МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Основы машинного обучения»

Тема: Кластеризация

Вариант 145Б

Студент гр. 1303 Самохин К. А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Изучить способы кластеризации наборов данных при помощи ЯП Python и библиотек Pandas, Seabort, Matplotlib, NumPy, Scikit-learn. При помощи перечисленных библиотек провести кластеризацию предоставленных датафреймов.

Задание.

1. Подготовка наборов данных:
   1.1. Загрузите наборы.
   1.2. Проверьте корректность загрузки.
   1.3. Постройте диаграмму рассеяния набора данных. Опишите форму данных.
   1.4. Подготовьте наборы данных проведя стандартизация или нормировку данных. Обоснуйте выбор операции.
2. K-Means:
   2.1. Проведите исследование оптимального количества кластеров методов локтя. Сделайте выводы, о наиболее подходящем количестве кластеров.
   2.2. Проведите исследование оптимального количества кластеров методом силуэта. Сделайте выводы, о наиболее подходящем количестве кластеров.
   2.3. Проведите кластеризацию алгоритмом K-means, с выбранным оптимальным количеством кластеров.
   2.4. Постройте диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров.
   2.5. Постройте диаграмму Вороного для результатов кластеризации. На диаграмме отметьте центроиды полученных кластеров.
   2.6. Постройте для каждого признака диаграмму “box-plot” или “violin-plot”, с разделением по кластерам. Сделайте выводы о разделении кластеров и успешности применения кластеризации K-means к набору данных.
   2.7. Рассчитайте для каждого кластера кол-во точек, среднее, СКО, минимум и максимум. Сопоставьте результаты с построенными графиками.
3. DBSCAN:
   3.1. Подберите параметры алгоритма DBSCAN, которые на ваш взгляд дают наилучшие результаты. Опишите процесс (почему и как изменяли параметры) подбора параметров.
   3.2. Постройте диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров.
   3.3. Сделайте выводы об успешности кластеризации.
4. Иерархическая кластеризация:

4.1. Проведите иерархическую кластеризацию при всех возможных параметрах linkage, используя количество кластеров, полученных в п.2 или п.3. Для каждого из результатов постройте дендрограмму. Сделайте выводы, о разделении кластеров и необходимости изменить количество кластеров (если считаете, что необходимо изменить количество кластеров, то повторите кластеризацию с другим количеством кластеров).

4.2. Постройте диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров. Используйте лучшие результаты, полученные для определенного параметра linkage.

4.3. Сравните результаты кластеризации с результатами, полученными в п.2 и п.3. Сделайте выводы о том, какой метод кластеризации подходит под каждый из наборов данных.

5. Изучение набора данных с большим количество признаков:

5.1. Для набора данных, отмеченного буквой вашего варианта, самостоятельно проведите кластеризацию. Метод выбираете самостоятельно, обосновав выбор. *Предварительно рекомендуется провести исследование и предобработку набор данных.*

5.2. Проведите анализ полученных кластеров индивидуально, и вместе. Можно использовать попарные диаграммы рассеяния, оценку плотности, построение box-plot и/или violin-plot, а также расчет характеристик кластера.

5.3. Сделайте выводы о смысловой нагрузке кластеров, какую содержательную информацию кластеры содержат.

Выполнение работы.

1. Подготовка наборов данных.

1.1. Загрузим данные из файлов, используя метод *read_csv* (см. листинги 1.1.1, 1.1.2, 1.1.3).

Листинг 1.1.1 – Считывание датасета из файла lab2_blobs.csv.

```
1  blobs = pd.read_csv('lab2_blobs.csv')
```

Листинг 1.1.2 – Считывание датасета из файла lab2_noisymoons.csv.

```
1  moons = pd.read_csv('lab2_noisymoons.csv')
```

Листинг 1.1.3 – Считывание датасета из файла lab2_luckyset.csv.

```
1  lucky = pd.read_csv('lab2_luckyset.csv')
```

1.2. Проверим корректность загруженных данных, вызвав у датафреймов метод *head* (см. листинги 1.2.1, 1.2.2, 1.2.3). Ко умолчанию данный метод выводит первые 5 строк. Вывод метода можно видеть в таблицах 1.2.1, 1.2.2, 1.2.3.

Листинг 1.2.1 – Вызов метода *head* для первого датафрейма.

```
1  blobs.head()
```

Таблица 1.2.1 – Вывод метода *head* для первого датафрейма.

	# x	y
0	-8.0267	-4.9731
1	-7.0422	-2.6454
2	8.9214	9.5679
3	1.0887	-0.2884
4	0.4739	-0.0737

Листинг 1.2.2 – Вызов метода *head* для второго датафрейма.

```
1  moons.head()
```

Таблица 1.2.2 – Вывод метода *head* для второго датафрейма.

	# x	y
0	-0.5237	0.8448
1	0.2002	0.9865
2	-0.4794	0.8188
3	0.5155	0.9515
4	1.8891	0.1536

Листинг 1.2.3 – Вызов метода *head* для третьего датафрейма.

```
1  lucky.head()
```

Таблица 1.2.3 – Вывод метода *head* для второго датафрейма.

	# x	y
0	-0.4743	0.3005
1	-0.6205	6.2994
2	2.5470	0.5894
3	-0.0678	4.9441
4	-0.0254	0.3081

1.3. Построим диаграммы рассеивания для каждого набора данных (см. рисунки 1.3.1, 1.3.2, 1.3.3). Рисование диаграмм будет осуществляться с использованием метода *scatterplot* библиотеки Seaborn (см. листинги 1.3.1, 1.3.2, 1.3.3).

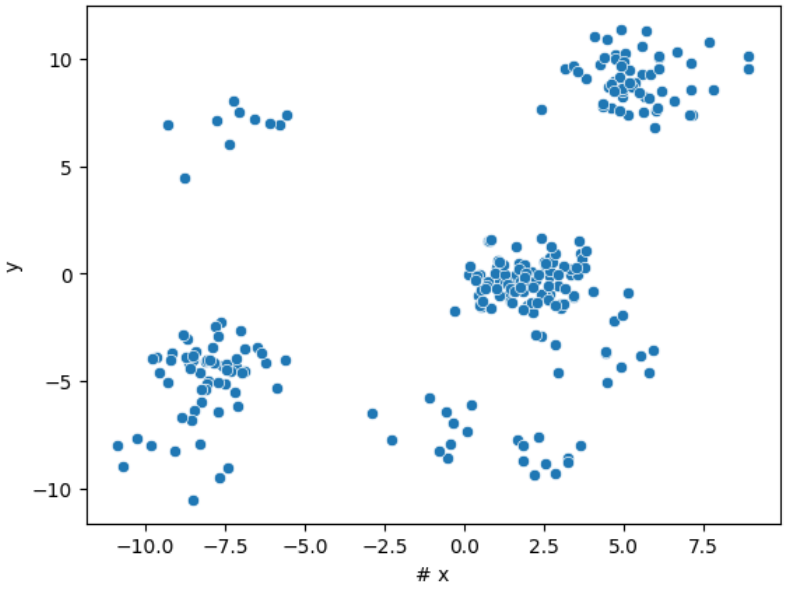

Рисунок 1.3.1 – Диаграмма рассеивания для первого датасета.

Листинг 1.3.1 – Рисование диаграммы рассеивания для первого датасета.

```
1  sb.scatterplot(blobs, x = "# x", y = "y")
```

Из рисунка 1.3.1 видно, что все наблюдения разделились на 3 наиболее плотных скопления, и также есть 2 гораздо менее плотных группы. У всех скоплений достаточно много выбросов. По характеру распределения точек можно заметить, что значения по обоим координатным осям отклоняются от нуля в отрицательную и положительную сторону на примерно одинаковую величину.

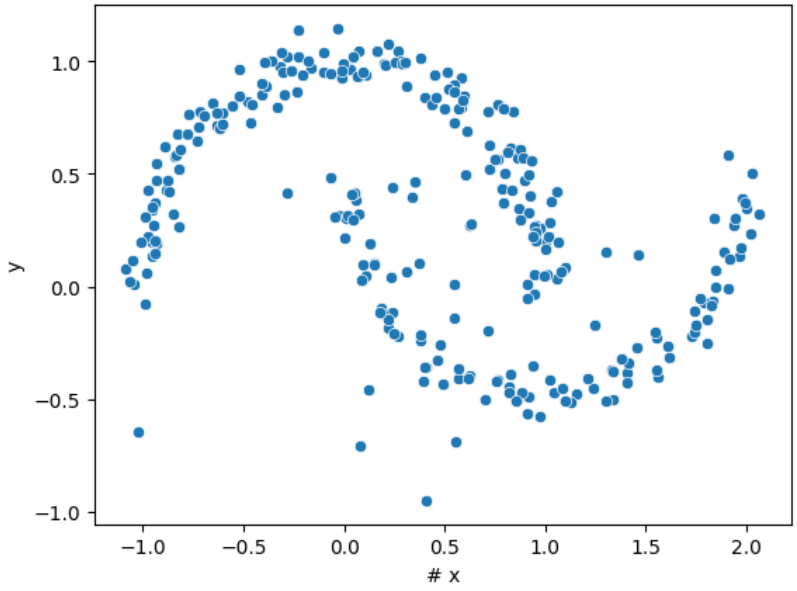

Рисунок 1.3.2 – Диаграмма рассеивания для второго датасета.

Листинг 1.3.2 – Рисование диаграммы рассеивания для второго датасета.

```
1  sb.scatterplot(moons, x = "# x", y = "y")
```

На рисунке 1.3.2 можно видеть, что все точки разделились на 2 наиболее крупных скопления, причём верхняя группа содержит в себе больше точек, чем нижняя, а из-за их схожего размера можно сделать вывод, что и плотность верхней группы выше. Также можно видеть выбросы, большая часть которых находится в центре диаграммы, между двумя скоплениями.

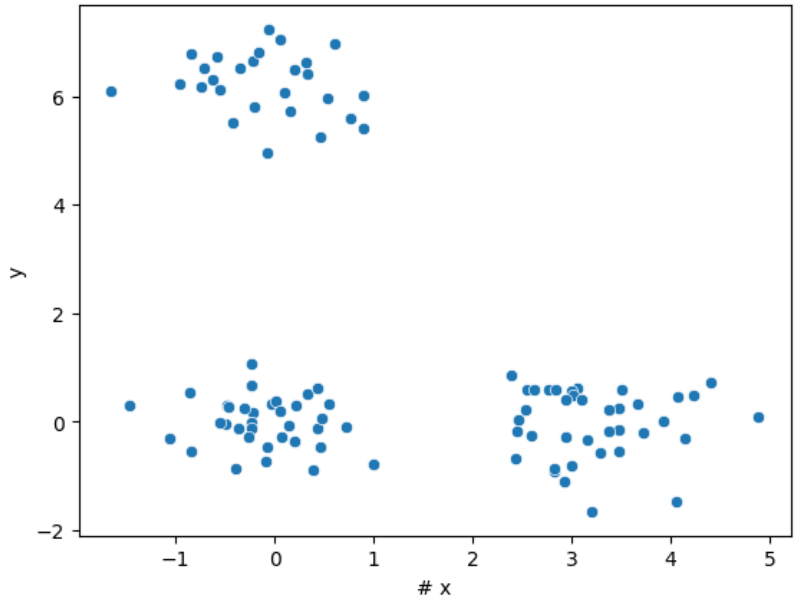

Рисунок 1.3.3 – Диаграмма рассеивания для третьего датасета.

Листинг 1.3.3 – Рисование диаграммы рассеивания для третьего датасета.

```
1  sb.scatterplot(lucky, x = "# x", y = "y")
```

На рисунке 1.3.3 несложно видеть, что наблюдения распределились по трём скоплениям. У всех трёх скоплений имеются выбросы, но в целом точки находятся в относительной близости к «своим» группам. Также можно заметить, что большая часть точек находится в положительной части плоскости (первая четверть координатной плоскости).

1.4. Подготовим наборы данных, проведя стандартизацию или нормировку данных. Прежде всего, нужно понять, какой метод подойдёт для конкретного датасета. Для этого построим гистограммы распределения с добавленной на них ядерной оценкой плотности (см. рисунки 1.4.1, 1.4.2, 1.4.3). Построение будем выполнять, используя метод *histplot* из библиотеки Seaborn (см. листинги 1.4.1, 1.4.2, 1.4.3).

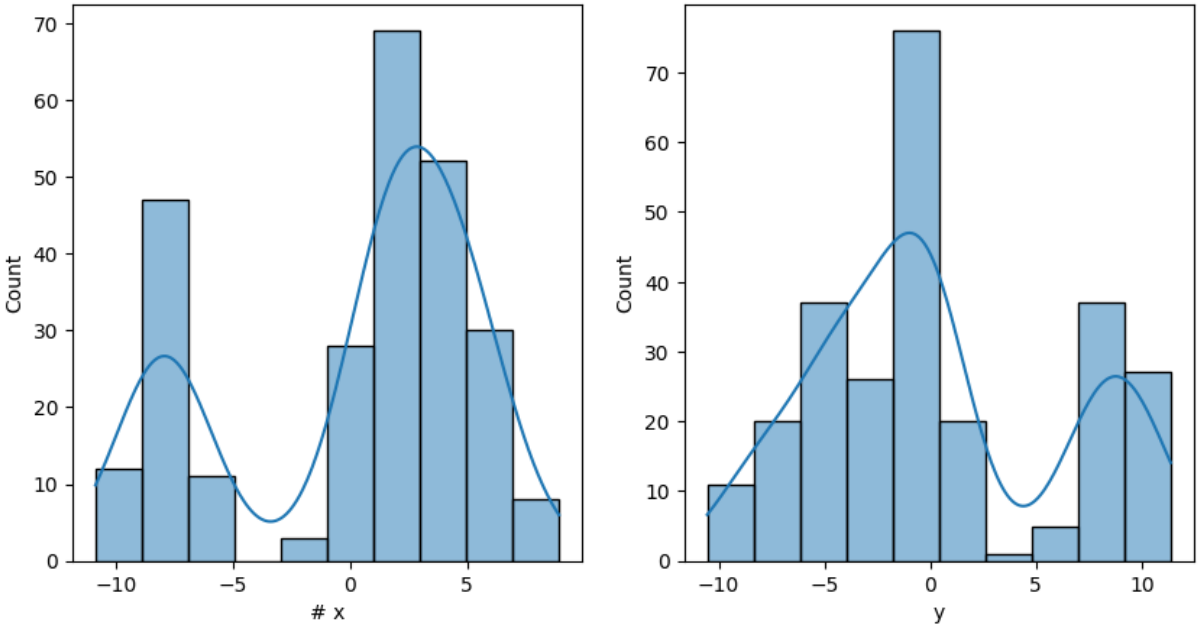

Рисунок 1.4.1 – Гистограмма распределения для первого датасета.

