

ЗАДАЧА КЛАСТЕРИЗАЦИИ НА НАБОРЕ ДАННЫХ МУЗЫКАЛЬНОГО СЕРВИСА DEEZER

Работу подготовили студенты группы
0392
Иванов С. Сидорина Д. Частухин Д.



Содержание:

Сколько слайдов
нужно в презентации?



1. Что такое Deezer?.....3
2. Что такое кластеризация?.....4
3. Отличия кластеризации от классификации.....5-6
4. Наша задача.....7
5. План действий.....8-9
6. Модели кластеризации.....10-13
7. Изучение датасета Deezer'a.....14-18
8. Разработка способов кластеризации.....19-36
9. Написание кода.....37-38
10. Обучение модели и предсказание сообществ.....39-42
11. Определение параметров кластеризации.....43-45
12. Оценка точности модели.....46-50
13. Результат работы.....51-56
14. Неудачные дубли.....57-62

01

ЧТО ТАКОЕ DEEZER

Это музыкальный сервис. Он позволяет пользователям слушать музыку и подкасты. В этом сервисе можно "выделять" понравившиеся жанры, группы и композиции. На основании предпочтений создаются плэйлисты музыкальных рекомендаций, которые радуют пользователей.

02

Кластеризация

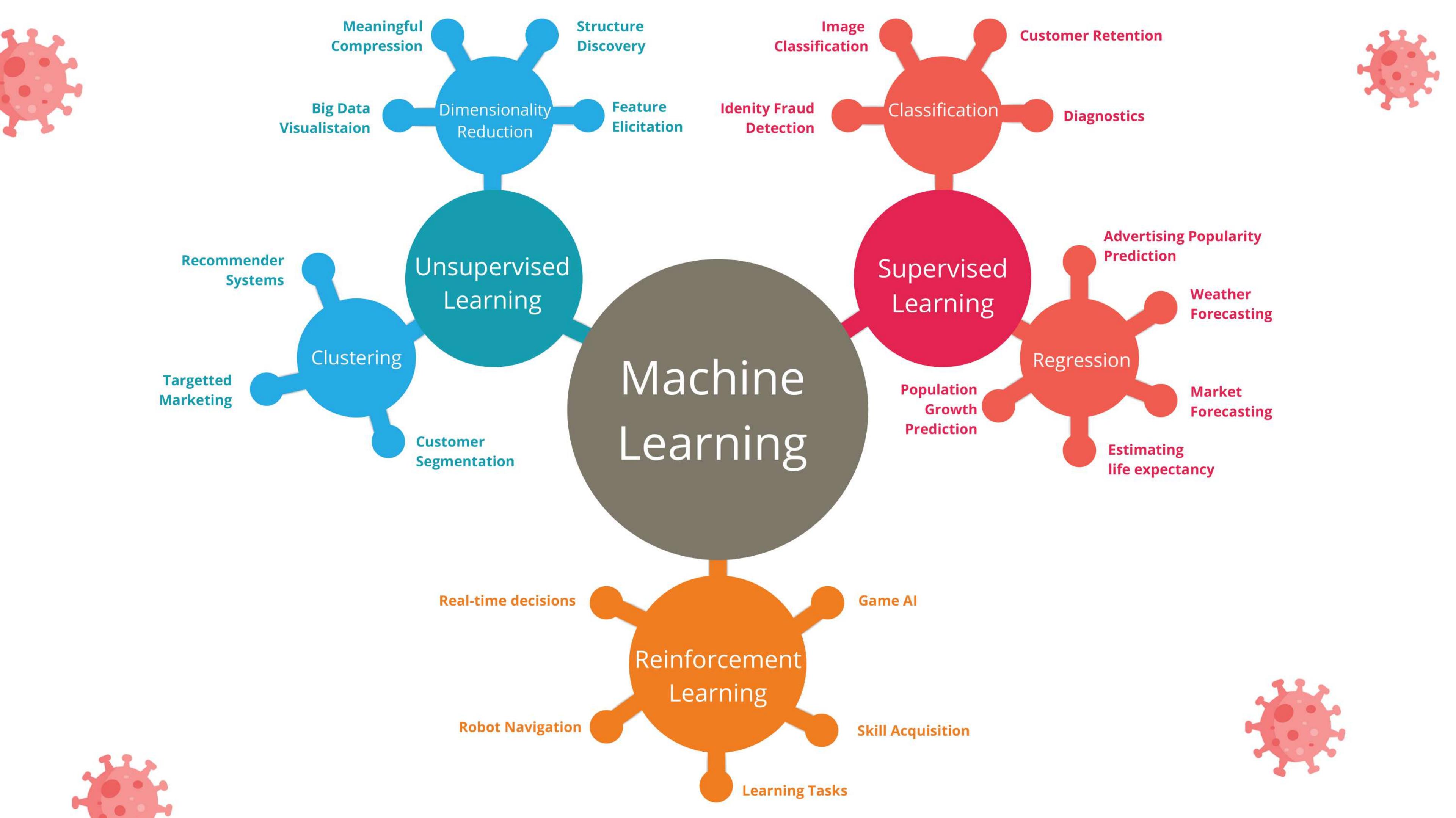
Это задача разбиения множества объектов на группы, называемые кластерами.

Внутри каждой группы должны оказаться «похожие» объекты, а объекты разных группы должны быть как можно более отличны.



Кластеризация/ Классификация

Кластеризация может показаться очень похожей на классификацию. В чем же отличие? Некоторые авторы утверждают, что принципиальное отличие в количестве кластеров, которое нельзя задать заранее. Другие – разница в методах обучения. Кластеризация использует методы обучения без учителя.



Наша задача

Нам нужно было разбить на кластеры группу пользователей, учитывая их музыкальные предпочтения и дружественные связи.

Модель, построенная для этой задачи, даёт возможность объединять пользователей по дружественным связям или по музыкальным предпочтениям. Такую модель можно использовать для рекомендации общей музыки пользователям, находящимся в одних сообществах.



05

План действий:

- Изучить модели кластеризации, работающие на графах (обучение без учителя)
- Изучить датасет музыкального сервиса Deezer
- Разработать способы кластеризации для конкретной задачи, используя набор инструментов для работы с графиками



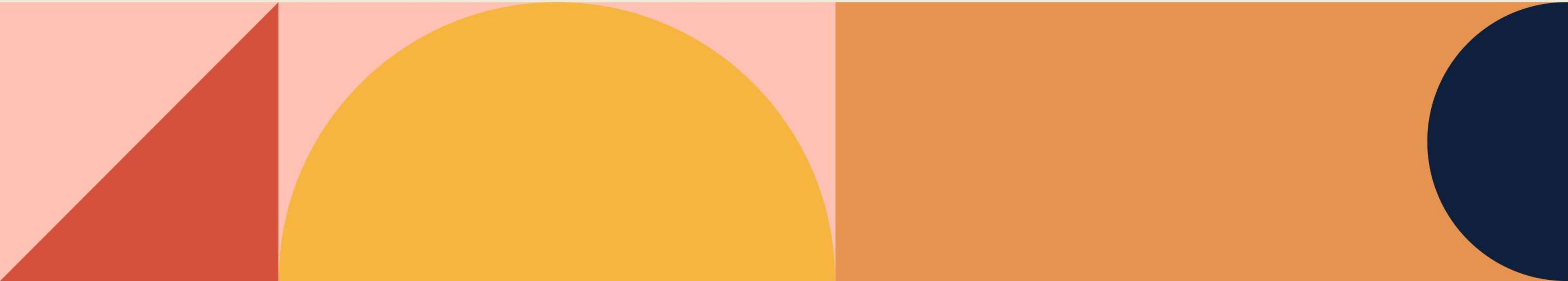
План действий:

- Написать код
- Обучить модель данными и предсказать сообщества
- Найти способы оценить точность модели
- Обучить другие модели этими данными и сравнить точности разных моделей



06

Модели кластеризации



Классификация кластерных методов



Методы по способу обработки данных

Иерархические

Строят не одно разбиение на кластеры, а целую систему вложенных разбиений

Неиерархические

Строят одно разбиение объектов на кластеры

Агломеративные

Последовательное объединение исходных элементов с соответствующим уменьшением числа кластеров

Дивизимные

Последовательное "дробление" исходных элементов с соответствующим увеличением числа кластеров

Итеративные

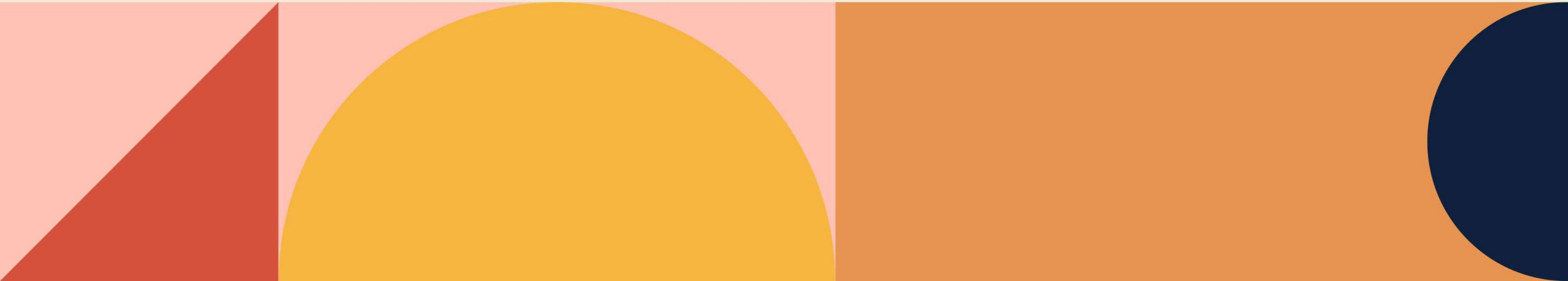
В процессе деления новые кластеры создаются до тех пор, пока не будет выполнено условие прекращения

Алгоритмы кластеризации



07

Изучение датасета Deezer'a



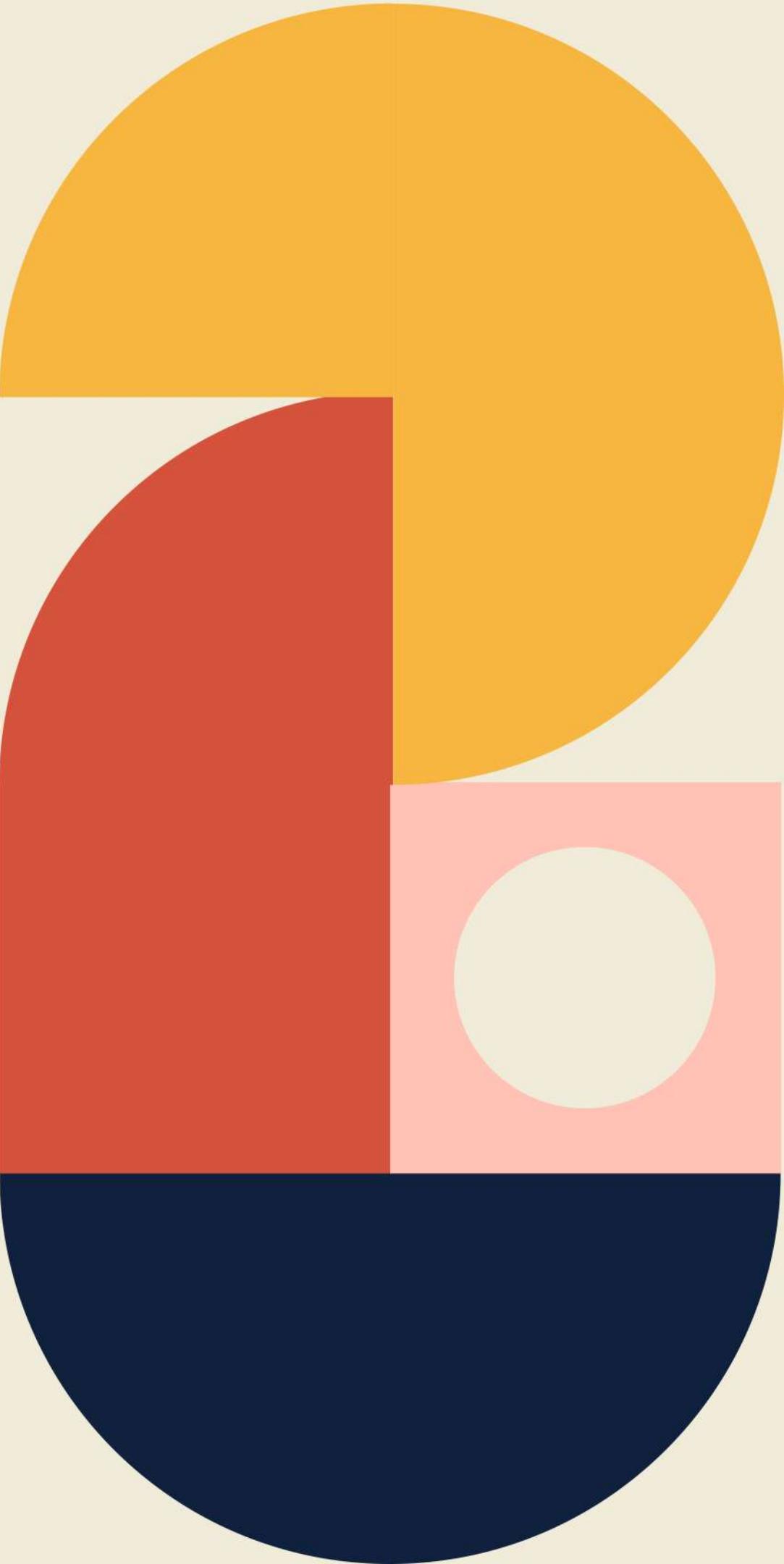
Обработка данных

У нас было 2 способа обработки данных.

В обоих случаях использовали библиотеку *pandas*.

Первый способ заключался в парсинге данных целиком. Второй - в понижении размерности путем объединения схожих жанров.

