    31.     0.112]
[10.    31.     0.223]
[10.    31.     0.334]
[10.    31.     0.445]
[10.    31.     0.556]
[10.    31.     0.667]
[10.    31.     0.778]
[10.    31.     0.889]
```

```
[10. 31.  1.]
[1.0e+01 3.6e+01 1.0e-03]
[10.    36.     0.112]
[10.    36.     0.223]
[10.    36.     0.334]
[10.    36.     0.445]
[10.    36.     0.556]
[10.    36.     0.667]
[10.    36.     0.778]
[10.    36.     0.889]
[10. 36.  1.]
[1.0e+01 4.1e+01 1.0e-03]
[10.    41.     0.112]
[10.    41.     0.223]
[10.    41.     0.334]
[10.    41.     0.445]
[10.    41.     0.556]
[10.    41.     0.667]
[10.    41.     0.778]
[10.    41.     0.889]
[10. 41.  1.]
[1.0e+01 4.6e+01 1.0e-03]
[10.    46.     0.112]
[10.    46.     0.223]
[10.    46.     0.334]
[10.    46.     0.445]
[10.    46.     0.556]
[10.    46.     0.667]
[10.    46.     0.778]
[10.    46.     0.889]
[10. 46.  1.]
[1.5e+01 1.0e+00 1.0e-03]
[15.     1.     0.112]
[15.     1.     0.223]
[15.     1.     0.334]
[15.     1.     0.445]
[15.     1.     0.556]
[15.     1.     0.667]
[15.     1.     0.778]
[15.     1.     0.889]
[15.  1.  1.]
[1.5e+01 6.0e+00 1.0e-03]
[15.     6.     0.112]
[15.     6.     0.223]
[15.     6.     0.334]
[15.     6.     0.445]
[15.     6.     0.556]
[15.     6.     0.667]
[15.     6.     0.778]
```

```
[15.     6.     0.889]
[15.  6.  1.]
[1.5e+01 1.1e+01 1.0e-03]
[15.    11.     0.112]
[15.    11.     0.223]
[15.    11.     0.334]
[15.    11.     0.445]
[15.    11.     0.556]
[15.    11.     0.667]
[15.    11.     0.778]
[15.    11.     0.889]
[15. 11.  1.]
[1.5e+01 1.6e+01 1.0e-03]
[15.    16.     0.112]
[15.    16.     0.223]
[15.    16.     0.334]
[15.    16.     0.445]
[15.    16.     0.556]
[15.    16.     0.667]
[15.    16.     0.778]
[15.    16.     0.889]
[15. 16.  1.]
[1.5e+01 2.1e+01 1.0e-03]
[15.    21.     0.112]
[15.    21.     0.223]
[15.    21.     0.334]
[15.    21.     0.445]
[15.    21.     0.556]
[15.    21.     0.667]
[15.    21.     0.778]
[15.    21.     0.889]
[15. 21.  1.]
[1.5e+01 2.6e+01 1.0e-03]
[15.    26.     0.112]
[15.    26.     0.223]
[15.    26.     0.334]
[15.    26.     0.445]
[15.    26.     0.556]
[15.    26.     0.667]
[15.    26.     0.778]
[15.    26.     0.889]
[15. 26.  1.]
[1.5e+01 3.1e+01 1.0e-03]
[15.    31.     0.112]
[15.    31.     0.223]
[15.    31.     0.334]
[15.    31.     0.445]
[15.    31.     0.556]
[15.    31.     0.667]
```

```
[15.    31.     0.778]
[15.    31.     0.889]
[15. 31.  1.]
[1.5e+01 3.6e+01 1.0e-03]
[15.    36.     0.112]
[15.    36.     0.223]
[15.    36.     0.334]
[15.    36.     0.445]
[15.    36.     0.556]
[15.    36.     0.667]
[15.    36.     0.778]
[15.    36.     0.889]
[15. 36.  1.]
[1.5e+01 4.1e+01 1.0e-03]
[15.    41.     0.112]
[15.    41.     0.223]
[15.    41.     0.334]
[15.    41.     0.445]
[15.    41.     0.556]
[15.    41.     0.667]
[15.    41.     0.778]
[15.    41.     0.889]
[15. 41.  1.]
[1.5e+01 4.6e+01 1.0e-03]
[15.    46.     0.112]
[15.    46.     0.223]
[15.    46.     0.334]
[15.    46.     0.445]
[15.    46.     0.556]
[15.    46.     0.667]
[15.    46.     0.778]
[15.    46.     0.889]
[15. 46.  1.]
[2.e+01 1.e+00 1.e-03]
[20.     1.     0.112]
[20.     1.     0.223]
[20.     1.     0.334]
[20.     1.     0.445]
[20.     1.     0.556]
[20.     1.     0.667]
[20.     1.     0.778]
[20.     1.     0.889]
[20.  1.  1.]
[2.e+01 6.e+00 1.e-03]
[20.     6.     0.112]
[20.     6.     0.223]
[20.     6.     0.334]
[20.     6.     0.445]
[20.     6.     0.556]
```