Листинг 1.4.1 – Построение гистограммы распределения для первого датасета.

```
1  fig, axs = plt.subplots(1,2)
2  sb.histplot(blobs, x='# x', kde=True, ax=axs[0])
3  sb.histplot(blobs, x='y', kde=True, ax=axs[1])
```

Из рисунка 1.4.1 видно, что разность между минимальным и максимальным значением не слишком велика, к тому же относительно нуля значения распределены схоже. На гистограмме также отчётливо видны два пика для обоих признаков, что сильно отличает распределение от нормального. Исходя из приведённых наблюдений было принято решение использовать *MaxAbsScaler* при нормировке.

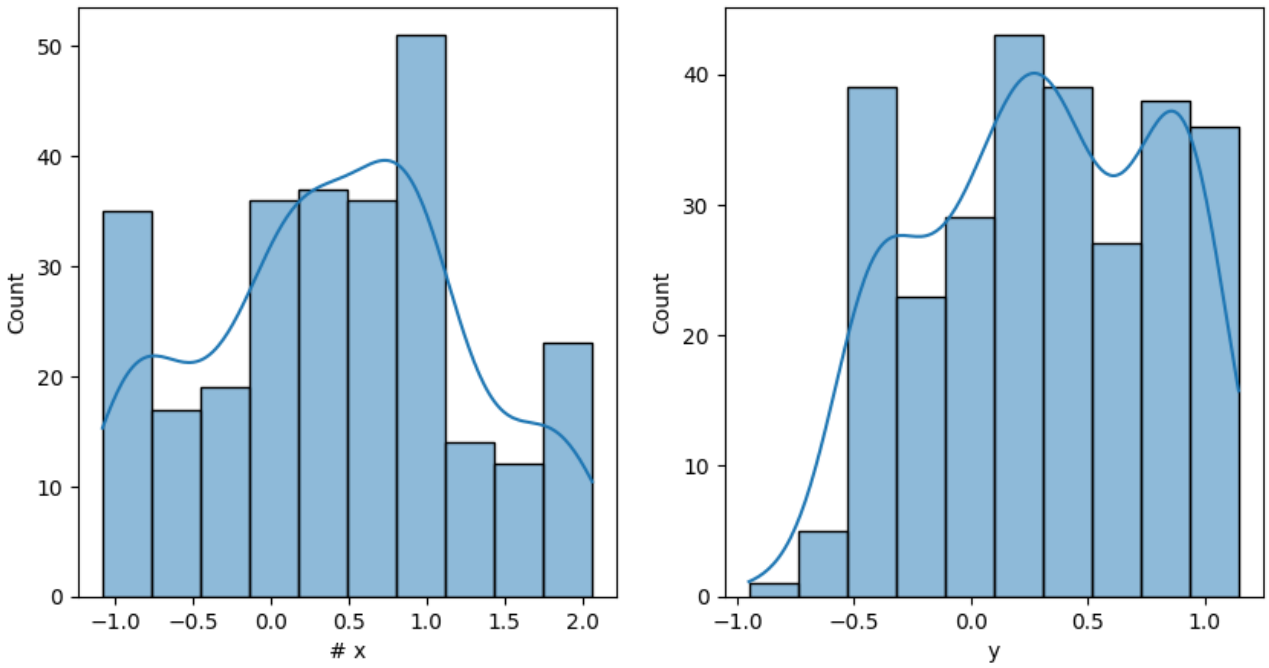

Рисунок 1.4.2 – Гистограмма распределения для второго датасета.

Листинг 1.4.2 – Построение гистограммы распределения для второго датасета.

```
1  fig, axs = plt.subplots(1,2)
2  sb.histplot(moons, x='# x', kde=True, ax=axs[0])
3  sb.histplot(moons, x='y', kde=True, ax=axs[1])
```

На рисунке 1.4.2 видно, что диаграмма ядерной оценки плотности имеет ярко выраженный пик и отдалённо напоминает нормальное распределение. На основании этого было решено использовать *StandardScaler* для обработки второго датасета.

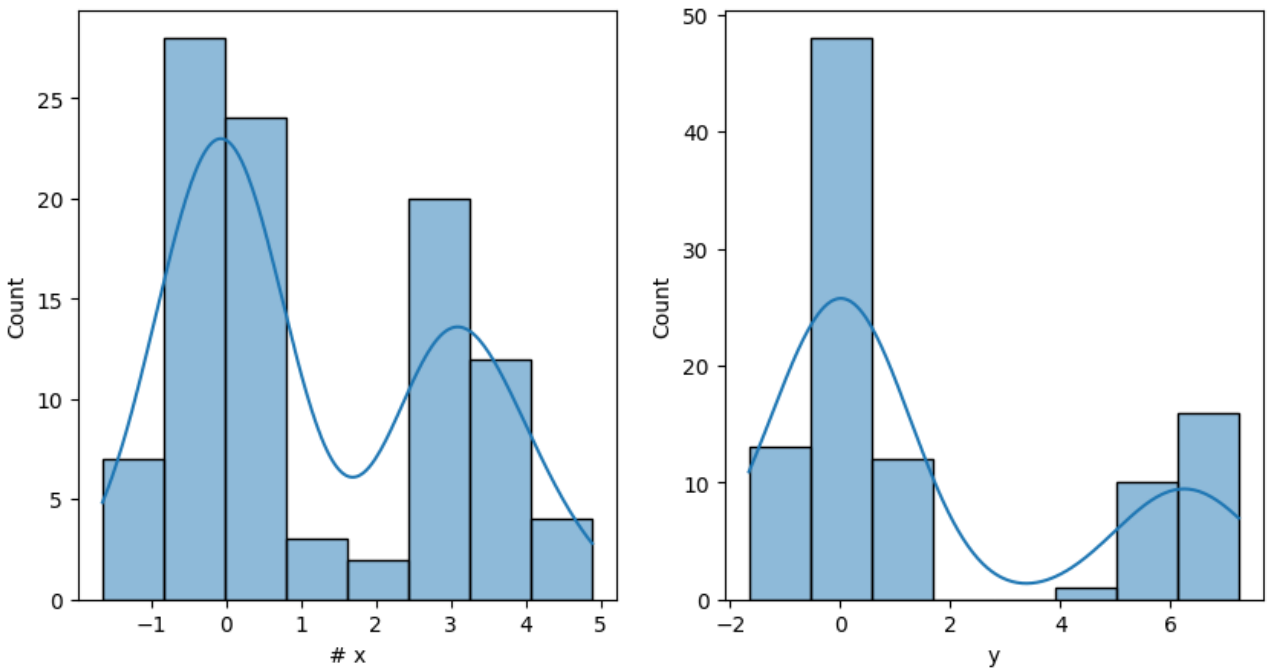

Рисунок 1.4.3 – Гистограмма распределения для третьего датасета.

Листинг 1.4.3 – Построение гистограммы распределения для третьего датасета.

```
1  fig, axs = plt.subplots(1,2)
2  sb.histplot(lucky, x='# x', kde=True, ax=axs[0])
3  sb.histplot(lucky, x='y', kde=True, ax=axs[1])
```

На рисунке 1.4.3 видно два выраженных пика на обоих графиках. Так как распределение не походе на нормальное и разность между максимальным и минимальным значениями мала, для обработки третьего датасета будет использован *MaxAbsScaler*.

Код преобразования датасетов представлен в листингах 1.4.4, 1.4.5, 1.4.6.

Листинг 1.4.4 – Нормировка первого датасета.

```
1  mblobs = mascaler.fit_transform(blobs)
2  mblobs = pd.DataFrame(mblobs, columns=["# x", "y"])
```

Листинг 1.4.5 – Стандартизация второго датасета.

```
1  mmoons = sscaler.fit_transform(moons)
2  mmoons = pd.DataFrame(mmoons, columns=["# x", "y"])
```

Листинг 1.4.6 – Нормировка третьего датасета.

```
mlucky = mascaler.fit_transform(lucky)
mlucky = pd.DataFrame(mlucky, columns=["# x", "y"])
```

2. K-Means.

2.1. Проведём исследование оптимального количества кластеров методом локтя. Для этого была написана функция, строящая график зависимости инерции от количества кластеров, принимающая на вход датафрейм (см. листинг 2.1.1).

Листинг 2.1.1 – Функция *elbow_method()*.

```
def elbow_method(data):
    inert_list = []
    for i  in range(10):
        temp_km = KMeans(i+1, n_init = 5)
        temp_km.fit(data)
        inert_list.append(temp_km.inertia_)

    plt.plot(list(range(1,11)),inert_list)
    plt.ylabel('Инерция')
    plt.xlabel('Количество кластеров')
    plt.xticks(list(range(1,11)))
    plt.grid()
    plt.show()
```

Далее представленная функция была применена к изменённым датафреймам, которые мы получили в пункте 1.4. Построенные графики можно видеть на рисунках 2.1.1, 2.1.2, 2.1.3.

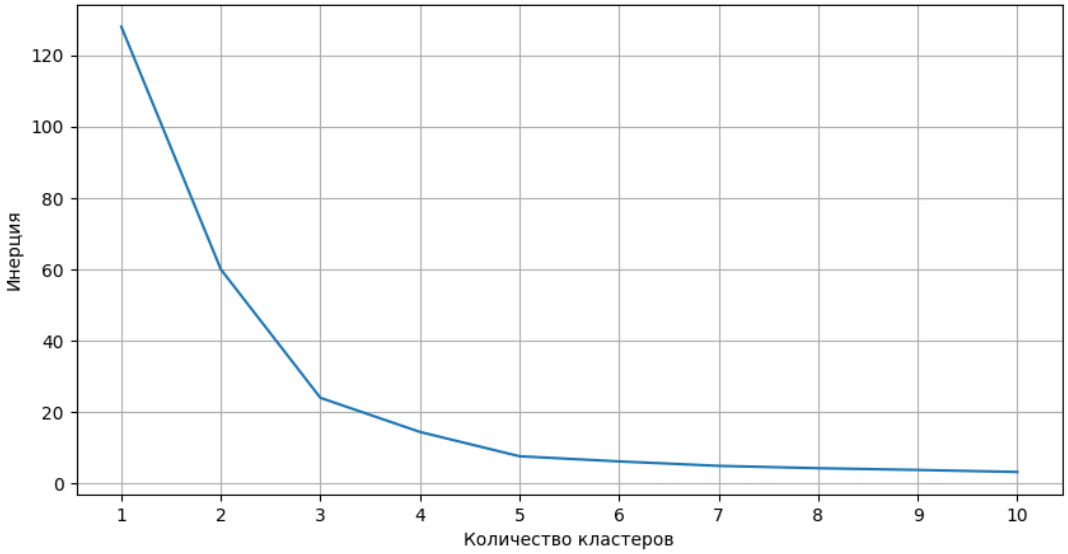

Рисунок 2.1.1 – Зависимость инерции от числа кластеров для первого датафрейма.

Видно, что на графике 2.1.1 «локтем» является промежуток 3-5, и после неё скорость убывания графика уменьшается. Оптимальное число кластеров для первого датафрейма возьмём равным 3.

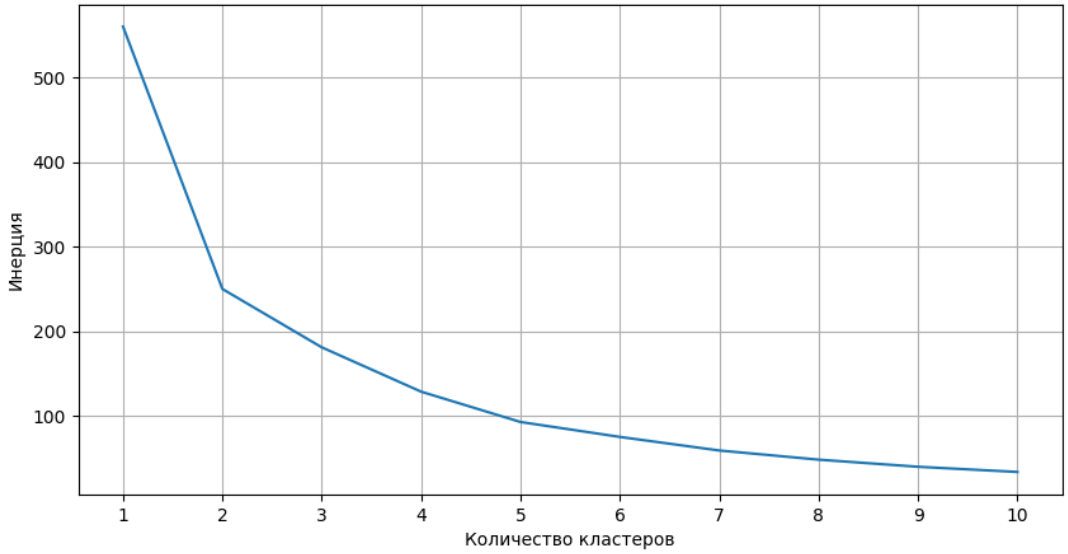

Рисунок 2.1.2 - Зависимость инерции от числа кластеров для второго датафрейма.

На рисунке 2.1.2 не вооруженным взглядом видно, что «локтем» является промежуток 7-8. «Локтем» также можно было считать и промежуток 2-3, но значение инерции в ней слишком велико. Поэтому число кластеров будет равно 7.