// Pandas — библиотека, помогающая оптимизировать код и упрощать работу с данными. Также Pandas добавляет новые структуры данных — серии и датафреймы. //



Реализация первого способа:

```
[ ] # Список жанров для каждой вершины
with open('Data/HU_genres.json') as json_data:
    data = json.load(json_data)

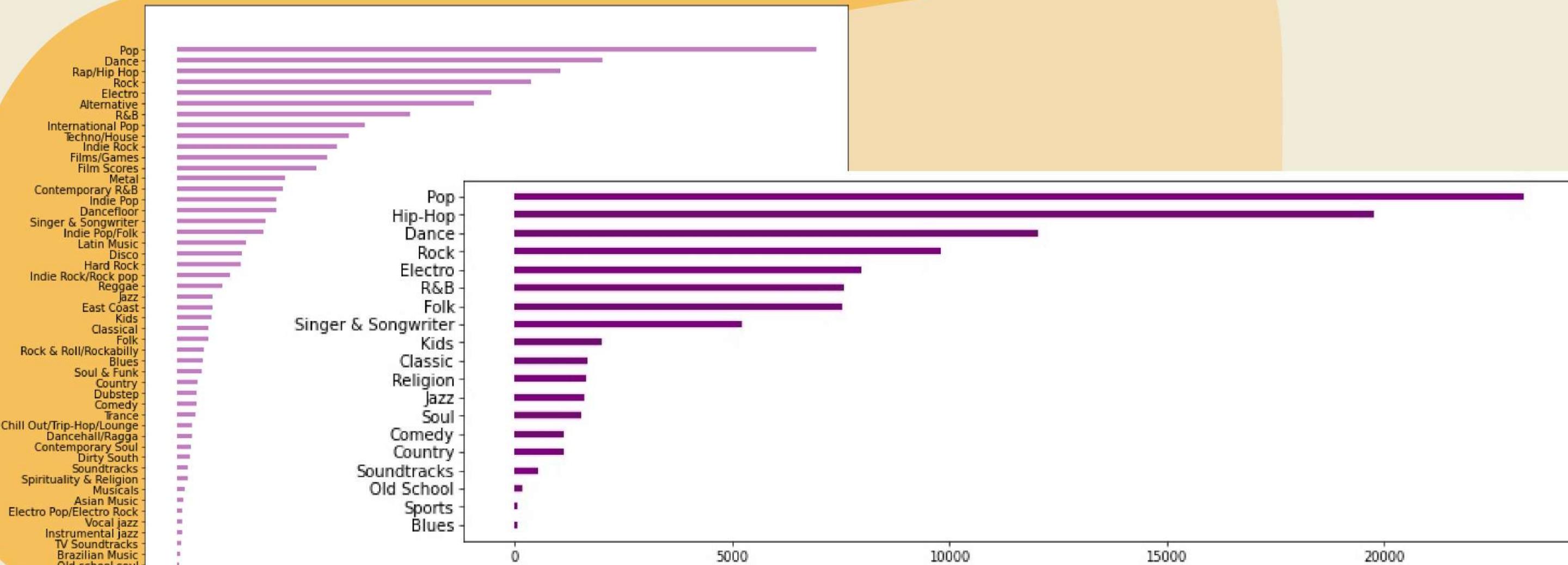
[ ] # Из словаря получаем все 84 жанра и заносим их в список
genres_list = []

for k in data.keys():
    for g in data[k]:
        if not(g in genres_list):
            genres_list.append(g)
        if len(genres_list) == 84:
            break

[ ] # Сортируем список с жанрами
genres_list.sort()
# Создаём пустой DataFrame с жанрами в виде столбцов
genres = pd.DataFrame(columns=genres_list)

[▶] # Для каждой строки создаём словарь с жанрами, которые содержит эта вершина, и добавляем их в DataFrame
for i in range(len(data)):
    genres_dict = dict.fromkeys(data[str(i)], 1)
    genres = genres.append(genres_dict, ignore_index=True)
# Остальные жанры зануляем
genres = genres.fillna(0)
```

Объединение жанров для второго способа:



```

new_genres['Blues'] = genres[['Acoustic Blues', 'Chicago Blues', 'Country Blues', 'Electric Blues']].sum(axis=1)
new_genres['Country'] = genres[['Alternative Country', 'Country', 'Country Blues', 'Bluegrass', 'Urban Cowboy']].sum(axis=1)
new_genres['R&B'] = genres[['Contemporary R&B', 'R&B', 'Oldschool R&B']].sum(axis=1)
new_genres['Old School'] = genres[['Old School', 'Old school soul', 'Oldschool R&B']].sum(axis=1)
new_genres['Classic'] = genres[['Classical', 'Classical Period', 'Opera', 'Romantic', 'Musicals', 'Baroque', 'Early Music', 'Renaissance', 'Modern']].sum(axis=1)
new_genres['Soundtracks'] = genres[['TV Soundtracks', 'Soundtracks', 'TV shows & movies', 'Film Scores', 'Films/Games', 'Bollywood', 'Game Scores']].sum(axis=1)
new_genres['Rock'] = genres[['Electro Pop/Electro Rock', 'Hard Rock', 'Rock', 'Rock & Roll/Rockabilly', 'Indie Rock', 'Alternative', 'Metal']].sum(axis=1)
new_genres['Electro'] = genres[['Electro Pop/Electro Rock', 'Electro', 'Electro Hip Hop', 'Grime', 'Techno/House', 'Dub', 'Dubstep']].sum(axis=1)
new_genres['Jazz'] = genres[['Instrumental jazz', 'Jazz', 'Jazz Hip Hop', 'Vocal jazz']].sum(axis=1)
new_genres['Hip-Hop'] = genres[['Electro Hip Hop', 'Chill Out/Trip-Hop/Lounge', 'Electro Hip Hop', 'Rap/Hip Hop', 'West Coast', 'East Coast', 'Dirty South']].sum(axis=1)
new_genres['Dance'] = genres[['Dance', 'Dancefloor', 'Dancehall/Ragga', 'Reggae', 'Techno/House', 'Disco']].sum(axis=1)
new_genres['Folk'] = genres[['Folk', 'African Music', 'Asian Music', 'Brazilian Music', 'Indian Music', 'Indie Pop', 'Indie Rock/Rock pop', 'Latin Music', 'Bolero', 'Corridos', 'Ranchera']].sum(axis=1)
new_genres['Pop'] = genres[['Indie Pop', 'Electro Pop/Electro Rock', 'International Pop', 'Indie Pop/Folk', 'Indie Rock/Rock pop', 'Pop', 'Tropical', 'Norteño', 'Ska', 'Nursery Rhymes']].sum(axis=1)
new_genres['Kids'] = genres[['Kids', 'Kids & Family', 'Nursery Rhymes']].sum(axis=1)
new_genres['Soul'] = genres[['Contemporary Soul', 'Soul & Funk', 'Old school soul']].sum(axis=1)
new_genres['Religion'] = genres[['Spirituality & Religion', 'Trance']].sum(axis=1)
new_genres['Sports'] = genres[['Sports']].sum(axis=1)
new_genres['Singer & Songwriter'] = genres[['Singer & Songwriter']].sum(axis=1)
new_genres['Comedy'] = genres[['Comedy']].sum(axis=1)
new_genres = new_genres.sort_index(axis=1)
new_genres['friend_cluster'] = friend_coord
new_genres.to_pickle('new_genres.pickle')

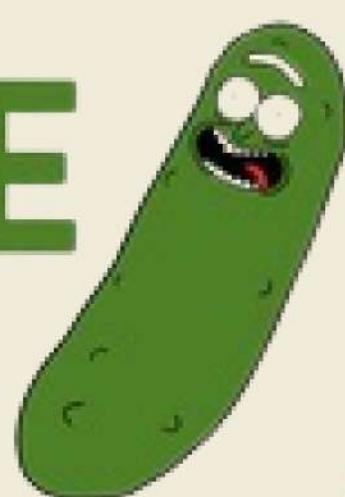
```

"Маринуем огурчики"

Чтобы сэкономить время и каждый раз не считывать данные, было решено использовать формат ***pickle*** (англ. соленый огурец).

Код:

```
[ ] # Сохраняем фрейм в файл формата pickle для дальнейшего использования  
genres.to_pickle('genres.pickle')  
  
[ ] # Загружаем фрейм  
genres = pd.read_pickle('Data/genres.pickle')
```

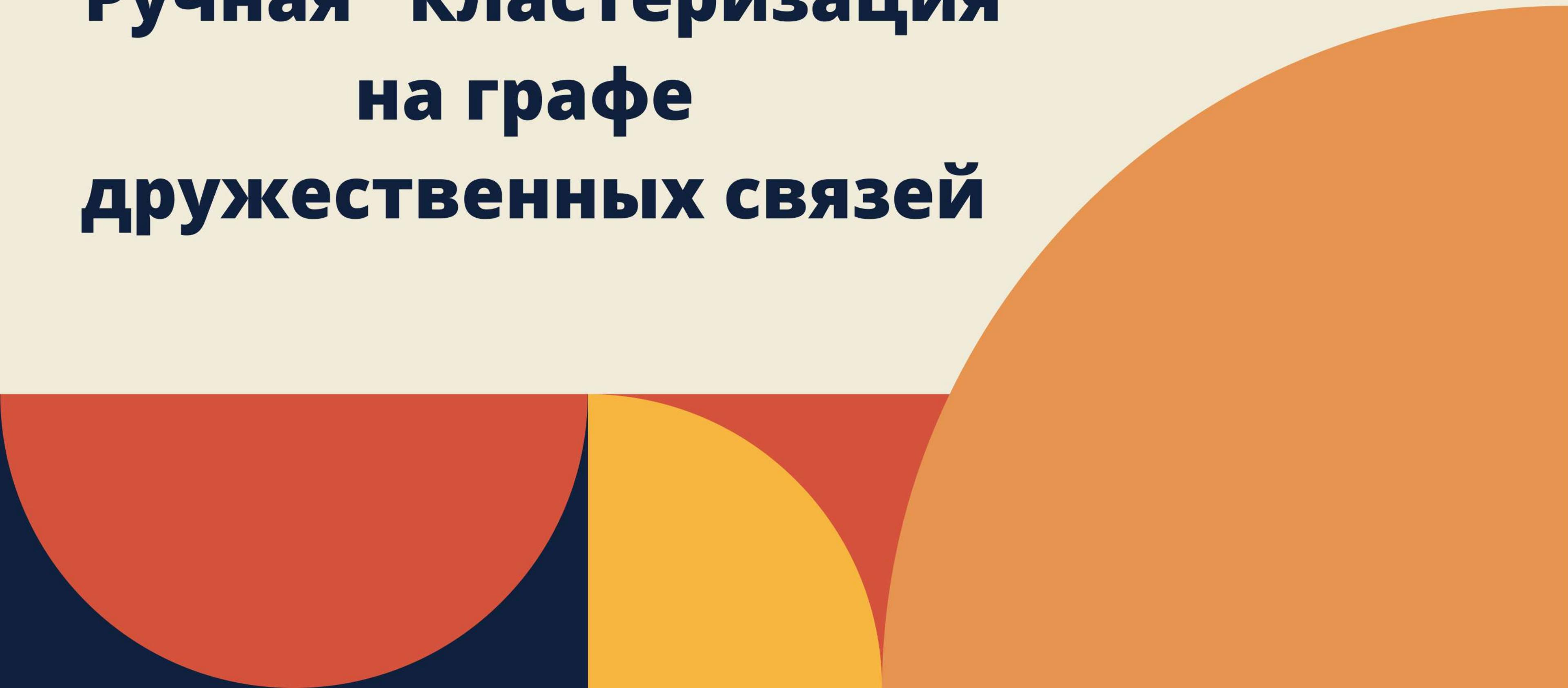


I'M
PICKLE
RICK!

Разработка способов кластеризации



"Ручная" кластеризация на графе дружественных связей



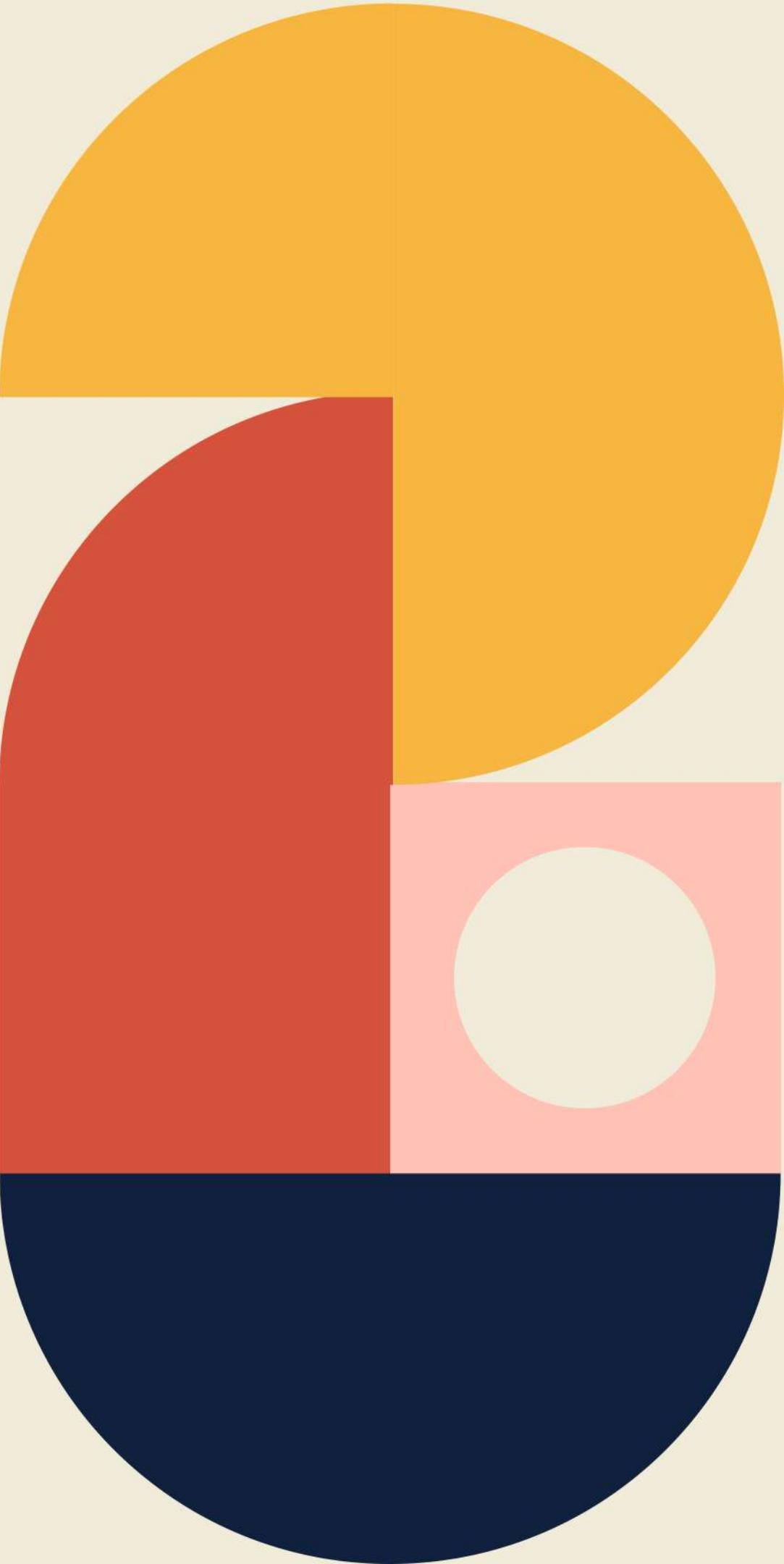
Принцип алгоритма:

Ручная иерархическая кластеризация делается для того, чтобы сократить время выполнения алгоритма: для алгоритмов **NetworkX** требуется матрица расстояний, которую в силу большого объема данных посчитать не удалось.



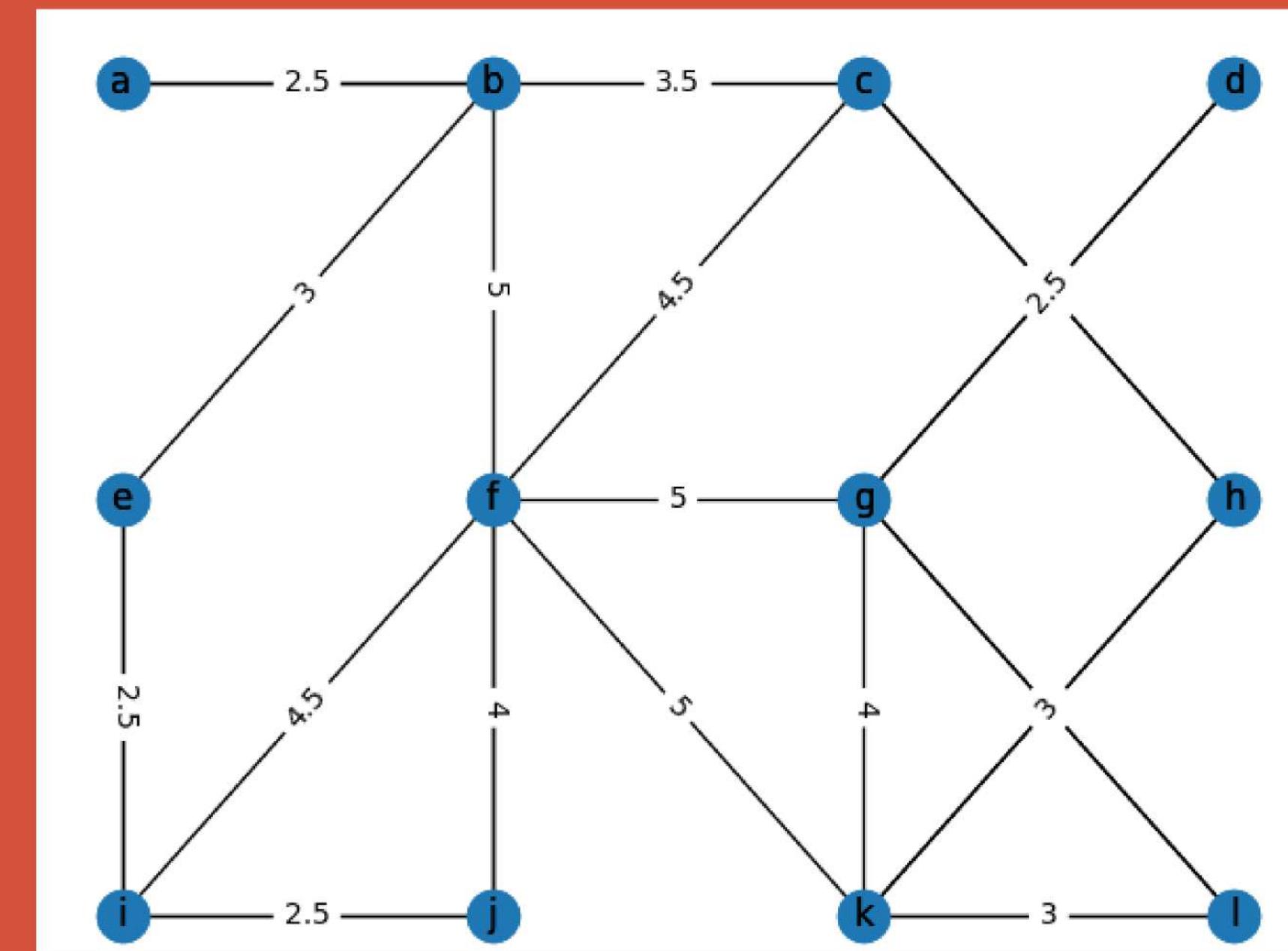
Шаги алгоритма:

1. Считаем вес рёбер, исходя из степеней вершин.
2. Строим минимальное остовное дерево (алгоритм Краскала).
3. Удаляем ребра так, чтобы получились компоненты связности.

