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 156М

Студентка гр. 1304 Чернякова А.Д.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Изучить различные методы кластеризации (K-Means, DBSCAN, Иерархическая кластеризация) на трех наборах данных различной сложности.

Задание.

1) Подготовка набора данных
2) K-Means
3) DBSCAN
4) Иерархическая кластеризация

Выполнение работы.

1. Подготовка наборов данных:

Согласно варианту будет рассмотрено 3 набора данных: lab2_blobs.csv, lab2_luckyset.csv, lab2_circles.csv.

1.1. Загрузим три набора данных (для blobs см. листинг 1.1.1, для luckyset см. листинг 1.1.2, для circles см. листинг 1.1.3).

Листинг 1.1.1 - Загрузка набора данных lab2_blobs.csv

```
df_bl = pd.read_csv('lab2_blobs.csv')
```

Листинг 1.1.2 - Загрузка набора данных lab2_luckyset.csv

```
df_ls = pd.read_csv('lab2_luckyset.csv')
```

Листинг 1.1.3 - Загрузка набора данных lab2_circles.csv

```
df_cc = pd.read_csv('lab2_circles.csv')
```

1.2. Проверим корректность загрузки с помощью метода head() - по умолчанию выводит первые 5 строк (наблюдений) набора данных (для blobs см. табл. 1.2.1, для luckyset см. табл. 1.2.2, для circles см. табл. 1.2.3).

Таблица 1.2.1 - Результат работы метода head() для blobs

	# 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

Таблица 1.2.2 - Результат работы метода head() для luckyset

	# 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

Таблица 1.2.3 - Результат работы метода head() для circles

	# x	y
0	0.3400	0.3297
1	0.6849	0.7212
2	0.0085	0.2924
3	-0.8343	-0.3787
4	0.1230	-1.0068

Все данные загружены корректно.

1.3. Построим диаграмму рассеяния набора данных и опишем форму данных. Для blobs смотреть листинг 1.3.1 (код отрисовки) и рисунок 1.3.1. Для luckyset смотреть листинг 1.3.2 (код отрисовки) и рисунок 1.3.2. Для circles смотреть листинг 1.3.3 (код отрисовки) и рисунок 1.3.3.

Листинг 1.3.1 - Отображение диаграммы рассеяния для blobs

```
plt.scatter(df_bl['# x'], df_bl['y'], alpha = 0.7)
plt.title('Диаграмма рассеяния blobs')
plt.xlabel('# x')
plt.ylabel('y')
plt.show()
```

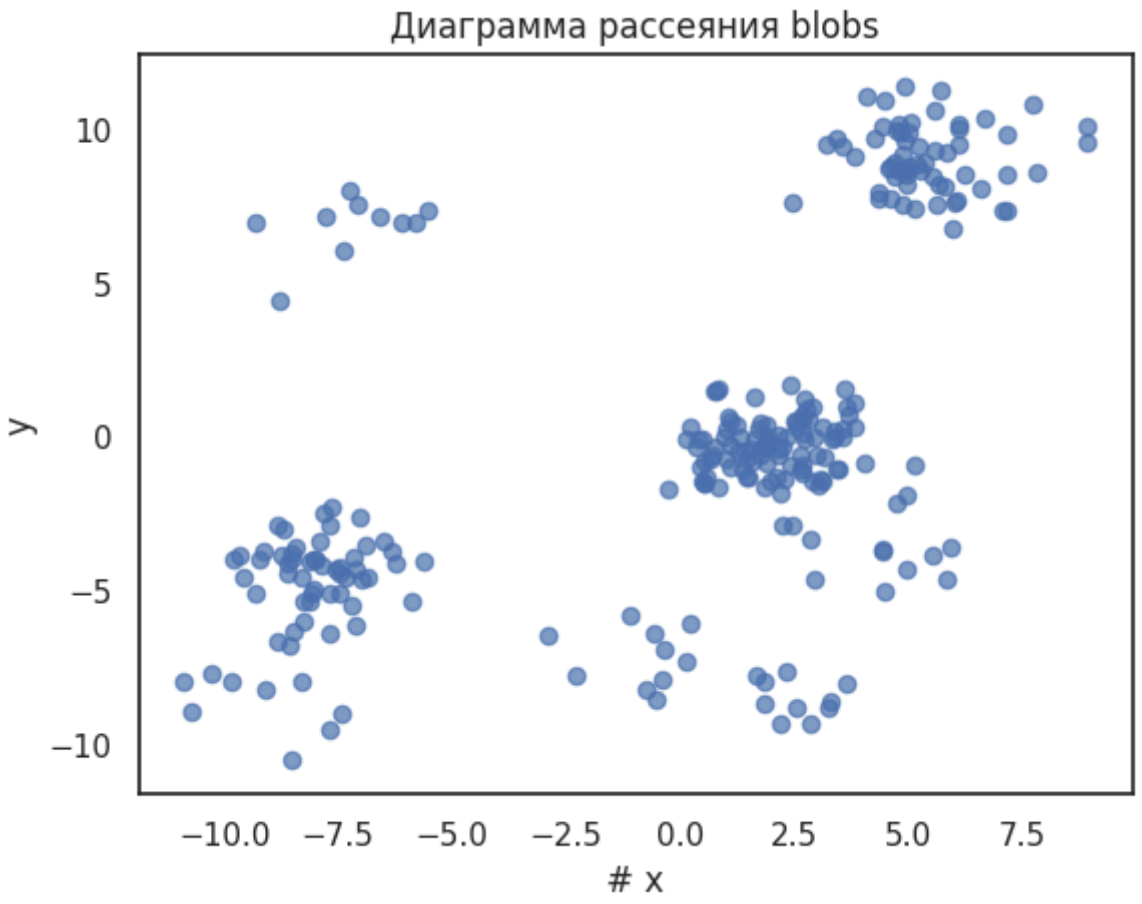

Рисунок 1.3.1 - Диаграмма рассеяния для blobs

По диаграмме рассеивания 1.3.1 нельзя увидеть зависимости между переменными, точки скапливаются группами в разных местах. Можно выделить 4-8 таких групп (кластеров).

Листинг 1.3.2 - Отображение диаграммы рассеяния для luckyset

```
plt.scatter(df_ls['# x'], df_ls['y'], alpha = 0.7)
plt.title('Диаграмма рассеяния luckyset')
plt.xlabel('# x')
plt.ylabel('y')
plt.show()
```

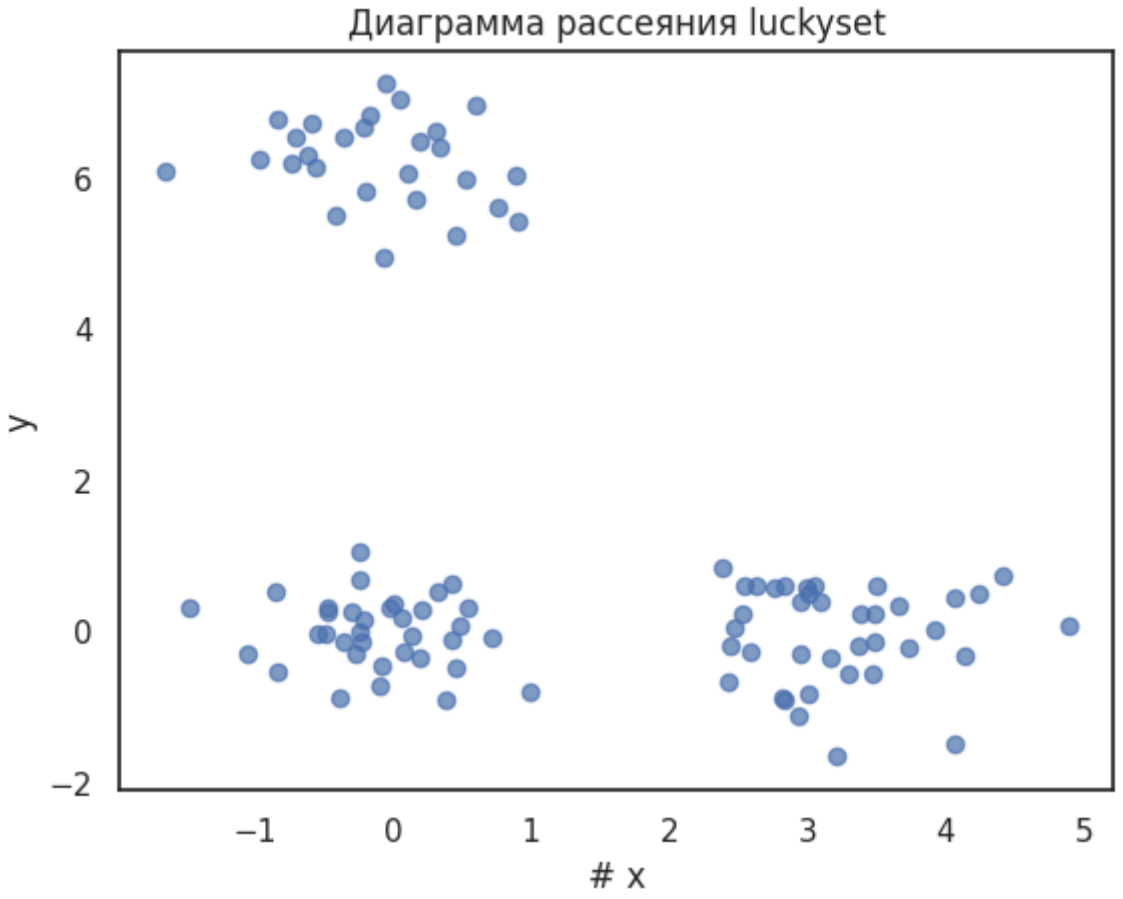

Рисунок 1.3.2 - Диаграмма рассеяния для luckyset

По диаграмме рассеивания 1.3.2 нельзя увидеть прямую зависимость между переменными, точки скапливаются группами в разных местах. Можно выделить 3 такие группы (кластера).

Листинг 1.3.3 - Отображение диаграммы рассеяния для circles

```
plt.scatter(df_cc['# x'], df_cc['y'], alpha = 0.7)
plt.title('Диаграмма рассеяния circles')
plt.xlabel('# x')
plt.ylabel('y')
plt.show()
```

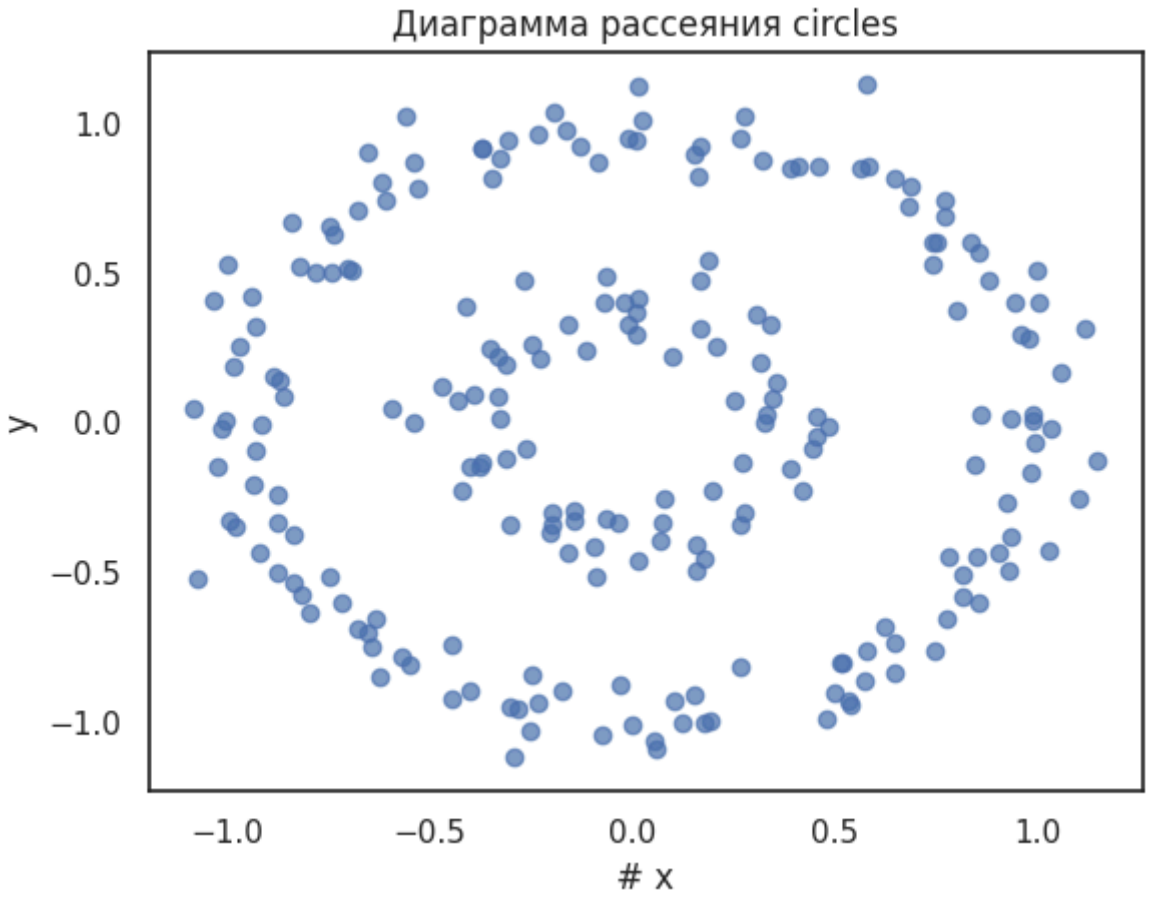

Рисунок 1.3.3 - Диаграмма рассеяния для circles

По диаграмме рассеивания 1.3.3 можно увидеть зависимость между переменными (уравнение окружности/эллипса). Можно выделить 2 кольца/эллипса (кластера).

1.4. Подготовим наборы данных проведя нормировку данных по модулю. Данный способ предобработки данных выбран чтобы сохранить знак (для blobs см. листинг 1.4.1, для luckyset см. листинг 1.4.2, для circles см.

листинг 1.4.3). Первые 5 строк (наблюдений) для blobs см. табл. 1.4.1, для luckyset см. табл. 1.4.2, для circles см. табл. 1.4.3.

Листинг 1.4.1 - Нормировка по модулю для blobs

```
numpy_arr = df_bl[["# x", "y"]].to_numpy()
scaler = MaxAbsScaler()
scaled_bl = scaler.fit_transform(numpy_arr)
scaled_bl
```

Листинг 1.4.2 - Нормировка по модулю для luckyset

```
numpy_arr = df_ls[["# x", "y"]].to_numpy()
scaler = MaxAbsScaler()
scaled_ls = scaler.fit_transform(numpy_arr)
scaled_ls
```

Листинг 1.4.3 - Нормировка по модулю для circles

```
numpy_arr = df_cc[["# x", "y"]].to_numpy()
scaler = MaxAbsScaler()
scaled_cc = scaler.fit_transform(numpy_arr)
scaled_cc
```

Таблица 1.4.1 - Нормировка по модулю для blobs

# x	y
-0.737612571	-0.437434030
-0.647142069	-0.232689466
0.819830913	0.841592780
0.100045947	-0.0253676729
0.043548980	-0.00648265428

Таблица 1.4.2 - Нормировка по модулю для luckyset

# x	y

-0.09707923	0.04154397
-0.1270033	0.87088881
0.52131731	0.08148425
-0.01387723	0.68351928
-0.00519885	0.04259467

Таблица 1.4.3 - Нормировка по модулю для circles

# x	y
0.295986768	0.292287234
0.596239227	0.639361702
0.00739966919	0.259219858
-0.726299295	-0.335726950
0.107077566	-0.892553191

2. K-Means:

K-Means - один из популярнейших и простых алгоритмов кластеризации.

2.1. Проведем исследование оптимального количества кластеров методов локтя.

Сначала опишем функцию для реализации метода локтя (см. листинг 2.1). Здесь 10 раз прогоняется алгоритм K-средних, задавая количество кластеров (от 1 до 10) и для каждого из кластеров алгоритм KMeans отработает 5 раз и выбирается лучшее (n_init = 5). Далее данные подставляются и находятся центроиды, рассчитывается инерция, добавляется в список инерций для визуализации зависимости инерции от количества кластеров. Далее отображается график, по которому можно определить оптимальное количество кластеров.

Листинг 2.1 - Реализация метода локтя

```
def loktya(df):
  inert_list = []
  for i  in range(10):
      temp_km = KMeans(i+1, n_init = 5)
      temp_km.fit(df)
      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()
```

Вызовем данную функцию для каждого набора данных (для blobs см. рис. 2.1.1, для luckyset см. рис. 2.1.2, для circles см. рис. 2.1.3).