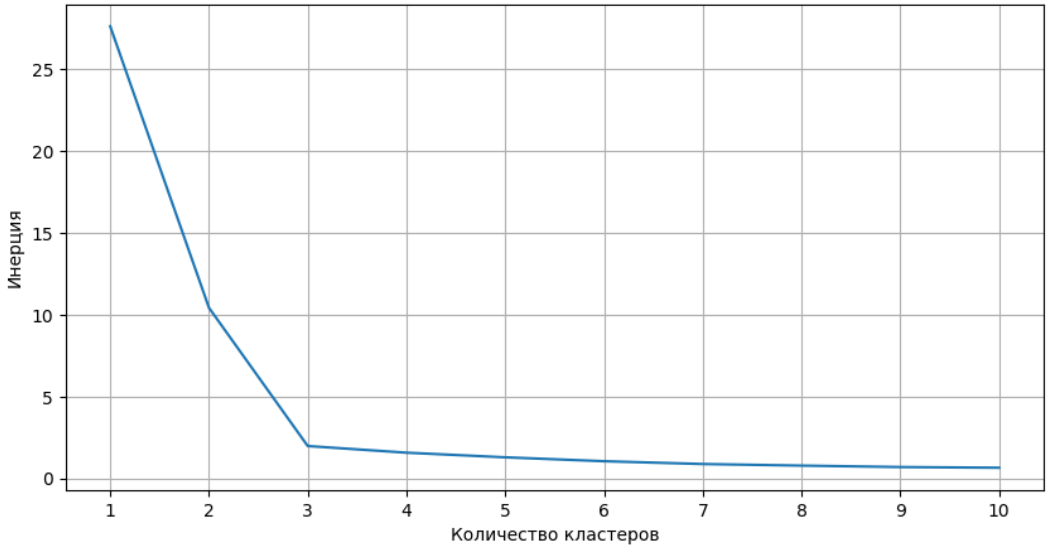

Рисунок 2.1.3 - Зависимость инерции от числа кластеров для третьего датафрейма.

На рисунке 2.1.3 видно, что «локоть» находится в точке 3, и после неё скорость убывания графика резко падает. Для третьего датафрейма оптимальное число кластеров будет равно 3.

2.2. Проведём исследование оптимального количества кластеров методом силуэта. Для этого напишем функцию, строящую график зависимости среднего значения коэффициента силуэта от количества кластеров, принимающая на вход датафрейм (см. листинг 2.2.1).

Листинг 2.2.1 – Функция *silhouette_method()*.

```
def silhouette_method(data):
    sil_list = []
    for i in range(1,10):
        temp_km = KMeans(i+1, n_init = 5)
        temp_clust = temp_km.fit_predict(data)
        sil_list.append(silhouette_score(data, temp_clust))

    plt.plot(list(range(2,11)),sil_list)
    plt.ylabel('Среднее значение коэффициента силуэта')
    plt.xlabel('Количество кластеров')
    plt.xticks(list(range(1,11)))
    plt.grid()
    plt.show()
```

Теперь применим функцию *silhouette_method()* к трём изменённым датафреймам, полученным в пункте 1.4. Полученные графики см. на рисунках 2.2.1, 2.2.2, 2.2.3.

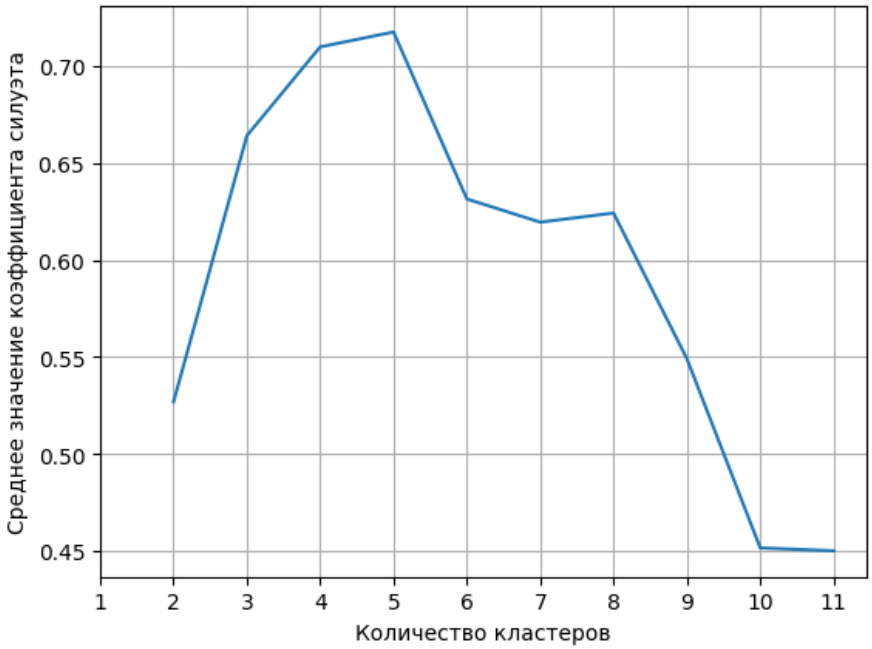

Рисунок 2.2.1 – Зависимость среднего значения коэффициента силуэта от числа кластеров для первого датафрейма.

На рисунке 2.2.1 коэффициент силуэта принимает максимальное значение в точке 5, это и будет число кластеров.

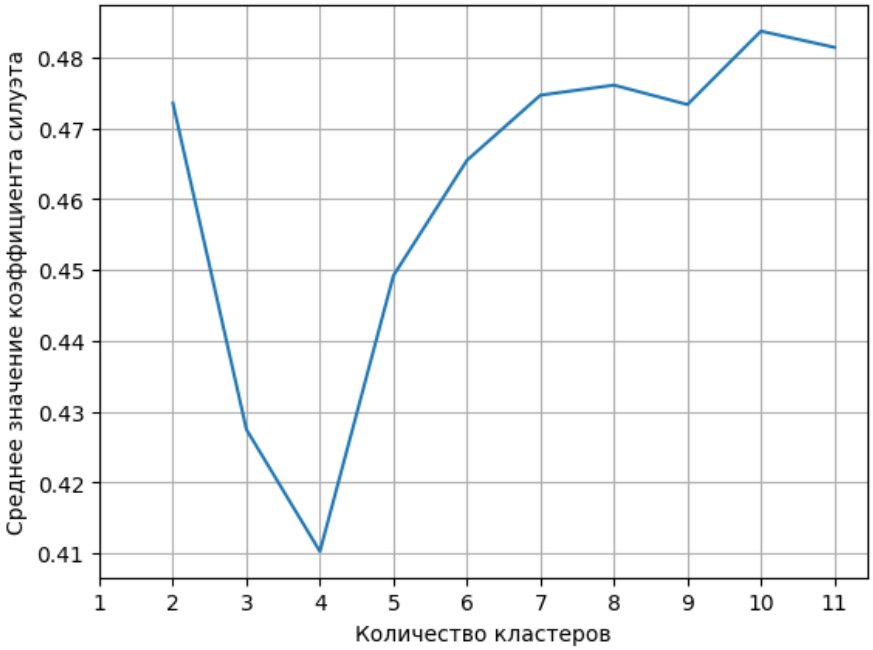

Рисунок 2.2.2 – Зависимость среднего значения коэффициента силуэта от числа кластеров для второго датафрейма.

Коэффициент силуэта на рисунке 2.2.2 принял максимальное значение в точке 10. Следовательно, число кластеров у второго датафрейма равно 10.

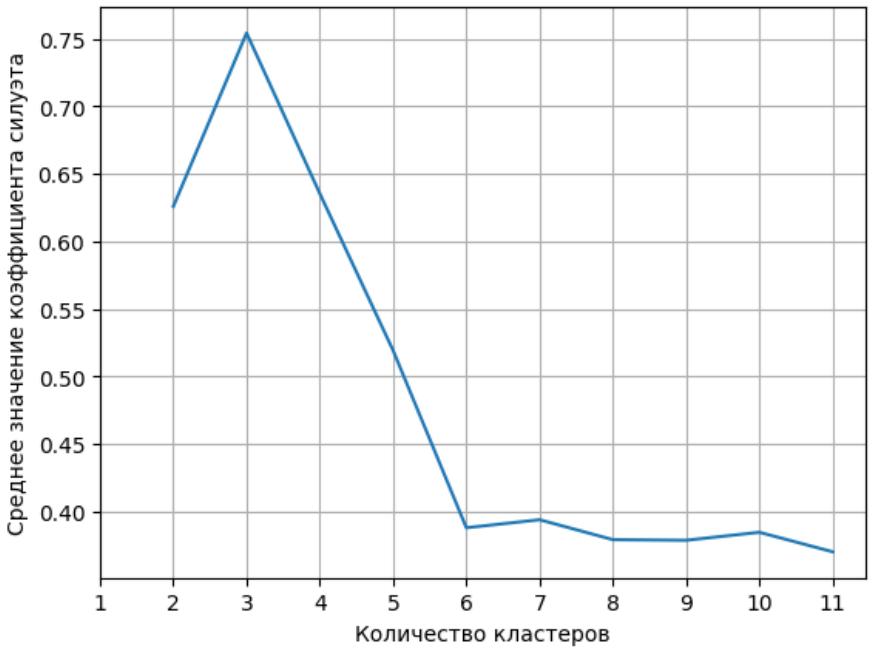

Рисунок 2.2.3 – Зависимость среднего значения коэффициента силуэта от числа кластеров для третьего датафрейма.

Значение коэффициента силуэта на рисунке 2.2.3 достигает своего максимума в точке 3, поэтому число кластеров в третьем датафрейме равно 3.

2.3. Проведём кластеризацию алгоритмом K-means, с выбранным числом кластеров (см. листинги 2.3.1, 2.3.3, 2.3.5). Вывод результатов кластеризации можно видеть в листингах 2.3.2, 2.3.4, 2.3.6 (вывод представлен вычисленной инерцией и координатами центроид).

Листинг 2.3.1 – Кластеризация первого набора алгоритмом K-means.

```
1  blobs_km5 = KMeans(n_clusters = 5)
2  blobs_clust5 = blobs_km5.fit_predict(mblobs)
3  blobs_cent5 = blobs_km5.cluster_centers_
4  print("Инерция: ", blobs_km5.inertia_)
5  blobs_cent5[:3]
```

Листинг 2.3.2 – Результаты кластеризации первого набора.

```
Инерция:  7.612980497383202
```

```
array([[-0.74445721, -0.44238897],
       [ 0.20715623, -0.05209791],
       [ 0.49527277,  0.79623692]])
```

Листинг 2.3.3 – Кластеризация второго набора алгоритмом K-means.

```
1  moons_km10 = KMeans(n_clusters = 10)
2  moons_clust10 = moons_km11.fit_predict(mmoons)
3  moons_cent10 = moons_km11.cluster_centers_
4  print("Инерция: ", moons_km11.inertia_)
5  moons_cent11[:3]
```

Листинг 2.3.4 – Результаты кластеризации второго набора.

```
Инерция:  33.93445355108556
array([[-0.12313202, -1.22126912],
       [-0.53002822,  1.39451517],
       [-0.38501534, -0.09659808]])
```

Листинг 2.3.5 – Кластеризация третьего набора алгоритмом K-means.

```
1  lucky_km3 = KMeans(n_clusters = 3)
2  lucky_clust3 = lucky_km3.fit_predict(mlucky)
3  lucky_cent3 = lucky_km3.cluster_centers_
4  print("Инерция: ", lucky_km3.inertia_)
5  lucky_cent3[:3]
```

Листинг 2.3.6 – Результаты кластеризации третьего набора.

```
Инерция:  1.97131818165591
array([[ 0.66111681, -0.00591853],
       [-0.02119034,  0.86006131],
       [-0.01996088, -0.00188256]])
```

2.4. Построим диаграммы рассеяния результатов кластеризации с выделением разным цветом разных кластеров (см. листинги 2.4.1, 2.4.2, 2.4.3). Диаграммы рассеяния представлены на рисунках 2.4.1, 2.4.2, 2.4.3.

Листинг 2.4.1 – Рисование результатов кластеризации первого набора.

```
1  kmblobs = mblobs.copy()
2  kmblobs['cluster'] = blobs_clust5
3  sb.scatterplot(kmblobs, x = '# x', y = 'y', hue = 'cluster', palette
= 'tab10')
```

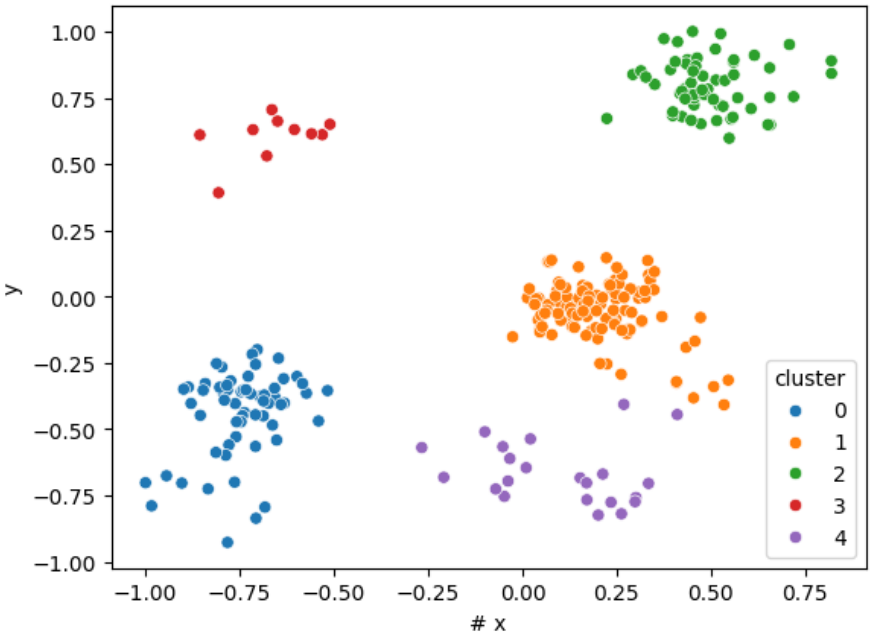

Рисунок 2.4.1 – Диаграмма рассеяния результатов кластеризации первого набора.