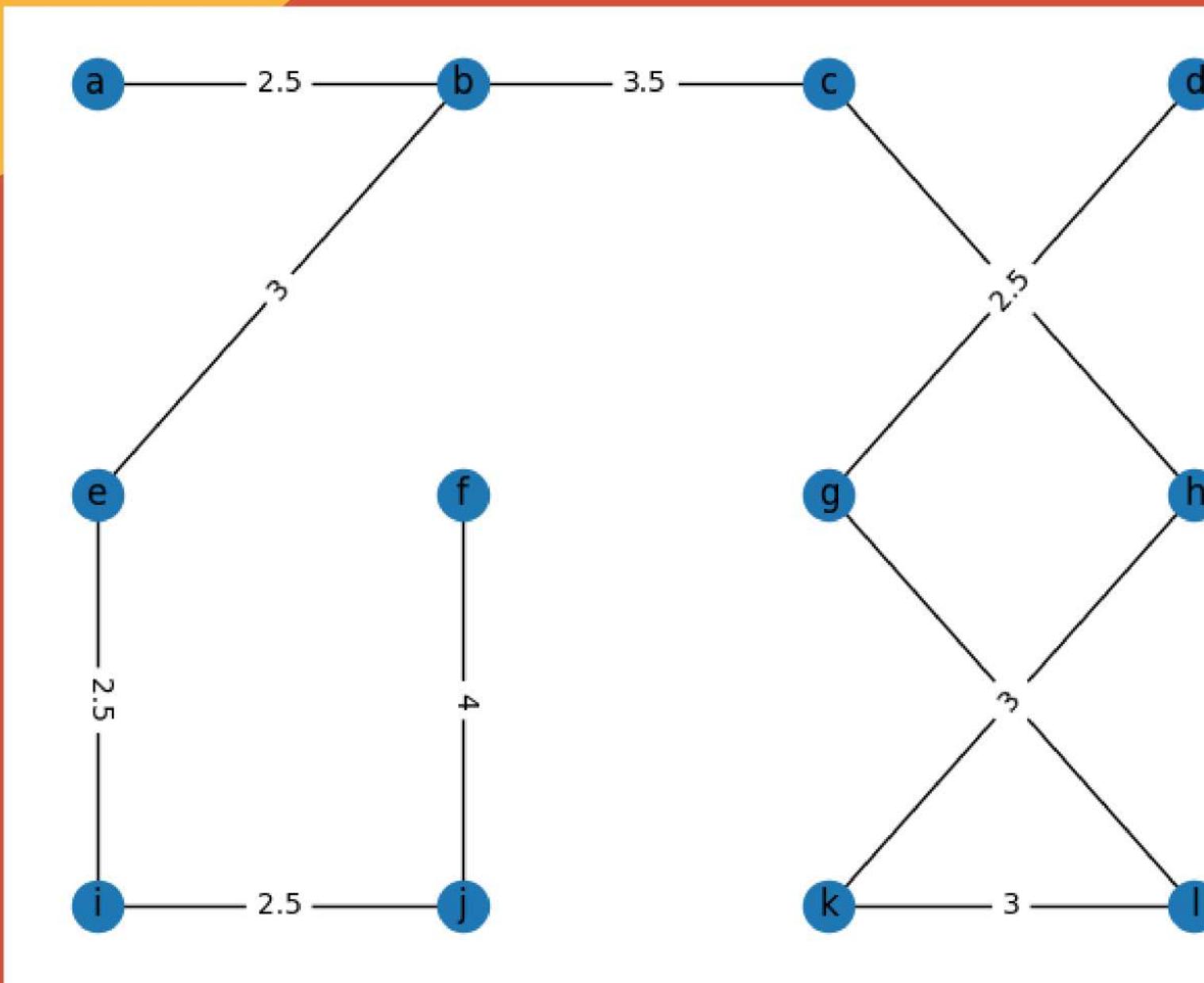


Пример работы:

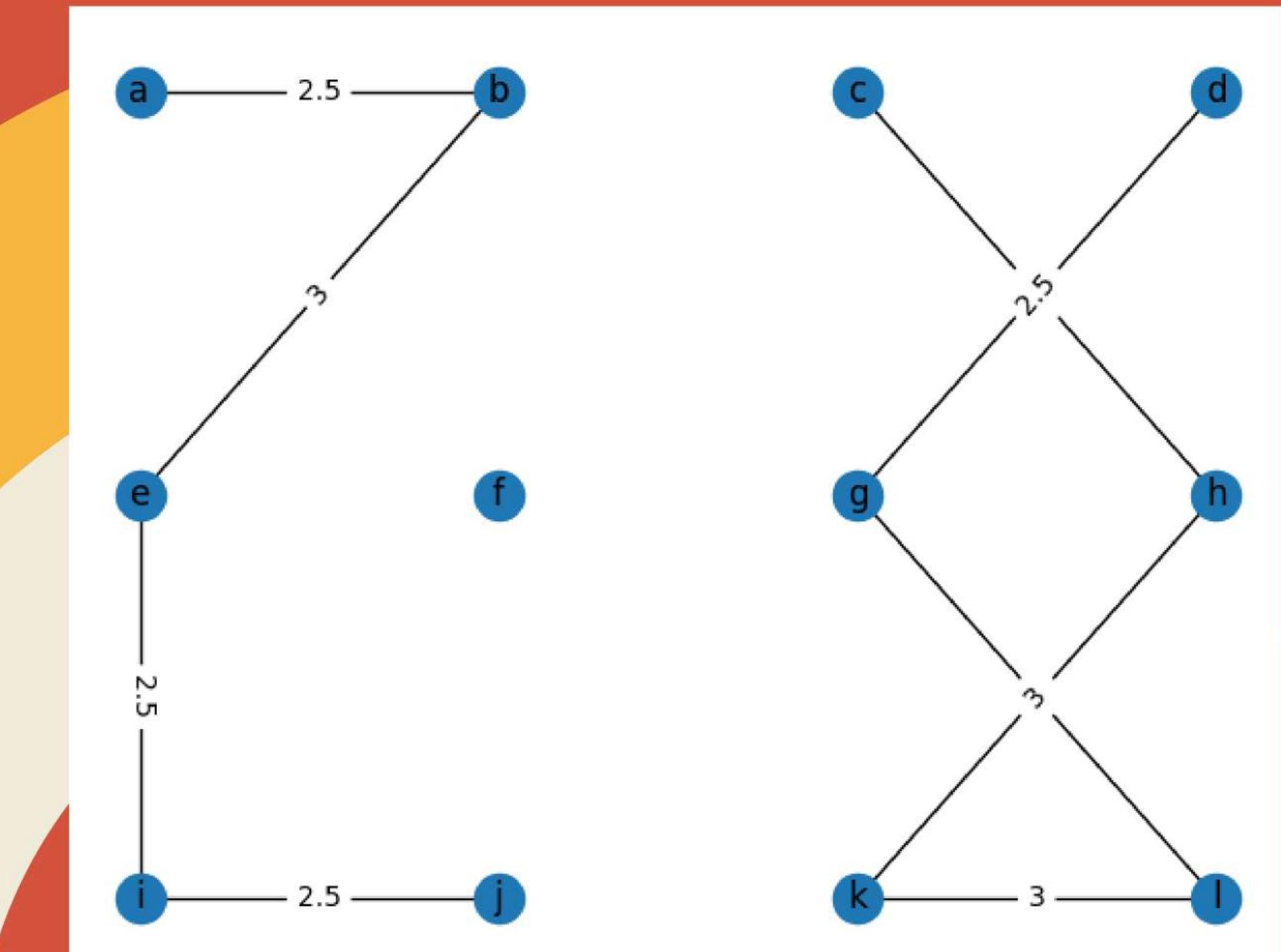
1. Считаем вес рёбер, исходя из степеней вершин.



2. Строим минимальное остовное дерево (алгоритм Краскала)

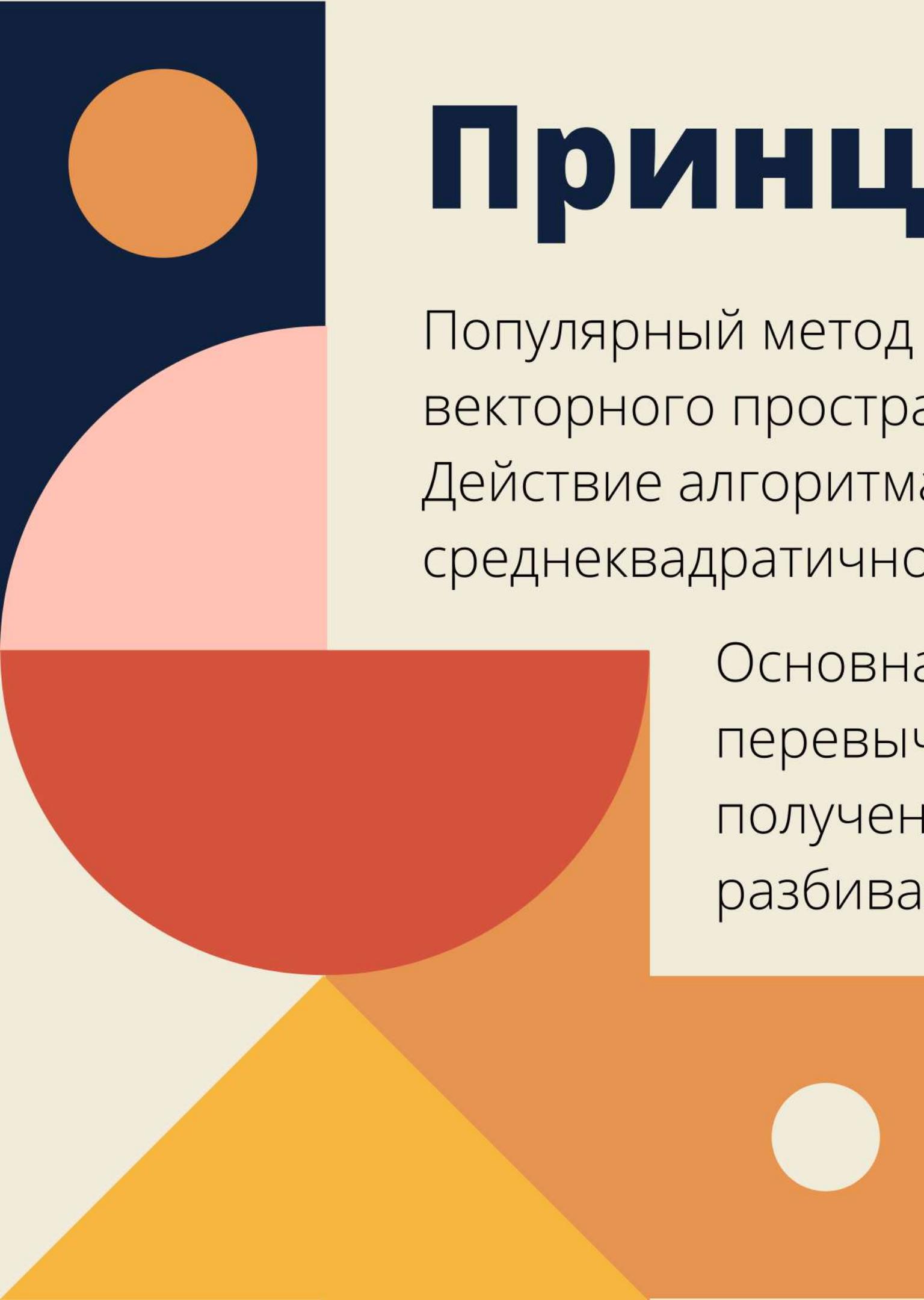


3. Удаляем ребра с наибольшим весом.



Алгоритм K-means





Принцип алгоритма:

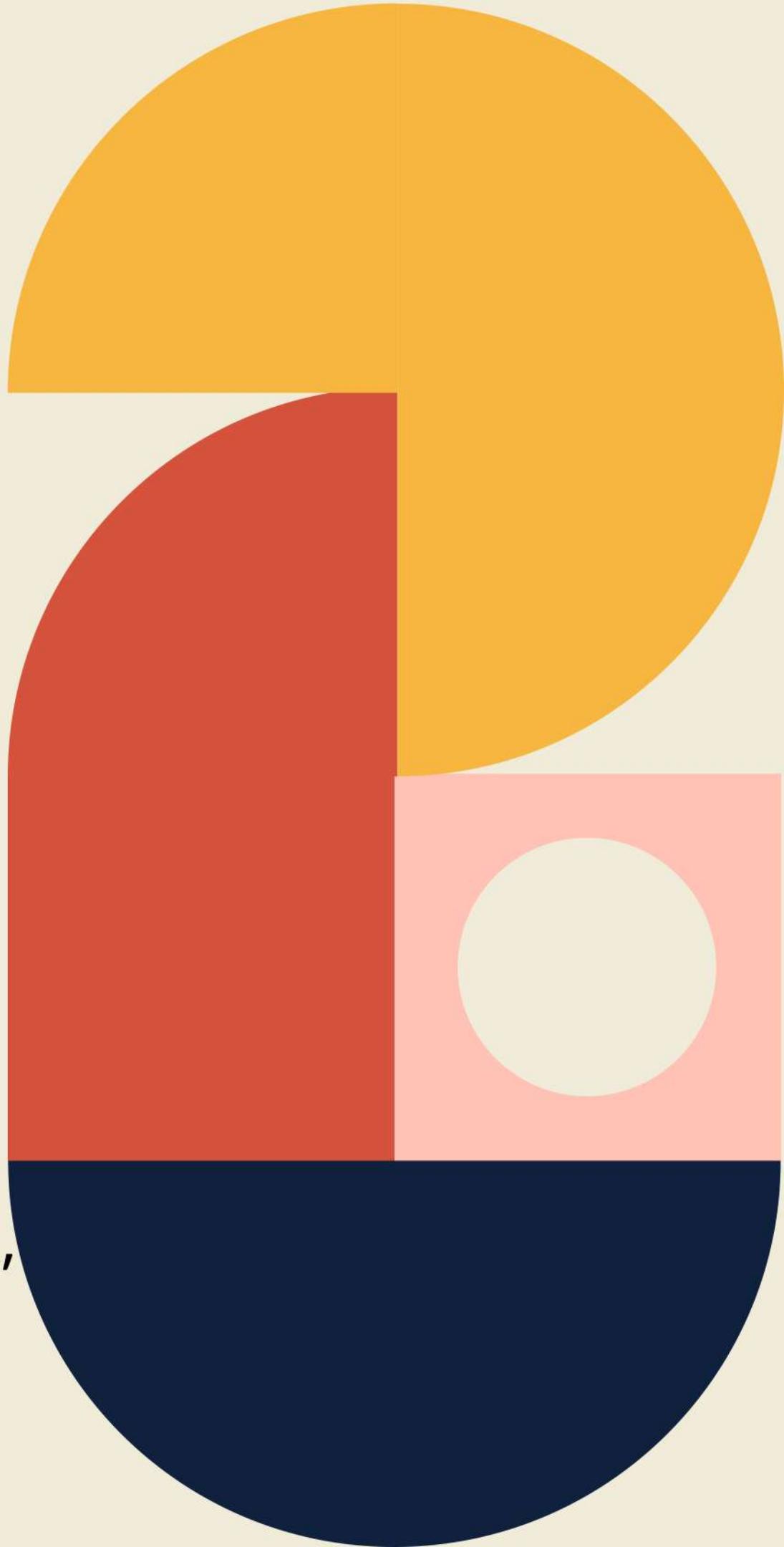
Популярный метод кластеризации. Он разбивает множество элементов векторного пространства на заранее известное число кластеров k . Действие алгоритма таково, что он стремится минимизировать среднеквадратичное отклонение на точках каждого кластера.