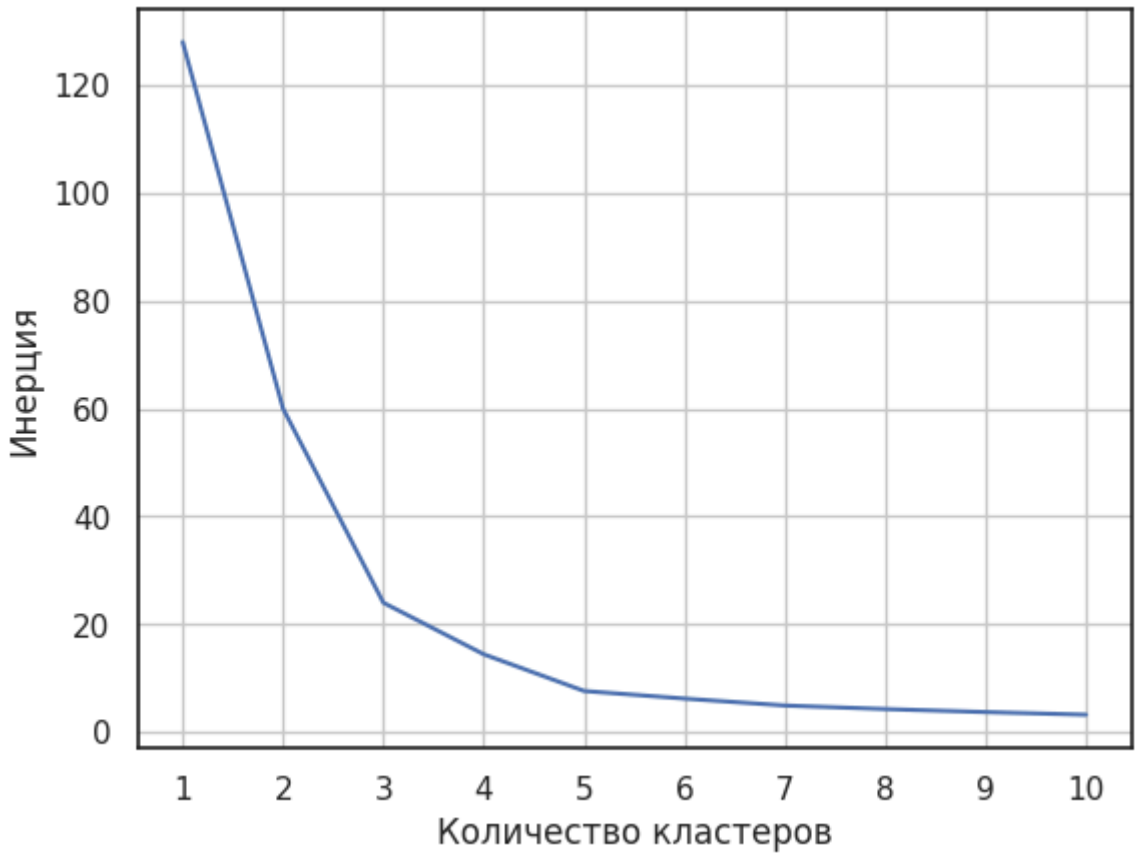

Рисунок 2.1.1 - Метод локтя для blobs

Для набора данных blobs оптимальным количеством кластеров будет 5, так как в этой точке скорость убывания графика уменьшается.

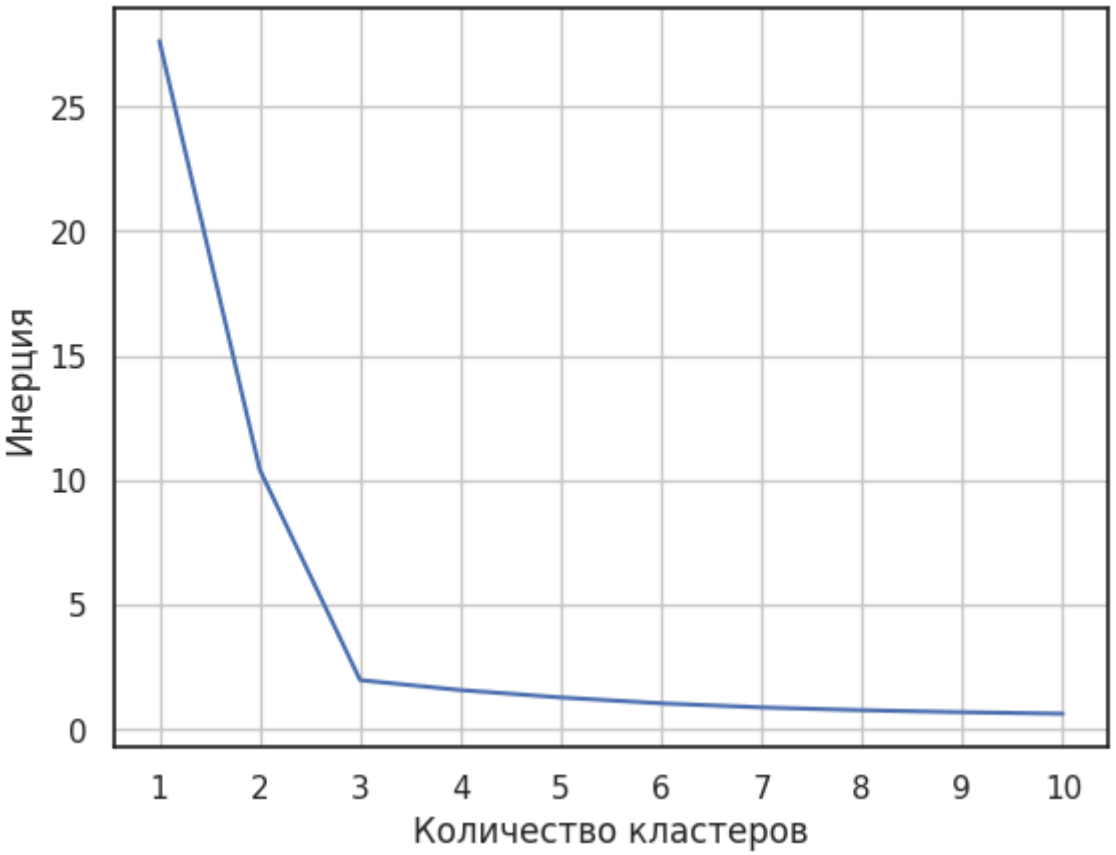

Рисунок 2.1.2 - Метод локтя для luckyset

Для набора данных luckeset оптимальным количеством кластеров будет 3, так как в этой точке скорость убывания графика уменьшается.

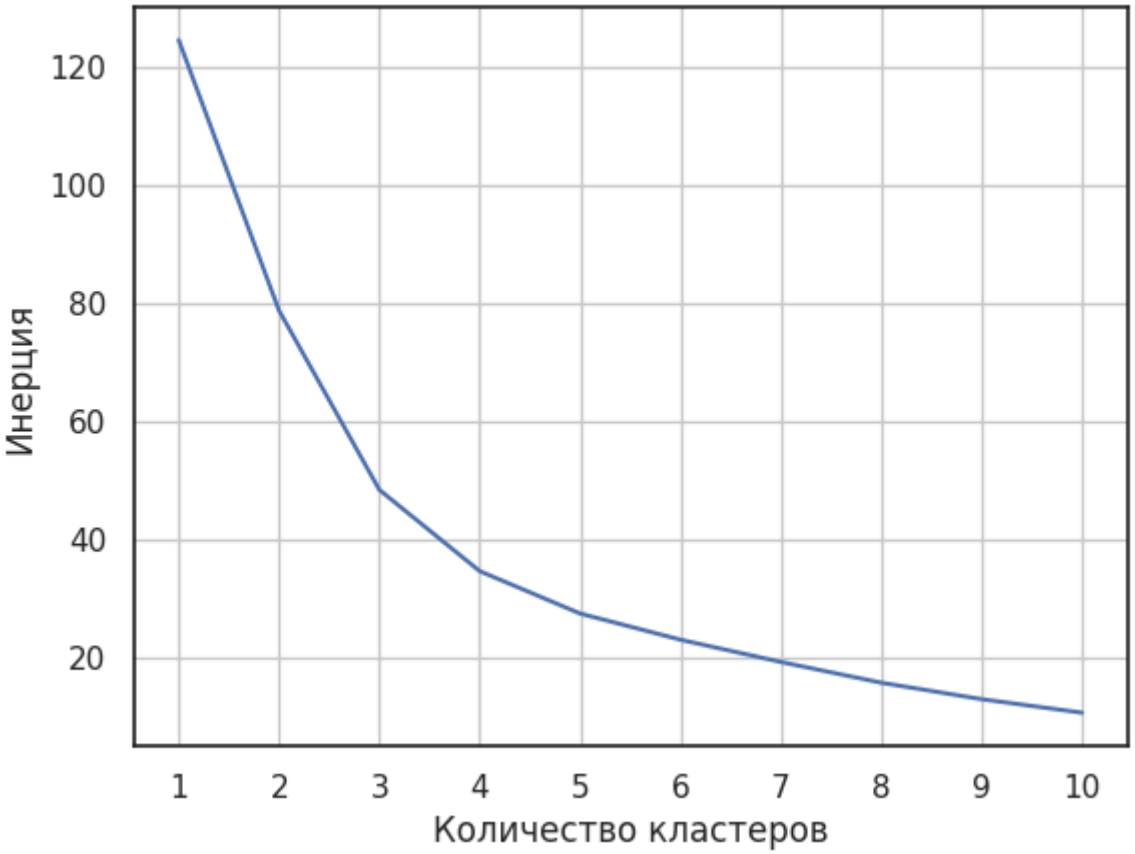

Рисунок 2.1.3 - Метод локтя для circles

Для набора данных circles оптимальным количеством кластеров будет 6-8, так как в этих точках скорость убывания графика уменьшается.

2.2. Проверим исследование оптимального количества кластеров методом силуэта.

Сначала опишем функцию для реализации метода силуэта (см. листинг 2.2). Здесь 10 раз прогоняется алгоритм K-средних, задавая количество кластеров (от 2 до 10, так как при количестве кластеров 1 коэффициент силуэта не будет рассчитан и будет ошибка) и для каждого из кластеров алгоритм KMeans отработает 5 раз и выбирается лучшее (n_init = 5). Далее применяется алгоритм кластеризации, вычисляется и добавляется в список sil_list среднее значение коэффициента силуэта для текущего числа кластеров. Затем отображается график, по которому можно определить оптимальное количество кластеров.

Листинг 2.2 - Реализация метода силуэта

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

Вызовем данную функцию для каждого набора данных (для blobs см. рис. 2.2.1, для luckyset см. рис. 2.2.2, для circles см. рис. 2.2.3).

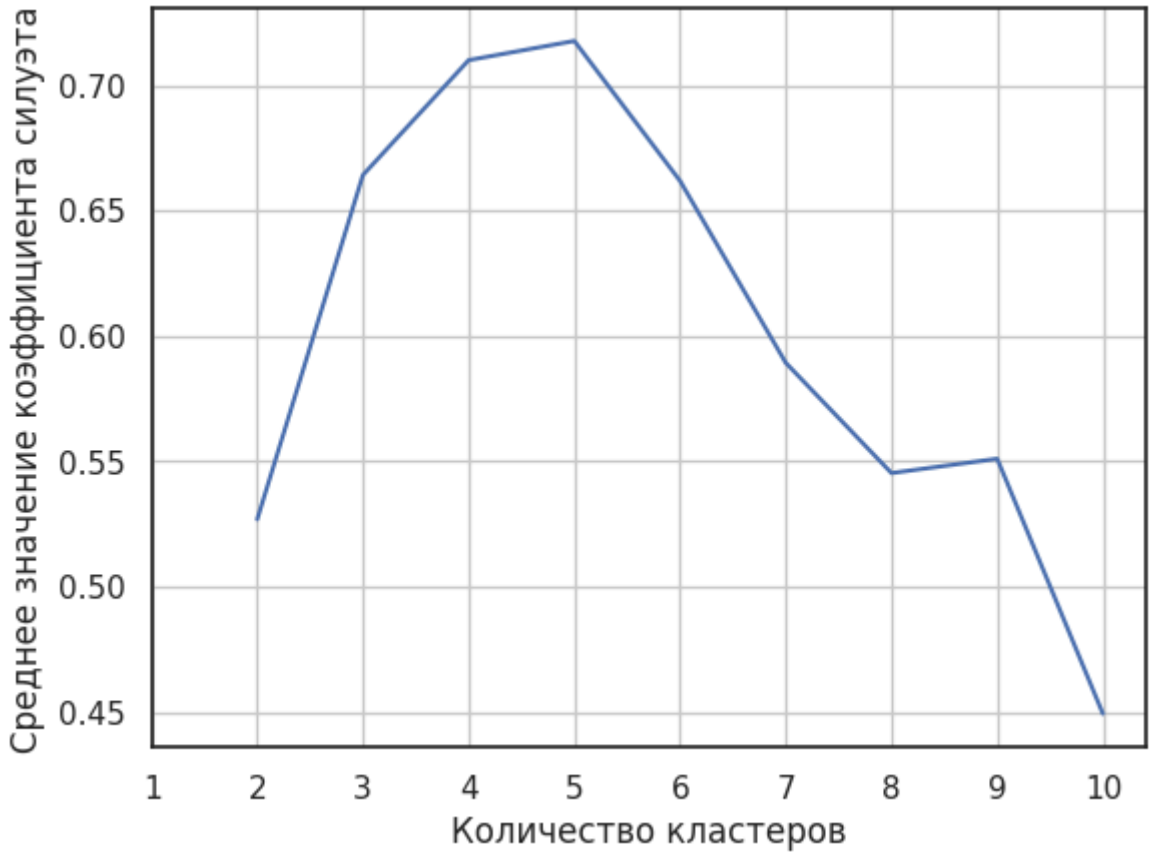

Рисунок 2.2.1 - Метод силуэта для blobs

Для набора данных blobs оптимальным количеством кластеров будет 5, так как среднее значение коэффициента силуэта при данном количестве кластеров максимально. Для данного набора оптимальное количество кластеров методом локтя и методом силуэта совпало.

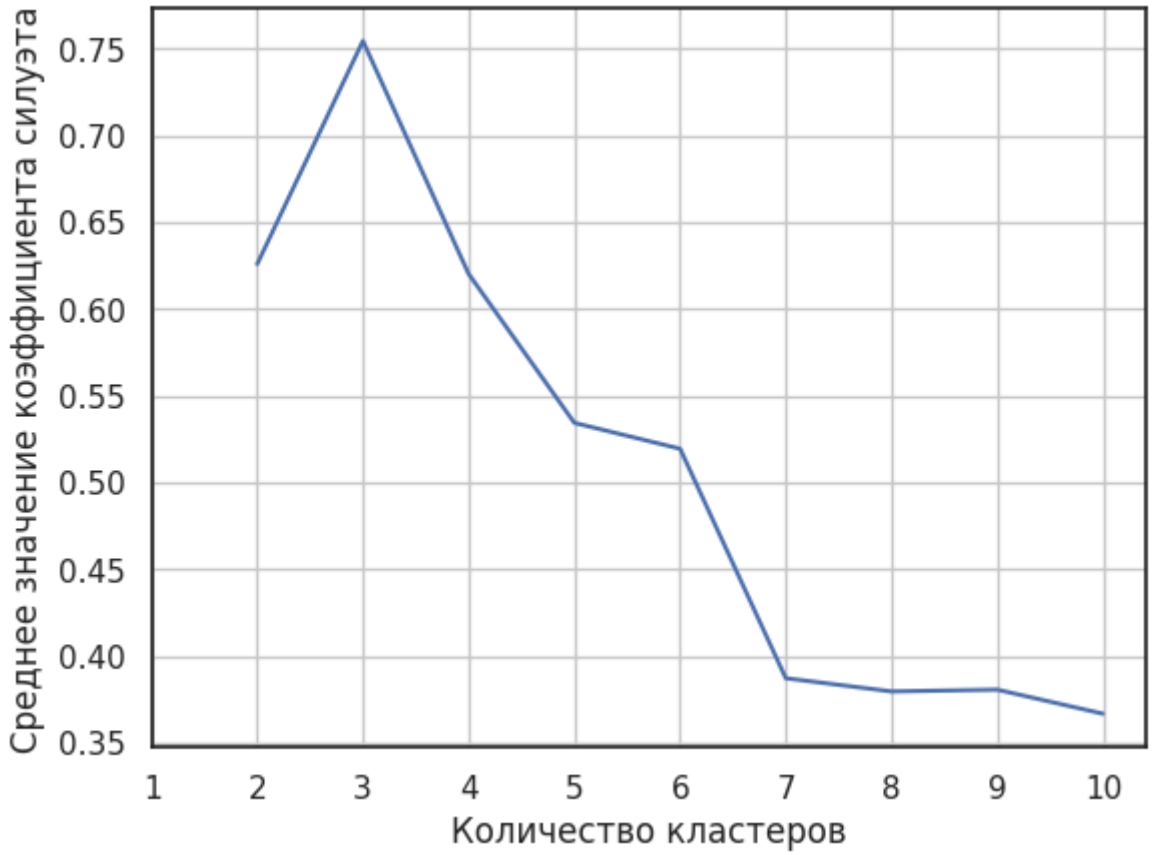

Рисунок 2.2.2 - Метод силуэта для luckyset

Для набора данных luckyset оптимальным количеством кластеров будет 3, так как среднее значение коэффициента силуэта при данном количестве кластеров максимально. Для данного набора оптимальное количество кластеров методом локтя и методом силуэта совпало.

