Лабораторная работа №9 творческая!

Выполнили Зимин Андрей Валерьевич и Жилин Андрей Игоревич

Импорт библиотек

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
import seaborn as sns
```

Задание 1

Выполнить кластеризацию на каком-нибудь датасете

Загрузка датасета

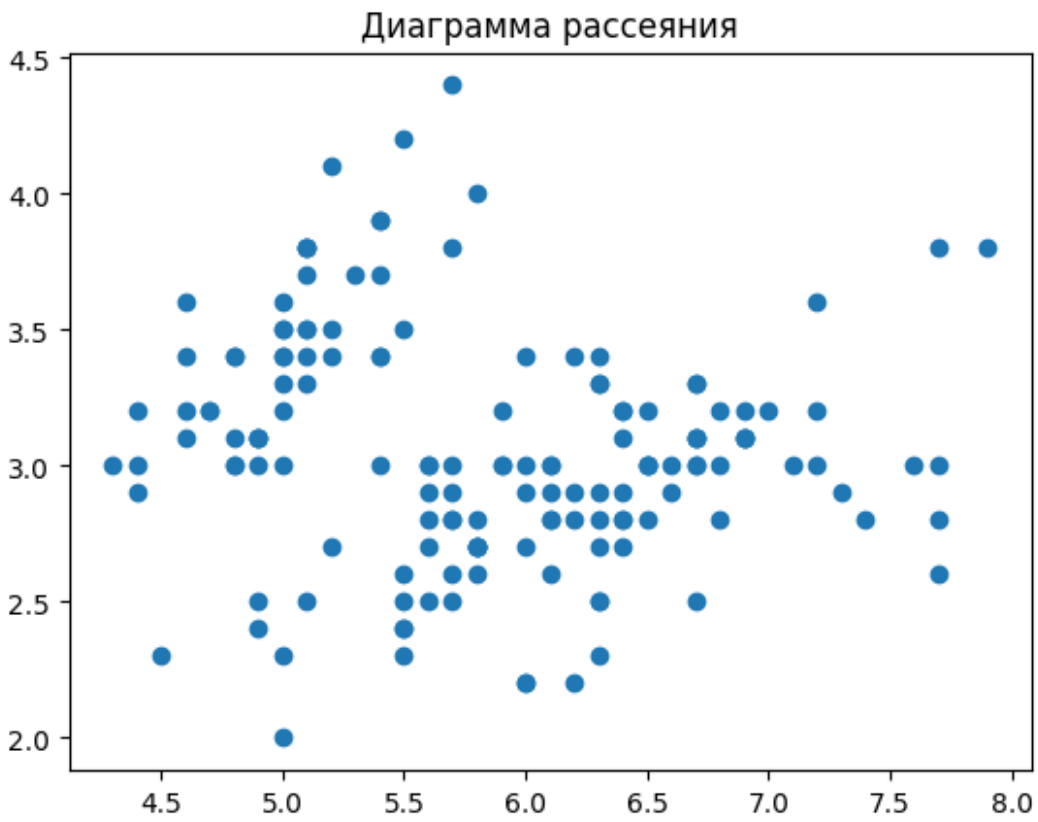
```
df = pd.read_csv('data/iris.csv')
X = df.drop(columns=['variety'])

X_numpy = X.values[:, :2]
X_numpy[:5]
```

```
array([[5.1, 3.5],
       [4.9, 3. ],
       [4.7, 3.2],
       [4.6, 3.1],
       [5. , 3.6]])
```

Импортировали датасет, посмотрим на распределение

```
plt.scatter(X_numpy[:, 0], X_numpy[:, 1])
plt.title("Диаграмма рассеяния")
```