```
[20.     6.     0.667]
[20.     6.     0.778]
[20.     6.     0.889]
[20.  6.  1.]
[2.0e+01 1.1e+01 1.0e-03]
[20.    11.     0.112]
[20.    11.     0.223]
[20.    11.     0.334]
[20.    11.     0.445]
[20.    11.     0.556]
[20.    11.     0.667]
[20.    11.     0.778]
[20.    11.     0.889]
[20. 11.  1.]
[2.0e+01 1.6e+01 1.0e-03]
[20.    16.     0.112]
[20.    16.     0.223]
[20.    16.     0.334]
[20.    16.     0.445]
[20.    16.     0.556]
[20.    16.     0.667]
[20.    16.     0.778]
[20.    16.     0.889]
[20. 16.  1.]
[2.0e+01 2.1e+01 1.0e-03]
[20.    21.     0.112]
[20.    21.     0.223]
[20.    21.     0.334]
[20.    21.     0.445]
[20.    21.     0.556]
[20.    21.     0.667]
[20.    21.     0.778]
[20.    21.     0.889]
[20. 21.  1.]
[2.0e+01 2.6e+01 1.0e-03]
[20.    26.     0.112]
[20.    26.     0.223]
[20.    26.     0.334]
[20.    26.     0.445]
[20.    26.     0.556]
[20.    26.     0.667]
[20.    26.     0.778]
[20.    26.     0.889]
[20. 26.  1.]
[2.0e+01 3.1e+01 1.0e-03]
[20.    31.     0.112]
[20.    31.     0.223]
[20.    31.     0.334]
[20.    31.     0.445]
```

```
[20.    31.     0.556]
[20.    31.     0.667]
[20.    31.     0.778]
[20.    31.     0.889]
[20. 31.  1.]
[2.0e+01 3.6e+01 1.0e-03]
[20.    36.     0.112]
[20.    36.     0.223]
[20.    36.     0.334]
[20.    36.     0.445]
[20.    36.     0.556]
[20.    36.     0.667]
[20.    36.     0.778]
[20.    36.     0.889]
[20. 36.  1.]
[2.0e+01 4.1e+01 1.0e-03]
[20.    41.     0.112]
[20.    41.     0.223]
[20.    41.     0.334]
[20.    41.     0.445]
[20.    41.     0.556]
[20.    41.     0.667]
[20.    41.     0.778]
[20.    41.     0.889]
[20. 41.  1.]
[2.0e+01 4.6e+01 1.0e-03]
[20.    46.     0.112]
[20.    46.     0.223]
[20.    46.     0.334]
[20.    46.     0.445]
[20.    46.     0.556]
[20.    46.     0.667]
[20.    46.     0.778]
[20.    46.     0.889]
[20. 46.  1.]
[2.5e+01 1.0e+00 1.0e-03]
[25.     1.     0.112]
[25.     1.     0.223]
[25.     1.     0.334]
[25.     1.     0.445]
[25.     1.     0.556]
[25.     1.     0.667]
[25.     1.     0.778]
[25.     1.     0.889]
[25.  1.  1.]
[2.5e+01 6.0e+00 1.0e-03]
[25.     6.     0.112]
[25.     6.     0.223]
[25.     6.     0.334]
```

```
[25.     6.     0.445]
[25.     6.     0.556]
[25.     6.     0.667]
[25.     6.     0.778]
[25.     6.     0.889]
[25.  6.  1.]
[2.5e+01 1.1e+01 1.0e-03]
[25.    11.     0.112]
[25.    11.     0.223]
[25.    11.     0.334]
[25.    11.     0.445]
[25.    11.     0.556]
[25.    11.     0.667]
[25.    11.     0.778]
[25.    11.     0.889]
[25. 11.  1.]
[2.5e+01 1.6e+01 1.0e-03]
[25.    16.     0.112]
[25.    16.     0.223]
[25.    16.     0.334]
[25.    16.     0.445]
[25.    16.     0.556]
[25.    16.     0.667]
[25.    16.     0.778]
[25.    16.     0.889]
[25. 16.  1.]
[2.5e+01 2.1e+01 1.0e-03]
[25.    21.     0.112]
[25.    21.     0.223]
[25.    21.     0.334]
[25.    21.     0.445]
[25.    21.     0.556]
[25.    21.     0.667]
[25.    21.     0.778]
[25.    21.     0.889]
[25. 21.  1.]
[2.5e+01 2.6e+01 1.0e-03]
[25.    26.     0.112]
[25.    26.     0.223]
[25.    26.     0.334]
[25.    26.     0.445]
[25.    26.     0.556]
[25.    26.     0.667]
[25.    26.     0.778]
[25.    26.     0.889]
[25. 26.  1.]
[2.5e+01 3.1e+01 1.0e-03]
[25.    31.     0.112]
[25.    31.     0.223]
```

```
[25.    31.     0.334]
[25.    31.     0.445]
[25.    31.     0.556]
[25.    31.     0.667]
[25.    31.     0.778]
[25.    31.     0.889]
[25. 31.  1.]
[2.5e+01 3.6e+01 1.0e-03]
[25.    36.     0.112]
[25.    36.     0.223]
[25.    36.     0.334]
[25.    36.     0.445]
[25.    36.     0.556]
[25.    36.     0.667]
[25.    36.     0.778]
[25.    36.     0.889]
[25. 36.  1.]
[2.5e+01 4.1e+01 1.0e-03]
[25.    41.     0.112]
[25.    41.     0.223]
[25.    41.     0.334]
[25.    41.     0.445]
[25.    41.     0.556]
[25.    41.     0.667]
[25.    41.     0.778]
[25.    41.     0.889]
[25. 41.  1.]
[2.5e+01 4.6e+01 1.0e-03]
[25.    46.     0.112]
[25.    46.     0.223]
[25.    46.     0.334]
[25.    46.     0.445]
[25.    46.     0.556]
[25.    46.     0.667]
[25.    46.     0.778]
[25.    46.     0.889]
[25. 46.  1.]
[3.e+01 1.e+00 1.e-03]
[30.     1.     0.112]
[30.     1.     0.223]
[30.     1.     0.334]
[30.     1.     0.445]
[30.     1.     0.556]
[30.     1.     0.667]
[30.     1.     0.778]
[30.     1.     0.889]
[30.  1.  1.]
[3.e+01 6.e+00 1.e-03]
[30.     6.     0.112]
[30.     6.     0.223]
```