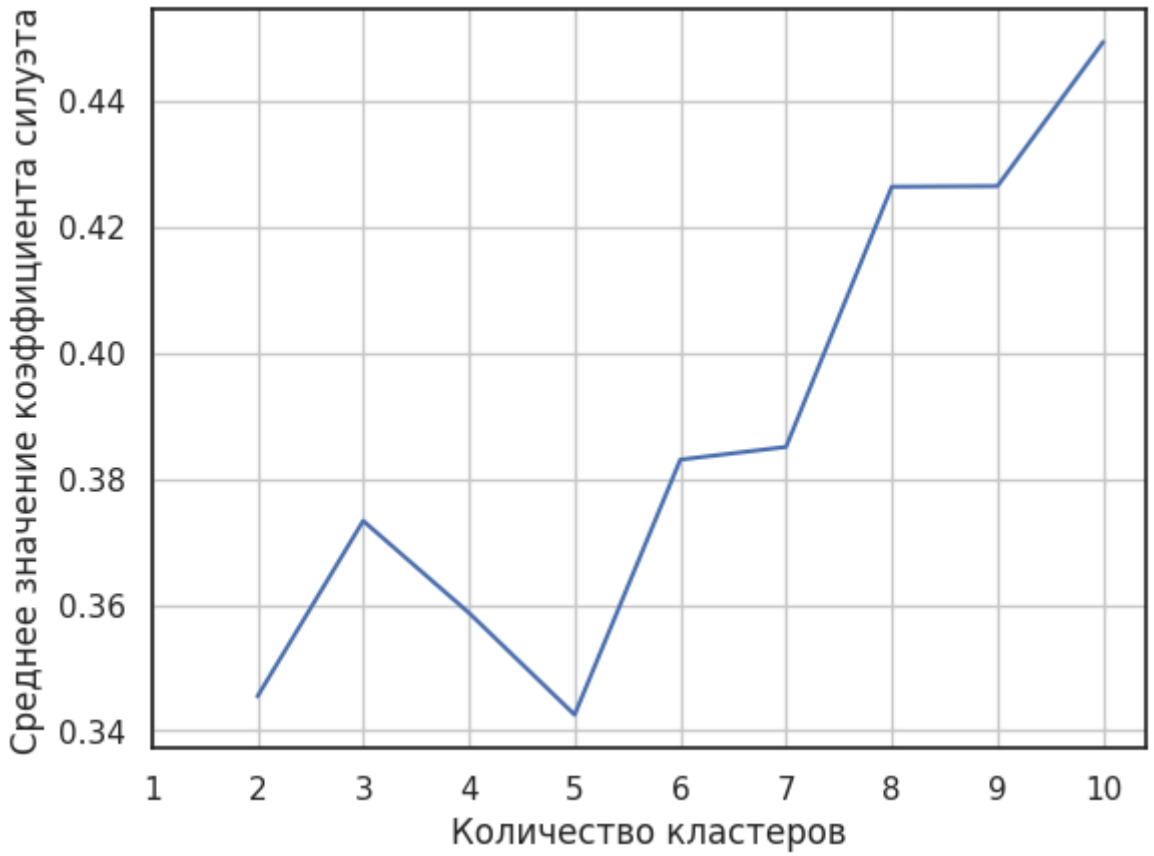

Рисунок 2.2.3 - Метод силуэта для circles

Для набора данных luckyset оптимальным количеством кластеров будет 10 (также есть пики при 3 и 8), так как среднее значение коэффициента силуэта при данном количестве кластеров максимально. Для данного набора оптимальное количество кластеров методом локтя (6-8) и методом силуэта (10, 8, 3. Выберем 8 кластеров.

2.3. Проведем кластеризацию алгоритмом K-means, с выбранным оптимальным количеством кластеров (см. листинг 2.3).

Листинг 2.3 - Кластеризация алгоритмом K-means

```
# Создаем объект KMeans с оптимальным количеством кластеров
kmeans = KMeans(n_clusters=k_optimal)
# Производим кластеризацию
clusters = kmeans.fit_predict(df)
```

2.4. Построим диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров (см. листинг 2.4). Здесь создается объект KMeans с оптимальным количеством кластеров и производится кластеризация. Получаем новый датафрейм из набора данных со столбцами # x, y. После кластеризации к DataFrame norm_df добавляется новый столбец 'cluster', содержащий метки кластеров, которым принадлежат соответствующие точки данных. Затем с помощью библиотеки Seaborn строится диаграмма рассеяния (scatterplot), где каждая точка данных представлена на графике с координатами из столбцов '# x' и 'y'. Цвет каждой точки определяется по меткам кластеров, переданным через аргумент hue='cluster'. Цвета точек выбираются из цветовой палитры 'tab10'.

Листинг 2.4 - Диаграмма рассеяния результатов кластеризации

```
import seaborn as sns

def cluster(k_optimal, df):
  # Создаем объект KMeans с оптимальным количеством кластеров
  kmeans = KMeans(n_clusters=k_optimal)
  # Производим кластеризацию
  clusters = kmeans.fit_predict(df)
  norm_df = pd.DataFrame(df, columns = ['# x', 'y'])
  norm_df['cluster'] = clusters
  sns.scatterplot(norm_df, x = '# x', y = 'y', hue = 'cluster',
palette = 'tab10')
  plt.show()
```

Отображение диаграммы рассеяния результатов кластеризации с выделением разным цветом разных кластеров для blobs смотреть на рисунке 2.4.1, для luckyset смотреть на рисунке 2.4.2, для circles смотреть на рисунке 2.4.3.

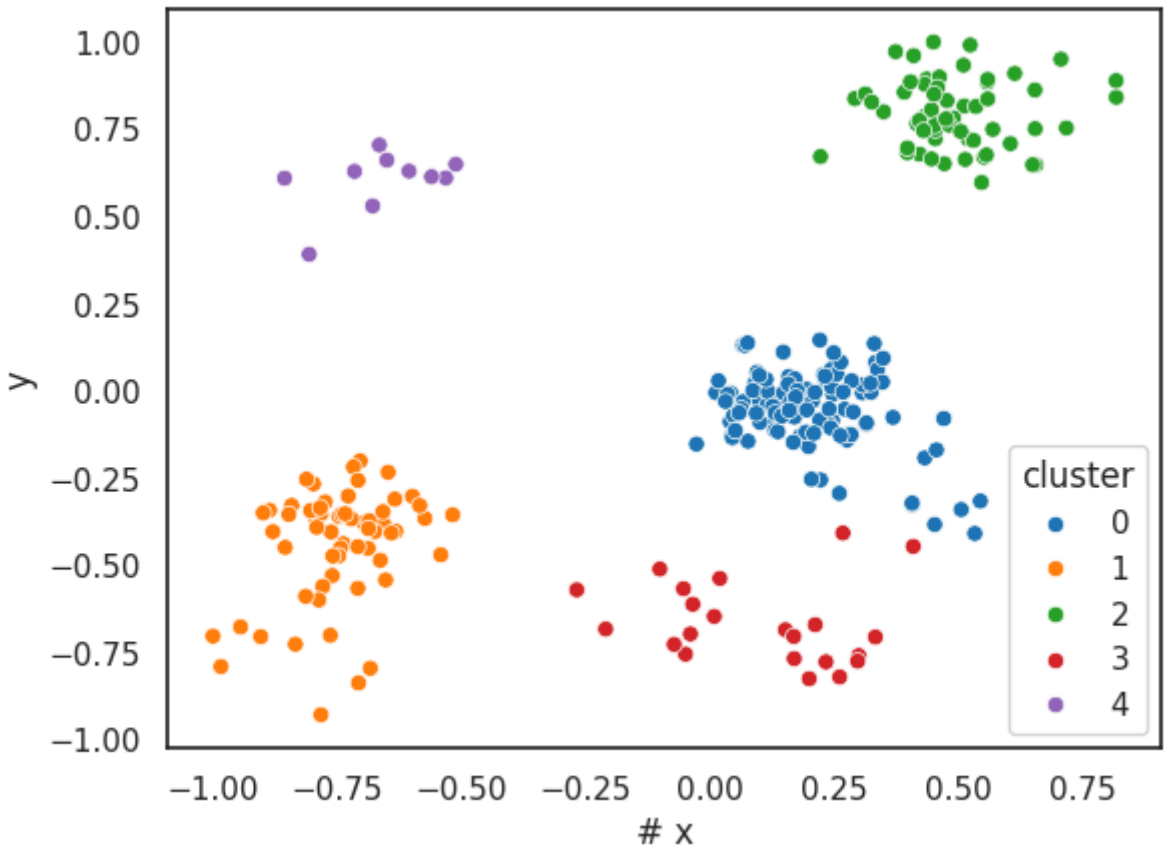

Рисунок 2.4.1 - Диаграмма рассеяния с кластерами для blobs

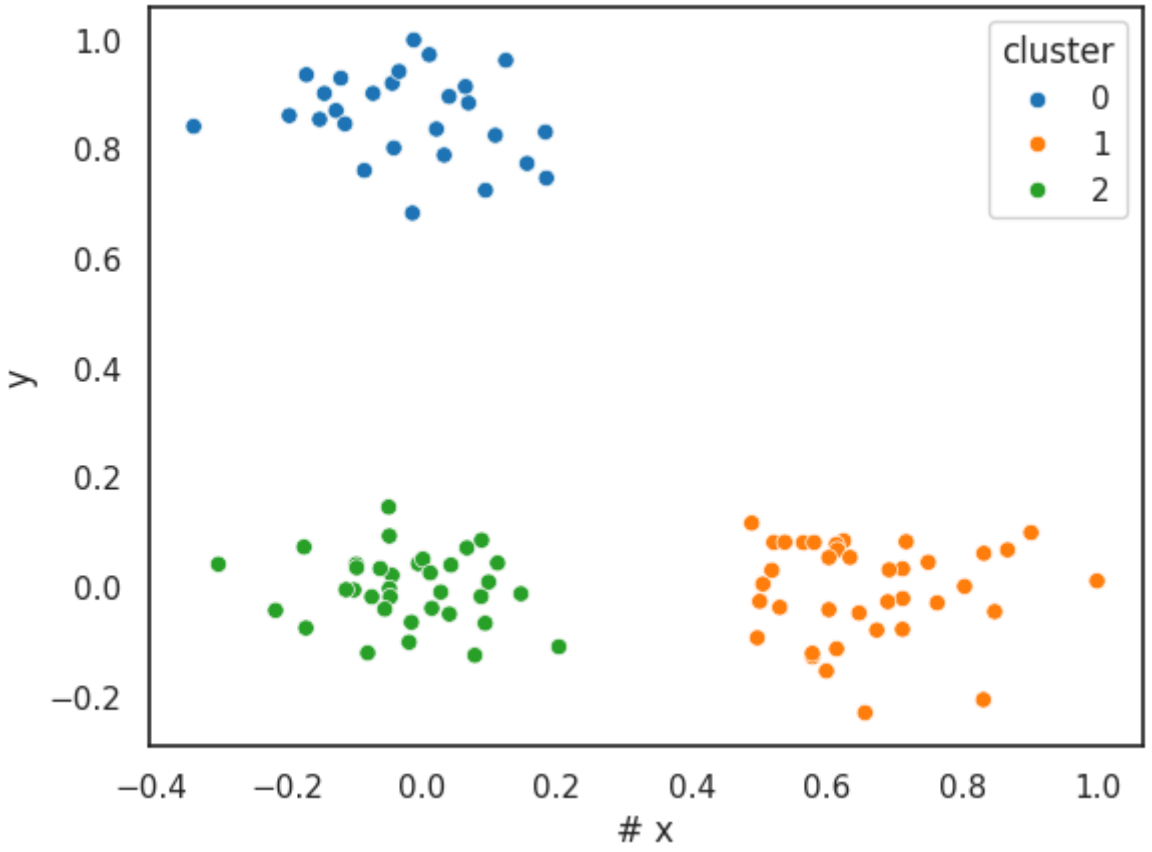

Рисунок 2.4.2 - Диаграмма рассеяния с кластерами для luckyset

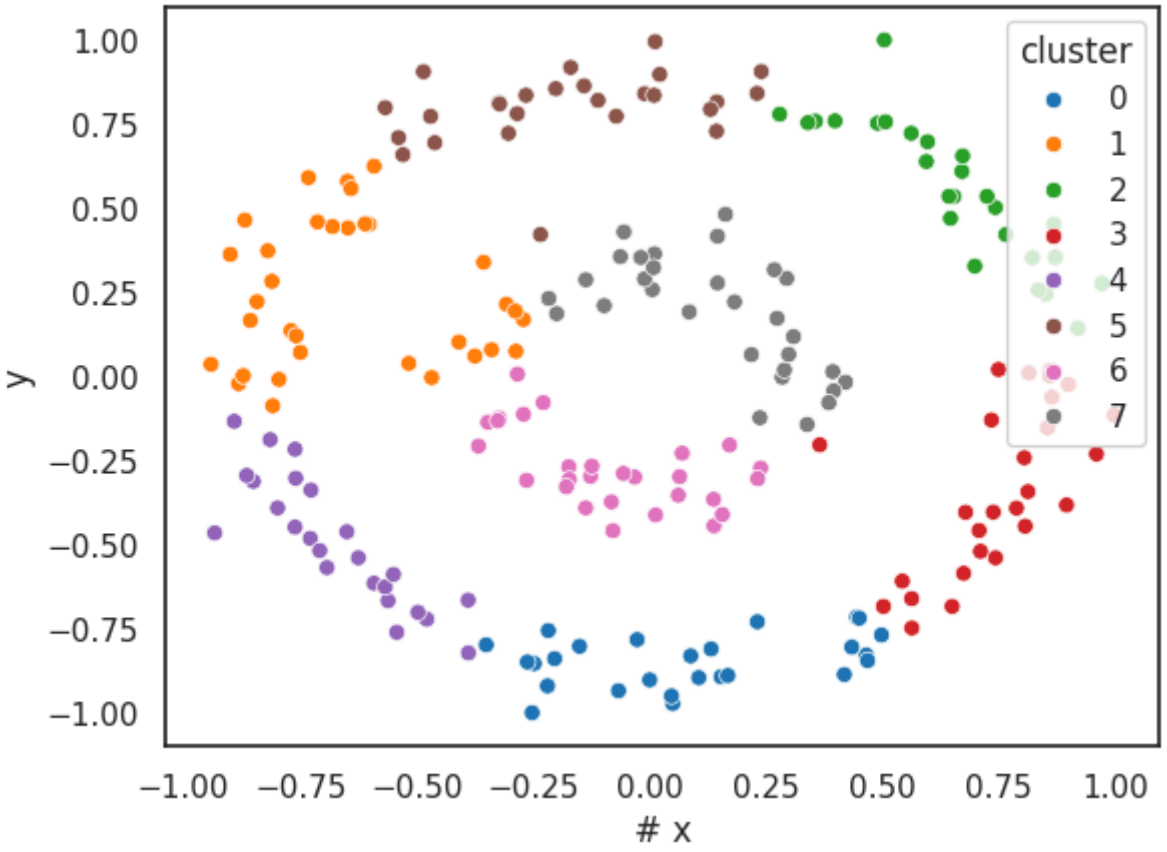

Рисунок 2.4.3 - Диаграмма рассеяния с кластерами для circles

2.5. Напишем функцию для построения диаграммы Вороного для результатов кластеризации (см. листинг 2.5). Здесь kmeans - объект KMeans с оптимальным количеством кластеров. Затем выполняется кластеризация методом KMeans. Получаем центроиды кластеров. Затем начинается построение диаграммы Вороного:

1) Создается сетка значений xx и yy с помощью np.meshgrid(), которая используется для определения областей Вороного (с шагом h).
2) Вызывается kmeans.predict(), чтобы получить метки кластеров для всех точек в сетке, формируя массив z_clust, который представляет собой метки кластеров для каждой точки в сетке.
3) z_clust затем преобразуется в форму, совместимую с размером сетки.
4) plt.imshow(): Рисуется диаграмма Вороного, где каждая область представляет собой кластер. Цвета областей определяются с помощью

цветовой карты plt.cm.Paired. nearest отвечает за отображение без размытия, extent - за размер.