```
Text(0.5, 1.0, 'Диаграмма рассеяния')
```



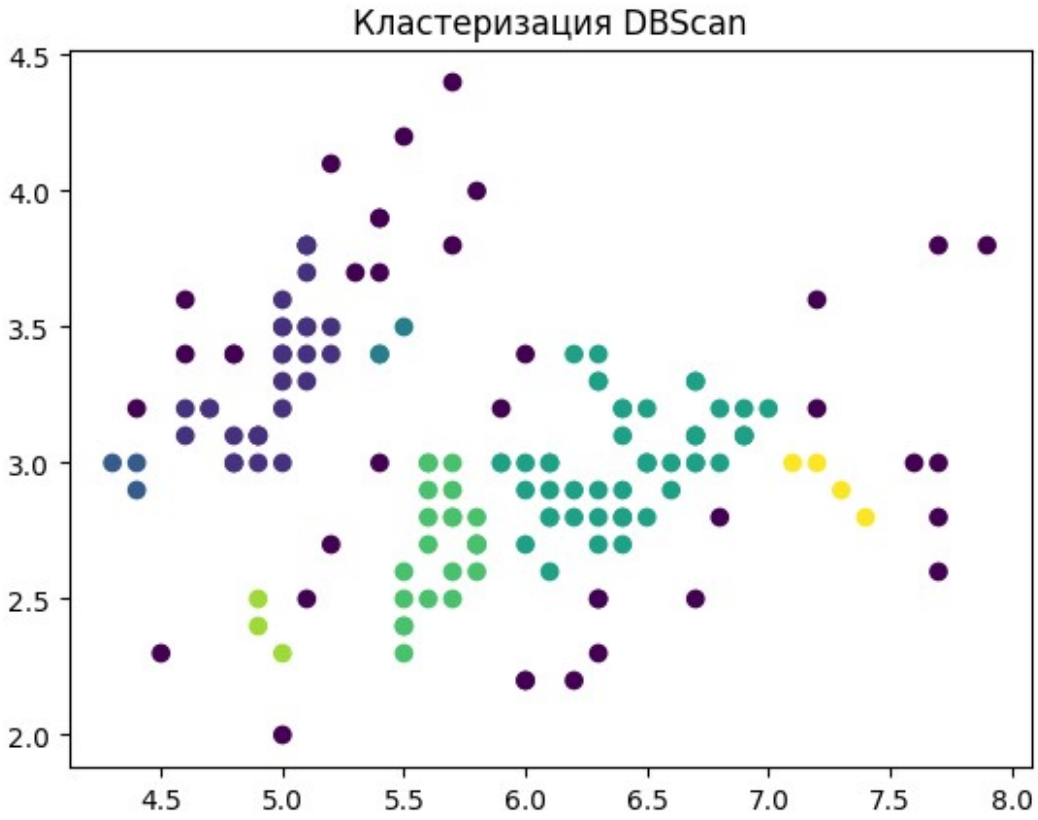
С кластерами тут сложно. Посмотрим что получится

```
class DBScan:
    def __init__(self, esp, min_pts, metric):
        self.esp = esp
        self.min_pts = min_pts
        self.metric = metric

    def fit_predict(self, X):
        # построим матрицу расстояний distance от каждого объекта до
каждого
        n = X.shape[0]
        self.distance = np.zeros((n, n))
        for i in range(n-1):
            for j in range(i+1, n):
                value = self.metric(X[i, :], X[j, :])
                self.distance[i][j] = value
                self.distance[j][i] = value

        # определение корневых точек
        root = [sum(self.distance[i, :] < self.esp) >= self.min_pts
for i in range(n)]
        # определение граничных точек
        border = [(not root[i]) and (True in [root[j] and
```

```
(self.distance[i, j] < self.esp) for j in range(n)]) for i in
range(n)]
        # определение выбросов
        noise = [not (root[i] or border[i]) for i in range(n)]
        #return root, border, noise

        # Создаем метки классов
        labels = -1 * np.ones(n, dtype=int)  # Изначально все метки -1
(шум)
        cluster_id = 0

        # Присваиваем метки классам
        for i in range(n):
            if root[i] and labels[i] == -1:  # Если это корневая точка
и она еще не имеет метки
                labels[i] = cluster_id  # Присваиваем метку кластеру
                self._expand_cluster(i, labels, cluster_id)  #
Расширяем кластер
                cluster_id += 1  # Увеличиваем идентификатор кластера
для следующего кластера

        return labels

    def _expand_cluster(self, point_idx, labels, cluster_id):
        # Получаем индексы соседей для текущей точки
        neighbors = [j for j in range(len(labels)) if
self.distance[point_idx, j] < self.esp]

        # Присваиваем метки всем соседям
        for neighbor_idx in neighbors:
            if labels[neighbor_idx] == -1:  # Если сосед еще не
помечен
                labels[neighbor_idx] = cluster_id  # Присваиваем метку
текущему кластеру
                self._expand_cluster(neighbor_idx, labels, cluster_id)
# Рекурсивно расширяем кластер

model = DBScan(esp=0.15, min_pts=3, metric=lambda x1, x2: sum((x1-
x2)**2)**0.5)
labels = model.fit_predict(X_numpy)
plt.scatter(X_numpy[:, 0], X_numpy[:, 1], c=labels)
plt.title("Кластеризация DBScan")
labels
```