Листинг 2.4.2 – Рисование результатов кластеризации второго набора.

```
1  kmmoons = mmoons.copy()
2  kmmoons['cluster'] = moons_clust11
3  sb.scatterplot(kmmoons, x = '# x', y = 'y', hue = 'cluster', palette
= 'tab10')
```

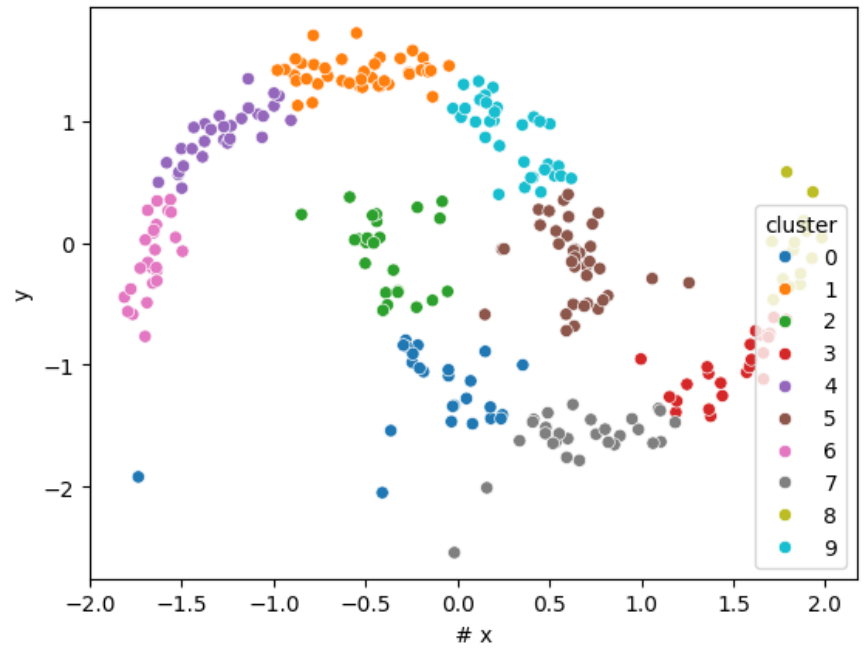

Рисунок 2.4.2 – Диаграмма рассеяния результатов кластеризации второго набора.

Листинг 2.4.3 – Рисование результатов кластеризации третьего набора.

```
1  kmlucky = mlucky.copy()
2  kmlucky['cluster'] = lucky_clust3
3  sb.scatterplot(kmlucky, x = '# x', y = 'y', hue = 'cluster', palette
= 'tab10')
```

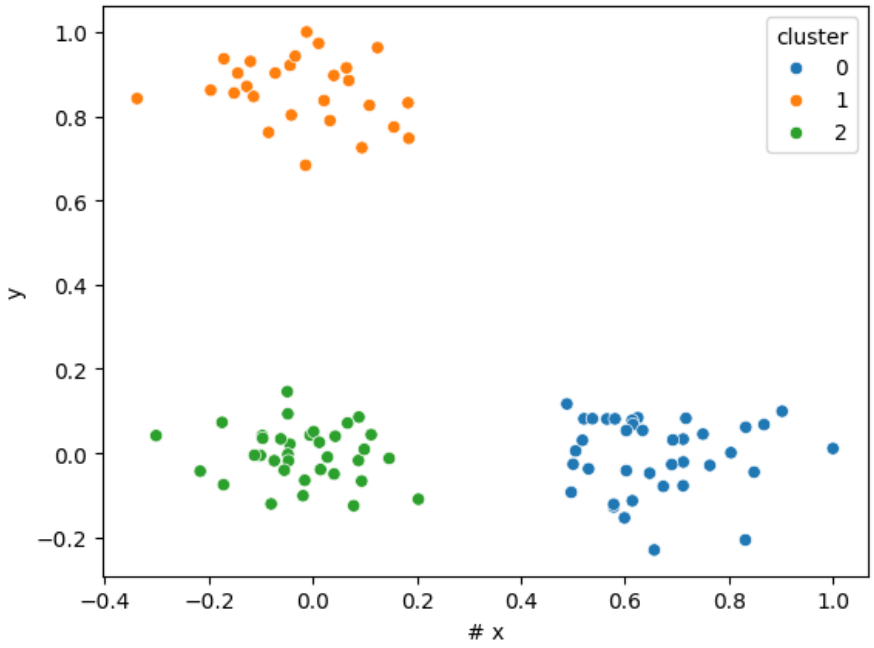

Рисунок 2.4.3 – Диаграмма рассеяния результатов кластеризации третьего набора.

2.5. Построим диаграмму Вороного для результатов кластеризации (см. листинги 2.5.1, 2.5.2, 2.5.3). Полученные диаграммы представлены на рисунках 2.5.1, 2.5.2, 2.5.3.

Листинг 2.5.1 – Построение диаграммы Вороного для первого датасета.

```
1  h = 0.02
2  x_min, x_max = kmblobs["# x"].min() - 0.5, kmblobs["# x"].max() + 0.5
3  y_min, y_max = kmblobs["y"].min() - 0.5, kmblobs["y"].max() + 0.5
4   xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
5  z_clust = blobs_km5.predict(np.c_[xx.ravel(), yy.ravel()])
6  z_clust = z_clust.reshape(xx.shape)
```

```
7     plt.imshow(z_clust,  interpolation="nearest",  extent=(xx.min(),
xx.max(),  yy.min(),  yy.max()),  cmap=plt.cm.Paired,  aspect="auto",
origin="lower") # рисуем области
8  plt.plot(kmblobs["#  x"],  kmblobs["y"],  "k.",  markersize=4)  #рисуем
точки
9   plt.scatter(_cent5[:,  0],  blobs_cent5[:,  1],  marker="x",  s=169,
linewidths=3,     color="w", =10)
10 plt.show()
```

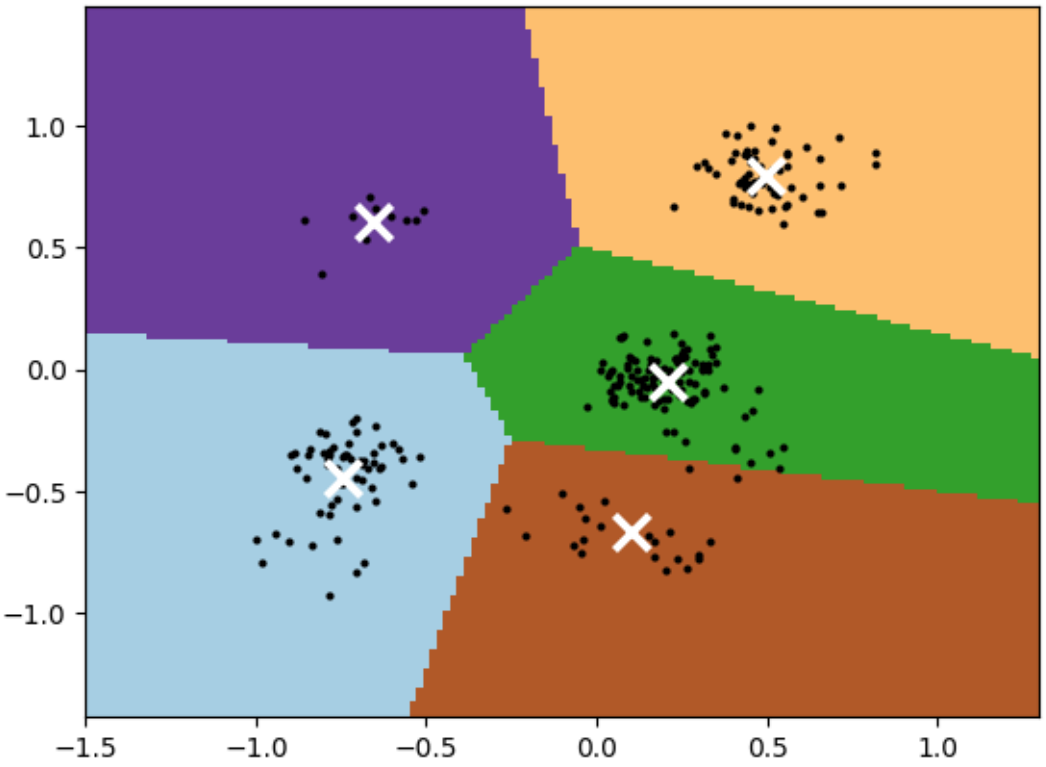

Рисунок 2.5.1 – Диаграмма Вороного для первого датафрейма.

Листинг 2.5.1 – Построение диаграммы Вороного для второго датасета.

```
1   h = 0.02
2   x_min, x_max = kmmoons["# x"].min() - 0.5, kmmoons["# x"].max() + 0.5
3   y_min, y_max = kmmoons["y"].min() - 0.5, kmmoons["y"].max() + 0.5
4  xx,  yy  =  np.meshgrid(np.arange(x_min,  x_max,  h),  np.arange(y_min,
y_max, h))
5   z_clust = moons_km10.predict(np.c_[xx.ravel(), yy.ravel()])
6   z_clust = z_clust.reshape(xx.shape)
7    plt.imshow(z_clust,    interpolation="nearest",    extent=(xx.min(),
xx.max(),  yy.min(),  yy.max()),  cmap=plt.cm.Paired,  aspect="auto",
origin="lower") # рисуем области
```

```
8  plt.plot(kmmoons["#  x"],  kmmoons["y"],  "k.",  markersize=4)  #рисуем
точки
9    plt.scatter(moons_cent11[:,  0],  moons_cent10[:,  1],    marker="x",
s=169, linewidths=3, color="w", zorder=10)
10 plt.show()
```

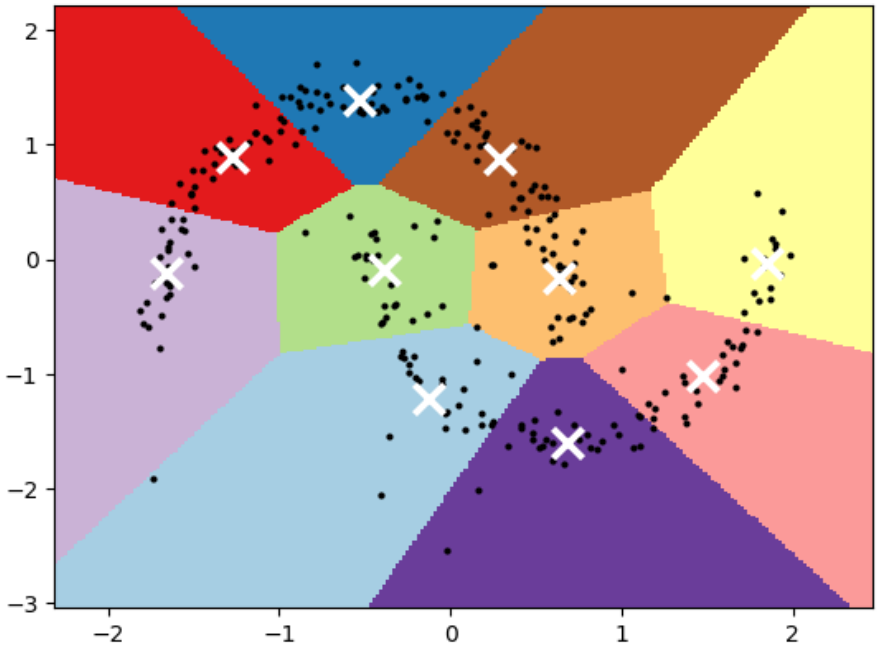

Рисунок 2.5.1 – Диаграмма Вороного для второго датафрейма.

Листинг 2.5.1 – Построение диаграммы Вороного для третьего датасета.

```
1  h = 0.02
2  x_min, x_max = kmlucky["# x"].min() - 0.5, kmlucky["# x"].max() + 0.5
3  y_min, y_max = kmlucky["y"].min() - 0.5, kmlucky["y"].max() + 0.5
4  xx,  yy  =  np.meshgrid(np.arange(x_min,  x_max,  h),  np.arange(y_min,
y_max, h))
5  z_clust = lucky_km3.predict(np.c_[xx.ravel(), yy.ravel()])
6  z_clust = z_clust.reshape(xx.shape)
7   plt.imshow(z_clust,   interpolation="nearest",   extent=(xx.min(),
xx.max(),   yy.min(),   yy.max()),   cmap=plt.cm.Paired,   aspect="auto",
origin="lower") # рисуем области
8  plt.plot(kmlucky["#  x"],  kmlucky["y"],  "k.",  markersize=4)  #рисуем
точки
9  plt.scatter(lucky_cent3[:, 0], lucky_cent3[:, 1], marker="x", s=169,
linewidths=3, color="w", zorder=10)
10 plt.show()
```

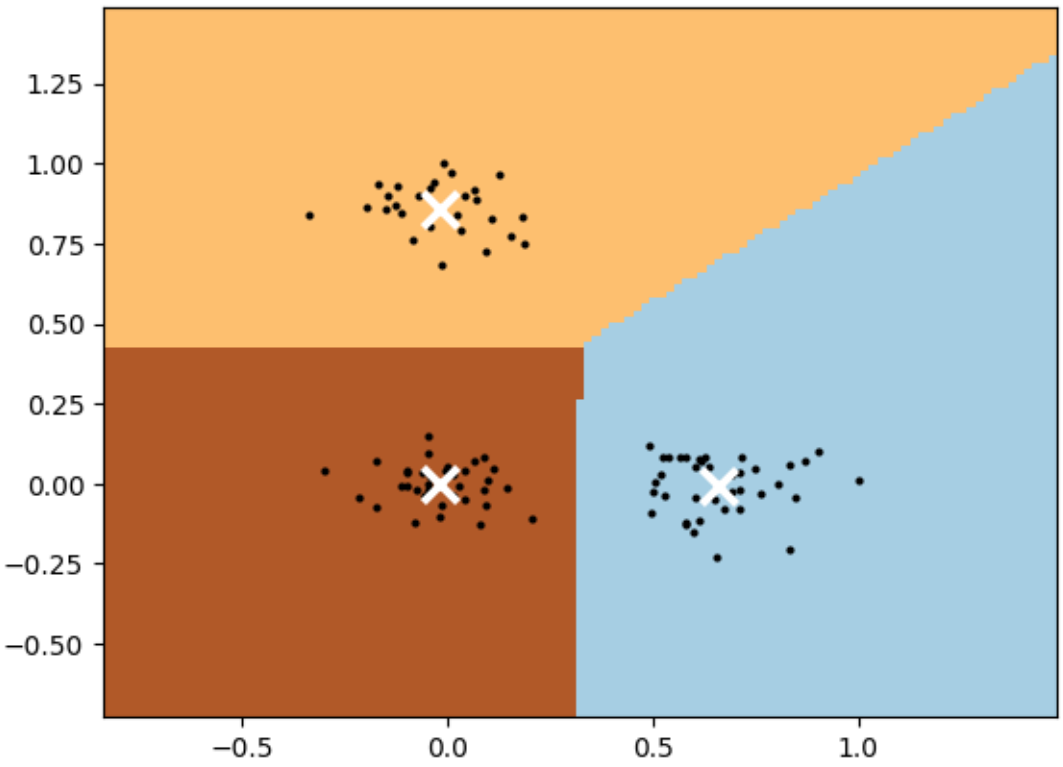

Рисунок 2.5.3 – Диаграмма Вороного для третьего датафрейма.