5) plt.plot(): Рисуются точки данных на диаграмме, каждая точка помечена черным цветом и имеет размер 4.
6) plt.scatter(): Рисуются центры кластеров, обозначенные крестиками. Центры кластеров визуализируются белым цветом.
7) plt.show(): Отображает построенную диаграмму.

Этот код предоставляет визуальное представление о распределении кластеров и их границ на плоскости данных.

Листинг 2.5 - Реализация функции для построения диаграммы Вороного

```
def voron(k_optimal, df):
  kmeans = KMeans(n_clusters=k_optimal)
  clusters = kmeans.fit_predict(df)
  norm_cent = kmeans.cluster_centers_

  h = 0.02
  x_min, x_max = df[:, 0].min() - 0.5, df[:, 0].max() + 0.5
  y_min, y_max = df[:, 1].min() - 0.5, df[:, 1].max() + 0.5
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
  z_clust = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
  z_clust = z_clust.reshape(xx.shape)
  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
  plt.plot(df[:, 0], df[:, 1], "k.", markersize=4) #рисуем точки
  plt.scatter(
      norm_cent[:, 0],
      norm_cent[:, 1],
      marker="x",
      s=169,
      linewidths=3,
      color="w",
      zorder=10,
  )
  plt.show()
```

Диаграммы Вороного для blobs сомотреть на рисунке 2.5.1, для luckyset смотреть на рисунке 2.5.2, для circles смотреть на рисунке 2.5.3.

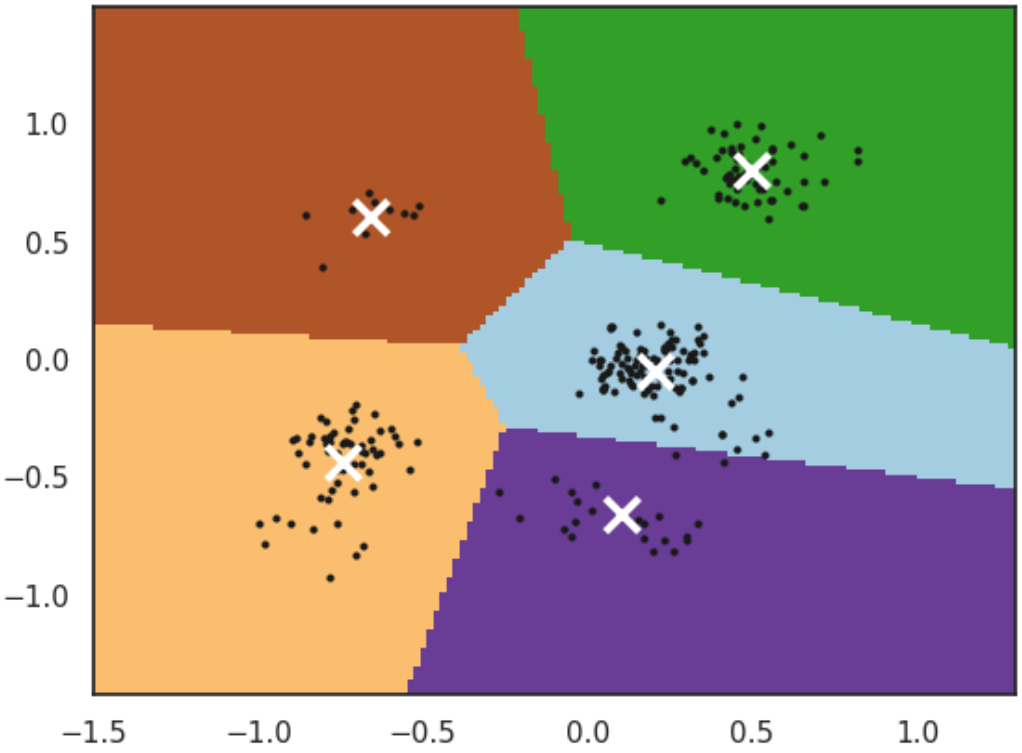

Рисунок 2.5.1 - Диаграмма Вороного для blobs

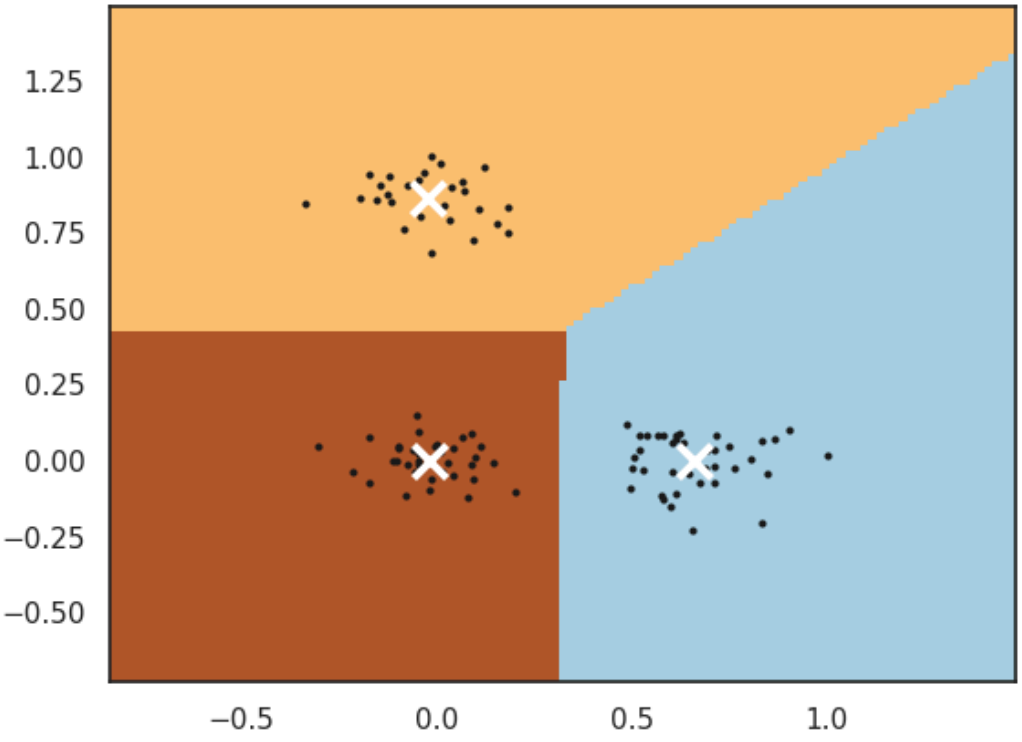

Рисунок 2.5.2 - Диаграмма Вороного для luckyset

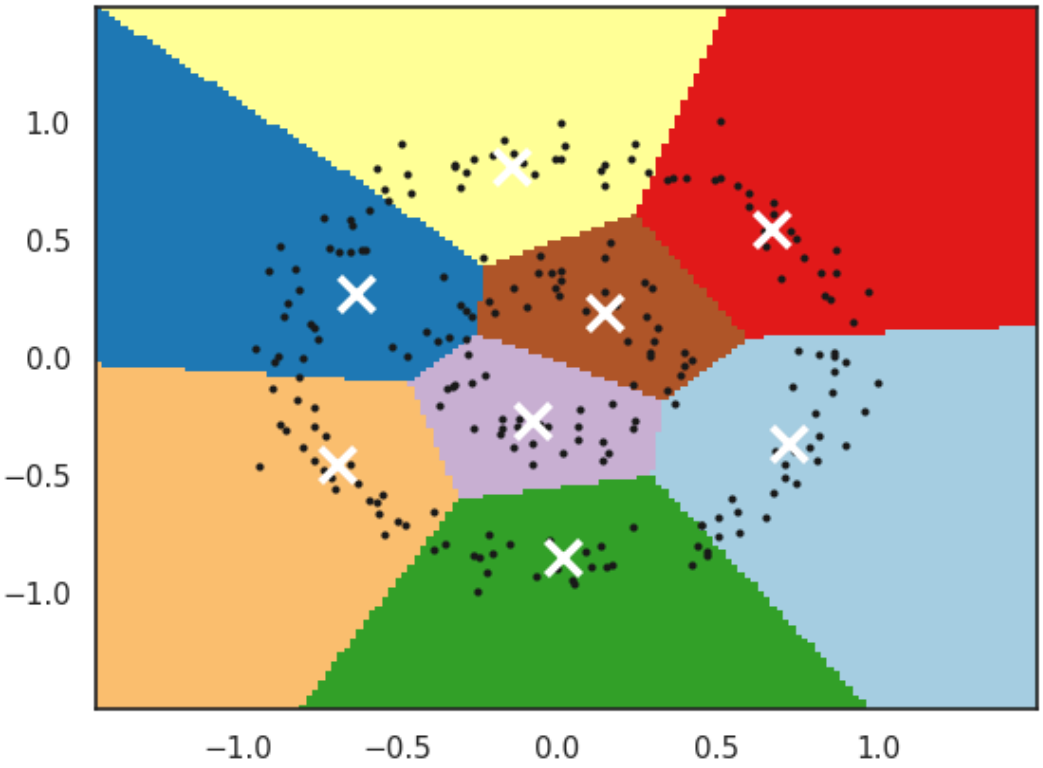

Рисунок 2.5.3 - Диаграмма Вороного для circles

Диаграмма Вороного представляет разбиение плоскости таким образом, что в образуются множества с близкими точками к одному из элементов этого множества. По построенным диаграммам Вороного можно увидеть, где области разных кластеров соприкасаются.

2.6. Построим для каждого признака диаграмму “violin-plot” с разделением по кластерам (см. листинг 2.6).

Листинг 2.6 - Построение диаграммы “violin-plot” для каждого признака

```
def violin_plots(k_optimal, df):
  kmeans = KMeans(n_clusters=k_optimal)
  clusters = kmeans.fit_predict(df)
  norm_df = pd.DataFrame(df, columns = ['# x', 'y'])
  norm_df['cluster'] = clusters

  # Построение violin plot для признака 'x' с разделением по
```

```
кластерам
  sns.violinplot(data=norm_df, x='# x', hue='cluster',
palette='tab10')
  plt.title('Violin Plot для признака x с разделением по
кластерам')
  plt.xlabel('# x')
  plt.show()

  # Построение violin plot для признака 'y' с разделением по
кластерам
  sns.violinplot(data=norm_df, x='y', hue='cluster',
palette='tab10')
  plt.title('Violin Plot для признака y с разделением по
кластерам')
  plt.xlabel('y')
  plt.show()
```

Результат работы функции для blobs смотреть на рисунках 2.6.1 и 2.6.2, для luckyset - на рисунках 2.6.3 и 2.6.4, для circles - на рисунках 2.6.5 и 2.6.6.

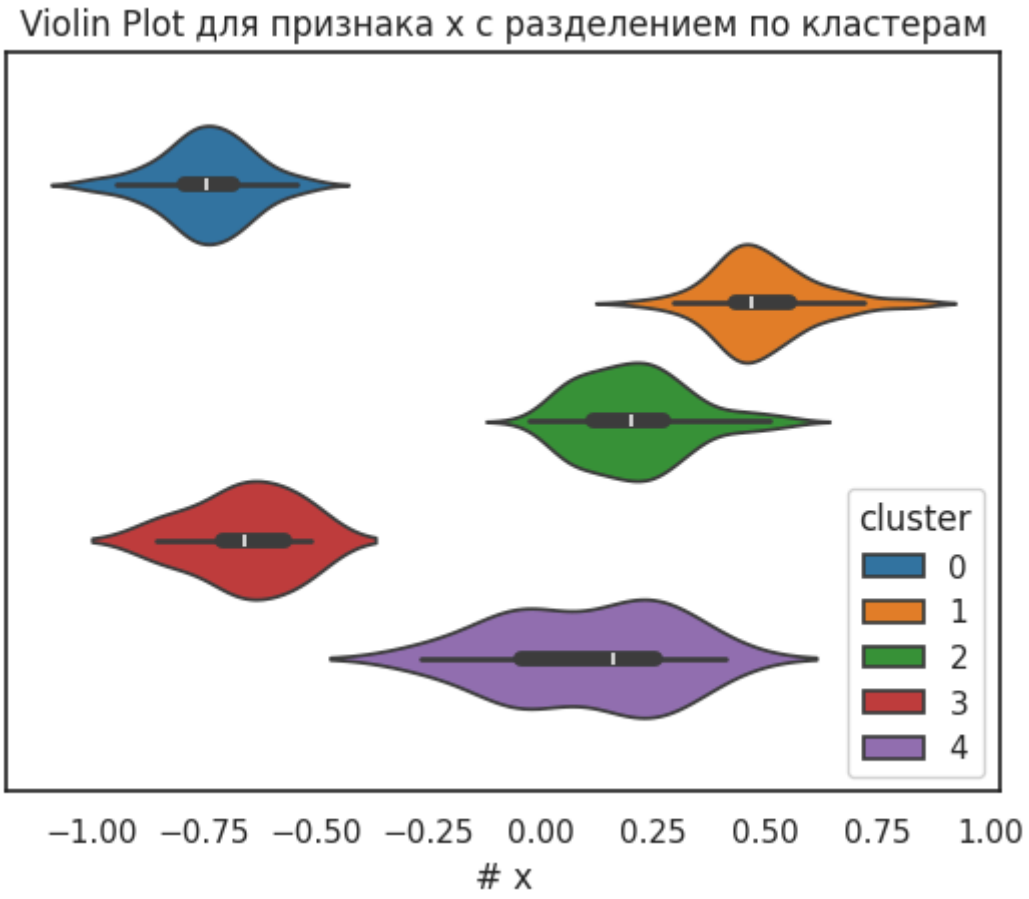

Рисунок 2.6.1 - “violin-plot” # x для blobs

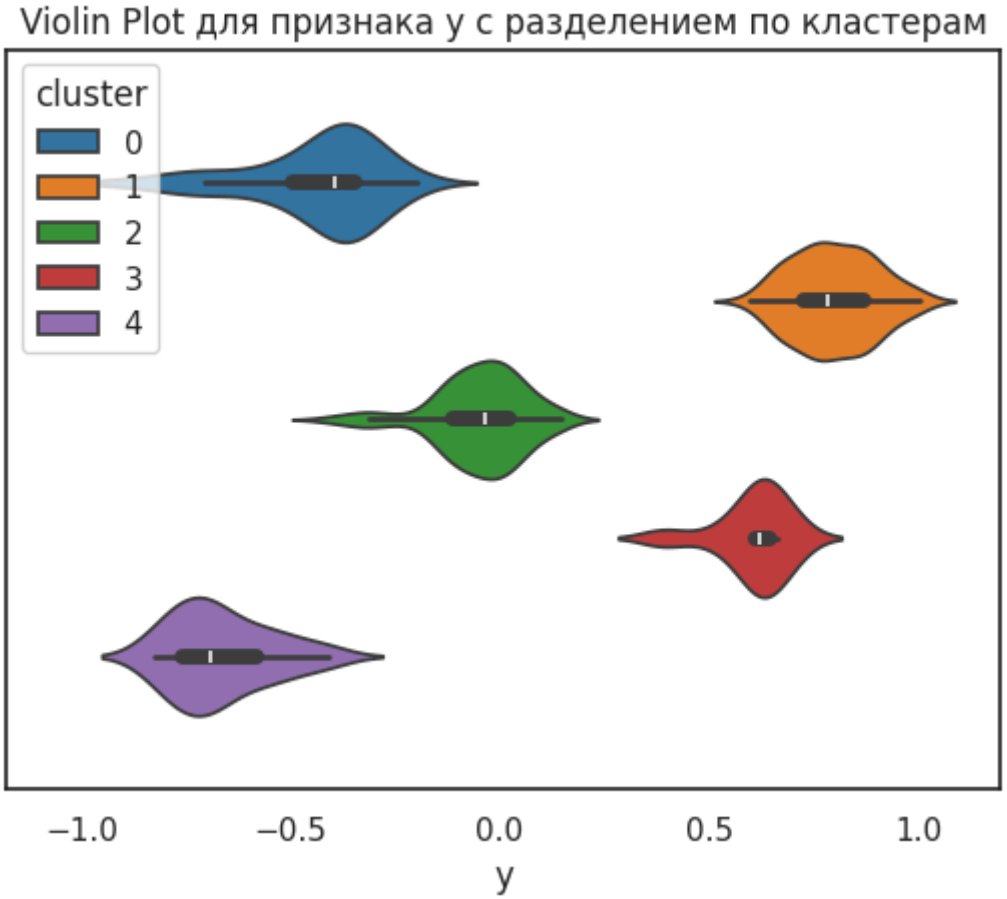

Рисунок 2.6.2 - “violin-plot” y для blobs

По двум рисункам 2.6.1 и 2.6.2 можно заметить, что данные хорошо разделены, не считая небольшого количества на границах. Кластеризация K-means к набору данных blobs успешна применена.