```
array([ 0,  0,  0,  0,  0, -1, -1,  0,  1,  0, -1, -1,  0,  1, -1, -1,
-1,
        0, -1,  0,  2,  0, -1,  0, -1,  0,  0,  0,  0,  0,  0,  2, -1,
-1,
        0,  0,  2,  0,  1,  0,  0, -1, -1,  0,  0,  0,  0,  0, -1,  0,
3,
```

```
       3,  3,  4,  3,  4,  3,  5,  3, -1, -1,  3, -1,  3,  4,  3,  4,
4,
      -1,  4, -1,  3, -1,  3,  3,  3, -1,  3,  3,  4,  4,  4,  4,  3,
-1,
      -1,  3, -1,  4,  4,  4,  3,  4,  5,  4,  4,  4,  3, -1,  4,  3,
4,
       6,  3,  3, -1,  5,  6, -1, -1,  3,  3,  3,  4,  4,  3,  3, -1,
-1,
      -1,  3,  4, -1,  3,  3, -1,  3,  3,  3,  6,  6, -1,  3,  3,  3,
-1,
       3,  3,  3,  3,  3,  3,  4,  3,  3,  3, -1,  3,  3,  3])
```



Определились некоторые классы. Выглядит неплохо. Также dbscan определил выбросы. Теперь посмотрим что скажет kmeans.

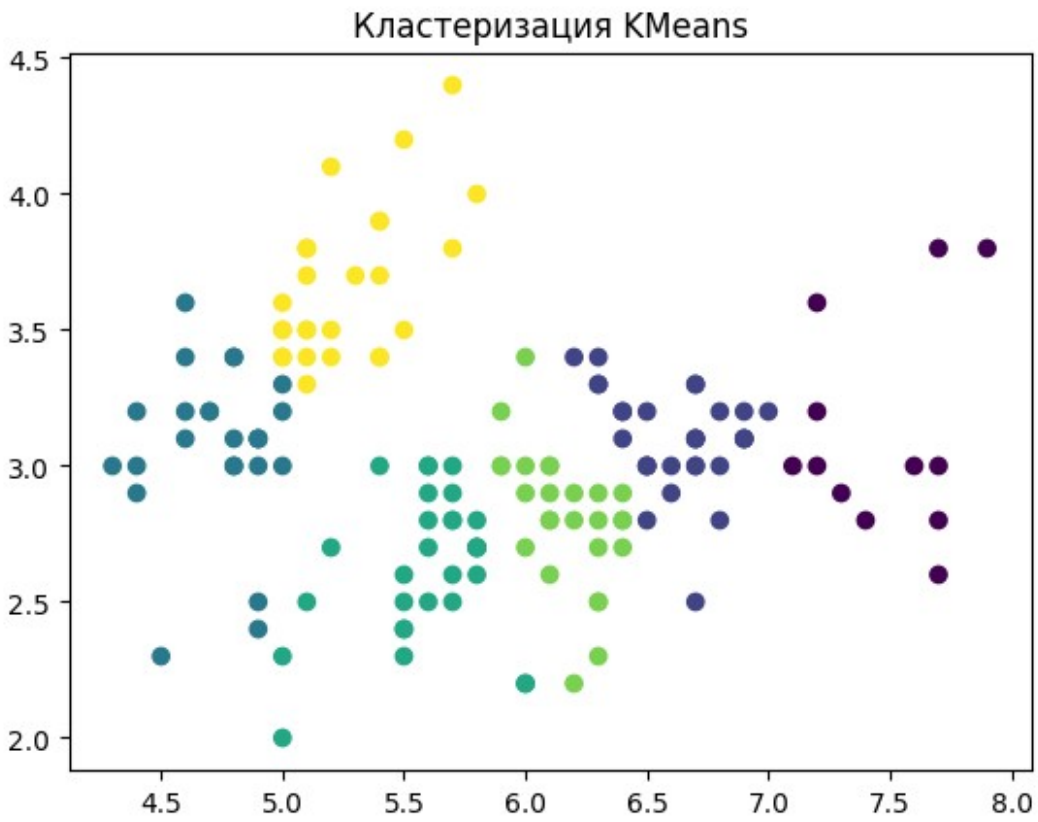
```python
def kmeans(X, k, max_iters=100):
    # Шаг 1: Случайным образом инициализируем центры кластеров
    centers = X[np.random.choice(X.shape[0], k, replace=False)]

    for _ in range(max_iters):
        # Шаг 2: Присваиваем точки к ближайшему центру
        distances = np.linalg.norm(X[:, np.newaxis] - centers, axis=2)
        labels = np.argmin(distances, axis=1)
```