Основная идея заключается в том, что на каждой итерации пересчитываются центр масс для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь в соответствии с тем, какой

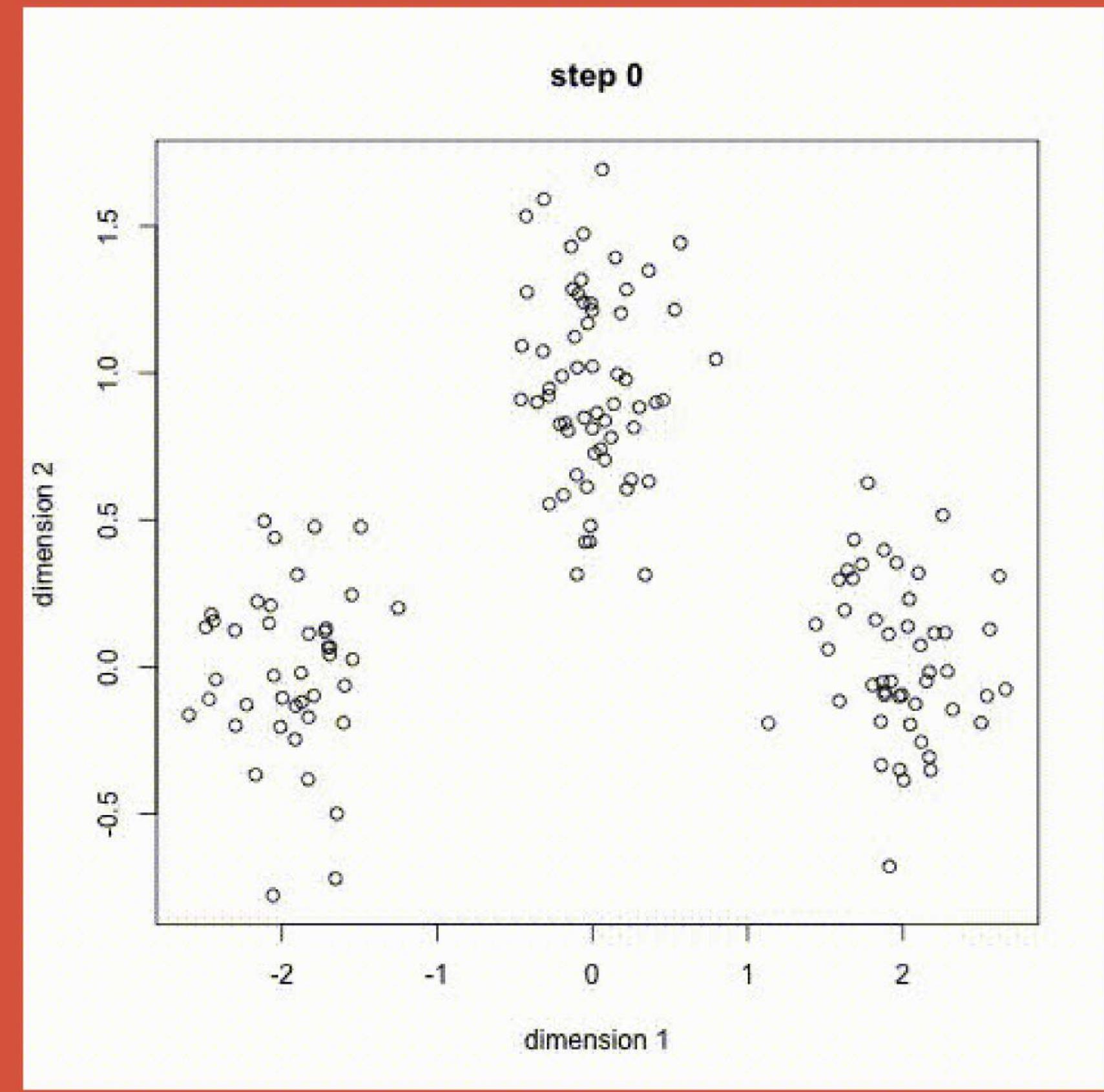
из новых центров оказался ближе по выбранной метрике. Алгоритм завершается, когда на какой-то итерации не происходит изменения кластеров.

Шаги алгоритма:

- 1) Выбрать количество кластеров (k) и получить данные $data_points \{x\}$.
- 2) Поместить центроиды $1, 2 \dots k$ случайным образом.
- 3) Повторять шаги 4 и 5 до сходимости или до конца фиксированного количества итераций:
- 4) **for each** $data_points \{x\}$:
 - a. Найти ближайший центроид
 - b. Пометить точку, как принадлежащую кластеру, соответствующему центроиду
- 5) **for each** $cluster j = 1 \dots k$:
 - a. Новый центроид = среднее значение всех точек, присвоенных этому кластеру
- 6) **end.**

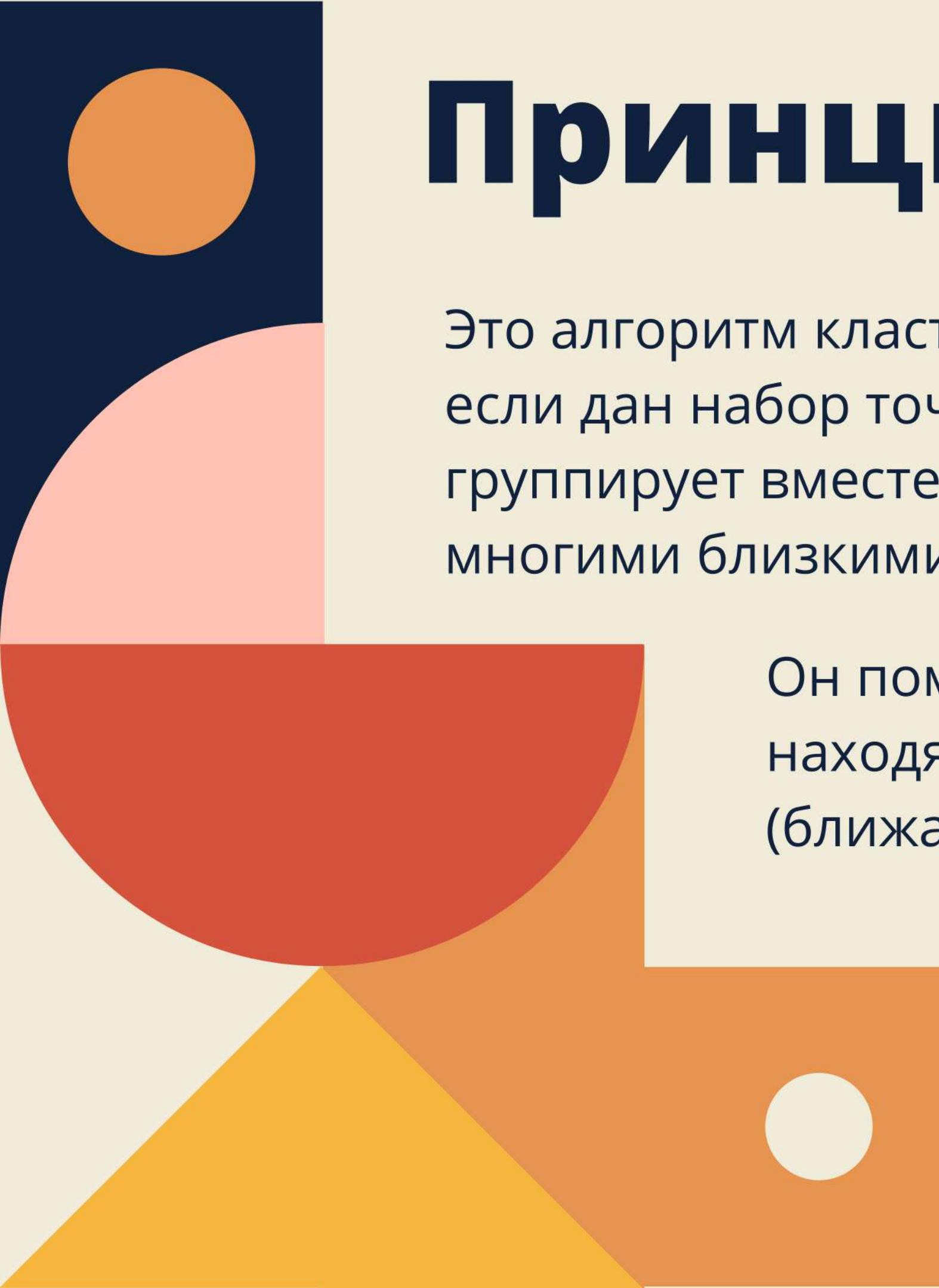


Пример работы:



Алгоритм DBSCAN





Принцип алгоритма:

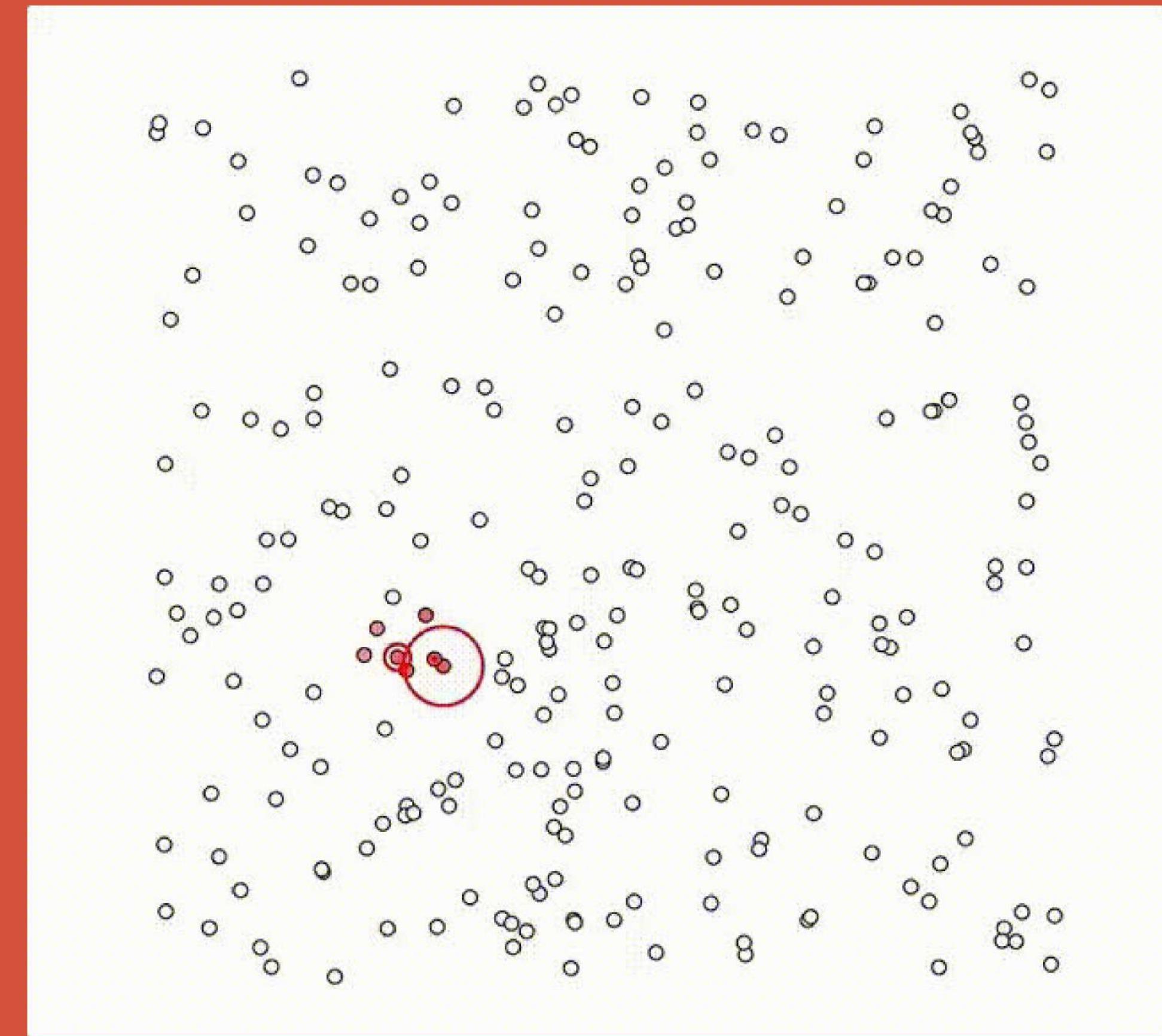
Это алгоритм кластеризации, основанной на плотности — если дан набор точек в некотором пространстве, алгоритм группирует вместе точки, которые тесно расположены (точки со многими близкими соседями).

Он помечает, как выбросы, точки, которые находятся одиноко в областях с малой плотностью (ближайшие соседи которых лежат далеко).

Шаги алгоритма:

```
DBSCAN(DB, metric, eps, minPts)
{
    C = 0                                // Счётчик кластеров
    for each point P in database DB
    {
        if label(P) ≠ undefined then continue          // Точка была просмотрена во внутреннем цикле
        Neighbors N = RangeQuery(DB, metric, P, eps)      // Находим соседей
        if |N| < minPts then                            // Проверка плотности
        {
            label(P) = Noise                           // Помечаем как шум
            continue
        }
        C = C + 1                                // следующая метка кластера
        label(P) = C                             // Помечаем точку как базовую
        Seed set S = N \ {P}                      // Соседи для расширения
        for each point Q in S
        {
            if label(Q) == Noise then label(Q) = C           // Заменяем метку метку "шум" на "границная точка"
            if label(Q) ≠ undefined then continue          // Была просмотрена
            label(Q) = C                               // Помечаем соседа
            Neighbors N = RangeQuery(DB, metric, Q, eps)  // Находим соседей
            if |N| ≥ minPts then                        // Проверяем плотность
            {
                S = S ∪ N                           // Добавляем соседей в набор граничных точек
            }
        }
    }
}
```

Пример работы:



Алгоритм иерархической кластеризации





Принцип алгоритма:

Это совокупность алгоритмов упорядочивания данных, направленных на создание иерархии (дерева) вложенных кластеров.

Выделяют два класса методов иерархической кластеризации:

- Агломеративные методы: новые кластеры создаются путем объединения более мелких кластеров и, таким образом, дерево создается от листьев к стволу;
- Дивизивные методы: новые кластеры создаются путем деления более крупных кластеров на более мелкие и, таким образом, дерево создается от ствола к листьям.

Шаги агломеративного алгоритма:

- 1) Назначаем каждый элемент кластером.
- 2) Итеративно объединяем ближайшие кластеры в один.