Рисунок 2.6.3 - “violin-plot” # x для luckyset

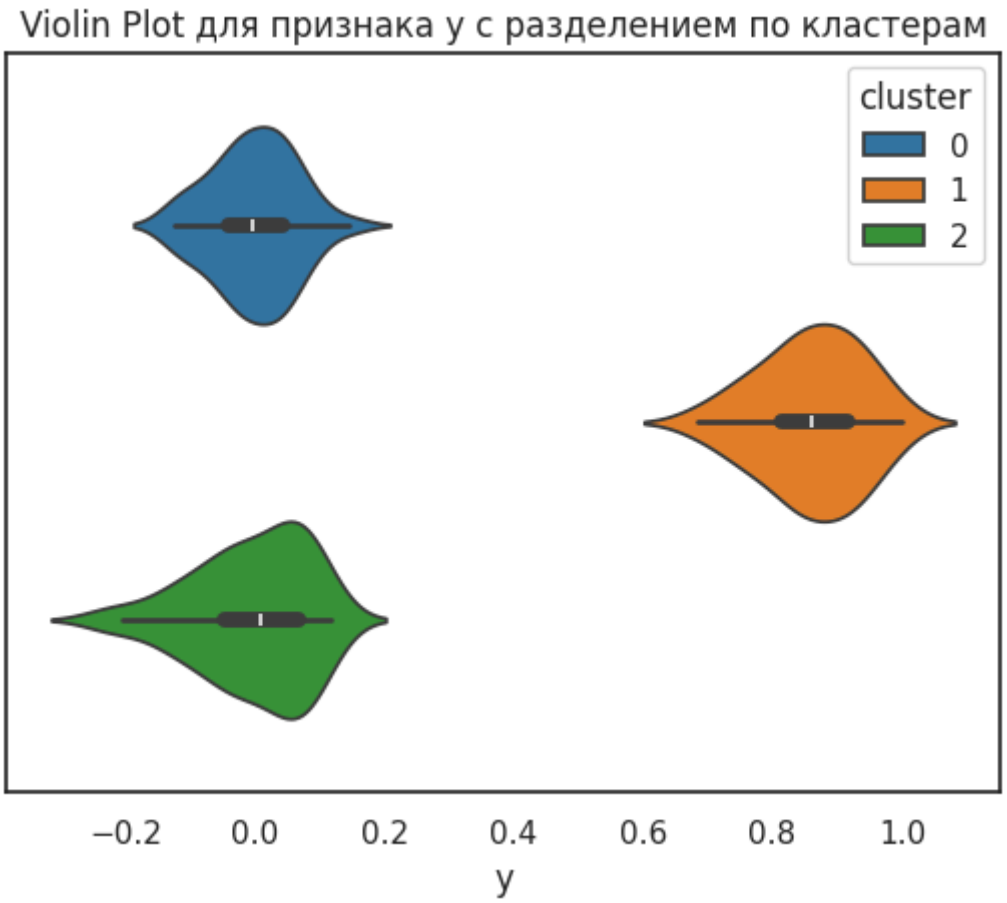

Рисунок 2.6.4 - “violin-plot” y для luckyset

По двум рисункам 2.6.3 и 2.6.4 можно заметить, что данные четко разделены. Кластеризация K-means к набору данных luckyset успешна применена.

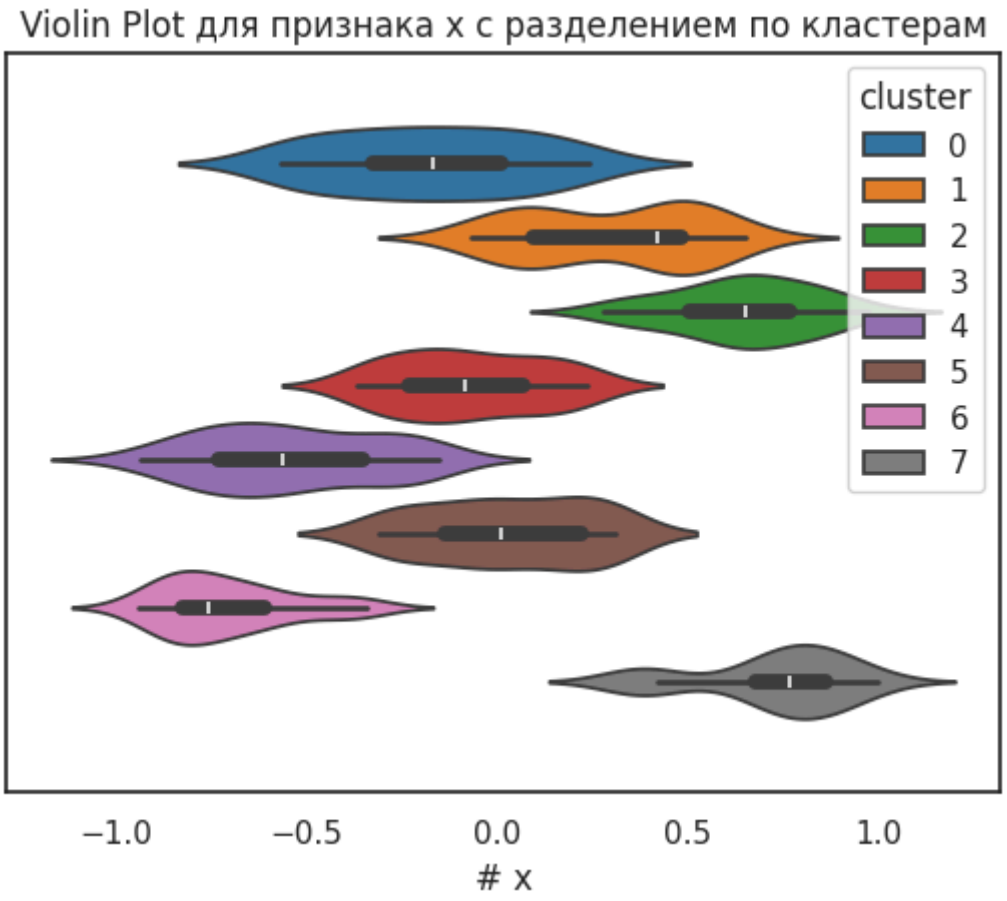

Рисунок 2.6.5 - “violin-plot” # x для circles

Рисунок 2.6.6 - “violin-plot” y для circles

По двум рисункам 2.6.5 и 2.6.6 можно заметить, что данные хорошо разделены, однако не так четко как на рисунках 2.6.3 и 2.6.4, небольшое количество на границах все-таки пересекается. Но можно сказать, что кластеризация K-means к набору данных circles успешна применена.

2.7. Рассчитаем для каждого кластера количество точек, среднее, СКО, минимум и максимум (см. листинг 2.7). Создается объект KMeans с указанным числом кластеров k_optimal. Производится кластеризация данных методом KMeans. Создается DataFrame norm_df, в который также добавляется столбец 'cluster', содержащий номер кластера для каждой точки. Для каждого кластера выделяются данные из norm_df, относящиеся к этому кластеру, с помощью метода loc (фильтрация) и выводятся статистические характеристики данных с помощью метода describe(). Это включает в себя количество точек (count),

среднее значение (mean), стандартное отклонение (std), минимальное (min) и максимальное (max) значения, а также квартили данных.

Листинг 2.7 - Расчет характеристик для каждого класса

```
def my_describe(k_optimal, df):
  kmeans = KMeans(n_clusters=k_optimal)
  clusters = kmeans.fit_predict(df)
  norm_df = pd.DataFrame(df, columns = ['# x', 'y'])
  norm_df['cluster'] = clusters

  for i in range(k_optimal):
    df_ = norm_df.loc[(norm_df['cluster'] == i)]
    print(df_.describe())
```

Характеристики для 5 кластеров набора данных blobs представлены на рисунках 2.7.1-2.7.5.

	# x	y	cluster
count	108.000000	108.000000	108.0
mean	0.207156	-0.052098	0.0
std	0.121496	0.111806	0.0
min	-0.026006	-0.408231	0.0
25%	0.116479	-0.113345	0.0
50%	0.199573	-0.034617	0.0
75%	0.274150	0.021946	0.0
max	0.546012	0.145934	0.0

Рисунок 2.7.1 - Характеристики первого кластера blobs

	# x	y	cluster
count	60.000000	60.000000	60.0
mean	-0.744457	-0.442389	1.0
std	0.100434	0.162729	0.0
min	-1.000000	-0.927398	1.0
25%	-0.794022	-0.495824	1.0
50%	-0.741532	-0.392398	1.0
75%	-0.684403	-0.344819	1.0
max	-0.517065	-0.200030	1.0

Рисунок 2.7.2 - Характеристики второго кластера blobs

	# x	y	cluster
count	60.000000	60.000000	60.0
mean	0.495273	0.796237	2.0
std	0.114945	0.096912	0.0
min	0.223920	0.597407	2.0
25%	0.432439	0.722644	2.0
50%	0.468503	0.784115	2.0
75%	0.554524	0.869938	2.0
max	0.819831	1.000000	2.0

Рисунок 2.7.3 - Характеристики третьего кластера blobs

	# x	y	cluster
count	10.000000	10.000000	10.0
mean	-0.657565	0.603129	3.0
std	0.113105	0.086650	0.0
min	-0.855771	0.391598	3.0
25%	-0.705681	0.609943	3.0
50%	-0.657168	0.621622	3.0
75%	-0.570904	0.644745	3.0
max	-0.510743	0.704815	3.0

Рисунок 2.7.4 - Характеристики четвертого кластера blobs

	# x	y	cluster
count	22.000000	22.000000	22.0
mean	0.101515	-0.664923	4.0
std	0.187210	0.116720	0.0
min	-0.267056	-0.824247	4.0
25%	-0.045582	-0.756933	4.0
50%	0.161634	-0.689958	4.0
75%	0.255725	-0.579953	4.0
max	0.410761	-0.406833	4.0

Рисунок 2.7.5 - Характеристики пятого кластера blobs

Для четвертого и пятого кластеров (так как точек немного) можно убедиться, что количество точек совпало с количеством точек на графике. Также для всех кластеров видно, что минимальное и максимальное значения совпали. Оценить СКО и среднее значение для всех кластеров по графику сложно. Можно сделать вывод, что график не достаточен для описания всех характеристик кластеров, но очень удобен для визуализации.

Характеристики для 3 кластеров набора данных luckyset представлены на рисунках 2.7.6-2.7.8.

	# x	y	cluster
count	38.000000	38.000000	38.0
mean	0.661117	-0.005919	0.0
std	0.126125	0.087973	0.0
min	0.488507	-0.229840	0.0
25%	0.578448	-0.046922	0.0
50%	0.629470	0.008260	0.0
75%	0.716085	0.067912	0.0
max	1.000000	0.116848	0.0

Рисунок 2.7.6 - Характеристики первого кластера luckyset

	# x	y	cluster
count	35.000000	35.000000	35.0
mean	-0.019961	-0.001883	1.0
std	0.105778	0.063676	0.0
min	-0.301267	-0.124881	1.0
25%	-0.077103	-0.041592	1.0
50%	-0.018810	-0.004922	1.0
75%	0.054977	0.041565	1.0
max	0.203042	0.146088	1.0

Рисунок 2.7.7 - Характеристики второго кластера luckyset

	# x	y	cluster
count	27.000000	27.000000	27.0
mean	-0.021190	0.860061	2.0
std	0.126333	0.079213	0.0
min	-0.337864	0.683519	2.0
25%	-0.116882	0.814179	2.0
50%	-0.013877	0.861820	2.0
75%	0.066899	0.917762	2.0
max	0.184559	1.000000	2.0

Рисунок 2.7.8 - Характеристики третьего кластера luckyset

Для всех кластеров можно убедиться, что количество точек совпало с количеством точек на графике и минимальное и максимальное значения также совпали для всех кластеров. Оценить СКО и среднее значение по графику сложно. Можно сделать вывод, что график не достаточен для описания всех характеристик кластеров, но очень удобен для визуализации.

Характеристики для 8 кластеров набора данных circles представлены на рисунках 2.7.9-2.7.16.

	# x	y	cluster
count	29.000000	29.000000	29.0
mean	-0.087451	-0.259629	0.0
std	0.186297	0.130814	0.0
min	-0.366240	-0.456560	0.0
25%	-0.262993	-0.351152	0.0
50%	-0.122660	-0.294858	0.0
75%	0.066075	-0.202216	0.0
max	0.240010	0.076152	0.0

Рисунок 2.7.9 - Характеристики первого кластера circles

	# x	y	cluster
count	27.000000	27.000000	27.0
mean	-0.157166	0.799970	1.0
std	0.259329	0.106130	0.0
min	-0.566641	0.422518	1.0
25%	-0.321450	0.773493	1.0
50%	-0.168277	0.813652	1.0
75%	0.018325	0.848493	1.0
max	0.281013	0.995124	1.0

Рисунок 2.7.10 - Характеристики второго кластера circles

	# x	y	cluster
count	27.000000	27.000000	27.0
mean	0.747786	-0.329938	2.0
std	0.146718	0.247012	0.0
min	0.367111	-0.745213	2.0
25%	0.678332	-0.527305	2.0
50%	0.751197	-0.380142	2.0
75%	0.859015	-0.120523	2.0
max	1.000000	0.022074	2.0

Рисунок 2.7.11 - Характеристики третьего кластера circles

	# x	y	cluster
count	24.000000	24.000000	24.0
mean	-0.671516	-0.490363	3.0
std	0.154173	0.189766	0.0
min	-0.933142	-0.819149	3.0
25%	-0.769870	-0.632957	3.0
50%	-0.699617	-0.497207	3.0
75%	-0.557717	-0.329300	3.0
max	-0.387917	-0.131738	3.0

Рисунок 2.7.12 - Характеристики четвертого кластера circles

	# x	y	cluster
count	30.000000	30.000000	30.0
mean	0.146699	0.185768	4.0
std	0.189262	0.172781	0.0
min	-0.215200	-0.141223	4.0
25%	-0.005267	0.031959	4.0
50%	0.174153	0.217110	4.0
75%	0.294768	0.311636	4.0
max	0.422478	0.482535	4.0

Рисунок 2.7.13 - Характеристики пятого кластера circles

	# x	y	cluster
count	26.000000	26.000000	26.0
mean	0.084437	-0.842199	5.0
std	0.275643	0.076804	0.0
min	-0.349700	-0.996631	5.0
25%	-0.189257	-0.891910	5.0
50%	0.070427	-0.839007	5.0
75%	0.373139	-0.796188	5.0
max	0.499695	-0.712589	5.0

Рисунок 2.7.14 - Характеристики шестого кластера circles

	# x	y	cluster
count	25.000000	25.000000	25.0
mean	0.669954	0.541220	6.0
std	0.174101	0.210623	0.0
min	0.341604	0.144947	6.0
25%	0.563594	0.355142	6.0
50%	0.672673	0.535816	6.0
75%	0.824323	0.723138	6.0
max	0.973709	1.000000	6.0

Рисунок 2.7.15 - Характеристики седьмого кластера circles

	# x	y	cluster
count	32.000000	32.000000	32.0
mean	-0.651647	0.248665	7.0
std	0.204566	0.210786	0.0
min	-0.941151	-0.085461	7.0
25%	-0.812440	0.070191	7.0
50%	-0.696178	0.205541	7.0
75%	-0.503765	0.447939	7.0
max	-0.269783	0.625532	7.0

Рисунок 2.7.16 - Характеристики восьмого кластера circles

Для всех кластеров можно убедиться, что количество точек совпало с количеством точек на графике и минимальное и максимальное значения также совпали для всех кластеров. Оценить СКО и среднее значение по графику сложно. Можно сделать вывод, что график не достаточен для описания всех характеристик кластеров, но очень удобен для визуализации.

3. DBSCAN:

3.1. Подберем параметры алгоритма DBSCAN, которые дадут наилучшие результаты. Сначала напишем функцию для реализации алгоритма DBSCAN (см. листинг 3.1). Функция dbscan принимает три аргумента: data (набор данных для кластеризации), eps (максимальное расстояние между двумя точками, чтобы они считались соседями) и min_samples (минимальное количество точек, необходимое для образования плотного кластера). Создается объект DBSCAN с заданными параметрами eps и min_samples. Производится кластеризация данных методом DBSCAN, и каждой точке данных назначается метка кластера. Результаты сохраняются в переменной dbscan_clust. Далее происходит визуализация кластеров. Каждый кластер (с метками от 0 до n-1 и -1) отображается точками разного цвета ('r', 'b', 'g', 'y', 'purple'), а выбросы (с меткой -1) отображаются крестиками черного цвета ('k'). Точки каждого кластера отбираются из массива data в зависимости от их метки из dbscan_clust.