```
[30.     6.     0.334]
[30.     6.     0.445]
[30.     6.     0.556]
[30.     6.     0.667]
[30.     6.     0.778]
[30.     6.     0.889]
[30.  6.  1.]
[3.0e+01 1.1e+01 1.0e-03]
[30.    11.     0.112]
[30.    11.     0.223]
[30.    11.     0.334]
[30.    11.     0.445]
[30.    11.     0.556]
[30.    11.     0.667]
[30.    11.     0.778]
[30.    11.     0.889]
[30. 11.  1.]
[3.0e+01 1.6e+01 1.0e-03]
[30.    16.     0.112]
[30.    16.     0.223]
[30.    16.     0.334]
[30.    16.     0.445]
[30.    16.     0.556]
[30.    16.     0.667]
[30.    16.     0.778]
[30.    16.     0.889]
[30. 16.  1.]
[3.0e+01 2.1e+01 1.0e-03]
[30.    21.     0.112]
[30.    21.     0.223]
[30.    21.     0.334]
[30.    21.     0.445]
[30.    21.     0.556]
[30.    21.     0.667]
[30.    21.     0.778]
[30.    21.     0.889]
[30. 21.  1.]
[3.0e+01 2.6e+01 1.0e-03]
[30.    26.     0.112]
[30.    26.     0.223]
[30.    26.     0.334]
[30.    26.     0.445]
[30.    26.     0.556]
[30.    26.     0.667]
[30.    26.     0.778]
[30.    26.     0.889]
[30. 26.  1.]
[3.0e+01 3.1e+01 1.0e-03]
[30.    31.     0.112]
```

```
[30.    31.     0.223]
[30.    31.     0.334]
[30.    31.     0.445]
[30.    31.     0.556]
[30.    31.     0.667]
[30.    31.     0.778]
[30.    31.     0.889]
[30. 31.  1.]
[3.0e+01 3.6e+01 1.0e-03]
[30.    36.     0.112]
[30.    36.     0.223]
[30.    36.     0.334]
[30.    36.     0.445]
[30.    36.     0.556]
[30.    36.     0.667]
[30.    36.     0.778]
[30.    36.     0.889]
[30. 36.  1.]
[3.0e+01 4.1e+01 1.0e-03]
[30.    41.     0.112]
[30.    41.     0.223]
[30.    41.     0.334]
[30.    41.     0.445]
[30.    41.     0.556]
[30.    41.     0.667]
[30.    41.     0.778]
[30.    41.     0.889]
[30. 41.  1.]
[3.0e+01 4.6e+01 1.0e-03]
[30.    46.     0.112]
[30.    46.     0.223]
[30.    46.     0.334]
[30.    46.     0.445]
[30.    46.     0.556]
[30.    46.     0.667]
[30.    46.     0.778]
[30.    46.     0.889]
[30. 46.  1.]
[3.5e+01 1.0e+00 1.0e-03]
[35.     1.     0.112]
[35.     1.     0.223]
[35.     1.     0.334]
[35.     1.     0.445]
[35.     1.     0.556]
[35.     1.     0.667]
[35.     1.     0.778]
[35.     1.     0.889]
[35.  1.  1.]
[3.5e+01 6.0e+00 1.0e-03]
```

```
[35.     6.     0.112]
[35.     6.     0.223]
[35.     6.     0.334]
[35.     6.     0.445]
[35.     6.     0.556]
[35.     6.     0.667]
[35.     6.     0.778]
[35.     6.     0.889]
[35.  6.  1.]
[3.5e+01 1.1e+01 1.0e-03]
[35.    11.     0.112]
[35.    11.     0.223]
[35.    11.     0.334]
[35.    11.     0.445]
[35.    11.     0.556]
[35.    11.     0.667]
[35.    11.     0.778]
[35.    11.     0.889]
[35. 11.  1.]
[3.5e+01 1.6e+01 1.0e-03]
[35.    16.     0.112]
[35.    16.     0.223]
[35.    16.     0.334]
[35.    16.     0.445]
[35.    16.     0.556]
[35.    16.     0.667]
[35.    16.     0.778]
[35.    16.     0.889]
[35. 16.  1.]
[3.5e+01 2.1e+01 1.0e-03]
[35.    21.     0.112]
[35.    21.     0.223]
[35.    21.     0.334]
[35.    21.     0.445]
[35.    21.     0.556]
[35.    21.     0.667]
[35.    21.     0.778]
[35.    21.     0.889]
[35. 21.  1.]
[3.5e+01 2.6e+01 1.0e-03]
[35.    26.     0.112]
[35.    26.     0.223]
[35.    26.     0.334]
[35.    26.     0.445]
[35.    26.     0.556]
[35.    26.     0.667]
[35.    26.     0.778]
[35.    26.     0.889]
[35. 26.  1.]
```