Расстояние между кластерами можно считать по-разному:

Метод одиночной связи

$$R_{min}(U, V) = \min \rho(u, v), \quad u \in U, v \in V$$

Метод Уорда

$$R_{ward}(U, V) = \frac{|U||V|}{|U| + |V|} \rho^2 \left(\sum_{u \in U} \frac{u}{|U|}, \sum_{v \in V} \frac{v}{|V|} \right)$$

Метод полной связи

$$R_{max}(U, V) = \max \rho(u, v), \quad u \in U, v \in V$$

Метод средней связи

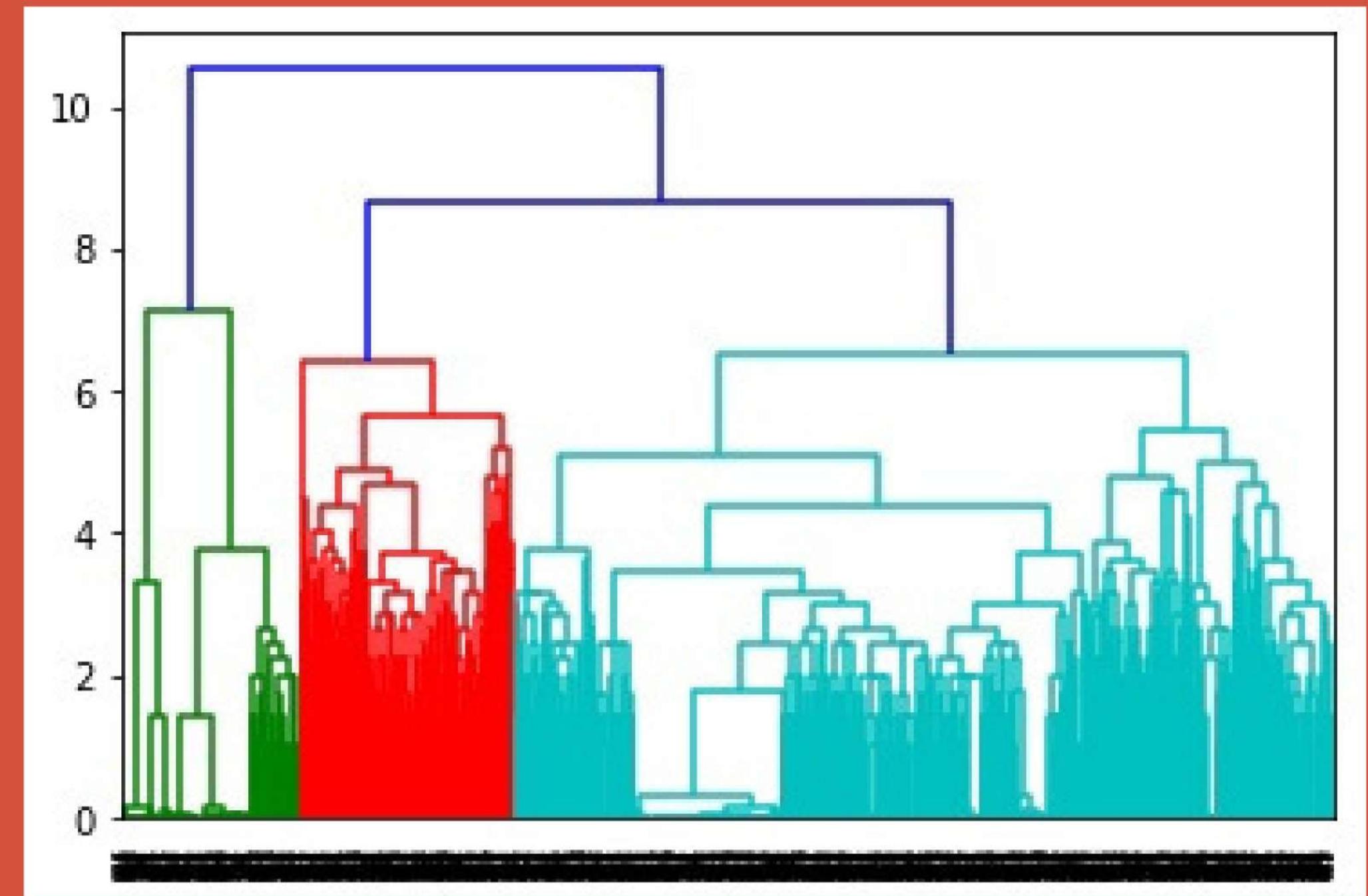
$$R_{avg}(U, V) = \frac{1}{|U||V|} \sum_{u \in U} \sum_{v \in V} \rho(u, v)$$

U, V - кластеры, между которыми измеряется расстояние, ρ - расстояние между элементами, определенное метрикой.



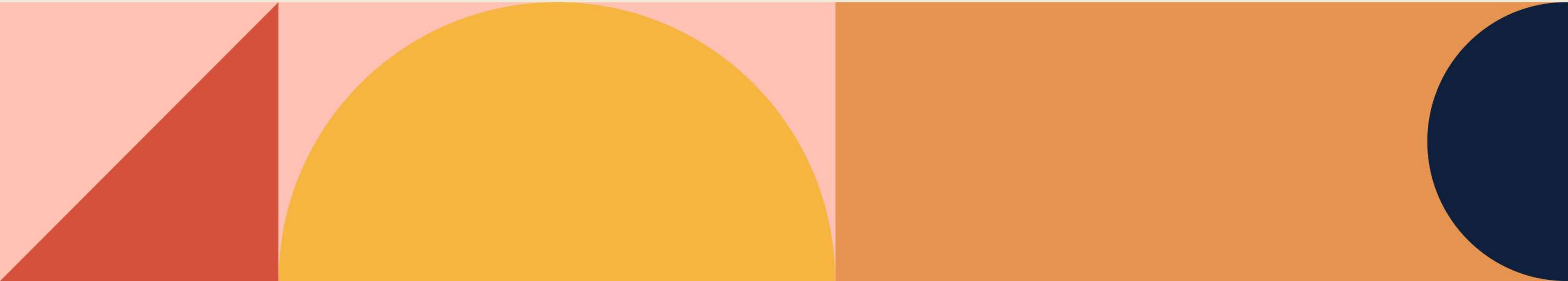
Дендрограмма

Это древовидная диаграмма, помогающая представить каждый шаг процесса последовательного укрупнения кластеров.



09

Написание кода



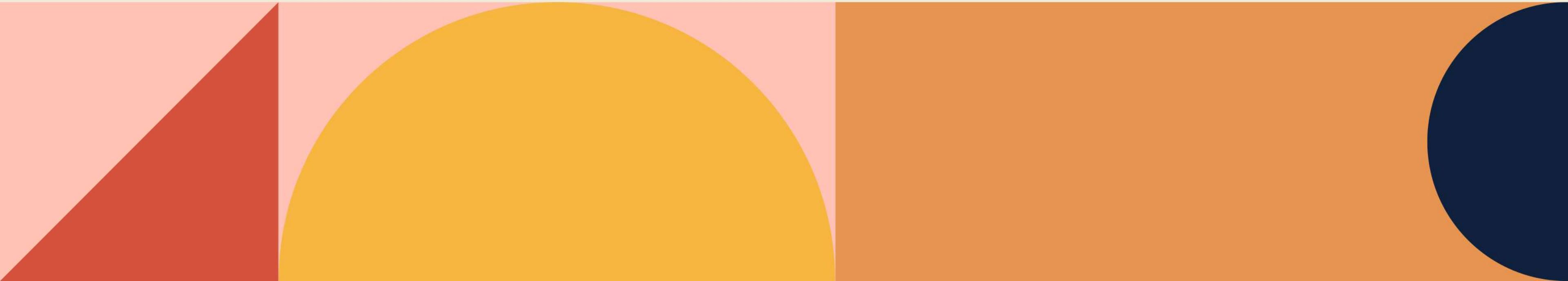
Ссылка на google colab:

[https://colab.research.google.com/drive/1uWylfLSIqw34Zh2wjAc2YbzqWRwnITUe?
usp=sharing](https://colab.research.google.com/drive/1uWylfLSIqw34Zh2wjAc2YbzqWRwnITUe?usp=sharing)

QR-код :



Обучение модели и предсказание сообществ



Метрики

Для исходных данных мы использовали **косинусное расстояние** в качестве метрики, так как **манхэттенская** и **евклидова** работают некорректно в случаях разреженных матриц координат и высоких размерностей ("проклятие размерности").

Уменьшив размерность, мы получили возможность использовать ранее неэффективные метрики, но косинусное расстояние потеряло эффективность.



Косинусное расстояние:

$$d_{cosine}(X, Y) = 1 + \frac{(X, Y)}{\|X\| \cdot \|Y\|}$$

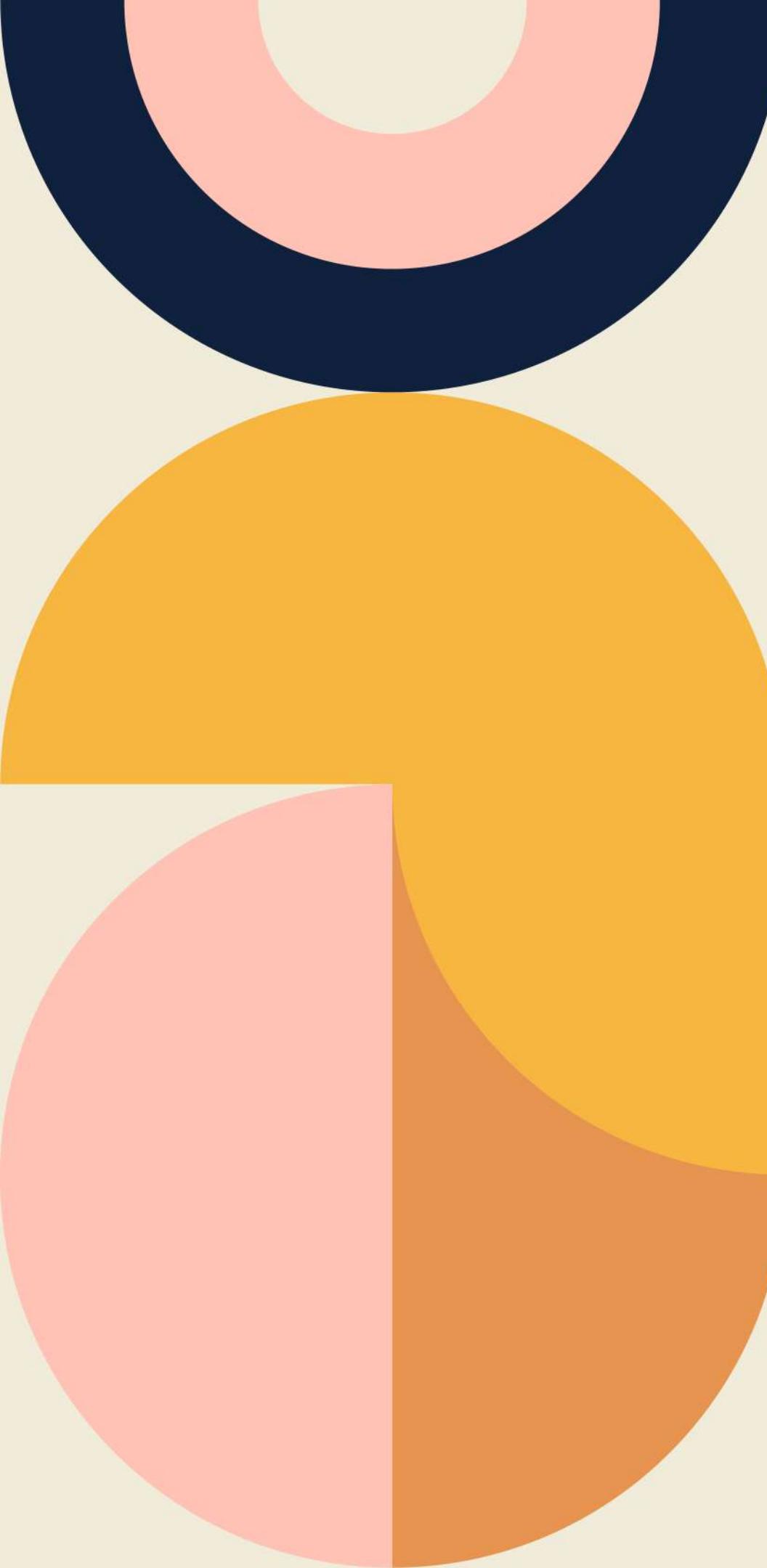
Евклидово расстояние:

$$d_e(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Манхэттенское расстояние:

$$d_m(X, Y) = \sum_{i=1}^n |x_i - y_i|$$

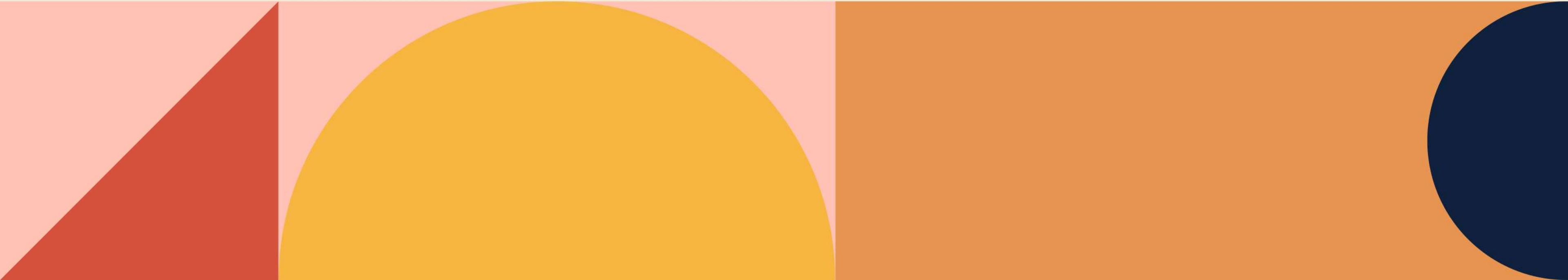
Где X, Y - векторы, x и y - координаты векторов, скалярное произведение определено как сумма произведений соответствующих координат векторов, норма индуцирована этим произведением.



Проклятие размерности

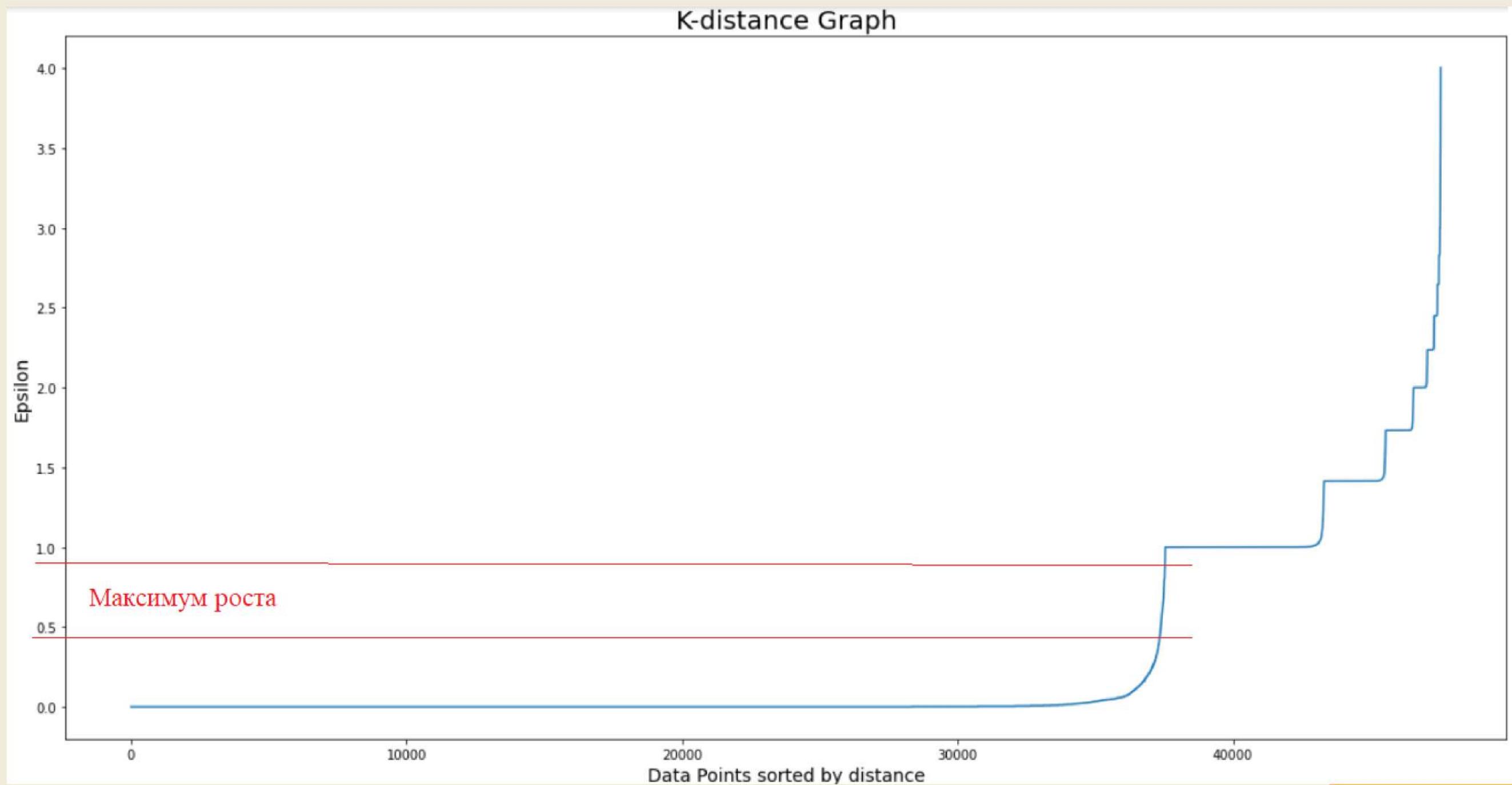
Каждое измерение, которое мы добавляем, экспоненциально увеличивает количество требуемых для обобщения образцов. Проклятие размерности чаще всего применяется к наборам данных: чем больше столбцов или переменных, тем экспоненциально больше выборок в этом датасете нам нужно проанализировать. Можно говорить и о весах, и о входных данных, принцип остается неизменным — многомерное пространство огромно!