2.6. Построим для каждого признака диаграмму “box-plot” с разделением по кластерам (см. листинги 2.6.1, 2.6.2, 2.6.3). Полученные диаграммы см. на рисунках 2.6.1, 2.6.2, 2.6.3.

Листинг 2.6.1 – Построение блочной диаграммы для первого набора.

```
1  fig, axs = plt.subplots(1,2)
2   sb.boxplot(kmblobs,   y="#   x",   hue="cluster",   palette="tab10",
ax=axs[0])
3  sb.boxplot(kmblobs, y="y", hue="cluster", palette="tab10",
ax=axs[1])
```

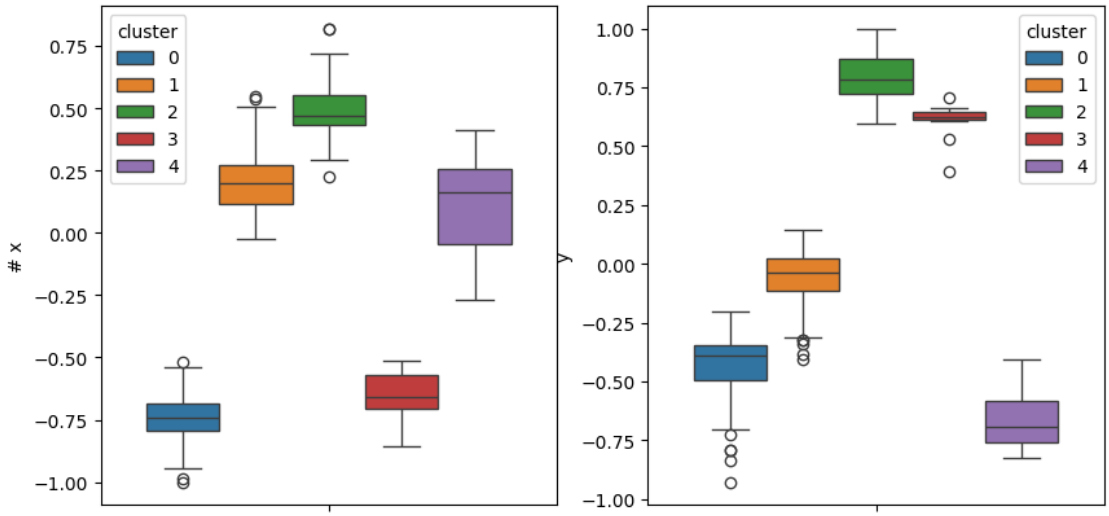

Рисунок 2.6.1 – Блочная диаграмма для первого набора.

Листинг 2.6.2 – Построение блочной диаграммы для второго набора.

```
1  fig, axs = plt.subplots(1,2)
2   sb.boxplot(kmblobs,   y="#   x",   hue="cluster",   palette="tab10",
ax=axs[0])
3  sb.boxplot(kmblobs, y="y", hue="cluster", palette="tab10",
ax=axs[1])
```

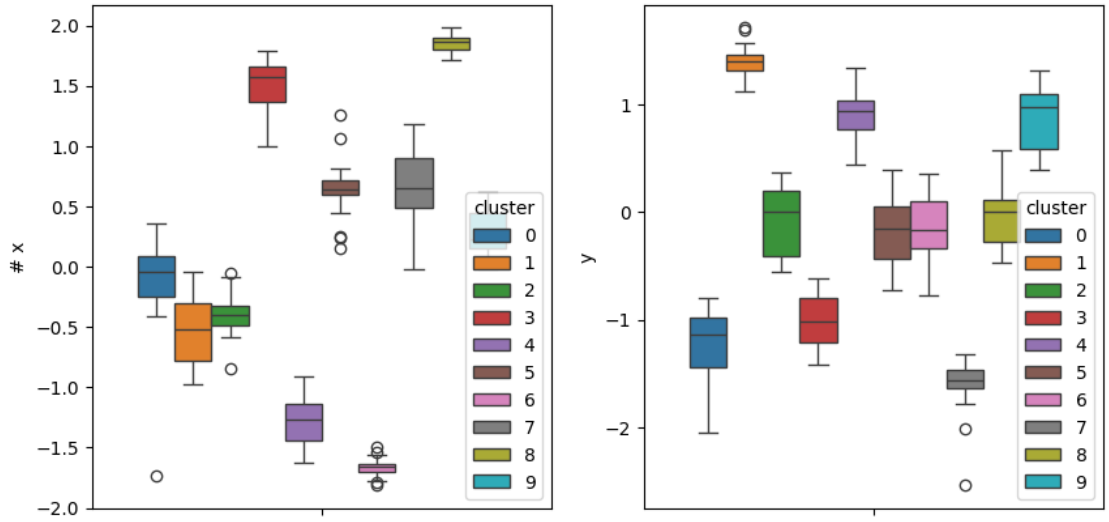

Рисунок 2.6.3 – Блочная диаграмма для второго набора.

Листинг 2.6.1 – Построение блочной диаграммы для третьего набора.

```
1  fig, axs = plt.subplots(1,2)
2   sb.boxplot(kmblobs,   y="#   x",   hue="cluster",   palette="tab10",
ax=axs[0])
```

```
3  sb.boxplot(kmblobs, y="y", hue="cluster", palette="tab10",
ax=axs[1])
```

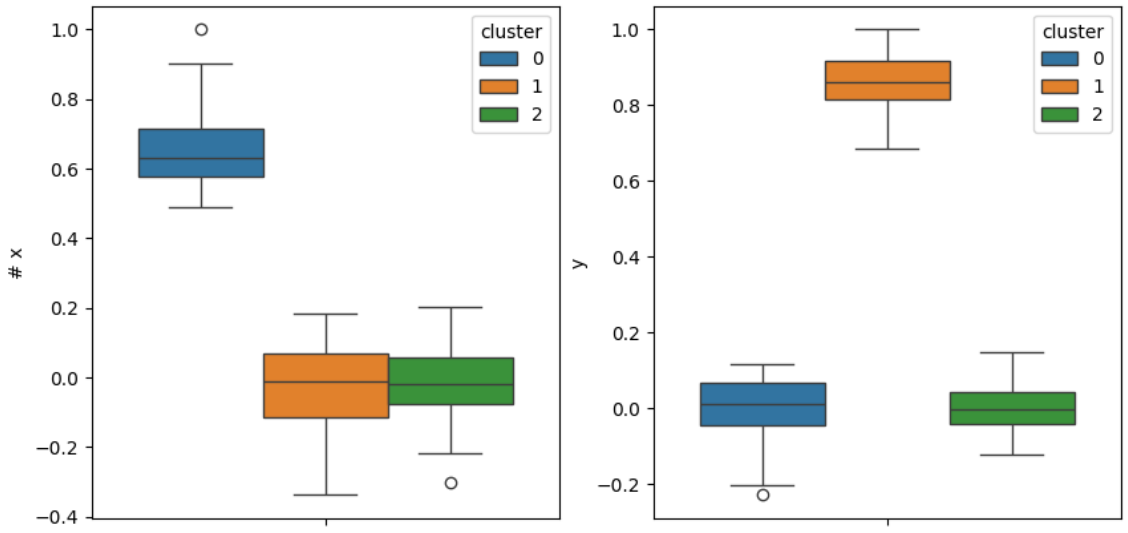

Рисунок 2.6.3 – Блочная диаграмма для третьего набора.

Блочные диаграммы позволяют определить распределение признаков, положение верхней и нижней квартилей, а также медиану, максимальное и наименьшее значения. Также видны выбросы, они обозначены выколотыми точками.

Что касается успешности кластеризации, можно отметить, что кластеризация набора lab2_blobs.csv прошла удачно, кластеры разделились достаточно чётко.

Аналогично и разделение набора lab2_luckyset.csv прошло крайне удачно, все кластеры разделены и нигде не пересекаются.

В свою очередь, на наборе lab2_noisymoons.csv метод K-means отработало плохо. Набор разделился на неоправданно большое число кластеров, многие из которых смешиваются и лежат слишком близко друг к другу.

2.7. Рассчитаем для каждого кластера кол-во точек, среднее, СКО, минимум и максимум (см. листинги 2.7.1, 2.7.3, 2.7.5). Результат работы см. в листингах 2.7.2, 2.7.4, 2.7.6.

Листинг 2.7.1 – Вычисление характеристик кластеров первого набора.

```
1  for i in range(5):
2      new_df = kmblobs[(kmblobs["cluster"] == i)].to_numpy()
3      print(f'Кластер {i}')
```

```
4     print("\tcount\t", [len(new_df)]*2)
5     print("\tmean\t", np.mean(new_df, axis=0)[:2:])
6     print("\tstd\t", np.std(new_df, axis=0)[:2:])
7     print("\tmin\t", np.min(new_df, axis=0)[:2:])
8     print("\tmax\t", np.max(new_df, axis=0)[:2:])
```

Листинг 2.7.2 – Характеристики кластеров первого набора.

```
Кластер 0
     count  [60, 60]
     mean   [-0.74445721 -0.44238897]
     std    [0.09959383 0.16136721]
     min    [-1.         -0.92739779]
     max    [-0.51706488 -0.20002991]
Кластер 1
     count  [108, 108]
     mean   [ 0.20715623 -0.05209791]
     std    [0.12093189 0.11128744]
     min    [-0.02600625 -0.4082313 ]
     max    [0.54601176 0.14593449]
Кластер 2
     count  [60, 60]
     mean   [0.49527277 0.79623692]
     std    [0.11398348 0.09610129]
     min    [0.22392024 0.59740694]
     max    [0.81983091 1.        ]
Кластер 3
     count  [10, 10]
     mean   [-0.65756479  0.60312874]
     std    [0.10730072 0.08220301]
     min    [-0.855771    0.39159806]
     max    [-0.51074251  0.70481493]
Кластер 4
     count  [22, 22]
     mean   [ 0.10151543 -0.66492306]
     std    [0.18290572 0.11403686]
     min    [-0.26705569 -0.82424706]
     max    [ 0.41076089 -0.40683274]
```

Листинг 2.7.3 – Вычисление характеристик кластеров второго набора.

```
for i in range(10):
    new_df = kmmoons[(kmmoons["cluster"] == i)].to_numpy()
    print(f'Кластер {i}')
    print("\tcount\t", [len(new_df)]*2)
    print("\tmean\t", np.mean(new_df, axis=0)[:2:])
    print("\tstd\t", np.std(new_df, axis=0)[:2:])
    print("\tmin\t", np.min(new_df, axis=0)[:2:])
    print("\tmax\t", np.max(new_df, axis=0)[:2:])
```

Листинг 2.7.4 – Характеристики кластеров второго набора.

```
Кластер 0
     count  [25, 25]
     mean   [-0.12313202 -1.22126912]
     std    [0.38824891 0.32379323]
     min    [-1.73617498 -2.04963387]
     max    [ 0.35507356 -0.79943922]
Кластер 1
     count  [40, 40]
     mean   [-0.53002822  1.39451517]
     std    [0.26353065 0.12190639]
     min    [-0.97991722  1.12464084]
     max    [-0.0458937  1.7194489]
Кластер 2
     count  [25, 25]
     mean   [-0.38501534 -0.09659808]
     std    [0.17600912 0.31246452]
     min    [-0.84818977 -0.55448238]
     max    [-0.05420952  0.37411748]
Кластер 3
     count  [23, 23]
     mean   [ 1.47730086 -1.02099192]
     std    [0.21129652 0.24068354]
     min    [ 0.99768175 -1.42250371]
     max    [ 1.78997473 -0.6128248 ]
Кластер 4
     count  [30, 30]
```

```
     mean   [-1.27721389  0.89497771]
     std    [0.19639792 0.21928702]
     min    [-1.62722565  0.44790946]
     max    [-0.90591845  1.34459378]
Кластер 5
     count  [37, 37]
     mean   [ 0.63867796 -0.15670566]
     std    [0.19296407 0.30002181]
     min    [ 0.14886527 -0.72158157]
     max    [1.25932885 0.3962754 ]
Кластер 6
     count  [25, 25]
     mean   [-1.66010328 -0.12554568]
     std    [0.07724224 0.30684853]
     min    [-1.81234309 -0.7685401 ]
     max    [-1.49585976  0.35419568]
Кластер 7
     count  [28, 28]
     mean   [ 0.68468056 -1.59919353]
     std    [0.29434536 0.22939807]
     min    [-0.0169691  -2.54158038]
     max    [ 1.18424541 -1.32696038]
Кластер 8
     count  [15, 15]
     mean   [ 1.84707671 -0.02914314]
     std    [0.07561418 0.28269193]
     min    [ 1.71079364 -0.46625726]
     max    [1.98196176 0.58024714]
Кластер 9
     count  [32, 32]
     mean   [0.28867503 0.87346479]
     std    [0.18038658 0.28294925]
     min    [-0.02468232  0.39688525]
     max    [0.61925157 1.32467198]
```