```
[3.5e+01 3.1e+01 1.0e-03]
[35.    31.     0.112]
[35.    31.     0.223]
[35.    31.     0.334]
[35.    31.     0.445]
[35.    31.     0.556]
[35.    31.     0.667]
[35.    31.     0.778]
[35.    31.     0.889]
[35. 31.  1.]
[3.5e+01 3.6e+01 1.0e-03]
[35.    36.     0.112]
[35.    36.     0.223]
[35.    36.     0.334]
[35.    36.     0.445]
[35.    36.     0.556]
[35.    36.     0.667]
[35.    36.     0.778]
[35.    36.     0.889]
[35. 36.  1.]
[3.5e+01 4.1e+01 1.0e-03]
[35.    41.     0.112]
[35.    41.     0.223]
[35.    41.     0.334]
[35.    41.     0.445]
[35.    41.     0.556]
[35.    41.     0.667]
[35.    41.     0.778]
[35.    41.     0.889]
[35. 41.  1.]
[3.5e+01 4.6e+01 1.0e-03]
[35.    46.     0.112]
[35.    46.     0.223]
[35.    46.     0.334]
[35.    46.     0.445]
[35.    46.     0.556]
[35.    46.     0.667]
[35.    46.     0.778]
[35.    46.     0.889]
[35. 46.  1.]
[4.e+01 1.e+00 1.e-03]
[40.     1.     0.112]
[40.     1.     0.223]
[40.     1.     0.334]
[40.     1.     0.445]
[40.     1.     0.556]
[40.     1.     0.667]
[40.     1.     0.778]
[40.     1.     0.889]
```

```
[40.  1.  1.]
[4.e+01 6.e+00 1.e-03]
[40.     6.     0.112]
[40.     6.     0.223]
[40.     6.     0.334]
[40.     6.     0.445]
[40.     6.     0.556]
[40.     6.     0.667]
[40.     6.     0.778]
[40.     6.     0.889]
[40.  6.  1.]
[4.0e+01 1.1e+01 1.0e-03]
[40.    11.     0.112]
[40.    11.     0.223]
[40.    11.     0.334]
[40.    11.     0.445]
[40.    11.     0.556]
[40.    11.     0.667]
[40.    11.     0.778]
[40.    11.     0.889]
[40. 11.  1.]
[4.0e+01 1.6e+01 1.0e-03]
[40.    16.     0.112]
[40.    16.     0.223]
[40.    16.     0.334]
[40.    16.     0.445]
[40.    16.     0.556]
[40.    16.     0.667]
[40.    16.     0.778]
[40.    16.     0.889]
[40. 16.  1.]
[4.0e+01 2.1e+01 1.0e-03]
[40.    21.     0.112]
[40.    21.     0.223]
[40.    21.     0.334]
[40.    21.     0.445]
[40.    21.     0.556]
[40.    21.     0.667]
[40.    21.     0.778]
[40.    21.     0.889]
[40. 21.  1.]
[4.0e+01 2.6e+01 1.0e-03]
[40.    26.     0.112]
[40.    26.     0.223]
[40.    26.     0.334]
[40.    26.     0.445]
[40.    26.     0.556]
[40.    26.     0.667]
[40.    26.     0.778]
```

```
[40.    26.     0.889]
[40. 26.  1.]
[4.0e+01 3.1e+01 1.0e-03]
[40.    31.     0.112]
[40.    31.     0.223]
[40.    31.     0.334]
[40.    31.     0.445]
[40.    31.     0.556]
[40.    31.     0.667]
[40.    31.     0.778]
[40.    31.     0.889]
[40. 31.  1.]
[4.0e+01 3.6e+01 1.0e-03]
[40.    36.     0.112]
[40.    36.     0.223]
[40.    36.     0.334]
[40.    36.     0.445]
[40.    36.     0.556]
[40.    36.     0.667]
[40.    36.     0.778]
[40.    36.     0.889]
[40. 36.  1.]
[4.0e+01 4.1e+01 1.0e-03]
[40.    41.     0.112]
[40.    41.     0.223]
[40.    41.     0.334]
[40.    41.     0.445]
[40.    41.     0.556]
[40.    41.     0.667]
[40.    41.     0.778]
[40.    41.     0.889]
[40. 41.  1.]
[4.0e+01 4.6e+01 1.0e-03]
[40.    46.     0.112]
[40.    46.     0.223]
[40.    46.     0.334]
[40.    46.     0.445]
[40.    46.     0.556]
[40.    46.     0.667]
[40.    46.     0.778]
[40.    46.     0.889]
[40. 46.  1.]
[4.5e+01 1.0e+00 1.0e-03]
[45.     1.     0.112]
[45.     1.     0.223]
[45.     1.     0.334]
[45.     1.     0.445]
[45.     1.     0.556]
[45.     1.     0.667]
```