```
        # Шаг 3: Обновляем центры кластеров
        new_centers = np.array([X[labels == i].mean(axis=0) for i in
range(k)])

        if np.all(centers == new_centers):
            break

        centers = new_centers

    return labels, centers

k = 6
labels, centers = kmeans(X_numpy, k)
labels
```

```
array([5, 2, 2, 2, 5, 5, 2, 5, 2, 2, 5, 2, 2, 2, 5, 5, 5, 5, 5, 5, 5,
5,
       2, 5, 2, 2, 5, 5, 5, 2, 2, 5, 5, 5, 2, 2, 5, 2, 2, 5, 5, 2, 2,
5,
       5, 2, 5, 2, 5, 2, 1, 1, 1, 3, 1, 3, 1, 2, 1, 3, 3, 4, 3, 4, 3,
1,
       3, 3, 4, 3, 4, 4, 4, 4, 4, 1, 1, 1, 4, 3, 3, 3, 3, 4, 3, 4, 1,
4,
       3, 3, 3, 4, 3, 3, 3, 3, 3, 4, 3, 3, 1, 3, 0, 4, 1, 0, 2, 0, 1,
0,
       1, 4, 1, 3, 3, 1, 1, 0, 0, 3, 1, 3, 0, 4, 1, 0, 4, 4, 4, 0, 0,
0,
       4, 4, 4, 0, 1, 1, 4, 1, 1, 1, 3, 1, 1, 1, 4, 1, 1, 4])
```

```
plt.scatter(X_numpy[:, 0], X_numpy[:, 1], c=labels)
plt.title("Кластеризация KMeans")
```

```
Text(0.5, 1.0, 'Кластеризация KMeans')
```



Получилось что-то достаточно похожее, однако kmeans не учитывает выбросы (из-за алгоритма, который лежит внутри)

Импорт библиотек

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
import seaborn as sns
```

Задание 2

Датасет взят из 2-го сезона соревнований "CodeRun" от компании Яндекс.

Название задачи: Внимание, выброс!
Текст задачи: Когда Меченый возглавил Институт изучения Зоны, его заинтересовали некоторые аномалии -- а именно выбросы, у которых определенные характеристики заметно отличались от остальных. Все его размышления были зафиксированы в КПК. К несчастью, КПК разгрызла псевдособака во время очередной вылазки, однако сами данные у Меченого остались в его лаборатории. Помогите ему заново отыскать аномальные выбросы среди имеющихся.

Загрузка датасета

```
data = []
with open('./data/attention_to_emission_input.txt') as file:
    n, m = map(int, file.readline().split())
    print(f"Количество объектов:  {n}")
    print(f"Количество признаков: {m}")
    data = [[float(i) for i in file.readline().split()] for i in
range(n)]
    data = np.array(data)

print("Первые 5 объектов датасета:")
data[:5, :]
```

```
Количество объектов:  50000
Количество признаков: 21
Первые 5 объектов датасета:
```

```
array([[  5364.99051,  38422.22028, -10923.25435,  35446.1303 ,
          -858.984  ,  10325.9377 ,  31904.70633,   3896.82932,
        -12560.83585,  29599.56183,  28665.86994, -44815.3644 ,
         48559.2683 ,  37184.85332, -30313.60702,   9709.31394,
          9718.03056, -12493.37933,   3392.91717,  23958.0602 ,
         19563.31532],
       [ 13531.96893,  -5139.29811,  32354.33063, -27853.16773,
        -44825.165  , -16643.67839,  -2880.86854,  42545.55648,
        -51394.84482, -30060.57781,  -9170.93692,  44487.83686,
        -37139.58191,  19280.63521,  40905.35507,  -9265.8577 ,
        -35556.67857, -22513.34943, -25921.42495, -19574.1461 ,
```

```
        -23971.2915 ],
       [-40730.57093, -34712.44863, -22597.97151, -43063.01155,
        -30992.24051, -36254.15373,   9707.03461, -10288.19612,
          1054.2411 , -32319.18406,  18003.3523 , -42117.80536,
         -9832.11708,  -8430.87892,   6071.1254 ,  43223.38662,
         15212.80469, -39302.27905,  40341.3013 , -48793.80385,
          -371.96934],
       [ 14816.35191, -39178.25777, -24568.2481 , -19624.49299,
        -21374.05467, -25541.88029, -12647.35314, -49323.34987,
         18013.92142, -25660.4997 ,  46607.39462,  20733.49367,
         49489.35534,  46996.66328,  45849.65176,  45189.46644,
         19055.486  , -14237.07279,  18186.92005, -14430.17366,
         13900.05528],
       [ 20572.53994, -51723.63587,  -9784.61354, -18859.97812,
         10487.19025,  21848.56175,  49811.29899, -19761.4122 ,
        -36658.56387,   7218.19133, -22418.96773,   8225.30182,
         14949.6515 ,  -5150.13807, -30716.35186,  12852.32345,
         45442.98334,  44789.80233, -12689.34215, -18737.5211 ,
         23913.19855]])
```