Листинг 3.1 - Реализация алгоритма DBSCAN

```
from sklearn.cluster import KMeans, DBSCAN

def dbscan(data, eps, min_samples):
  dbscan = DBSCAN(eps = eps, min_samples = min_samples)
  dbscan_clust = dbscan.fit_predict(data)
  plt.scatter(data[dbscan_clust == 0,0], data[dbscan_clust ==
0,1], c = 'r')
  plt.scatter(data[dbscan_clust == 1,0], data[dbscan_clust ==
1,1], c = 'b')
  plt.scatter(data[dbscan_clust == 2,0], data[dbscan_clust ==
2,1], c = 'g')
```

```
  plt.scatter(data[dbscan_clust == 3,0], data[dbscan_clust ==
3,1], c = 'y')
  plt.scatter(data[dbscan_clust == 4,0], data[dbscan_clust ==
4,1], c = 'purple')
  plt.scatter(data[dbscan_clust == -1,0], data[dbscan_clust ==
-1,1], c ='k', marker = 'x')
  plt.show()
```

Выбор параметров для blobs представлен на рисунках 3.1.1-3.1.4.

Первые параметры выбраны случайным образом: eps = 0.08, min_samples = 6.

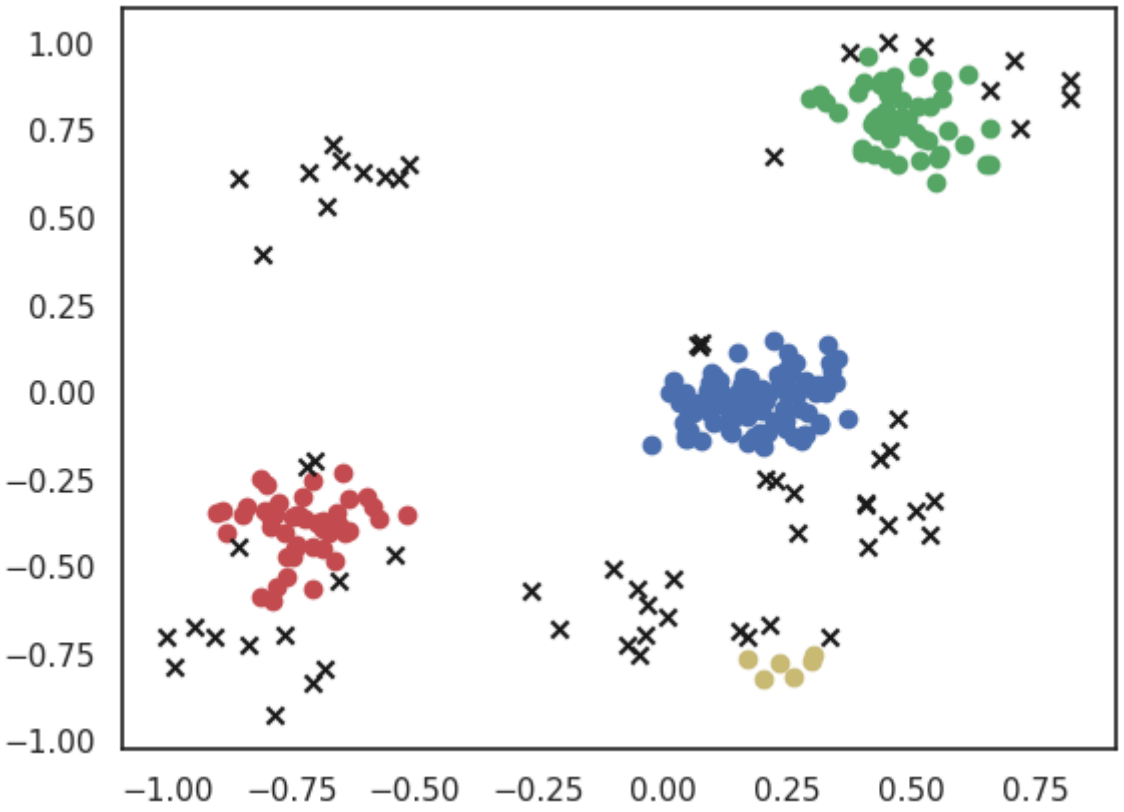

Рисунок 3.1.1 - eps = 0.08, min_samples = 6 для blobs

Анализируя данный график можно заметить, что 0.08 слишком маленький параметр, так как очень много точек являются шумами. Увеличим параметр eps до 0.1 (см. рис. 3.1.2).

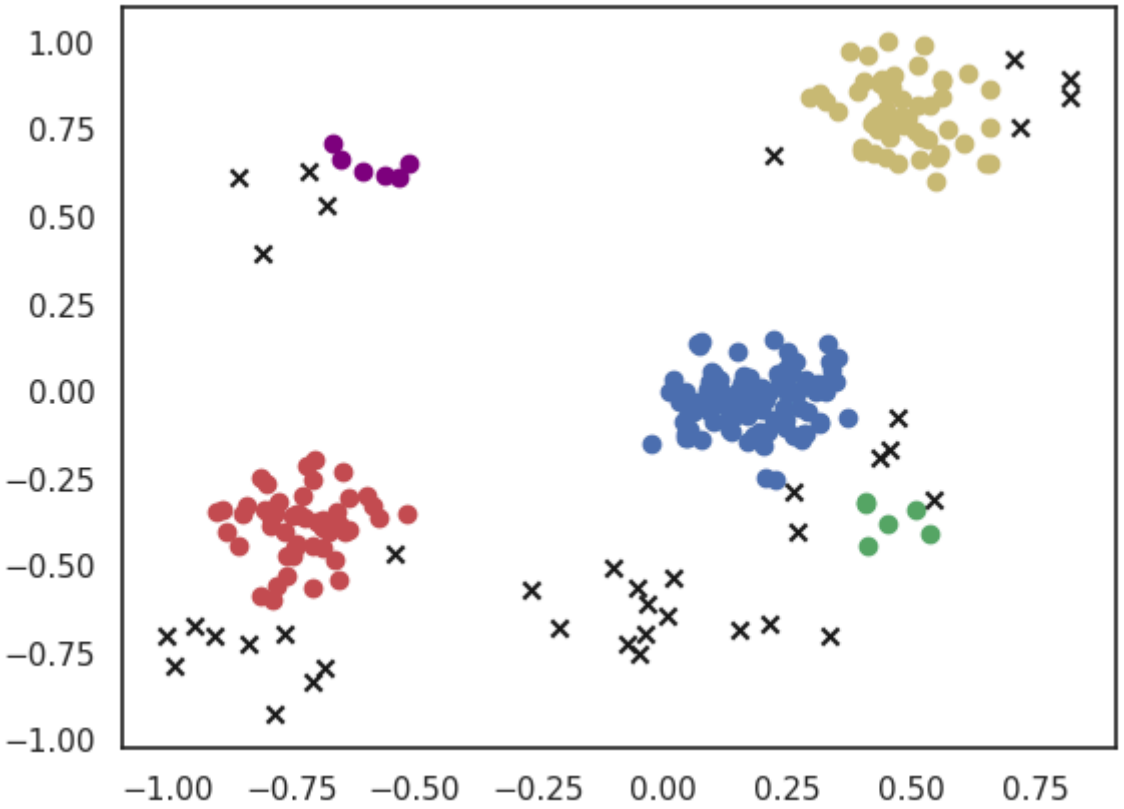

Рисунок 3.1.2 - eps = 0.1, min_samples = 6 для blobs

Анализируя данный график можно заметить, что с увеличением eps до 0.1, количество кластеров увеличилось с 4 до 5, но все равно очень много точек являются шумами. Увеличим параметр eps в 2 раза (см. рис. 3.1.3).

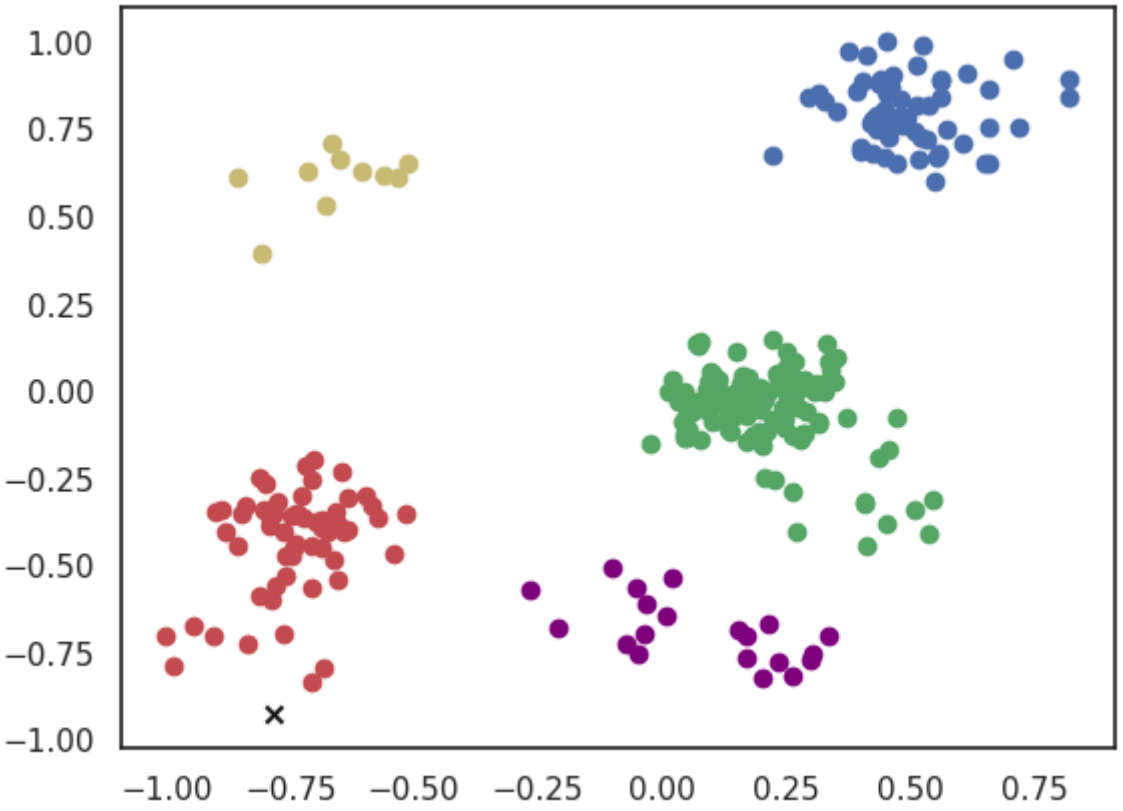

Рисунок 3.1.3 - eps = 0.2, min_samples = 6 для blobs

Анализируя данный график можно заметить, что с увеличением eps до 0.2, количество кластеров 5 (как и должно быть) и лишб одна точка является выбросом. Уменьшим параметр min_samples на 1 (см. рис. 3.1.4).

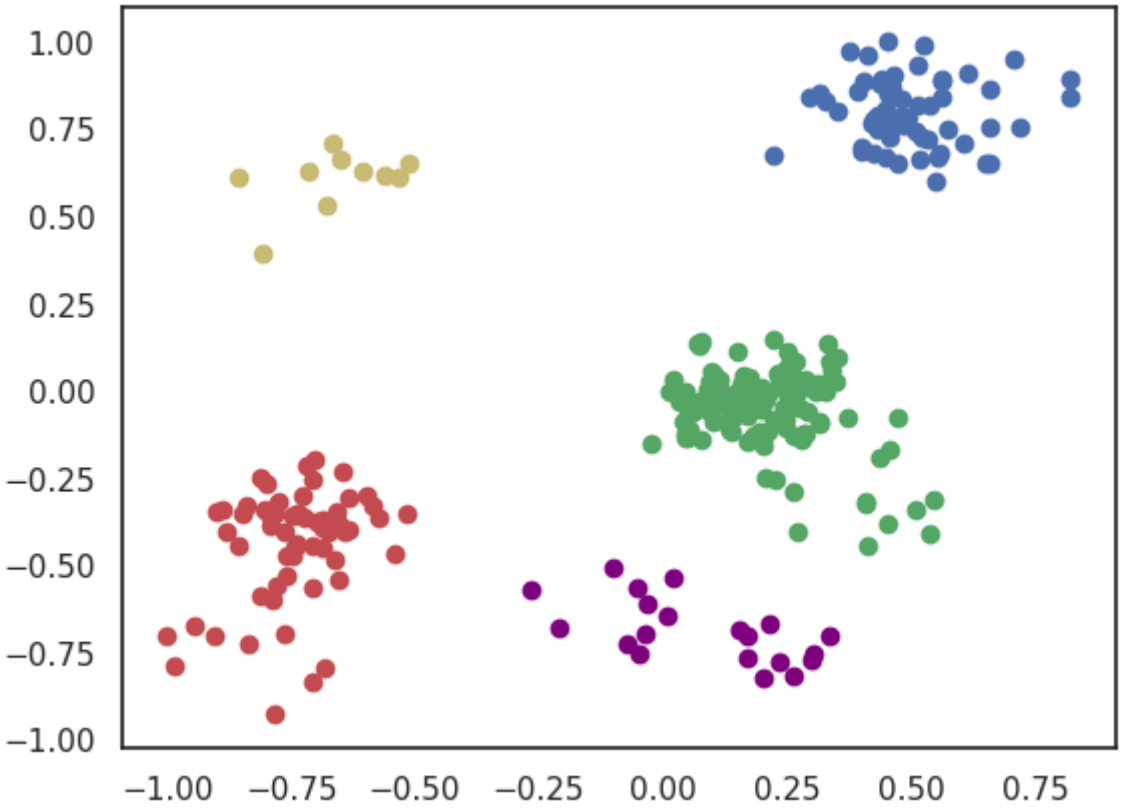

Рисунок 3.1.4 - eps = 0.2, min_samples = 5 для blobs

Анализируя данный график можно заметить, что при eps=0.2 и min_samples=5, количество кластеров 5 и нет выбросов. Подходящие параметры для blobs найдены.

Выбор параметров для luckyset представлен на рисунках 3.1.5-3.1.6.

Первые параметры выбраны случайным образом (такие же как и для blobs): eps = 0.08, min_samples = 6 (см. рис. 3.1.5).

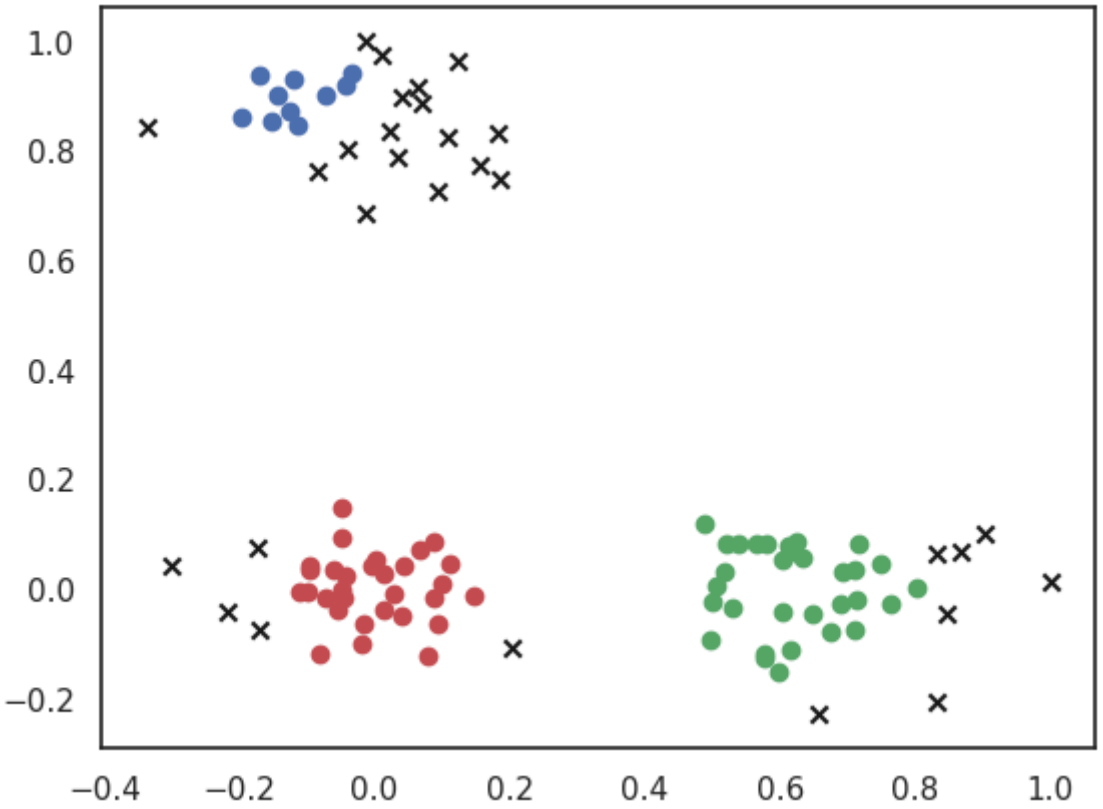

Рисунок 3.1.5 - eps = 0.08, min_samples = 6 для luckyset

Анализируя данный график можно заметить, что 0.08 слишком маленький параметр, так как очень много точек являются шумами. Однако кластеров уже трое, как и должно быть. Видно, что выбросы находятся на достаточно большом расстоянии, поэтому сразу увеличим параметр eps до 0.2, а параметр min_samples уменьшим на 1 (до 5) (см. рис. 3.1.6).