```
[45.     1.     0.778]
[45.     1.     0.889]
[45.  1.  1.]
[4.5e+01 6.0e+00 1.0e-03]
[45.     6.     0.112]
[45.     6.     0.223]
[45.     6.     0.334]
[45.     6.     0.445]
[45.     6.     0.556]
[45.     6.     0.667]
[45.     6.     0.778]
[45.     6.     0.889]
[45.  6.  1.]
[4.5e+01 1.1e+01 1.0e-03]
[45.    11.     0.112]
[45.    11.     0.223]
[45.    11.     0.334]
[45.    11.     0.445]
[45.    11.     0.556]
[45.    11.     0.667]
[45.    11.     0.778]
[45.    11.     0.889]
[45. 11.  1.]
[4.5e+01 1.6e+01 1.0e-03]
[45.    16.     0.112]
[45.    16.     0.223]
[45.    16.     0.334]
[45.    16.     0.445]
[45.    16.     0.556]
[45.    16.     0.667]
[45.    16.     0.778]
[45.    16.     0.889]
[45. 16.  1.]
[4.5e+01 2.1e+01 1.0e-03]
[45.    21.     0.112]
[45.    21.     0.223]
[45.    21.     0.334]
[45.    21.     0.445]
[45.    21.     0.556]
[45.    21.     0.667]
[45.    21.     0.778]
[45.    21.     0.889]
[45. 21.  1.]
[4.5e+01 2.6e+01 1.0e-03]
[45.    26.     0.112]
[45.    26.     0.223]
[45.    26.     0.334]
[45.    26.     0.445]
[45.    26.     0.556]
```

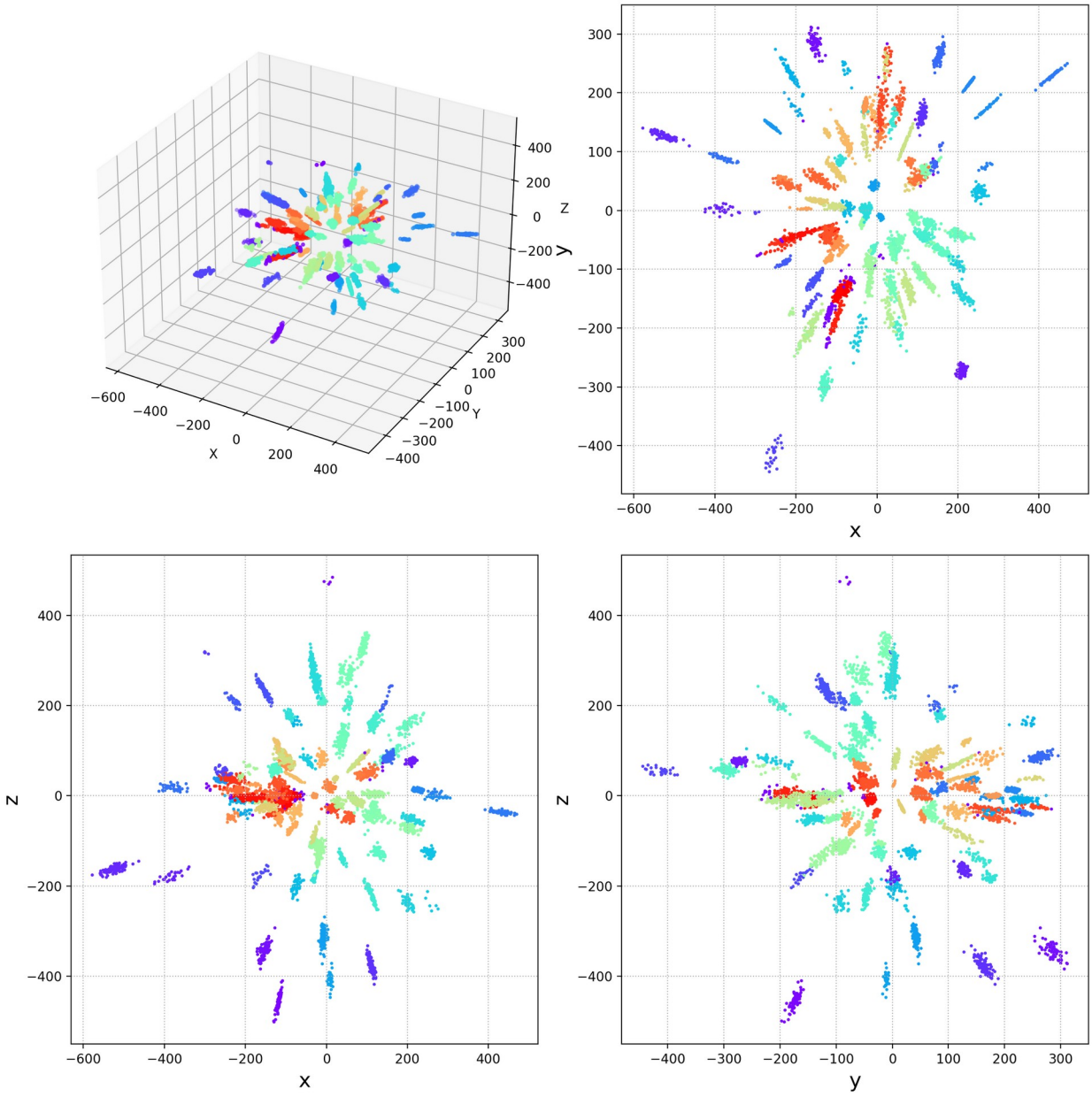
```
[45.    26.     0.667]
[45.    26.     0.778]
[45.    26.     0.889]
[45. 26.  1.]
[4.5e+01 3.1e+01 1.0e-03]
[45.    31.     0.112]
[45.    31.     0.223]
[45.    31.     0.334]
[45.    31.     0.445]
[45.    31.     0.556]
[45.    31.     0.667]
[45.    31.     0.778]
[45.    31.     0.889]
[45. 31.  1.]
[4.5e+01 3.6e+01 1.0e-03]
[45.    36.     0.112]
[45.    36.     0.223]
[45.    36.     0.334]
[45.    36.     0.445]
[45.    36.     0.556]
[45.    36.     0.667]
[45.    36.     0.778]
[45.    36.     0.889]
[45. 36.  1.]
[4.5e+01 4.1e+01 1.0e-03]
[45.    41.     0.112]
[45.    41.     0.223]
[45.    41.     0.334]
[45.    41.     0.445]
[45.    41.     0.556]
[45.    41.     0.667]
[45.    41.     0.778]
[45.    41.     0.889]
[45. 41.  1.]
[4.5e+01 4.6e+01 1.0e-03]
[45.    46.     0.112]
[45.    46.     0.223]
[45.    46.     0.334]
[45.    46.     0.445]
[45.    46.     0.556]
[45.    46.     0.667]
[45.    46.     0.778]
[45.    46.     0.889]
[45. 46.  1.]

param = param_combinations[np.argmax(ari_values)]
cluster = hdbscan.HDBSCAN(metric="euclidean",
                          min_cluster_size = int(param[0]),
                          algorithm="generic",
                          alpha = param[2],
```