Как мы видим:

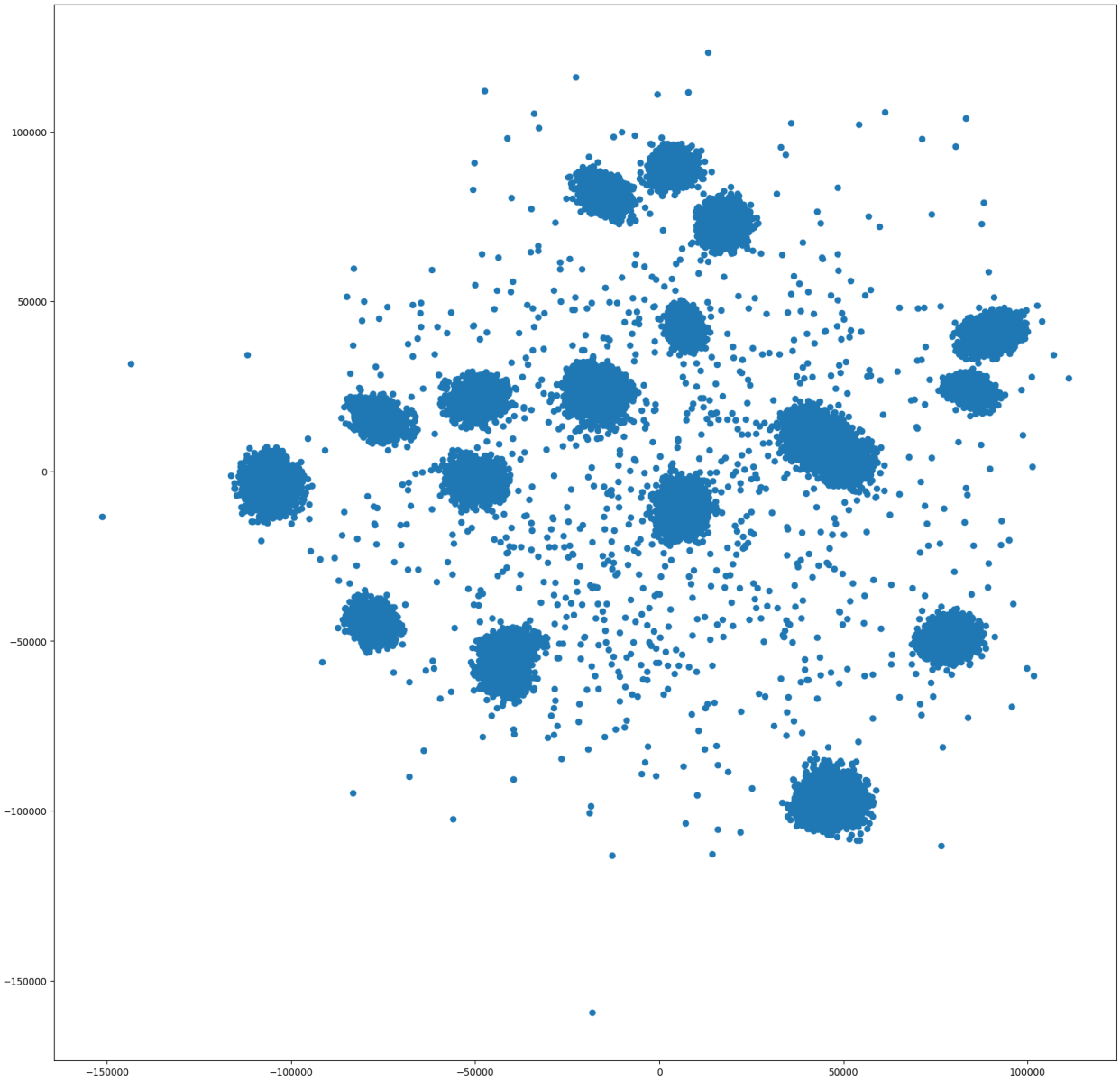
- датасет достаточно большой, с ним будет интересно работать
- данные без пропусков, все признаки непрерывные, из одного диапазона