Листинг 2.7.5 – Вычисление характеристик кластеров третьего набора.

```
1  for i in range(3):
2     new_df = kmlucky[(kmlucky["cluster"] == i)].to_numpy()
```

```
3     print(f'Кластер {i}')
4     print("\tcount\t", [len(new_df)]*2)
5     print("\tmean\t", np.mean(new_df, axis=0)[:2:])
6     print("\tstd\t", np.std(new_df, axis=0)[:2:])
7     print("\tmin\t", np.min(new_df, axis=0)[:2:])
8     print("\tmax\t", np.max(new_df, axis=0)[:2:])
```

Листинг 2.7.6 – Характеристики кластеров третьего набора.

```
Кластер 0
     count  [35, 35]
     mean   [-0.01996088 -0.00188256]
     std    [0.10425615 0.06275928]
     min    [-0.30126696 -0.12488076]
     max    [0.20304153 0.14608823]
Кластер 1
     count  [27, 27]
     mean   [-0.02119034  0.86006131]
     std    [0.12397173 0.07773195]
     min    [-0.33786356  0.68351928]
     max    [0.18455902 1.        ]
Кластер 2
     count  [38, 38]
     mean   [ 0.66111681 -0.00591853]
     std    [0.12445432 0.08680812]
     min    [ 0.48850728 -0.22983977]
     max    [1.         0.11684846]
```

Полученные данные показывают, что в наборах lab2_luckyset.csv и lab2_noisymoons.csv точки разделились на кластеры практически поровну, что также видно на построенных диаграммах.

Вычисленные числовые характеристики каждого кластера совпали с аналогичными характеристиками, которые можно узнать, глядя на блочную диаграмму.

Можно сделать вывод, что диаграммы целиком соответствуют вычисленным значениям.

3. DBSCAN.

3.1. Подберём параметры алгоритма DBSCAN, дающие наилучшие результаты (см. листинги 3.1.1, 3.1.2, 3.1.3).

Листинг 3.1.1 – Алгоритм DBSCAN для первого набора.

```
1  dbscan = DBSCAN(eps = 0.15, min_samples = 6)
2  blob_data = mblobs.to_numpy()
3  dbscan_clust = dbscan.fit_predict(blob_data)
4  print(set(dbscan_clust))
```

Параметры *eps*=0.15 и *min_samples*=6 были выбраны в качестве оптимальных. При уменьшении *min_samples* разбиение на кластеры не меняется, однако точки, помеченные как выбросы теряют эту характеристику, в свою очередь при увеличении *min_samples* разбиение также не меняется, но всё больше точек помечается как выбросы. Увеличение параметра *eps* приводит к уменьшению числа кластеров и «слиянию» двух групп, находящихся на достаточно большом отдалении друг от друга. Уменьшение *eps* сначала приводит к увеличению числа кластеров, а при дальнейшем уменьшении параметра большая часть точек, даже явно принадлежащих скоплениям, помечаются как выбросы.

Листинг 3.1.2 – Алгоритм DBSCAN для второго набора.

```
1  dbscan = DBSCAN(eps = 0.28, min_samples = 6)
2  moon_data = mmoons.to_numpy()
3  dbscan_clust = dbscan.fit_predict(moon_data)
4  print(set(dbscan_clust))
```

Параметры *eps*=0.28 и *min_samples*=6 были выбраны в качестве оптимальных. При уменьшении *min_samples* ничего не меняется, в свою очередь при увеличении *min_samples* увеличивается число мелких кластеров, лежащих близко друг к другу и часто смешивающихся, также растёт число выбросов. Увеличение параметра *eps* приводит к «слиянию» двух ярко выраженных кластеров, которые не должны быть объединены в одну группу. Уменьшение *eps* приводит к увеличению числа кластеров, лежащих вплотную к другим группам, и появлению множества выбросов.

Листинг 3.1.2 – Алгоритм DBSCAN для третьего набора.

```
dbscan = DBSCAN(eps = 0.1, min_samples = 7)
lucky_data = mlucky.to_numpy()
dbscan_clust = dbscan.fit_predict(lucky_data)
print(set(dbscan_clust))
```

Параметры *eps*=0.1 и *min_samples*=7 были выбраны в качестве оптимальных. При уменьшении *min_samples* разбиение не меняется, но уменьшается число точек, отмеченных как выбросы, в свою очередь при увеличении *min_samples* один из кластеров пропадает и помечается как выбросы. Увеличение параметра *eps* приводит к исчезновению выбросов. Уменьшение *eps* приводит к исчезновению одного из кластеров и значительному увеличению числа выбросов.

3.2. Построим диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров (см. листинги 3.2.1, 3.2.2, 3.2.3). Полученные диаграммы см. на рисунках 3.2.1, 3.2.2, 3.2.3.

Листинг 3.2.1 – Построение диаграммы рассеяния для первого набора после кластеризации.

```
dbsblobs = mblobs.copy()
dbsblobs['cluster'] = dbscan_clust
sb.scatterplot(dbsblobs, x = '# x', y = 'y', hue = 'cluster',
palette = 'tab10')
```

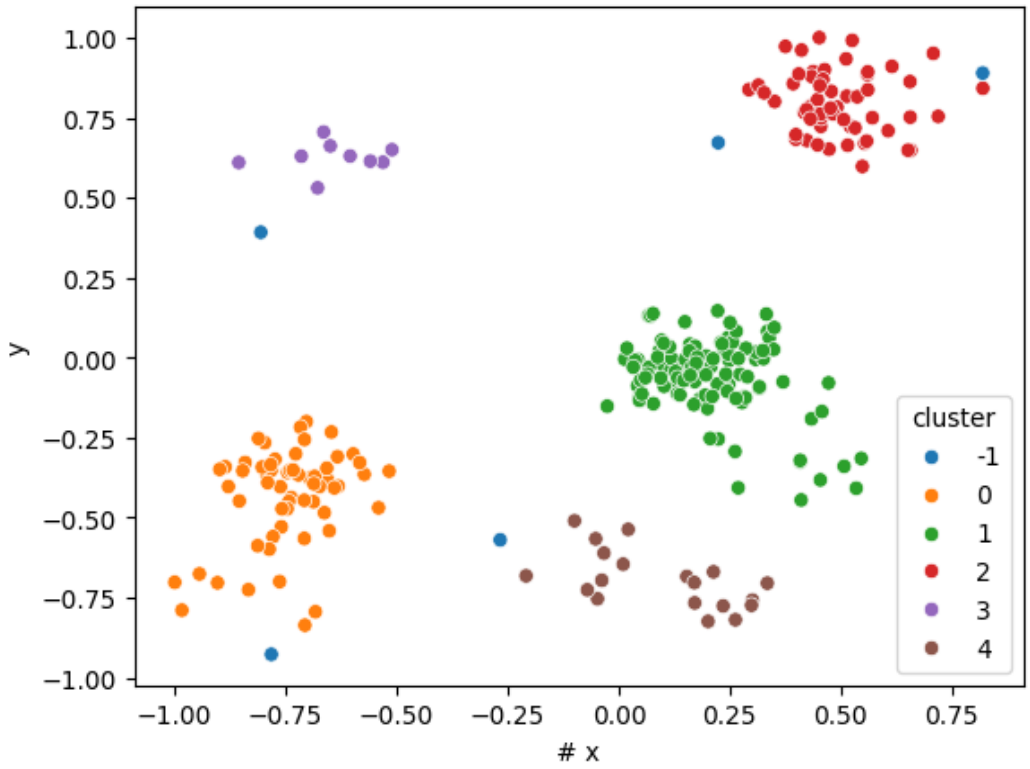

Рисунок 3.2.1 – Диаграмма рассеяния первого набора.

Листинг 3.2.2 – Построение диаграммы рассеяния для второго набора после кластеризации.

```
1  dbsmoons = mmoons.copy()
2  dbsmoons['cluster'] = dbscan_clust
3  sb.scatterplot(dbsmoons, x = '# x', y = 'y', hue = 'cluster',
palette = 'tab10')
```

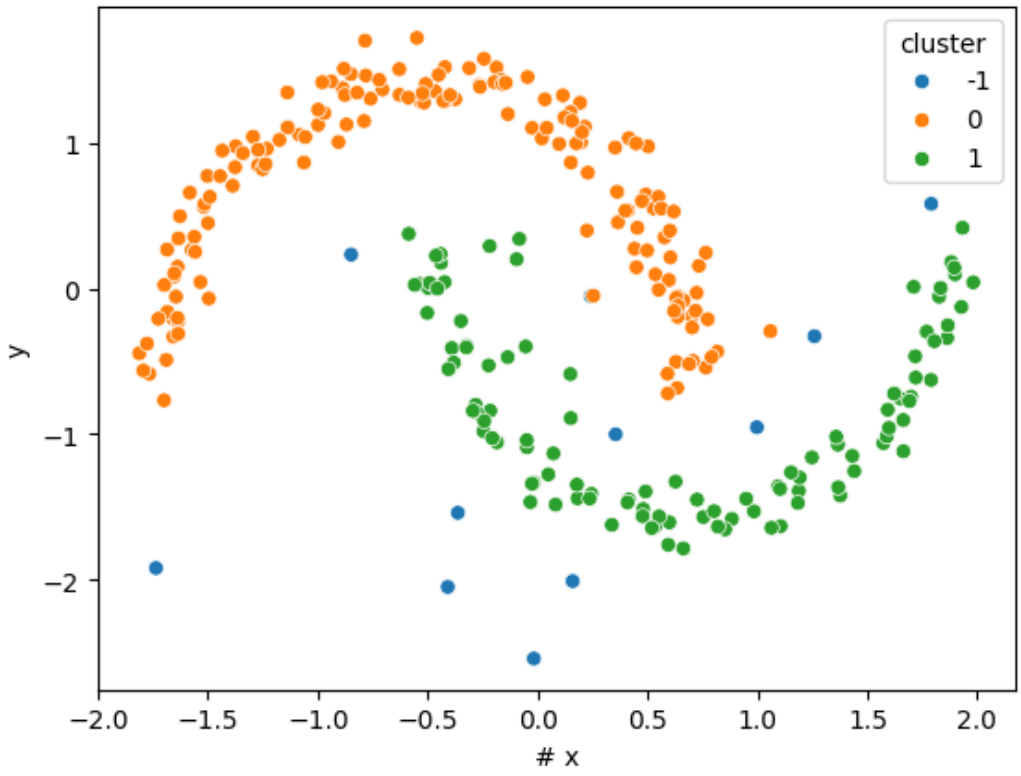

Рисунок 3.2.2 – Диаграмма рассеяния второго набора.

Листинг 3.2.3 – Построение диаграммы рассеяния для третьего набора после кластеризации.

```
1  dbslucky = mlucky.copy()
2  dbslucky['cluster'] = dbscan_clust
3  sb.scatterplot(dbslucky, x = '# x', y = 'y', hue = 'cluster',
palette = 'tab10')
```

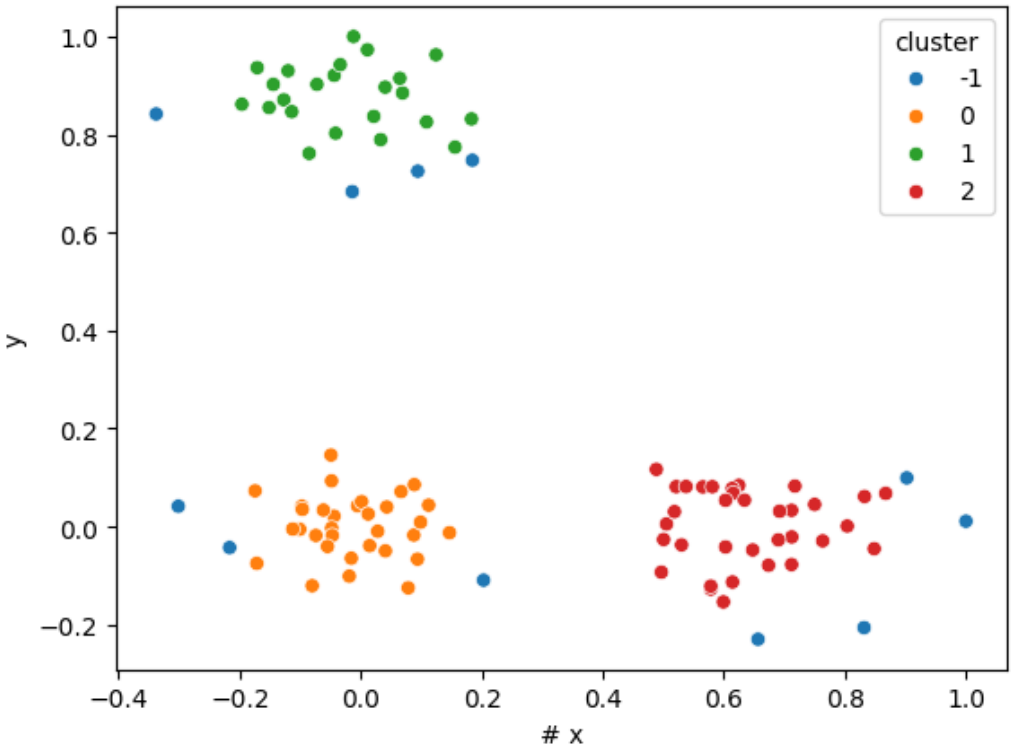

Рисунок 3.2.3 – Диаграмма рассеяния третьего набора.