Определение оптимальных параметров кластеризации



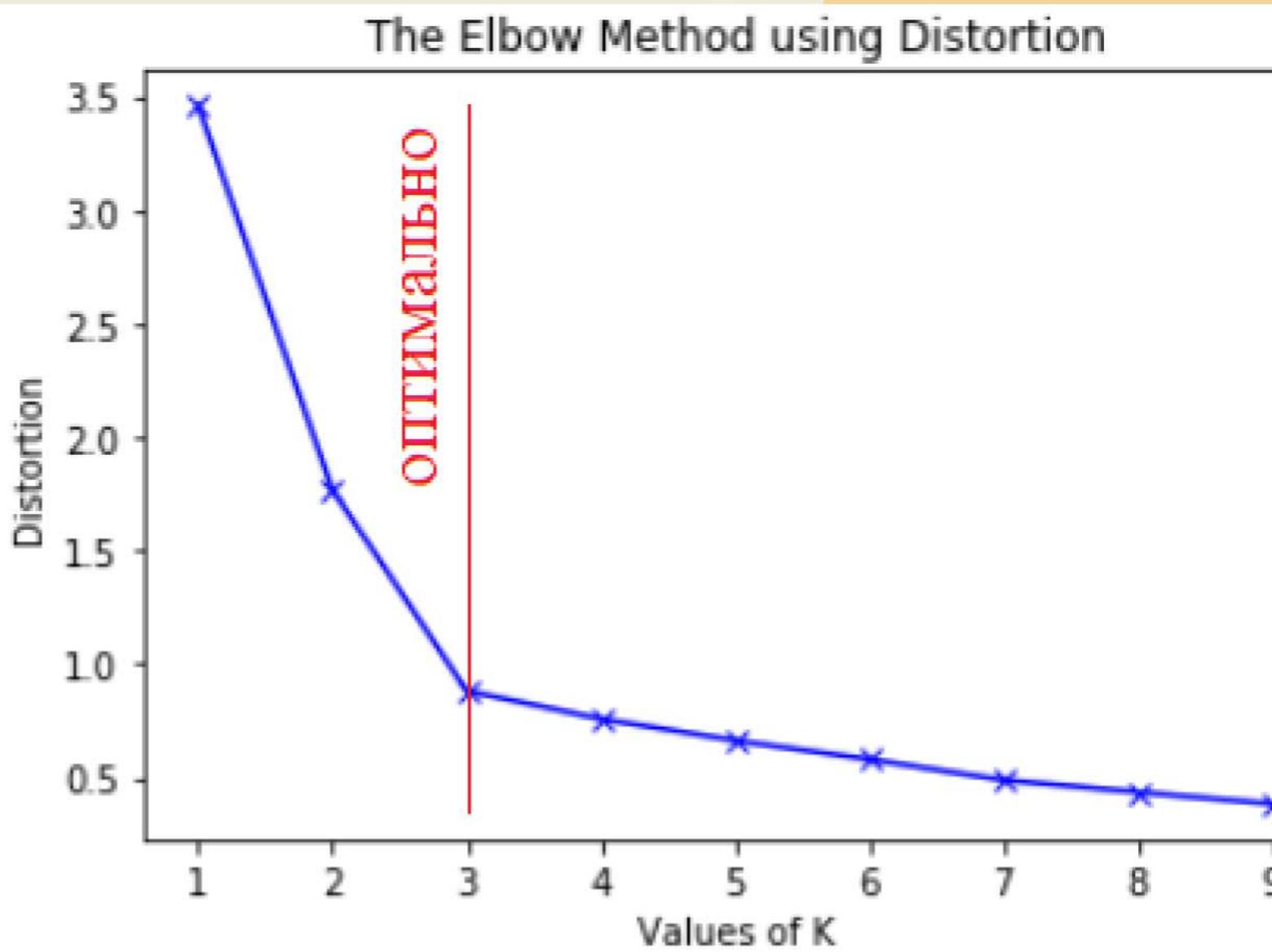
DBSCAN:

- 1) $\text{min_samples} > \dim(V)$
- 2) ε рассчитывается с помощью графика K -расстояний — дистанций между точкой и ближайшей к ней точкой данных для всех точек в наборе:



K-means:

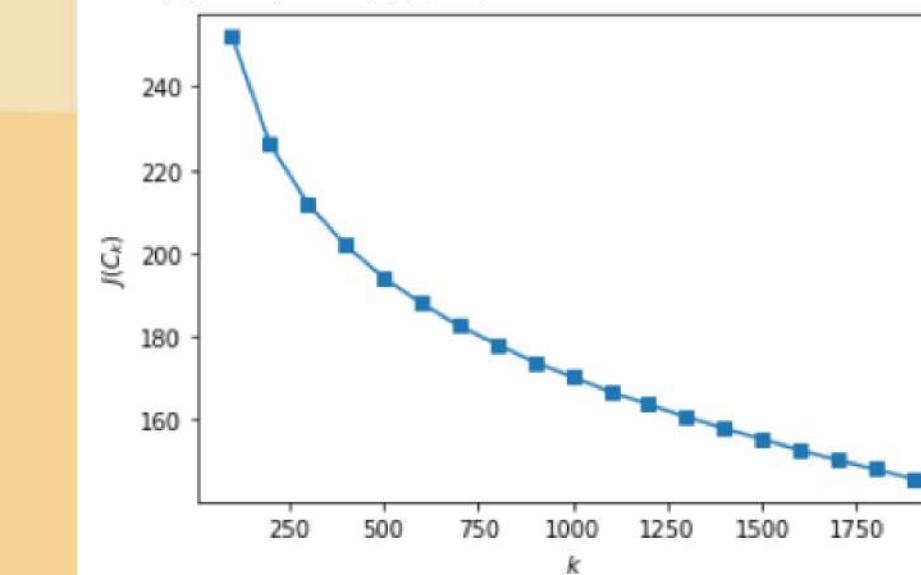
Количество кластеров определяется "методом локтя"
(в нашем случае определить не удалось):



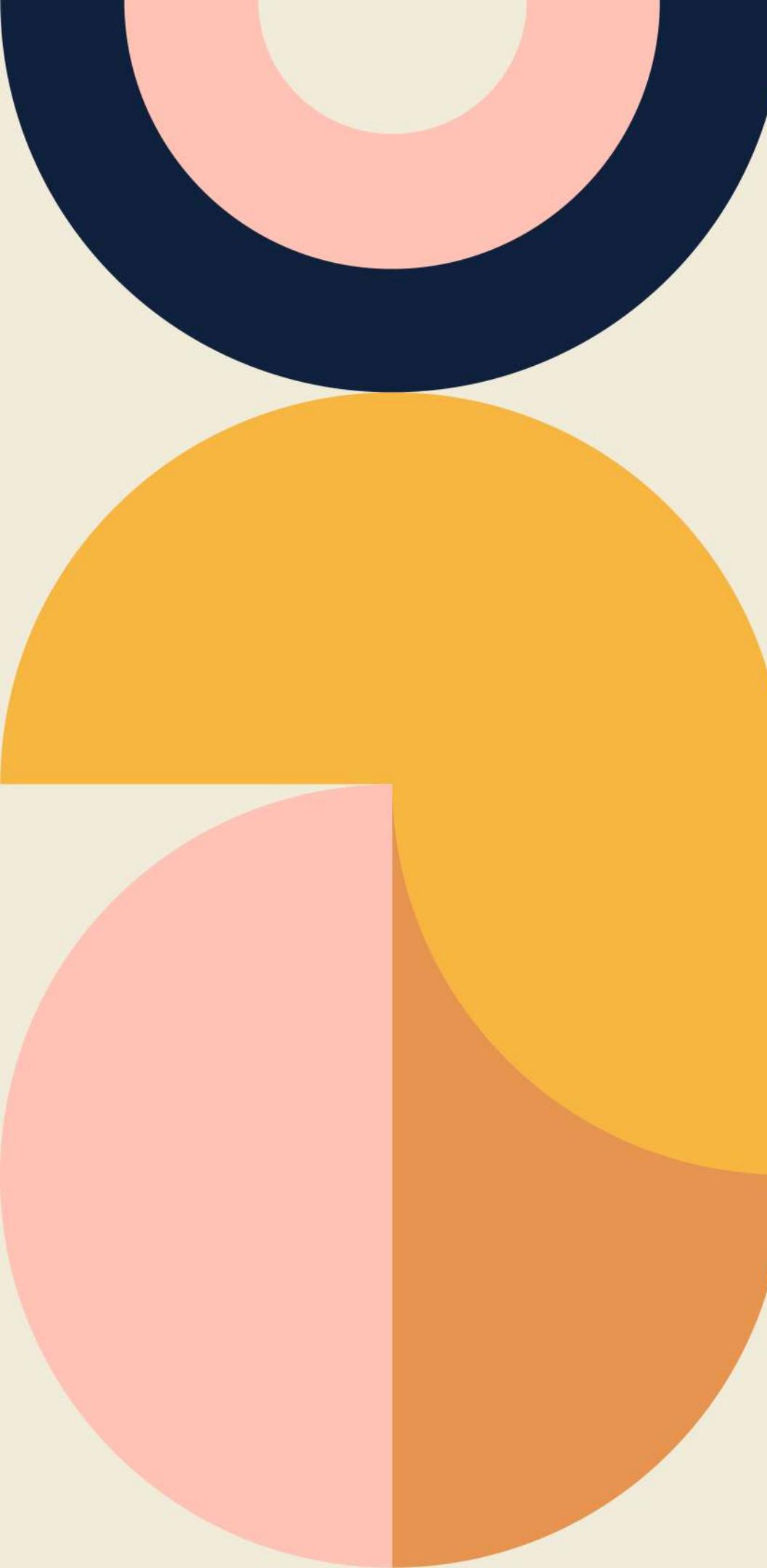
```
inertia = []
for k in range(100, 2000, 100):
    kmeans = KMeans(n_clusters=k, random_state=1).fit(X)
    inertia.append(np.sqrt(kmeans.inertia_))

plt.plot(range(100, 2000, 100), inertia, marker='s')
plt.xlabel('$k$')
plt.ylabel('$J(C_k)$')
```

Text(0, 0.5, '\$J(C_k)\$')



Оценка точности модели



Силуэт

Данный коэффициент не предполагает знания истинных меток объектов, и позволяет оценить качество кластеризации, используя только саму (неразмеченную) выборку и результат кластеризации. Сначала силуэт определяется отдельно для каждого объекта. Обозначим через **a** — среднее расстояние от данного объекта до объектов из того же кластера, через **b** — среднее расстояние от данного объекта до объектов из ближайшего кластера (отличного от того, в котором лежит сам объект).

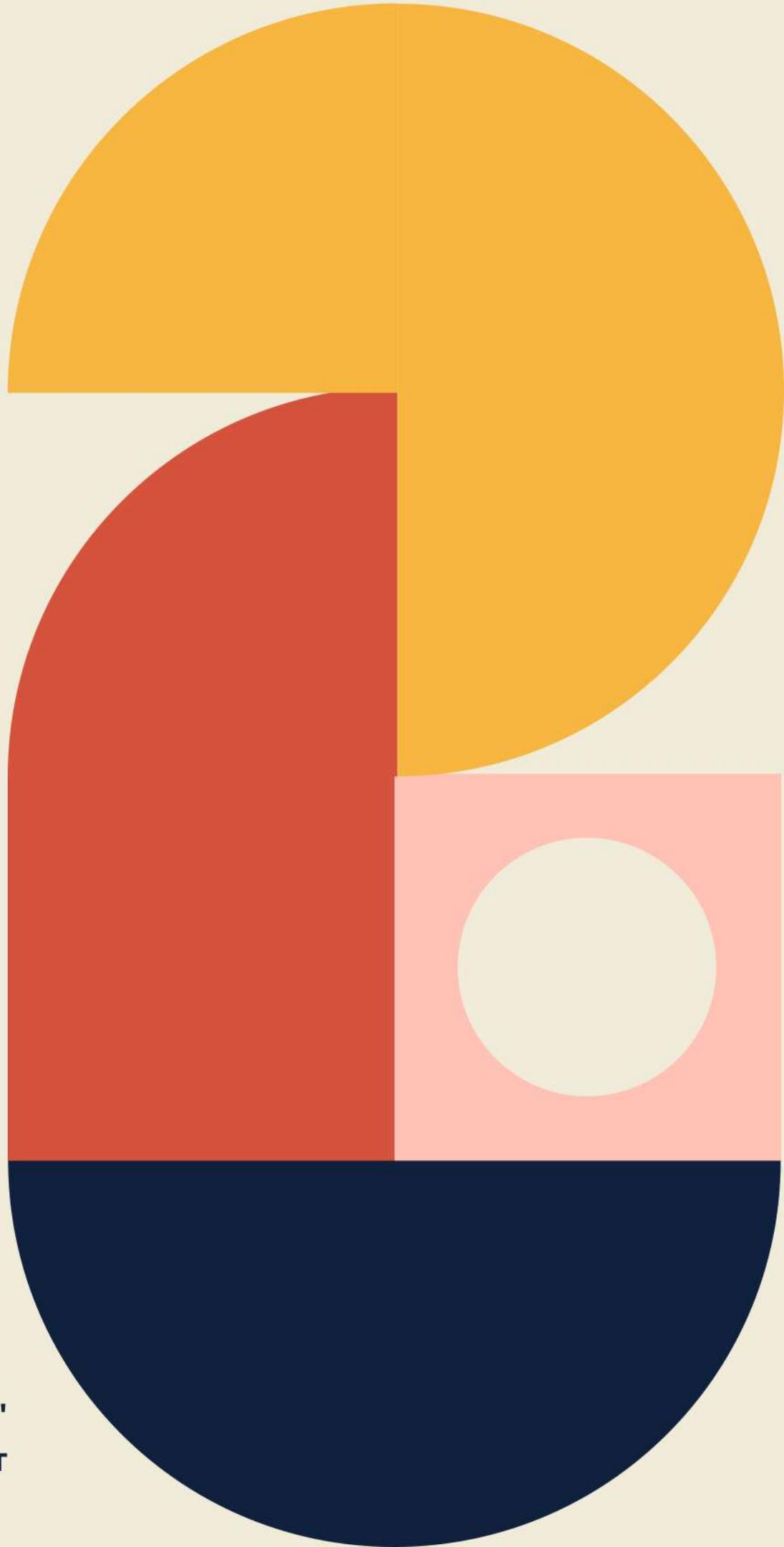
Тогда силуэтом данного объекта называется величина:

$$s = \frac{b - a}{\max(a, b)}$$

Силуэт показывает, насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров.