Вывод: скорее всего датасет синтетический

Посмотрим на диаграмму рассеяния. Здесь нам нужно взять только два признака для отрисовки, поэтому применяем метод главных компонент и смотрим на распределение

```
pca = PCA(n_components=2)
d2 = pca.fit_transform(data)
d2.shape
plt.figure(figsize=(20,20))
plt.scatter(d2[:, 0], d2[:, 1])
```

```
<matplotlib.collections.PathCollection at 0x1f099046f90>
```



Вывод: видны 17 плотных скоплений объектов, которые можно назвать кластерами. Оставшиеся объекты будем считать выбросами

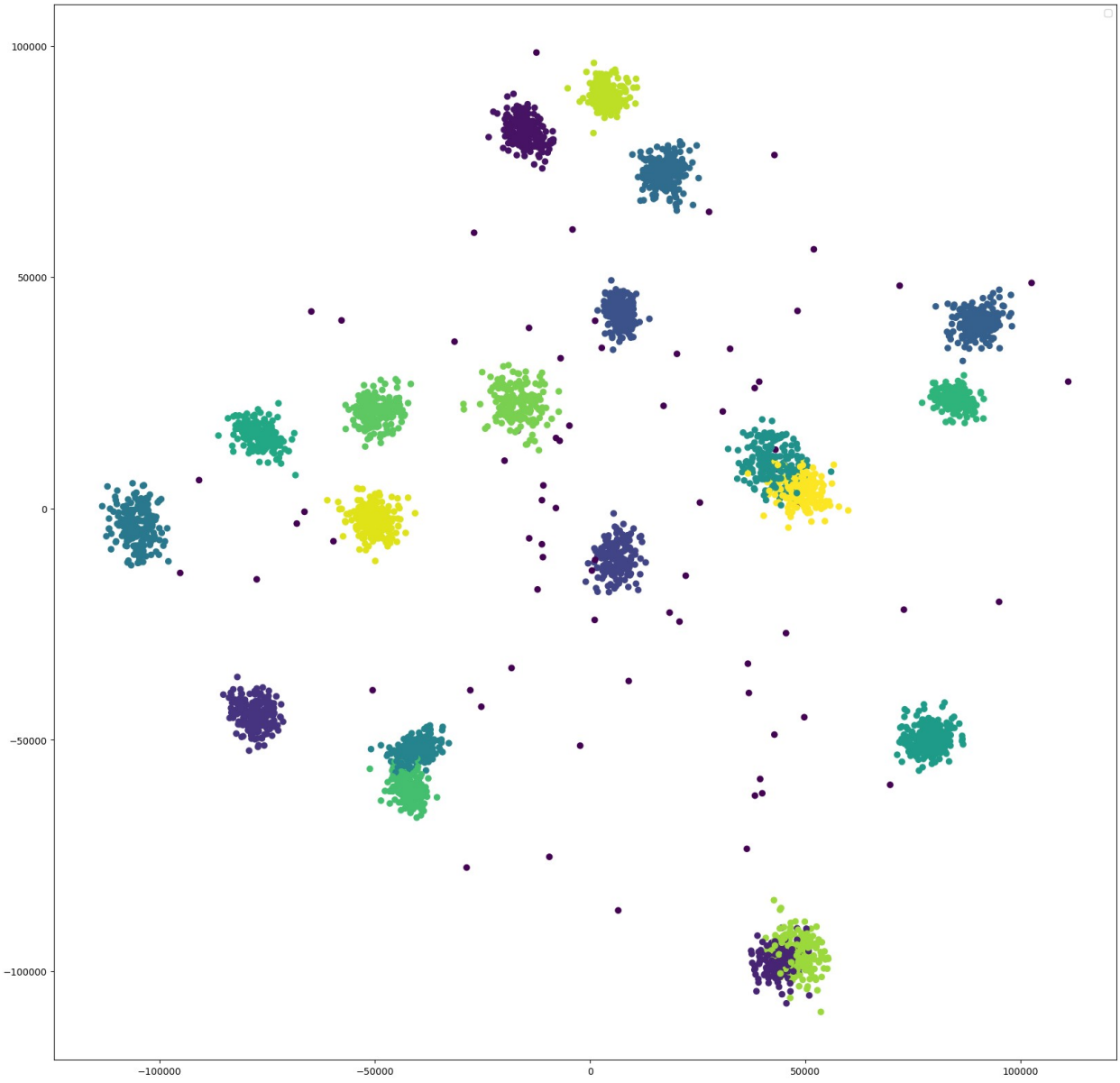
```
class DBScan:
    def __init__(self, esp, min_pts, metric):
        self.esp = esp
        self.min_pts = min_pts
        self.metric = metric

    def fit_predict(self, X):
        # построим матрицу расстояний distance от каждого объекта до
каждого
        n = X.shape[0]
        self.distance = np.zeros((n, n))
```

```
        for i in range(n-1):
            for j in range(i+1, n):
                value = self.metric(X[i, :], X[j, :])
                self.distance[i][j] = value
                self.distance[j][i] = value

        # определение корневых точек
        root = [sum(self.distance[i, :] < self.esp) >= self.min_pts
for i in range(n)]
        # определение граничных точек
        border = [(not root[i]) and (True in [root[j] and
(self.distance[i, j] < self.esp) for j in range(n)]) for i in
range(n)]
        # определение выбросов
        noise = [not (root[i] or border[i]) for i in range(n)]
        #return root, border, noise

        # Создаем метки классов
        labels = -1 * np.ones(n, dtype=int)  # Изначально все метки -1
(шум)
        cluster_id = 0

        # Присваиваем метки классам
        for i in range(n):
            if root[i] and labels[i] == -1:  # Если это корневая точка
и она еще не имеет метки