```
                            cluster_selection_method='eom',
                            min_samples = int(param[1]))
cluster.fit(data[['x','y','z']].to_numpy())
```

```
HDBSCAN(algorithm='generic', alpha=0.445, min_samples=21)
```

```
plot_galaxies(data,cluster)
```

```
ARI = 0.9154
```



Задача 5

Пусть дана выборка точек x_i, взятая из смеси гауссовых распределений:

$$p(x) = \alpha \cdot N_{\mu_1,\sigma_1}(x) + (1-\alpha) \cdot N_{\mu_2,\sigma_2}(x).$$

Тогда можно поставить задачу оценки параметров $\alpha, \mu_1, \mu_2, \sigma_1, \sigma_2$ по выборке $\{x_i\}$.

- Покажите, что задача максимизации обычного правдоподобия $\prod_i p(x_i) \to max_{\alpha,\mu_1,\mu_2}$ плохо определена. Какие значения параметров максимизируют такое правдоподобие?
- Сгенерируйте данные (просто два сгустка точек, хорошо видных при реализации) и найдите параметры $\alpha, \mu_1, \mu_2, \sigma_1, \sigma_2$ с помощью EM-алгоритма. Инициализировать параметры можно какими-то случайными значениями. EM-алгоритм состоит из двух чередующихся шагов:
    a. M(Maximization)-шаг. Относим каждую точку x_i к первой или второй гауссиане, сравнивая значения правдоподобия для каждой компоненты смеси:

$$a(x_i) = \begin{bmatrix} 1, & p_1(x_i) > p_2(x_i), \\ 2, & p_2(x_i) > p_1(x_i), \end{bmatrix}$$

где $p_1(x) = \alpha N_{\mu_1,\sigma_1}(x), p_2(x) = (1-\alpha) N_{\mu_2,\sigma_2}(x)$.

    a. E(Expectation)-шаг. Находим параметры μ_1, σ_1 и μ_2, σ_2, максимизируя правдоподобие (или его логарифм) отдельно по точкам, отнесенным к каждой гауссиане:

$$\prod_{x_i : a(x_i)=1} p_1(x_i) \to \max_{\mu_1,\sigma_1}$$
$$\prod_{x_i : a(x_i)=2} p_2(x_i) \to \max_{\mu_2,\sigma_2}$$

Примечание. При нахождении параметра α можно оптимизировать обычное правдоподобие $\prod_i p(x_i)$. Все такие максимизации правдоподобия осуществляются аналитически в общем виде для гауссовых распределений.

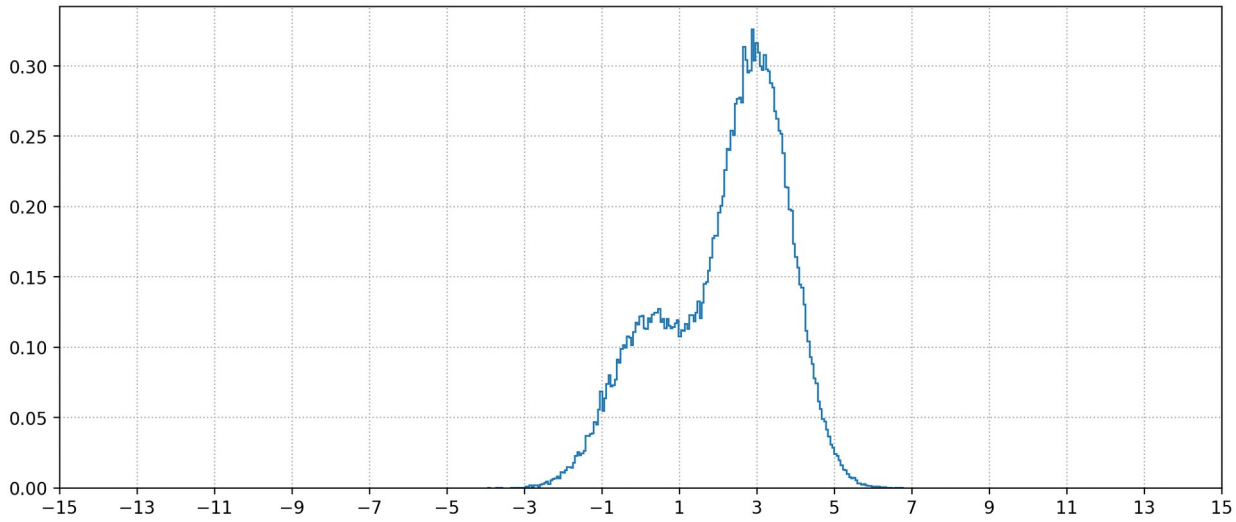
Реализуйте EM-алгоритм. Так как метод является итерационным, необходимо выбрать какой-либо критерий остановки, например, прекращать процес, если относительное изменение каждого параметра при очередном шаге меньше некоторого порога. С какой точностью удалось восстановить $\alpha, \mu_1, \mu_2, \sigma_1, \sigma_2$?