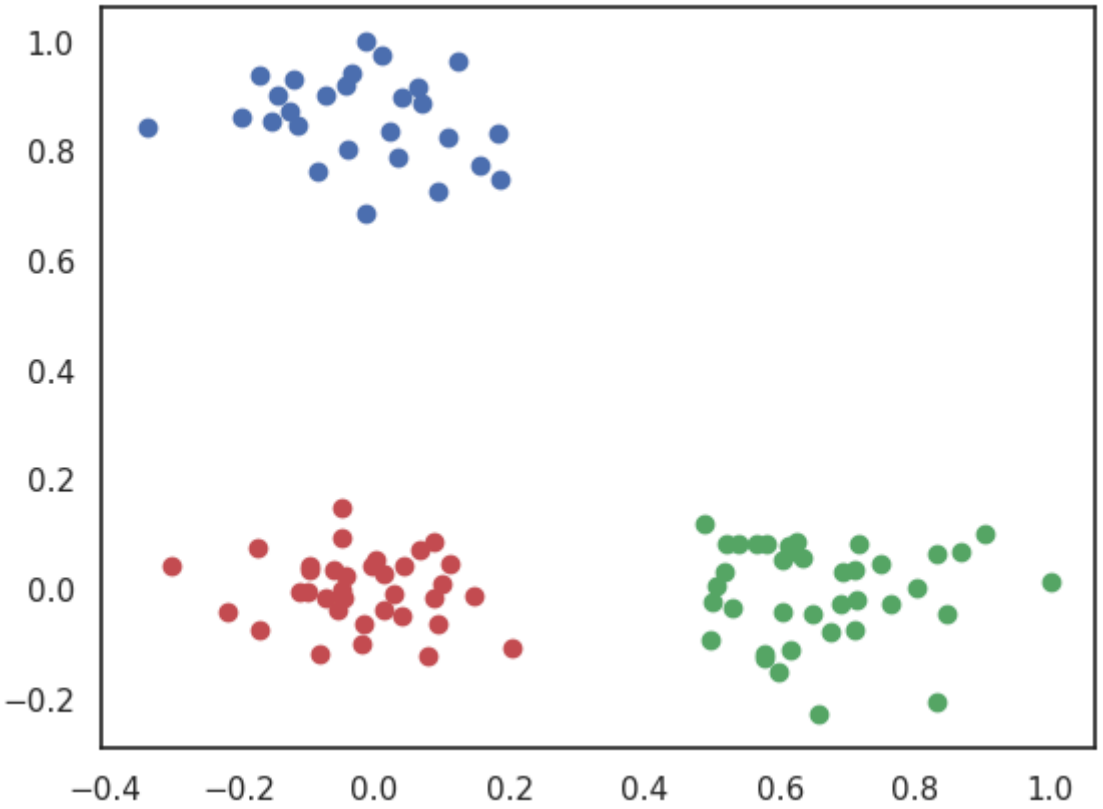

Рисунок 3.1.6 - eps = 0.2, min_samples = 5 для luckyset

Анализируя данный график можно заметить, что при eps=0.2 и min_samples=5, количество кластеров 3 и нет выбросов. Подходящие параметры для luckyset найдены.

Выбор параметров для circles представлен на рисунках 3.1.7-3.1.10.

Первые параметры выбраны случайным образом (такие же как для blobs и для luckyset): eps = 0.08, min_samples = 6 (см. рис. 3.1.7).

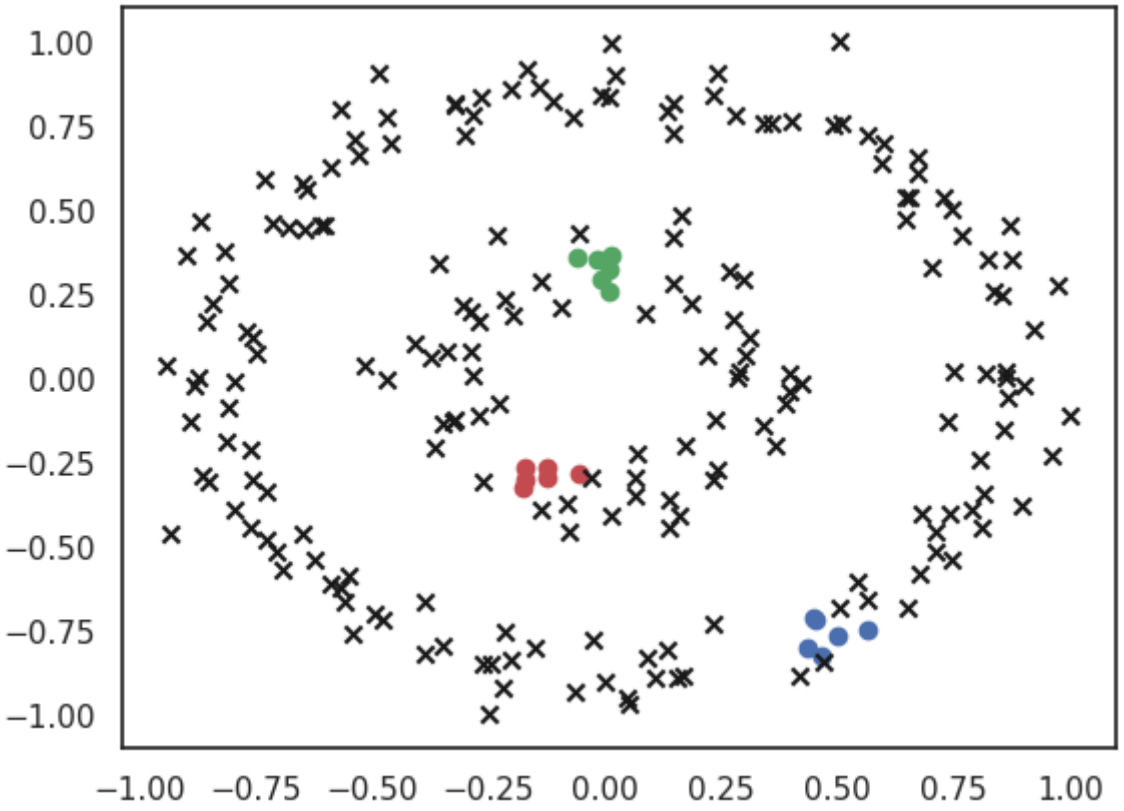

Рисунок 3.1.7 - eps = 0.08, min_samples = 6 для circles

Анализируя данный график можно заметить, что 0.08 слишком маленький параметр, так как почти все точки являются шумами. Увеличим параметр eps до 0.2, а парметр min_samples уменьшим до 5 (см. рис. 3.1.8).

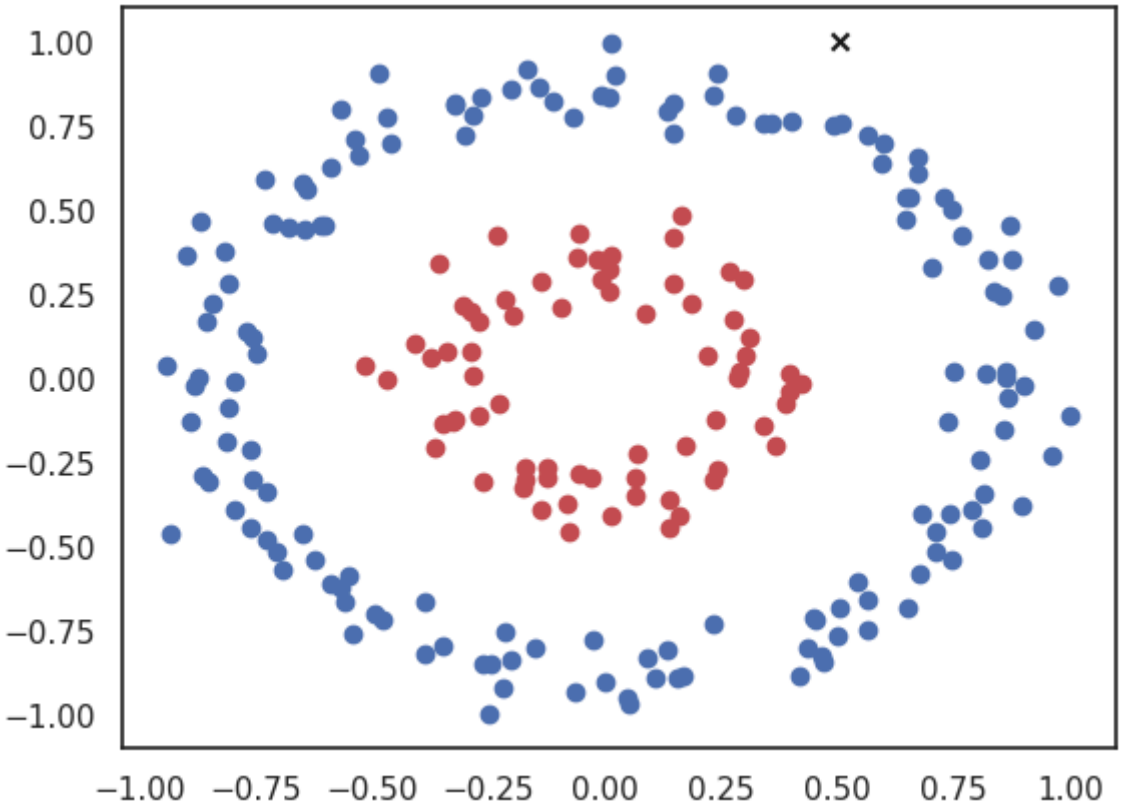

Рисунок 3.1.8 - eps = 0.2, min_samples = 5 для circles

Анализируя данный график можно заметить, что при eps=0.2 и min_samples=5, количество кластеров 2 и выбросом является всего одного значение. Можно сказать, что подходящие параметры для circles найдены, однако попробуем увеличить число кластеров - для этого увеличим параметр min_samples в два раза (см. рис. 3.1.9).

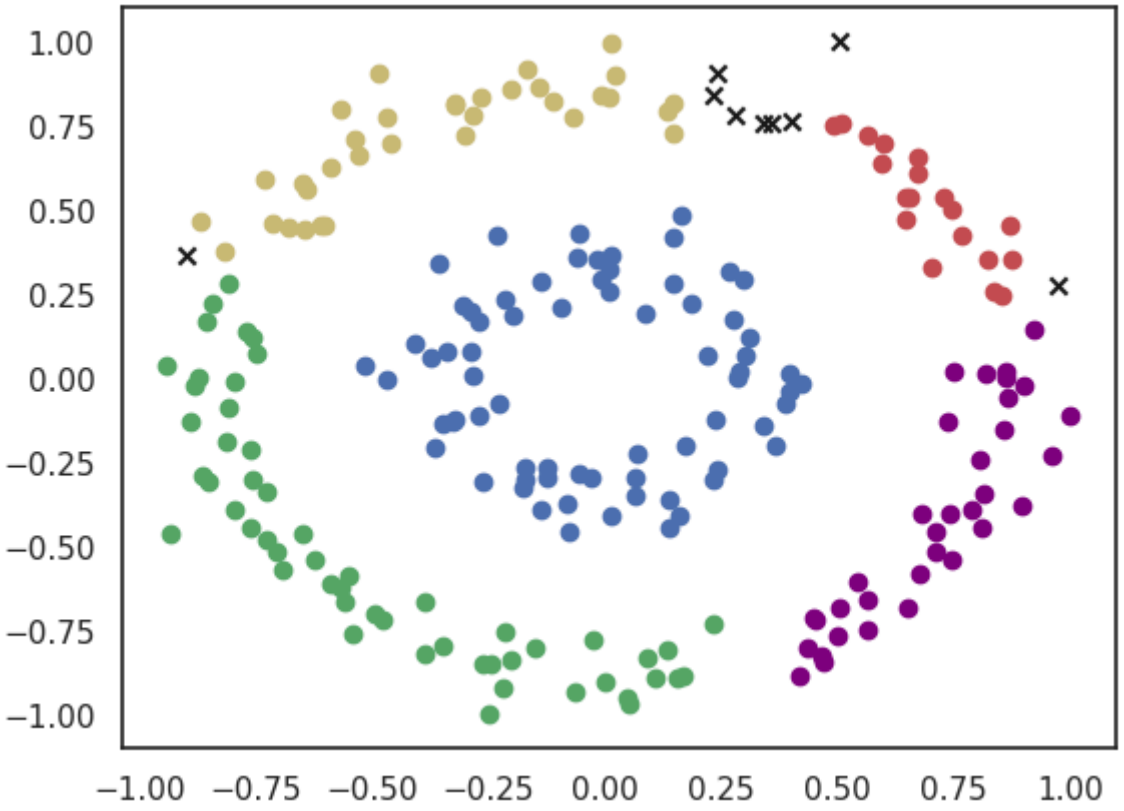

Рисунок 3.1.9 - eps = 0.2, min_samples = 10 для circles

Анализируя данный график можно заметить, что при eps=0.2 и min_samples=10, количество кластеров 5, но выбросом немало. Попробуем уменьшить число выбросов - для этого увеличим параметр min_samples на 1 и параметр esp на 0.01 (см. рис. 3.1.10).

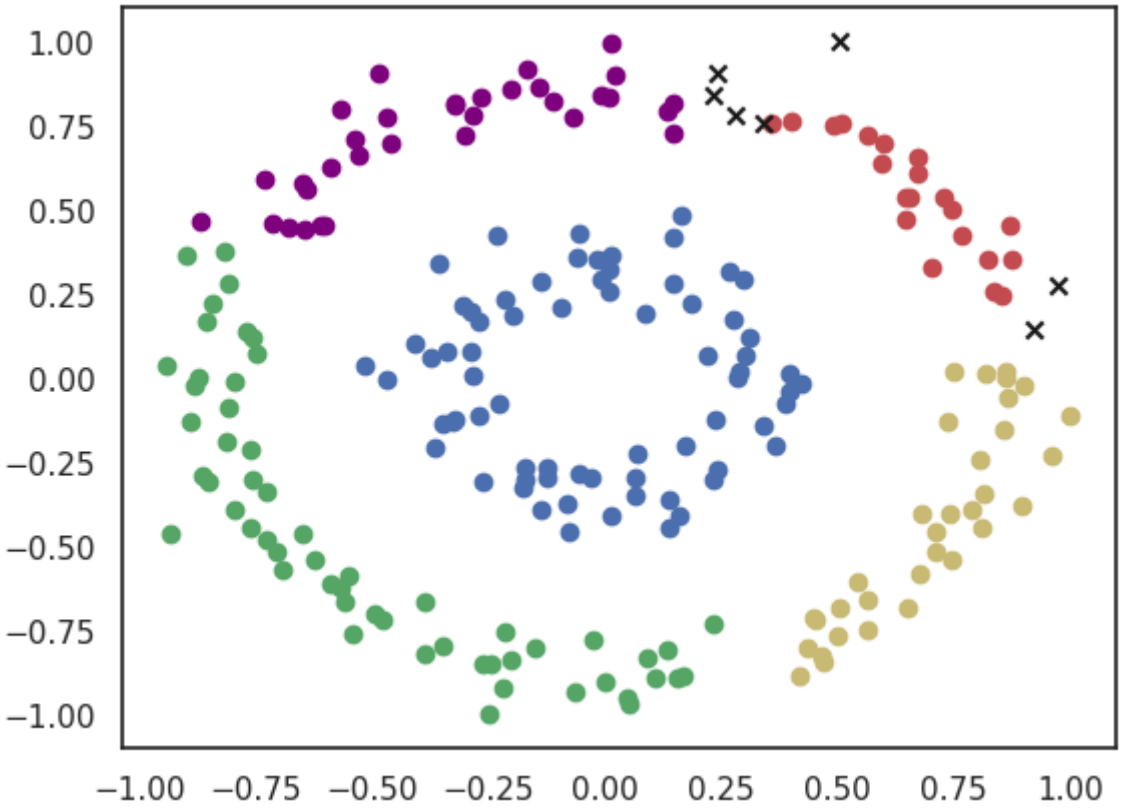

Рисунок 3.1.10 - eps = 0.21, min_samples = 11 для circles

Анализируя данный график можно заметить, что при eps=0.21 и min_samples=11, количество кластеров 5 и выбросов немного. Если считать такое число выбросов допустимым, то подходящие параметры для luckyset - eps=0.21 и min_samples=11 (5 кластеров). Если же выбросов многовато, то подходящие параметры для luckyset - eps=0.2 и min_samples=5 (2 кластера) (см. рисунок 3.1.8).