                labels[i] = cluster_id  # Присваиваем метку кластеру
                self._expand_cluster(i, labels, cluster_id)  #
Расширяем кластер
                cluster_id += 1  # Увеличиваем идентификатор кластера
для следующего кластера

        return labels

    def _expand_cluster(self, point_idx, labels, cluster_id):
        # Получаем индексы соседей для текущей точки
        neighbors = [j for j in range(len(labels)) if
self.distance[point_idx, j] < self.esp]

        # Присваиваем метки всем соседям
        for neighbor_idx in neighbors:
            if labels[neighbor_idx] == -1:  # Если сосед еще не
помечен
                labels[neighbor_idx] = cluster_id  # Присваиваем метку
текущему кластеру
                self._expand_cluster(neighbor_idx, labels, cluster_id)
# Рекурсивно расширяем кластер

model = DBScan(esp=100000, min_pts=5, metric=lambda x1, x2: sum((x1-
x2)**2)**0.5)
```

```
labels = model.fit_predict(data[:4000])
plt.figure(figsize=(20,20))
plt.scatter(d2[:4000, 0], d2[:4000, 1], c=labels)
plt.legend()
```

```
C:\Users\Sai\AppData\Local\Temp\ipykernel_612\3335116140.py:15:
UserWarning: No artists with labels found to put in legend.  Note that
artists whose label start with an underscore are ignored when legend()
is called with no argument.
  plt.legend()
```

```
<matplotlib.legend.Legend at 0x1f0999dce10>
```



т.к. асимптотика алгоритма o(n^3) мы не успели посчитать весь датасет (50000 объектов будут считаться двое суток). Поэтому мы кластеризовали первые 4000 объектов из 50000. Вывод: оказалось, объекты которые мы считали кластером, на самом деле могут относиться к двум разным кластером. Так произошло из-за того, что мы изначально смотрели только на одну проекцию, а в реальном 20-мерном прастрансве кластеров больше, чем мы предположили. Выбросы определены верно - задача выполнена.