```
class sum_of_two_gaussians:
    def __init__(self, alpha, sigma1, sigma2, mu1, mu2):
        self.alpha, self.sigma1, self.sigma2, self.mu1, self.mu2 =
alpha, sigma1, sigma2, mu1, mu2
    def pdf(self,x):
        return self.alpha * sts.norm(self.mu1, self.sigma1).pdf(x) +
(1 - self.alpha) * sts.norm(self.mu2, self.sigma2).pdf(x)
```

```
    def rvs(self, size):
        elements = np.linspace(-100, 100, int(1e6))
        probabilities = self.pdf(elements) * np.abs(elements[0] -
elements[1])
        return np.random.choice(elements, p = probabilities, size =
size) + sts.uniform(0, np.abs(elements[0] - elements[1])).rvs(size)

distribution = sum_of_two_gaussians(0.3, 1, 0.9, 0.2, 3)
sample = distribution.rvs(100000)
print(sample.shape)
```

```
(100000,)
```

```
plt.figure(figsize= (12, 5), dpi = 200)
plt.grid(ls = ':')
plt.hist(sample, bins = 200, histtype = 'step', density = True)
plt.xlim(-15, 15)
plt.xticks(np.arange(-15, 17, 2));
```

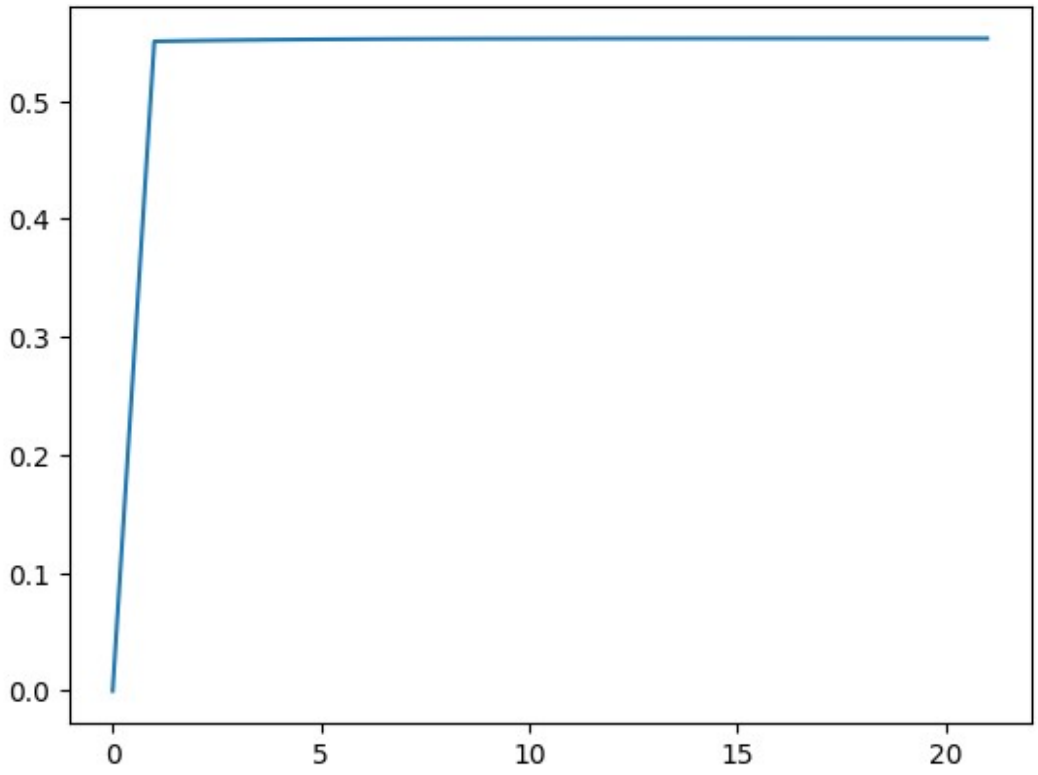



```
alpha, param_1, param_2 = np.random.rand(), np.random.rand(2) * 5,
np.random.rand(2) * 5
distribution_classes = []
iteration = 0
diff = 100
now_l = 10
history = {'classes': [], 'param_1': [], 'param_2': [], 'alpha': [],
'L': []}
history['L'].append(0)
while iteration < int(1e4) and (diff > 1e-3 or np.isnan(diff)):
    distribution_classes = alpha * sts.norm(*param_1).pdf(sample) >=
(1 - alpha) * sts.norm(*param_2).pdf(sample)
    sample1, sample2 = [sample[i] for i in range(len(sample)) if
distribution_classes[i]], [sample[i] for i in range(len(sample)) if
```