Данная величина лежит в диапазоне. Значения, близкие к -1, соответствуют плохим (разрозненным) кластеризациям, значения, близкие к 0, говорят о том, что кластеры пересекаются и накладываются друг на друга, значения, близкие к 1, соответствуют "плотным" четко выделенным кластерам. Таким образом, чем больше силуэт, тем более четко выделены кластеры, и они представляют собой компактные, плотно сгруппированные облака точек.

"Речи мои представляют отнюдь не пачкотню,
как вы изволите выражаться в присутствии дамы,
а вереницу плотно сгруппированных силлогизмов"
Кот Бегемот



Оценка методом силуэтов:

```
# считаем силуэт KMeans:  
score = silhouette_score(X, KM_labels, metric = 'euclidean')  
print(score)
```

0.34391742984970536

```
# считаем силуэт DBSCAN  
score = silhouette_score(X, DB_labels, metric = 'manhattan')  
print(score)
```

0.27002226731369666

```
# считаем силуэт Birch  
score = silhouette_score(X, BR_clustering.labels_, metric = 'euclidean')  
print(score)
```

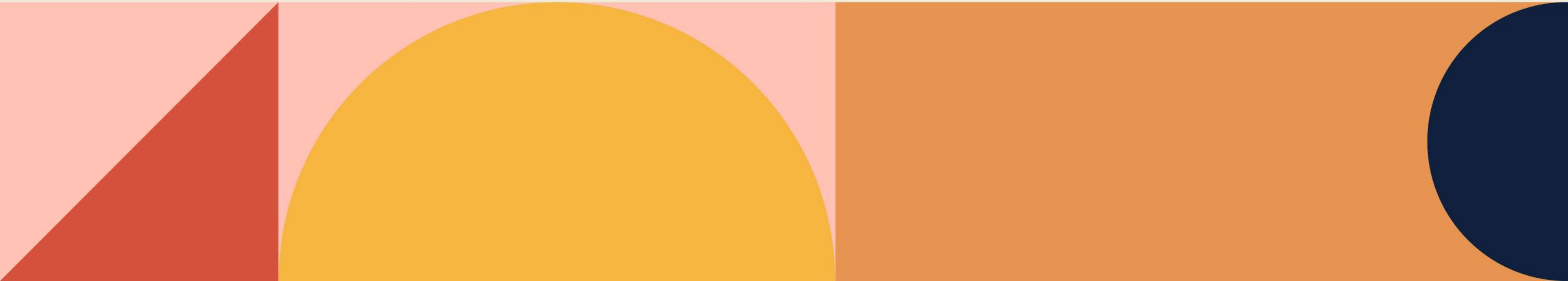
0.15051523576882603

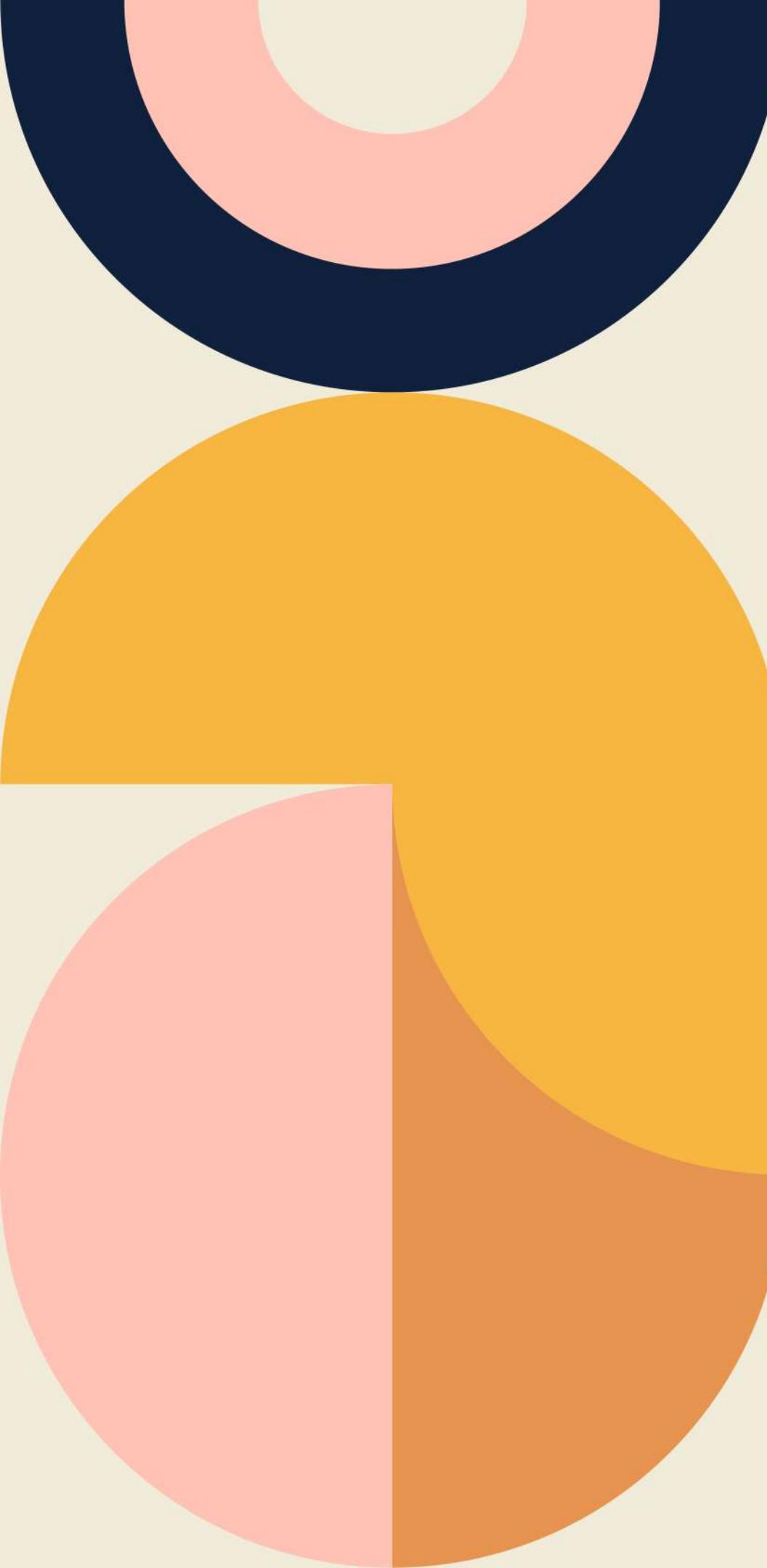
Оценка алгоритмов относительно друг друга через Rand индекс:

```
# оценим алгоритмы друг относительно друга
from sklearn import metrics
from sklearn.metrics.cluster import rand_score
labels_true = BR_labels
labels_pred = DB_labels
print('DBSCAN относительно Birch: ', metrics.rand_score(labels_true, labels_pred))
labels_pred = KM_labels
print('K-means относительно Birch: ', metrics.rand_score(labels_true, labels_pred))
labels_true = KM_labels
labels_pred = DB_labels
print('DBSCAN относительно K-means:', metrics.rand_score(labels_true, labels_pred))
```

DBSCAN относительно Birch: 0.7599607283768968
K-means относительно Birch: 0.9328819600599448
DBSCAN относительно K-means: 0.8143951734758463

Результат работы

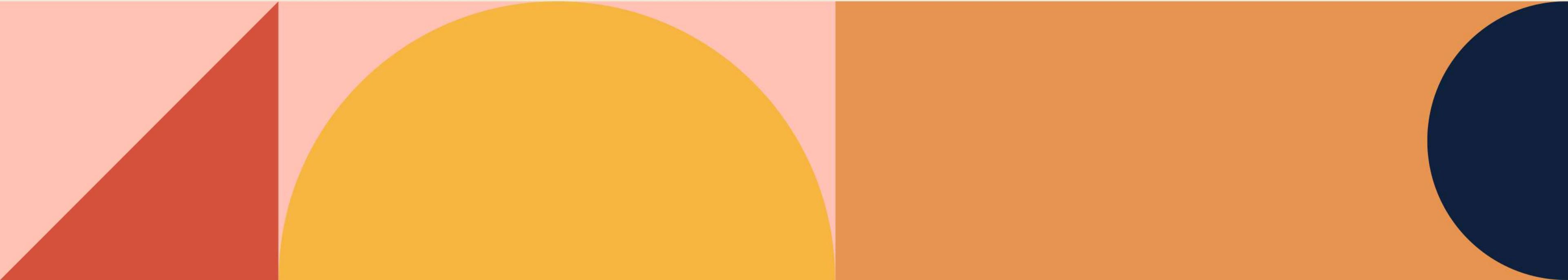




По итогу наша команда расшифровала исходные данные, выполнила парсинг двумя способами благодаря библиотеке pandas.

Мы сначала вручную выполнили кластеризацию по дружественным связям, а потом - кластеризацию по музыкальным предпочтениям с помощью функций библиотек scikit-learn и networkx методами K-means, DBSCAN, OPTICS, BIRCH, а затем сравнили между собой полученные результаты и оценили их методом силуэтов.

Примеры кластеров



Часть кластера № 202 по OPTICS:

	Blues	Classic	Comedy	Country	Dance	Electro	Folk	Hip-Hop	Jazz	Kids	Old School	Pop	R&B	Religion	Rock	Singer & Songwriter	Soul	Soundtracks	Sports
5261	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
6901	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
9354	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
11057	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
13816	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
13847	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
14523	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
18835	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
21375	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
21990	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
24918	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
25397	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
26408	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
31665	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
31867	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
35133	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
36740	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0
39330	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0		0.0	0.0	0.0

Часть кластера № 1 по k-means:

	Blues	Classic	Comedy	Country	Dance	Electro	Folk	Hip-Hop	Jazz	Kids	Old School	Pop	R&B	Religion	Rock	Singer & Songwriter	Soul	Soundtracks	Sports
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
44	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
60	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...
47496	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
47499	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
47506	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
47511	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
47521	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Часть кластера № 40 по k-means:

	Blues	Classic	Comedy	Country	Dance	Electro	Folk	Hip-Hop	Jazz	Kids	Old School	Pop	R&B	Religion	Rock	Singer & Songwriter	Soul	Soundtracks	Sports
207	0.0	0.0	0.0	0.0	2.0	0.0	0.0	1.0	0.0	0.0	0.0	3.0	1.0	1.0	3.0	0.0	0.0	0.0	0.0
233	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	3.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0
273	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0
732	0.0	0.0	0.0	0.0	2.0	0.0	1.0	1.0	0.0	0.0	0.0	3.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0
772	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	3.0	1.0	0.0	3.0	0.0	0.0	0.0	0.0
...
46829	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0
46833	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	3.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0
47014	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	3.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0
47200	0.0	0.0	0.0	0.0	1.0	2.0	1.0	1.0	0.0	0.0	0.0	3.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0
47430	0.0	0.0	0.0	0.0	2.0	1.0	1.0	0.0	0.0	0.0	0.0	2.0	1.0	0.0	3.0	0.0	1.0	0.0	0.0

Часть кластера № 3 по BIRCH:

	Blues	Classic	Comedy	Country	Dance	Electro	Folk	Hip-Hop	Jazz	Kids	Old School	Pop	R&B	Religion	Rock	Singer & Songwriter	Soul	Soundtracks	Sports
398	0.0	0.0	1.0	0.0	3.0	2.0	0.0	2.0	0.0	0.0	0.0	2.0	2.0	0.0	2.0	0.0	0.0	0.0	0.0
427	0.0	0.0	0.0	0.0	2.0	2.0	0.0	2.0	0.0	0.0	0.0	2.0	0.0	0.0	2.0	0.0	1.0	0.0	0.0
544	0.0	0.0	0.0	0.0	3.0	1.0	1.0	1.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	0.0
861	0.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0	0.0	0.0	0.0	3.0	2.0	0.0	3.0	0.0	0.0	0.0	0.0
895	0.0	0.0	0.0	0.0	2.0	1.0	0.0	2.0	0.0	0.0	0.0	2.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0
...
47157	0.0	0.0	0.0	0.0	2.0	1.0	1.0	1.0	0.0	0.0	0.0	3.0	2.0	0.0	4.0	0.0	0.0	0.0	0.0
47295	0.0	0.0	0.0	0.0	3.0	2.0	0.0	2.0	0.0	0.0	0.0	2.0	2.0	0.0	2.0	0.0	2.0	0.0	0.0
47305	0.0	0.0	0.0	0.0	2.0	2.0	1.0	1.0	0.0	0.0	0.0	3.0	1.0	0.0	3.0	0.0	0.0	0.0	0.0
47458	0.0	0.0	0.0	0.0	1.0	1.0	1.0	2.0	0.0	0.0	0.0	3.0	2.0	0.0	3.0	0.0	0.0	0.0	0.0
47526	0.0	0.0	0.0	0.0	1.0	2.0	1.0	3.0	0.0	0.0	0.0	2.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0

Часть кластера № 27 по BIRCH:

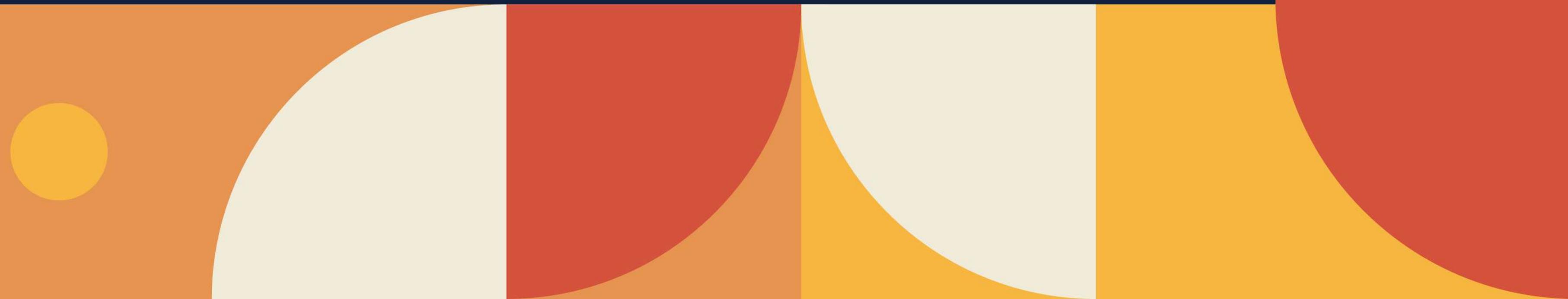
	Blues	Classic	Comedy	Country	Dance	Electro	Folk	Hip-Hop	Jazz	Kids	Old School	Pop	R&B	Religion	Rock	Singer & Songwriter	Soul	Soundtracks	Sports
332	0.0	0.0	0.0	2.0	2.0	0.0	0.0	2.0	1.0	0.0	0.0	1.0	2.0	0.0	2.0	0.0	2.0	0.0	0.0
773	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	2.0	0.0	0.0	2.0	2.0	0.0	1.0	1.0	2.0	0.0	0.0
867	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0
944	0.0	0.0	0.0	0.0	0.0	0.0	2.0	1.0	0.0	1.0	0.0	1.0	2.0	0.0	1.0	0.0	2.0	0.0	0.0
1178	0.0	0.0	0.0	0.0	2.0	1.0	1.0	1.0	2.0	0.0	0.0	1.0	2.0	0.0	1.0	0.0	1.0	0.0	0.0
...
46127	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0	0.0	1.0	0.0	0.0
46768	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	2.0	0.0	1.0	0.0	1.0	0.0	0.0
46892	0.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0	0.0	0.0	0.0	2.0	2.0	0.0	2.0	0.0	1.0	0.0	0.0
47306	0.0	0.0	0.0	0.0	1.0	1.0	0.0	2.0	1.0	0.0	0.0	2.0	2.0	0.0	2.0	0.0	1.0	0.0	0.0
47485	0.0	0.0	0.0	0.0	2.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	2.0	0.0	0.0

Часть кластера № 16 по DBSCAN:

Часть кластера № 72 по DBSCAN:

	Blues	Classic	Comedy	Country	Dance	Electro	Folk	Hip-Hop	Jazz	Kids	Old School	Pop	R&B	Religion	Rock	Singer & Songwriter	Soul	Soundtracks	Sports	
277	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0
382	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0
543	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0
1371	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0
1502	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0
...
45731	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0
45854	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0
45927	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0
46912	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0
47161	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0		0.0	0.0	0.0	0.0

Неудачные дубли:



Иерархическая кластеризация

Было потрачено много времени и сил на реализацию иерархической агломеративной кластеризации, однако при любой попытке ее запустить весь colab ломался из-за нехватки ОЗУ, из-за чего приходилось запускать все фрагменты кода заново до того момента, пока не создали файлы с отформатированными фреймами.