3.3. Сделаем выводы об успешности кластеризации.

Подводя итог кластеризации с использованием DBSCAN можно сказать, что для всех наборов она прошла успешно, причём результат кластеризации наборов lab2_blobs.csv и lab2_luckyset.csv никак не отличается от результата, полученного при использовании метода K-means, за исключением появления дополнительной группы, в которую определяются шумы. Набор lab2_noisymoons.csv также был кластеризован успешно, но результат кластеризации разительно отличается от результата, полученного в пункте 2 для этого же датасета. Кластеров стало 2, что соответствует ожидаемому результату кластеризации выборки такого вида.

4. Иерархическая кластеризация.

4.1. Проведём иерархическую кластеризацию при всех возможных параметрах *linkage* и для каждого из наборов построим соответствующие дендрограммы (см. рисунки 4.1.1, 4.1.2, 4.1.3). Функция для рисования дендрограмм представлена в листинге 4.1.1. Рисование дендрограмм для каждого датасета см. в листингах 4.1.2, 4.1.3, 4.1.4.

Листинг 4.1.1 – Функция рисования дендрограмм.

```
def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)
```

Листинг 4.1.2 – Рисование набора дендрограмм для первого датасета.

```
fig, axs = plt.subplots(2,2, figsize = (15, 15))
linkages = ["ward", "complete", "average", "single"]
for i in range(4):
    agc_blob = AgglomerativeClustering(distance_threshold=0, n_clusters=None, linkage=linkages[i])
    agc_blob = agc_blob.fit(mblobs)
    plot_dendrogram(agc_blob, truncate_mode="level", p=3, ax=axs[i//2, i%2])
    axs[i//2, i%2].set_title(f"{linkages[i]}")
    axs[i//2, i%2].set_xlabel("Количество точек в узле (или индекс точки, если нет скобок)")
    axs[i//2, i%2].set_ylabel("Расстояние")
```

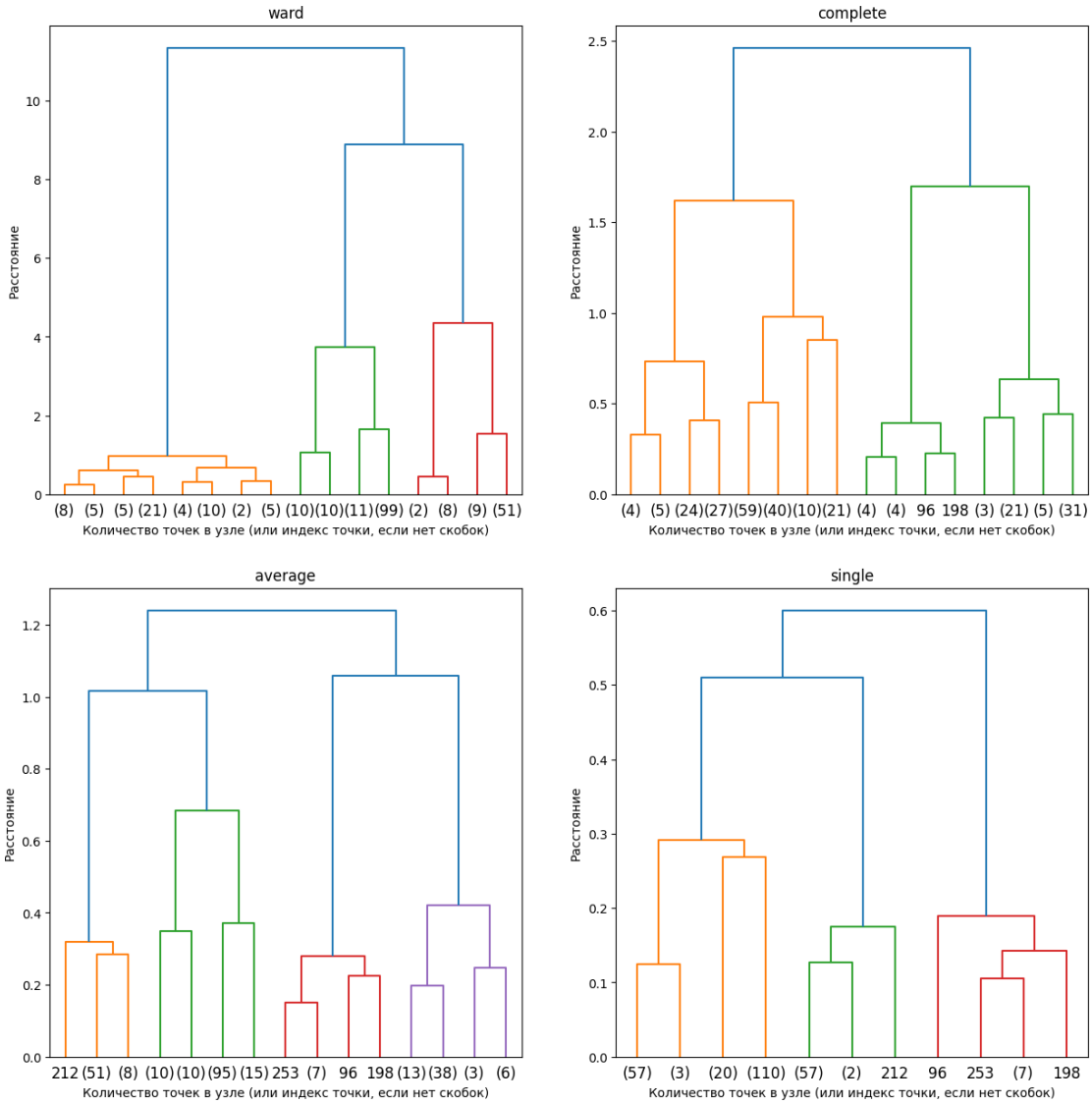

Рисунок 4.1.1 – Набор дендрограмм для первого датасета.

Из рисунка 4.1.1 можно понять, какое расстояние является походящим для кластеризации при каждом из значений параметра *linkage*.

- *ward*: расстояние 3, число кластеров равно 5
- *complete*: расстояние 1.2, число кластеров равно 4
- *average*: расстояние 0.5, число кластеров равно 5
- *single*: расстояние 0.2, число кластеров равно 5

Таким образом, число кластеров для параметров *ward*, *average* и *single* при оптимальном расстоянии совпало с числом кластеров, полученных в результате работы алгоритмов K-means и DBSCAN. Число кластеров для параметра *complete* равно 4, что входит в промежуток метода локтя (см. рисунок 2.1.1).

Листинг 4.1.3 – Рисование набора дендрограмм для второго датасета.

```
1  fig, axs = plt.subplots(2,2, figsize = (15, 15))
2  linkages = ["ward", "complete", "average", "single"]
3  for i in range(4):
4      agc_moon      =      AgglomerativeClustering(distance_threshold=0,
n_clusters=None, linkage=linkages[i])
5      agc_moon = agc_moon.fit(mmoons)
6      plot_dendrogram(agc_moon,       truncate_mode="level",       p=4,
ax=axs[i//2, i%2])
7      axs[i//2, i%2].set_title(f"{linkages[i]}")
8      axs[i//2,  i%2].set_xlabel("Количество  точек  в  узле  (или  индекс
точки, если нет скобок)")
9      axs[i//2, i%2].set_ylabel("Расстояние")
```

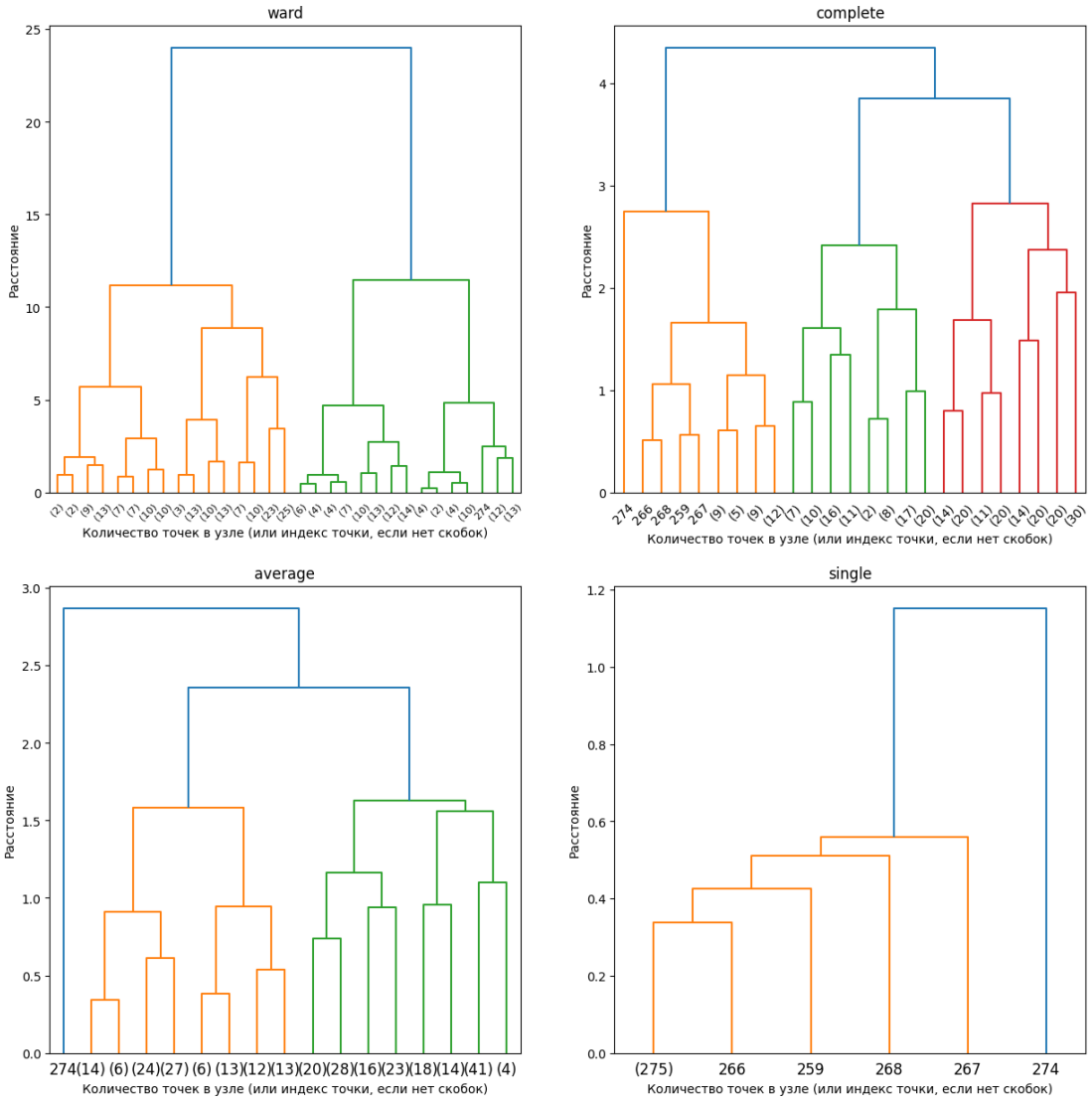

Рисунок 4.1.2 – Набор дендрограмм для второго датасета.

Из рисунка 4.1.2 можно понять, какое расстояние является походящим для кластеризации при каждом из значений параметра *linkage*.

- *ward*: расстояние 5, число кластеров равно 7
- *complete*: расстояние 2, число кластеров равно 7
- *average*: расстояние 1, число кластеров равно 8
- *single*: расстояние 0.4, число кластеров равно 5

Таким образом, число кластеров для параметров *ward* и *complete* равно 7, что входит в промежуток метода локтя (см. рисунок 2.1.2), число кластеров для параметра *average* равно 8, что также находится в промежутке метода локтя. А для *single* было получено новое значение кластеров, до этого не рассмотренное.

Листинг 4.1.4 – Рисование набора дендрограмм для третьего датасета.

```
1  fig, axs = plt.subplots(2,2, figsize = (15, 15))
2  linkages = ["ward", "complete", "average", "single"]
3  for i in range(4):
4      agc_lucky     =     AgglomerativeClustering(distance_threshold=0,
n_clusters=None, linkage=linkages[i])
5      agc_lucky = agc_lucky.fit(mlucky)
6      plot_dendrogram(agc_lucky,      truncate_mode="level",      p=3,
ax=axs[i//2, i%2])
7      axs[i//2, i%2].set_title(f"{linkages[i]}")
8      axs[i//2,  i%2].set_xlabel("Количество точек в узле (или индекс
точки, если нет скобок)")
9      axs[i//2, i%2].set_ylabel("Расстояние")
```