```
not distribution_classes[i]]
    param_1_prev, param_2_prev, alpha_prev = np.array(param_1),
np.array(param_2), alpha
    try:
        param_1 = sts.norm.fit(sample1)
        param_2 = sts.norm.fit(sample2)
    except:
        param_1 = np.random.random(2)
        alpha = np.random.random()
        param_2 = np.random.random(2)
    log_L = lambda a: 1e5 /
np.sum(np.log(np.abs(sum_of_two_gaussians(a, param_1[1], param_2[1],
param_1[0], param_2[0]).pdf(sample))))
    alpha = optimize.minimize(log_L, np.random.rand(), method =
'CG').x[0]
    while np.isnan(alpha):
        alpha = optimize.minimize(log_L, -np.random.rand() * alpha / 2
+ alpha , method = 'BFGS', bounds=(0.05, 0.94)).x[0]
    diff = np.abs(np.linalg.norm(np.hstack([param_1_prev -
np.array(param_1), param_2_prev - np.array(param_2), [alpha -
alpha_prev]])))
    now_l = np.abs(log_L(alpha))
    if not np.isnan(diff): print(diff, now_l)
    history['param_1'].append(param_1)
    history['param_2'].append(param_2)
    history['alpha'].append(alpha)
    history['classes'].append(distribution_classes)
    iteration += 1
    history['L'].append(now_l)
```

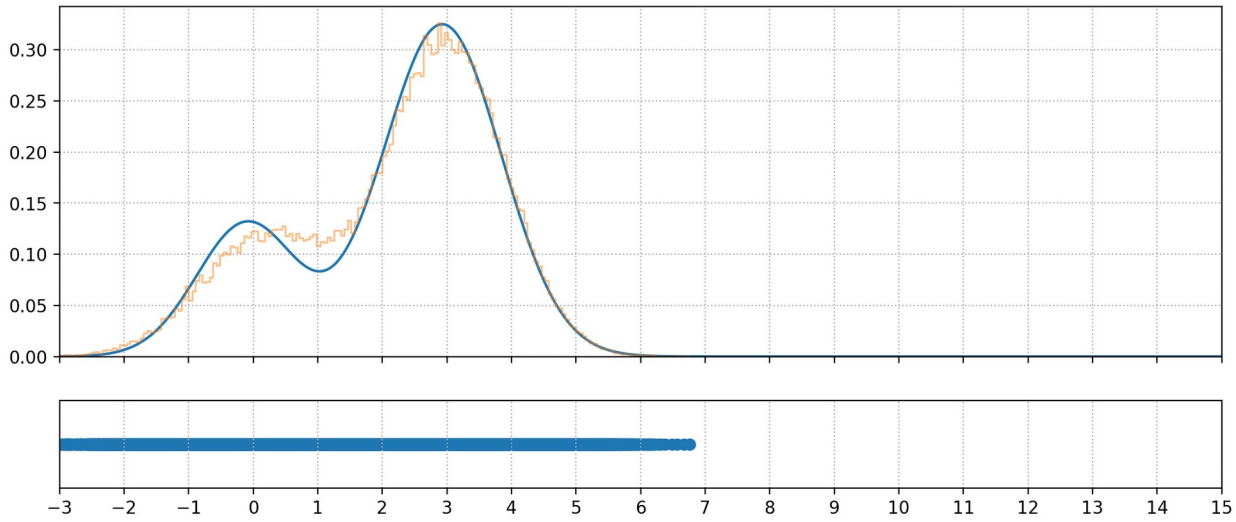
```
3.08528634464863 0.550649204248393
0.05369964571665602 0.5512368992458483
0.04740818181835482 0.5517022402744737
0.04014981167185894 0.5520563161861413
0.03484943274738697 0.5523330798866283
0.029265859710677238 0.5525408031943924
0.024795411007281266 0.5526992122652301
0.02012299848378661 0.5528152826736394
0.017072242643161758 0.552905667466895
0.012837780943155026 0.5529685669423531
0.008638692456803085 0.5530090906918005
0.006956739921813323 0.5530405634750326
0.006409437065791113 0.5530683618567527
0.004905724024265368 0.5530887959029446
0.00341059592533309 0.5531027586165006
0.002632151536890606 0.5531133479594575
0.0021526173968713412 0.553121867368311
0.0010166947022847948 0.5531258682373487
0.0010765808419598156 0.5531300841216783
```

```
0.0010761700761342096 0.5531342589693428
0.0006574638539351835 0.553136791316577

print(history['L'])

[0, 0.550649204248393, 0.5512368992458483, 0.5517022402744737,
0.5520563161861413, 0.5523330798866283, 0.5525408031943924,
0.5526992122652301, 0.5528152826736394, 0.552905667466895,
0.5529685669423531, 0.5530090906918005, 0.5530405634750326,
0.5530683618567527, 0.5530887959029446, 0.5531027586165006,
0.5531133479594575, 0.553121867368311, 0.5531258682373487,
0.5531300841216783, 0.5531342589693428, 0.553136791316577]

plt.plot(history['L'])

[<matplotlib.lines.Line2D at 0x7f2c9fe6c8e0>]
```




```
print(iteration)

21

print(param_1, param_2, alpha)

(-0.0981548230266496, 0.7750907562392725) (2.92877788104251,
0.9160219515700528) 0.2540451348754866
```

```
x_line = np.linspace(-5, 15, 1000)

fig, ax = plt.subplots(2, 1, figsize = (12, 5), dpi = 200,
gridspec_kw={'height_ratios': [4, 1]}, sharex = True)

ax[0].plot(x_line, sum_of_two_gaussians(alpha, param_1[1], param_2[1],
param_1[0], param_2[0]).pdf(x_line))
ax[0].hist(sample, bins = 200, histtype = 'step', alpha = 0.5, density
= True)
ax[1].scatter(sample, np.zeros_like(sample))
ax[1].set_yticks([])
ax[1].set_xticks(np.arange(-5, 17, 1))
ax[1].set_xlim(-3, 15)
ax[0].grid(ls = ':')
ax[1].grid(ls = ':')
```




```
x_line = np.linspace(-5, 15, 1000)

fig, ax = plt.subplots(2, 1, figsize = (12, 5), dpi = 200,
gridspec_kw={'height_ratios': [4, 1]}, sharex = True)
ax[0].hist(sample, bins = 200, histtype = 'step', alpha = 0.5, density
= True)
distribution_line, = ax[0].plot([], [])

def animate(i):
    distribution_line.set_data(x_line,
sum_of_two_gaussians(history['alpha'][i], history['param_1'][i][1],
history['param_2'][i][1], history['param_1'][i][0], history['param_2']
[i][0]).pdf(x_line))
    ax[1].clear()
    ax[1].scatter(sample, np.zeros_like(sample), c =
history['classes'][i])
    ax[1].set_yticks([])
```

```
    ax[1].set_xticks(np.arange(-5, 17, 1))
    ax[1].set_xlim(-3, 15)
    ax[0].grid(ls = ':')
    ax[1].grid(ls = ':')

    return [distribution_line]

def init():
    return animate(0)

anim = FuncAnimation(fig, animate, frames = len(history['alpha']),
interval = 20, blit = True, init_func = init)
anim.save('animation_9.mp4', fps = iteration // 5, extra_args=['-
vcodec', 'libx264'])
```