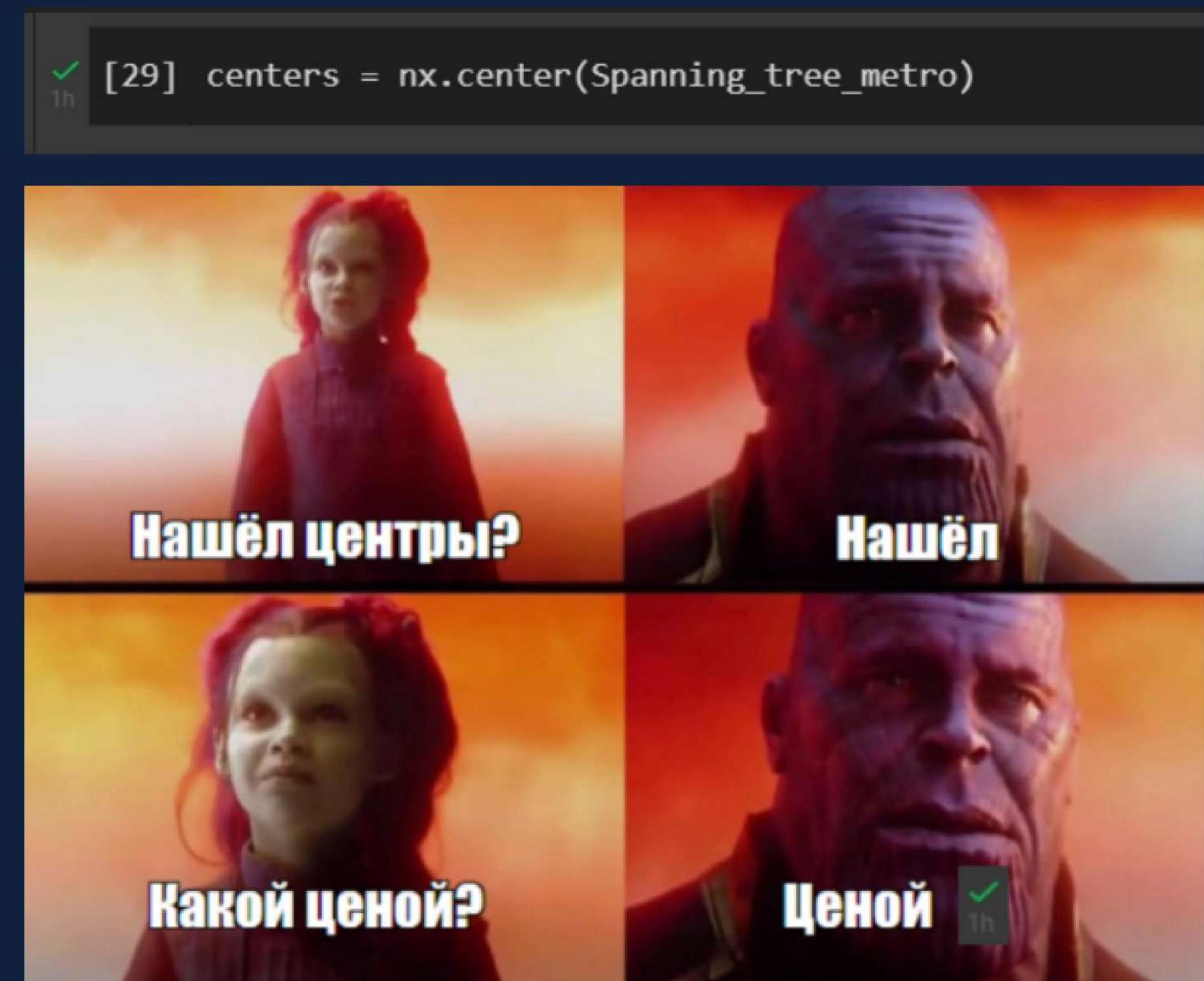


Идея с центрами

На поиск центров в графе дружбы
google colab потратил час.

Но эта информация оказалась
совершенно бесполезной...

```
✓ 0s    ▶ centers
[39706, 28442]
```



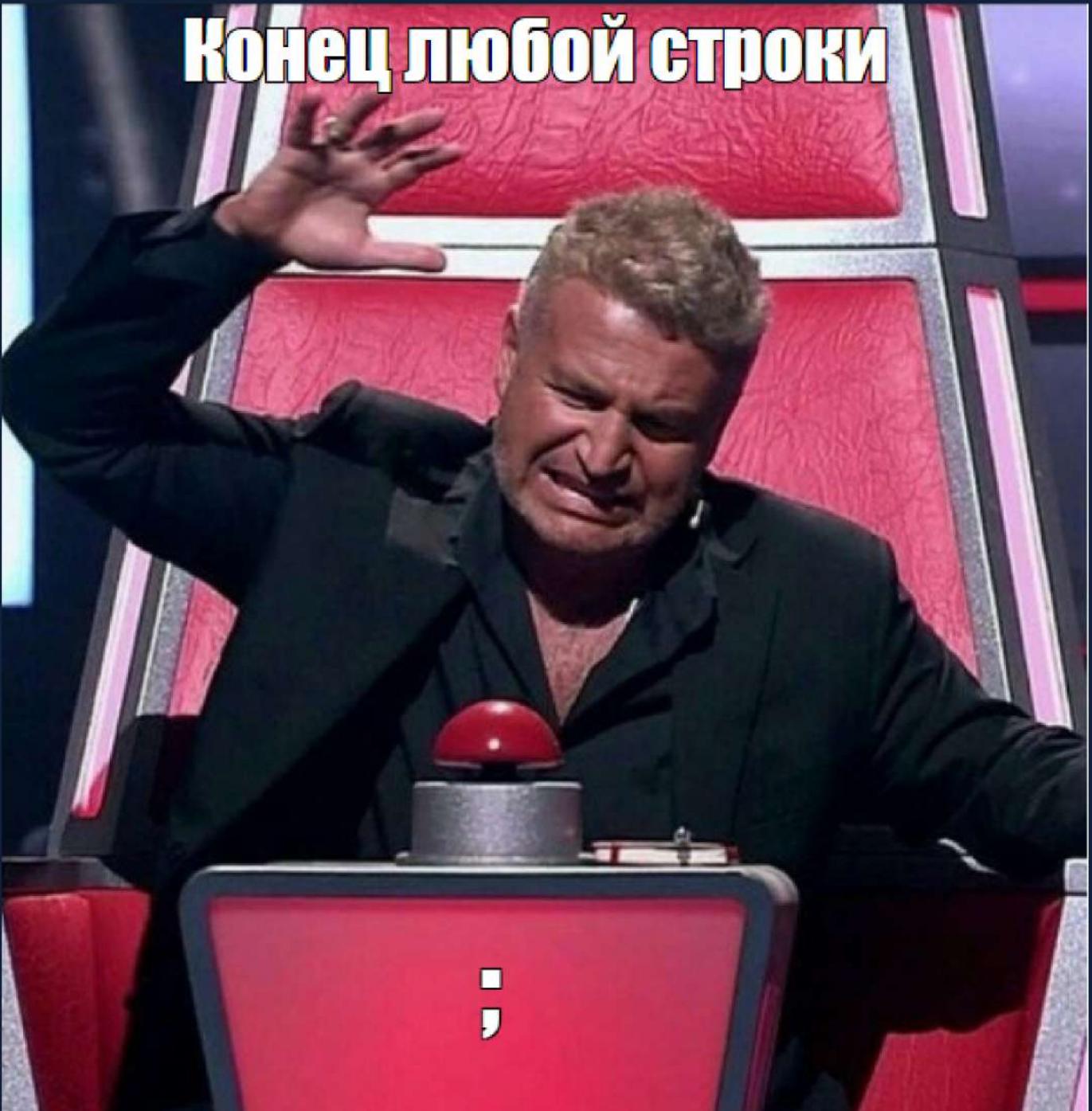
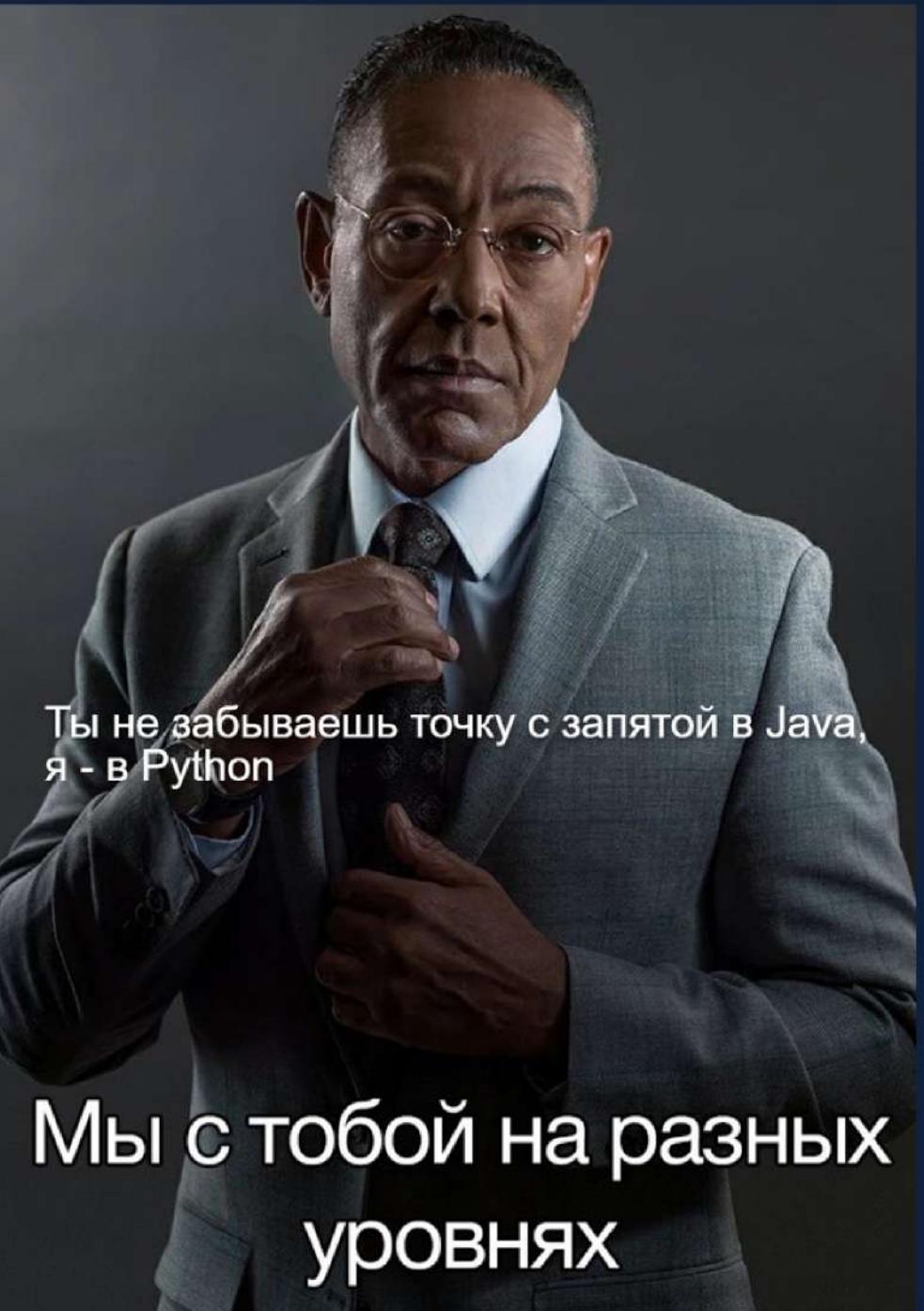
OPTICS

Алгоритм кластеризации OPTICS вселял в нас надежду на исправление главной проблемы DBSCAN в нашей реализации — обилие шума. Однако, настроить OPTICS нам удалось, и в лучшем случае он выдавал тот же результат, что и DBSCAN...



Точка с запятой

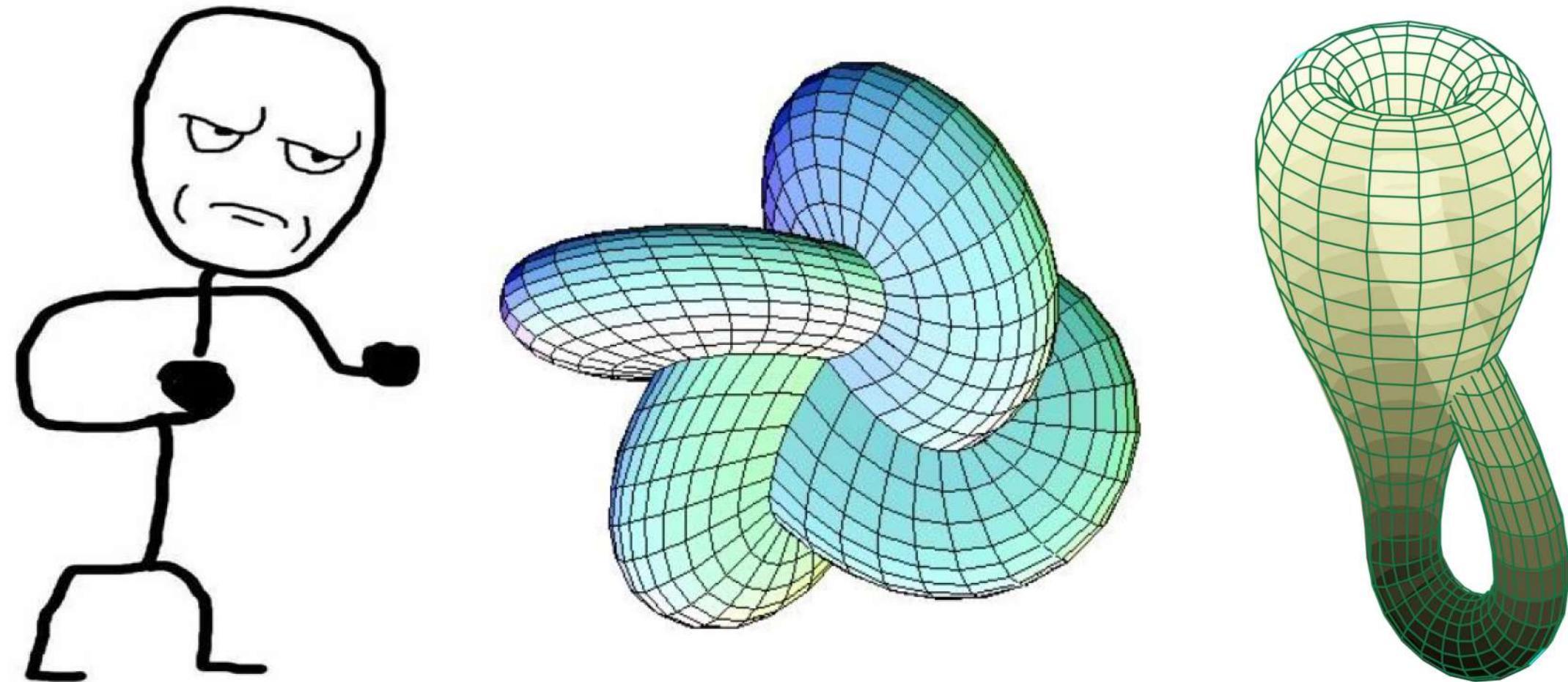
```
plt.plot(range(50, 1000, 50), inertia, marker='s');  
plt.xlabel('$k$')  
plt.ylabel('$J(c_k)$');|
```



Отрицательная размерность

```
ValueError: negative dimensions are not allowed
```

ну давай, давай, нападай



В одной из попыток
выполнить
иерархическую
агломеративную
кластеризацию вылезла
такая ошибка.