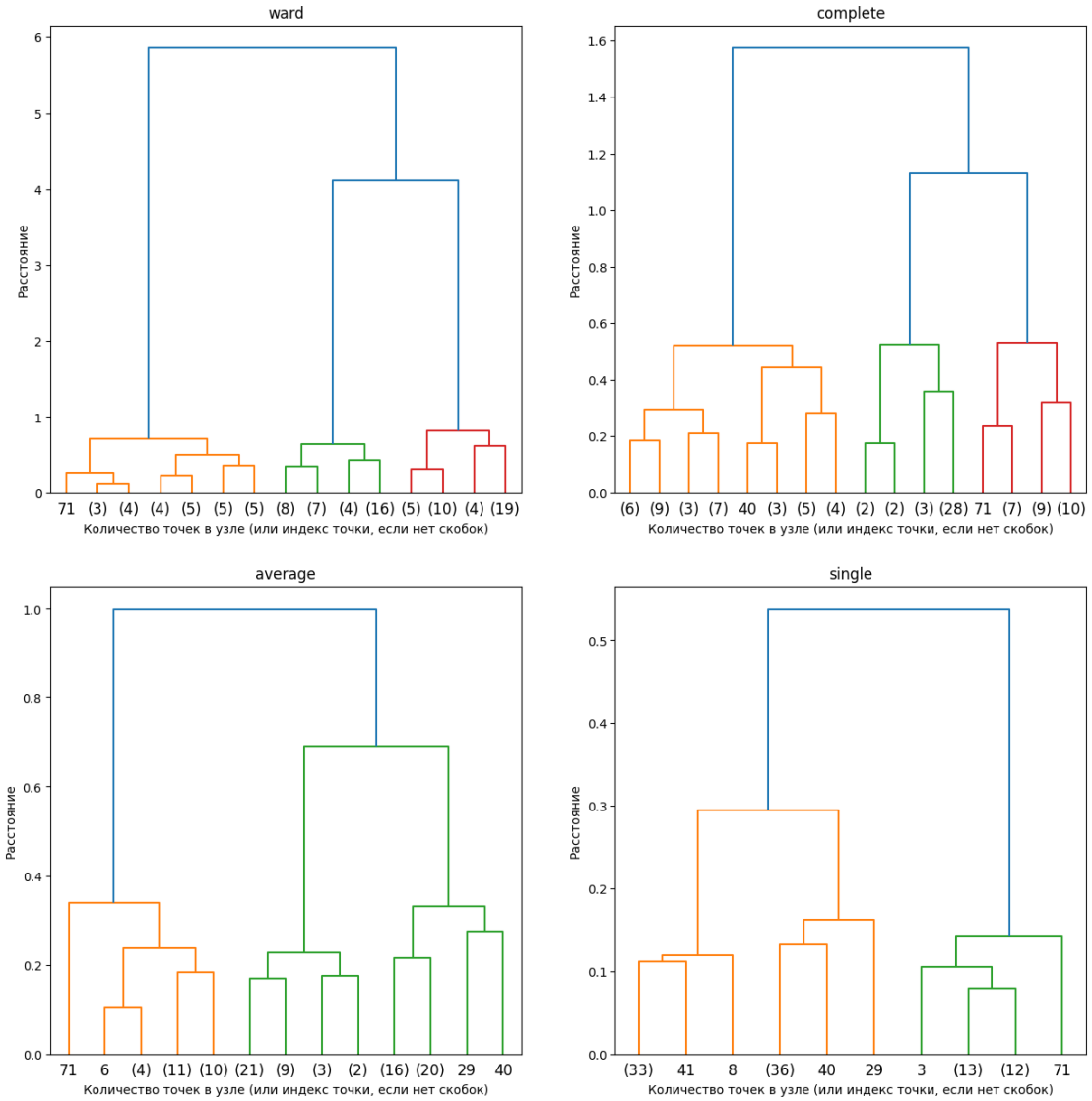

Рисунок 4.1.3 – Набор дендрограмм для третьего датасета.

Из рисунка 4.1.3 можно понять, какое расстояние является походящим для кластеризации при каждом из значений параметра *linkage*.

- *ward*: расстояние 2, число кластеров равно 3
- *complete*: расстояние 0.8, число кластеров равно 3
- *average*: расстояние 0.5, число кластеров равно 3
- *single*: расстояние 0.2, число кластеров равно 3

Таким образом, число кластеров для параметров *ward*, *complete*, *average* и *single* получилось одинаковым и равным 3, что полностью совпадает с числом кластеров, полученным при использовании методов K-means и DBSCAN.

4.2. Построим диаграмму рассеяния результатов иерархической кластеризации с выделением разным цветом разных кластеров (см. рисунки 4.2.1,

4.2.2 и 4.2.3). Построение диаграмм будет выполняться с использованием фанкции *draw_agg_plot()* (см. листинг 4.2.1), принимающей на вход датафрейм, расстояние и параметр *linkage*. Для lab2_blobs.csv возьмем *ward* и расстояние 3 (см. листинг 4.2.2). Для lab2_noisymoons.csv возьмем *ward* и расстояние 5 (см. листинг 4.2.3). Для lab2_luckyset.csv возьмем *ward* и расстояние 2 (см. листинг 4.2.4).

Листинг 4.2.1 – Функция *draw_agg_plot()*.

```
1  def draw_agg_plot(df, distance, linkage):
2      agg = AgglomerativeClustering(n_clusters=None,
distance_threshold=distance, linkage=linkage)
3      clust = agg.fit_predict(df)
4      clust_df = pd.DataFrame(data=df, columns=['# x', 'y'])
5      clust_df['cluster'] = clust
6      sb.scatterplot(data=clust_df, x='# x', y='y', hue='cluster',
palette='tab10')
```

Листинг 4.2.2 – Рисование диграммы для первого датасета.

```
1  draw_agg_plot(mblobs, 3, "ward")
```

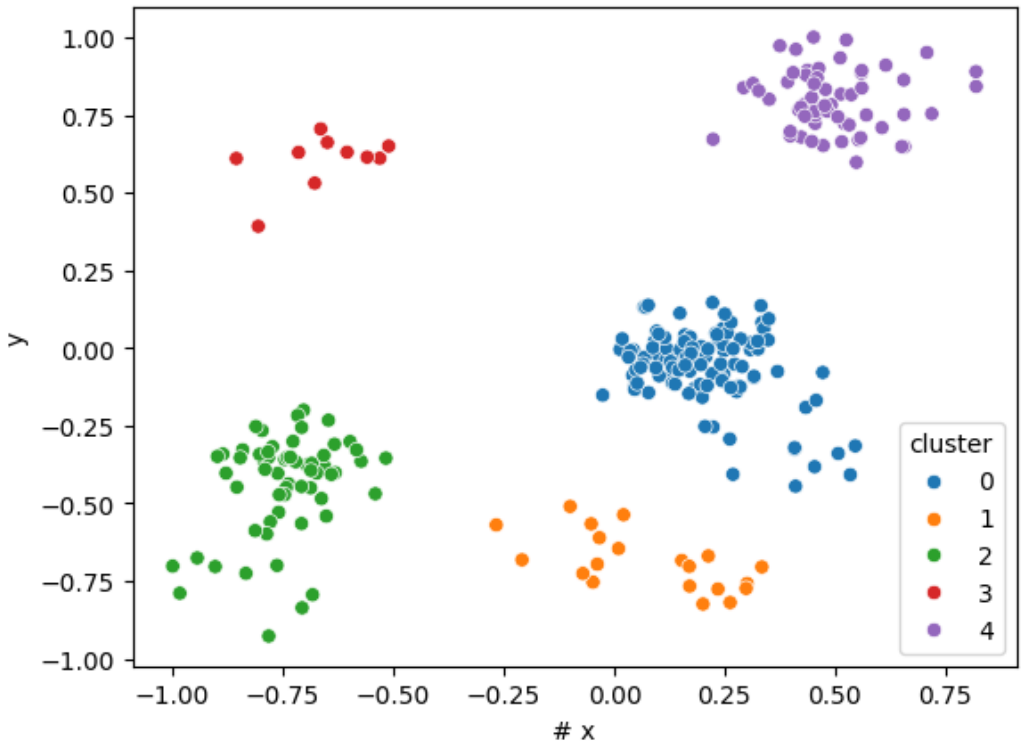

Рисунок 4.2.1 – Диаграмма рассеяния для первого датасета.

Листинг 4.2.3 – Рисование диграммы для второго датасета.

```
1  draw_agg_plot(mmoons, 5, "ward")
```

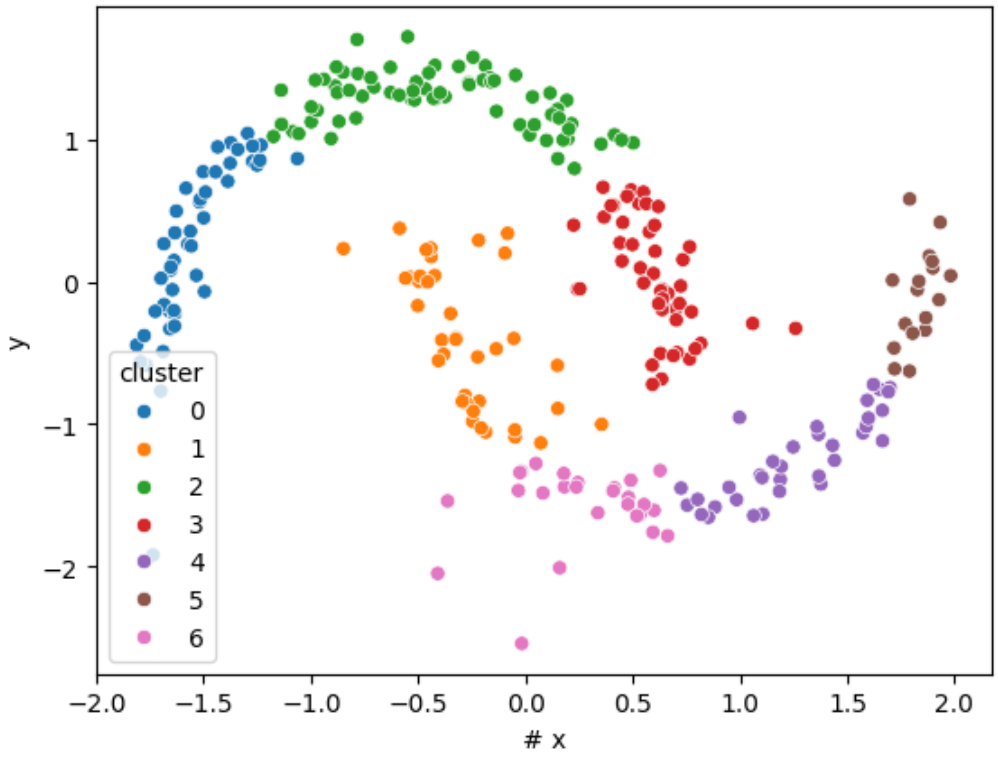

Рисунок 4.2.2 – Диаграмма рассеяния для второго датасета.

Листинг 4.2.4 – Рисование диграммы для третьего датасета.

```
1  draw_agg_plot(mlucky, 2, "ward")
```

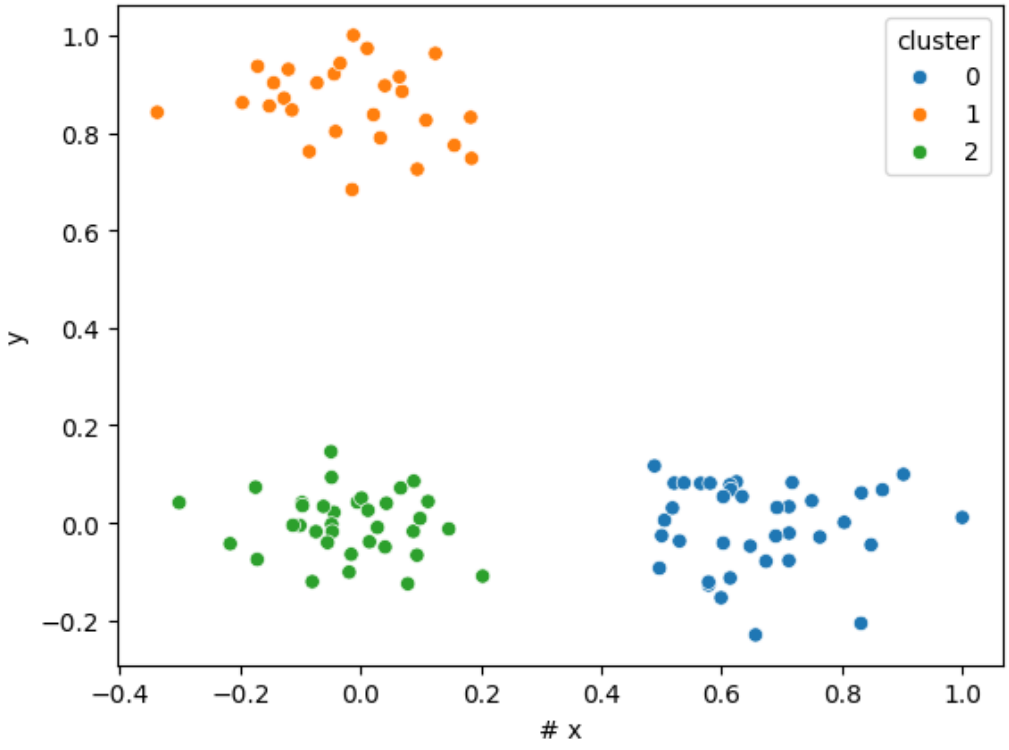

Рисунок 4.2.3 – Диаграмма рассеяния для третьего датасета.

4.3. Сравним результаты кластеризации с предыдущими результатами.

Для набора lab2_blobs, можно сказать, что все методы кластеризации показали себя хорошо. Результатом каждого метода было разделение набора на 5 кластеров, однако, на мой взгляд, метод DBSCAN справился лучше других, потому как он выделил шумы в отдельный кластер.

Для набора lab2_blobs лучшим оказался метод DBSCAN, потому что он разделил набор данных на 2 кластера, что полностью согласуется с визуальной оценкой распределения наблюдений. Два других метода поделили выборку на гораздо большее число кластеров, расположенных очень близко и местами смешивающихся, я счёл такие результаты не оптимальными.

Для набора lab2_blobs, можно сказать, что все методы кластеризации показали себя хорошо. Отличительной чертой этого набора является значительная резнесённость скоплений точек, что полностью убирает возможность смешения кластеров. Для кластеризации этого датасета нельзя выделить лучший, все отработали, как ожидалось.

Выводы.

В ходе выполнения лабораторной работы были изучены методы кластеризации данных. Для кластеризации использовались такие методы как K-means, DBSCAN и иерархическая кластеризация. Были изучены методы локтя и силуэта, используемые для определения оптимального количества кластеров при кластеризации. В ходе исследования было выяснено, что алгоритм K-means лучше всего подходит для кластеризации больших объёмов данных, однако он плохо работает, если распределение данных имеет сложную форму. В свою очередь, алгоритм DBSCAN хорошо работает на данных, имеющих сложную форму распределения, это обусловлено возможностью точечного контролирования кластеризации и отсеивания шумов. Иерархическая кластеризация показывает хороший результат, если нет данных об изначальном количестве кластеров, однако плохо работает на больших объёмах данных.