3.2. Диаграммы рассеяния результатов кластеризации с выделением разным цветом разных кластеров были представлены в пункте 3.1.

3.3. Каждая диаграмма была проанализирована, были найдены оптимальные параметры для алгоритма DBSCAN. Кластеризация данным алгоритмов оказалась успешной, так как на всех итоговых диаграммах

рассеяния видно четкое разделение кластеров, а также близость элементов внутри одного кластера

4. Иерархическая кластеризация:

4.1. Проведем иерархическую кластеризацию при всех возможных параметрах linkage (данный параметр показывает, как будут объединяться классы), используя количество кластеров полученных в п.2. Для каждого из результатов построим дендрограмму (см. листинг 4.1).

Всего бывает 4 параметра linkage:

1) ‘ward’ - минимизация дисперсии в кластерах. Аналог инерции из K-means. Параметр по умолчанию, работает только с евклидовым расстоянием.
2) ‘average’ - минимизирует среднее расстояние между каждыми точками двух кластеров. Альтернатива для не евклидовых расстояний.
3) ‘complete’ - минимизирует максимальное расстояние между точками двух кластеров
4) ‘single’ - минимизирует расстояние между ближайшими точками двух кластеров. Эффективна только в явно разделяемых кластерах со сложной формой. Не устойчива к шумам.

Листинг 4.1 - Построение дендрограмм

```
from scipy.cluster.hierarchy import dendrogram

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)

def my_dendrogram(scaled_data, clusters, data_name):
  linkage_arr = ['ward', 'average', 'complete', 'single']
  for i in range(4):
    agc = AgglomerativeClustering(n_clusters=clusters,
linkage=linkage_arr[i], compute_distances=True)
    ag_clust = agc.fit(scaled_data)
    plot_dendrogram(ag_clust, truncate_mode="level", p=3)
    plt.title("Dendogram "+ linkage_arr[i] + " " + data_name)
    plt.show()
```

Здесь функция plot_dendrogram взять из библиотечной реализации. Для отрисовки дендрограмм со всеми возможными параметрами linkage написана функция my_dendrogram. Для каждого параметра linkage создается объект кластеризации и производится кластеризация. Затем происходит построение дендрограммы. truncate_mode указывает режим обрезки дендрограммы ("lastp" - обрезает последние p узлов, "level" - обрезает дерево на уровне p, None - не производит обрезку). Параметр truncate_mode="level" в данной функции означает, что дендрограмма будет обрезана на уровне p. Аргумент p определяет уровень обрезки дендрограммы (для truncate_mode="level" он указывает на уровень обрезки, для truncate_mode="lastp" он указывает количество последних узлов для отображения).

Проведем иерархическую кластеризацию при всех возможных параметрах linkage для набора данных blobs. Четыре дендрограммы для всех параметров linkage представлены на рисунках 4.1.1-4.1.4.

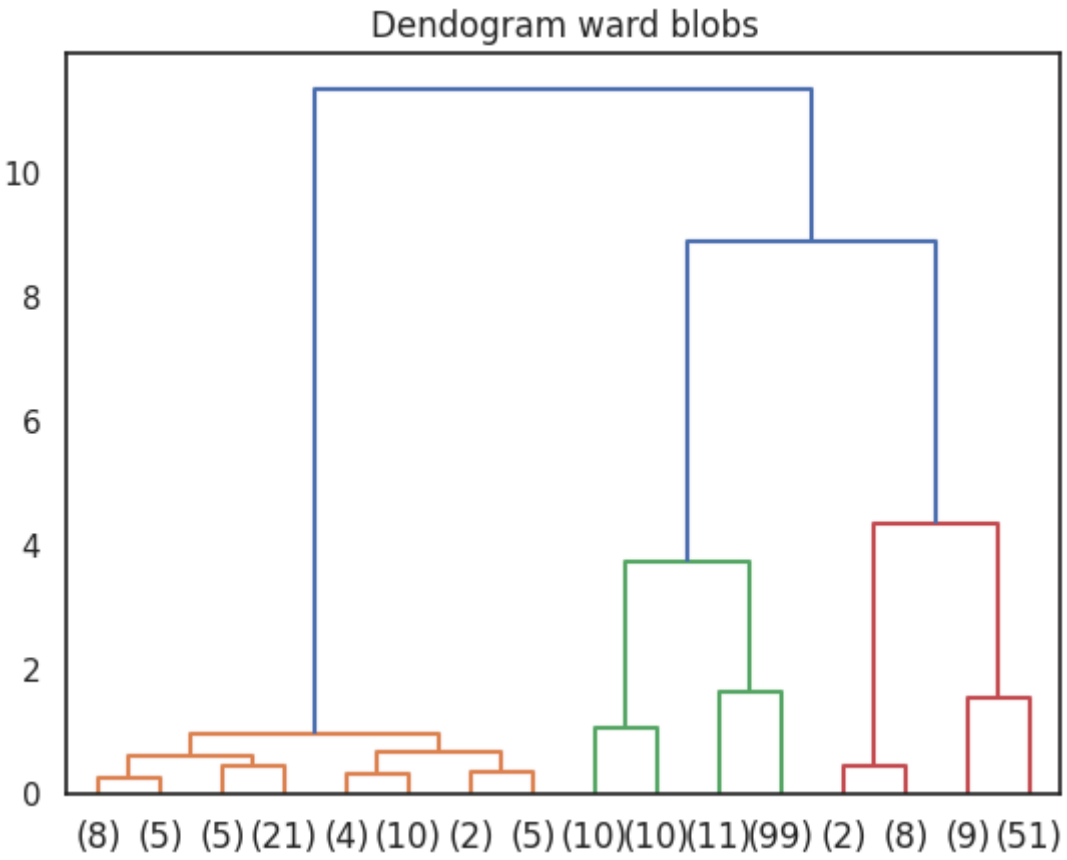

Рисунок 4.1.1 - Дендрограмма для blobs при linkage=ward

Анализируя рисунок 4.1.1 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 3, то получится 5 кластеров (оптимальное количество кластеров для blobs, найденное в пункте 2).

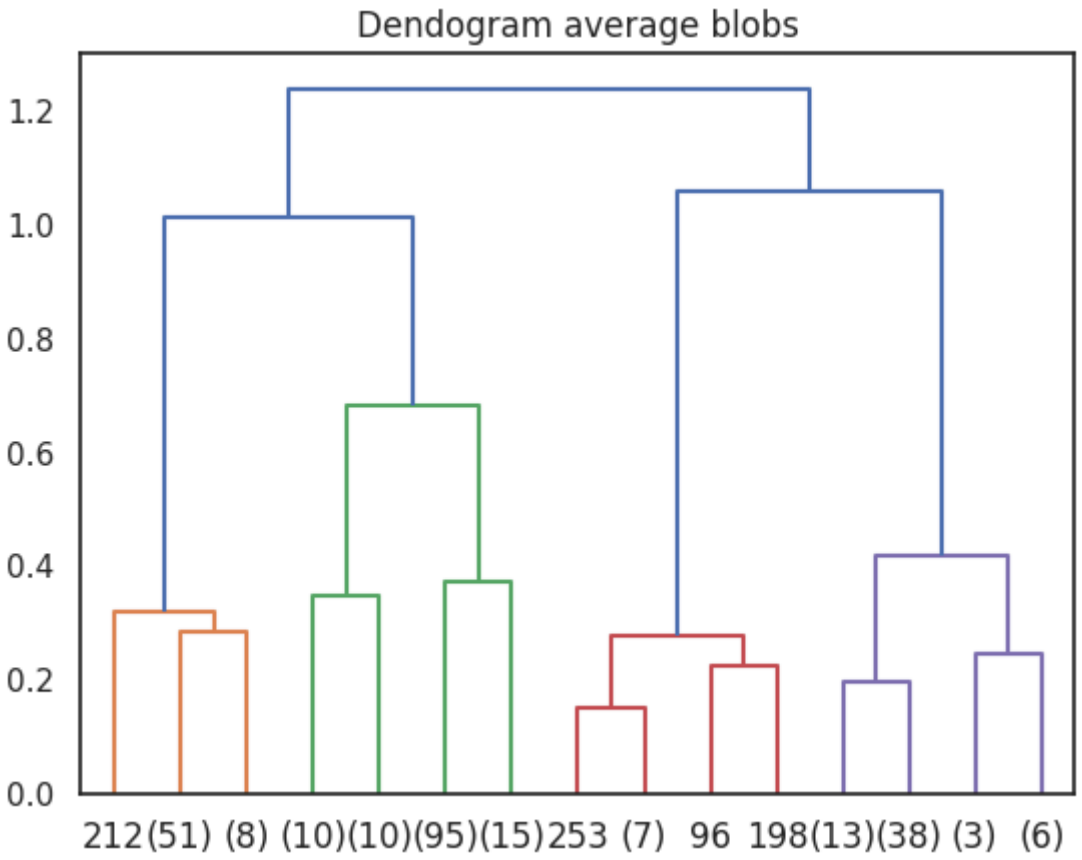

Рисунок 4.1.2 - Дендрограмма для blobs при linkage=average

Анализируя рисунок 4.1.2 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 0.5, то получится 5 кластеров (оптимальное количество кластеров для blobs, найденное в пункте 2).

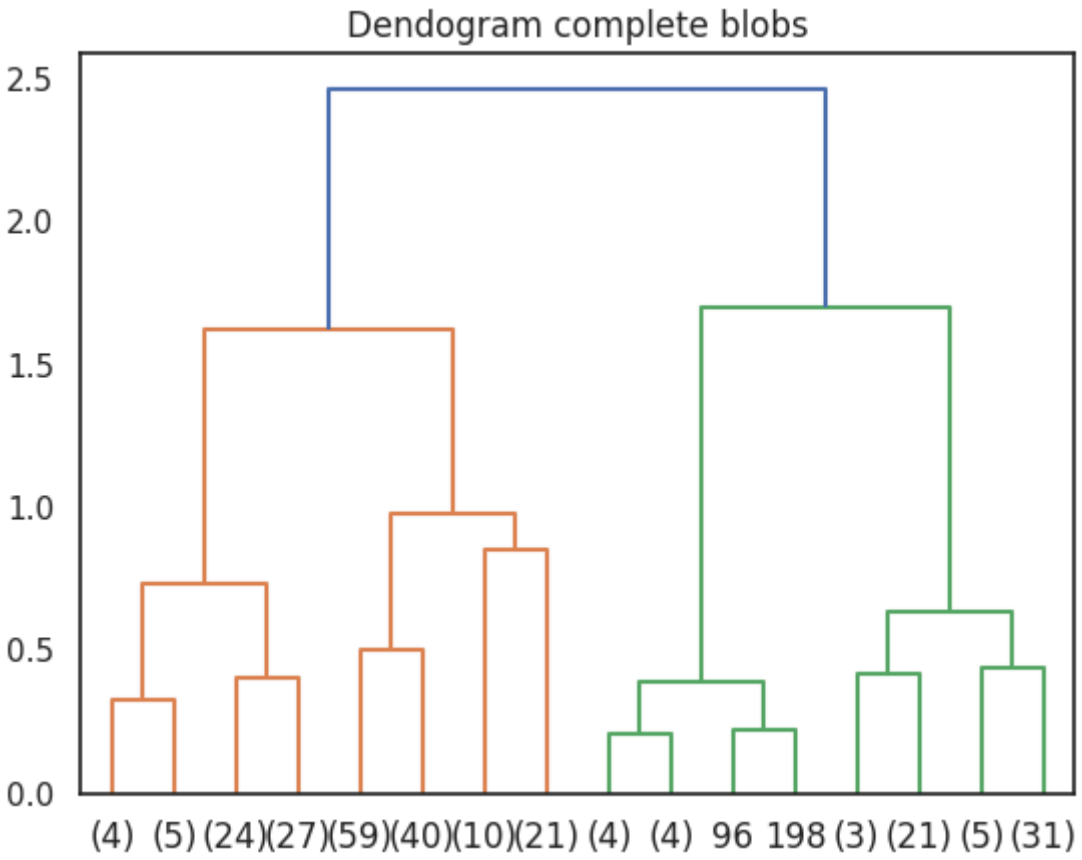

Рисунок 4.1.3 - Дендрограмма для blobs при linkage=complete

Анализируя рисунок 4.1.3 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 1, то получится 5 кластеров (оптимальное количество кластеров для blobs, найденное в пункте 2).

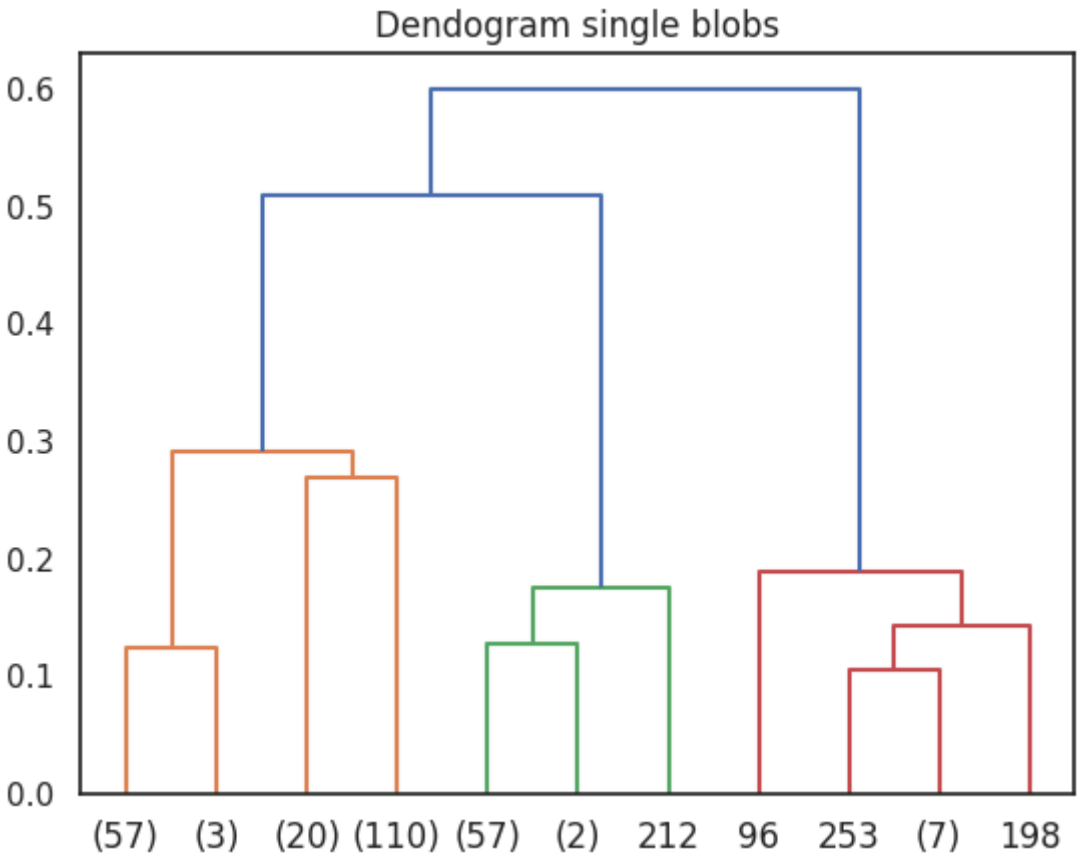

Рисунок 4.1.4 - Дендрограмма для blobs при linkage=single

Анализируя рисунок 4.1.4 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 0.2, то получится 5 кластеров (оптимальное количество кластеров для blobs, найденное в пункте 2).

Таким образом оптимальное количество кластеров для blobs равно 5, как и в пункте 2.

Проведем иерархическую кластеризацию при всех возможных параметрах linkage для набора данных luckyset. Четыре дендрограммы для всех параметров linkage представлены на рисунках 4.1.5-4.1.8.

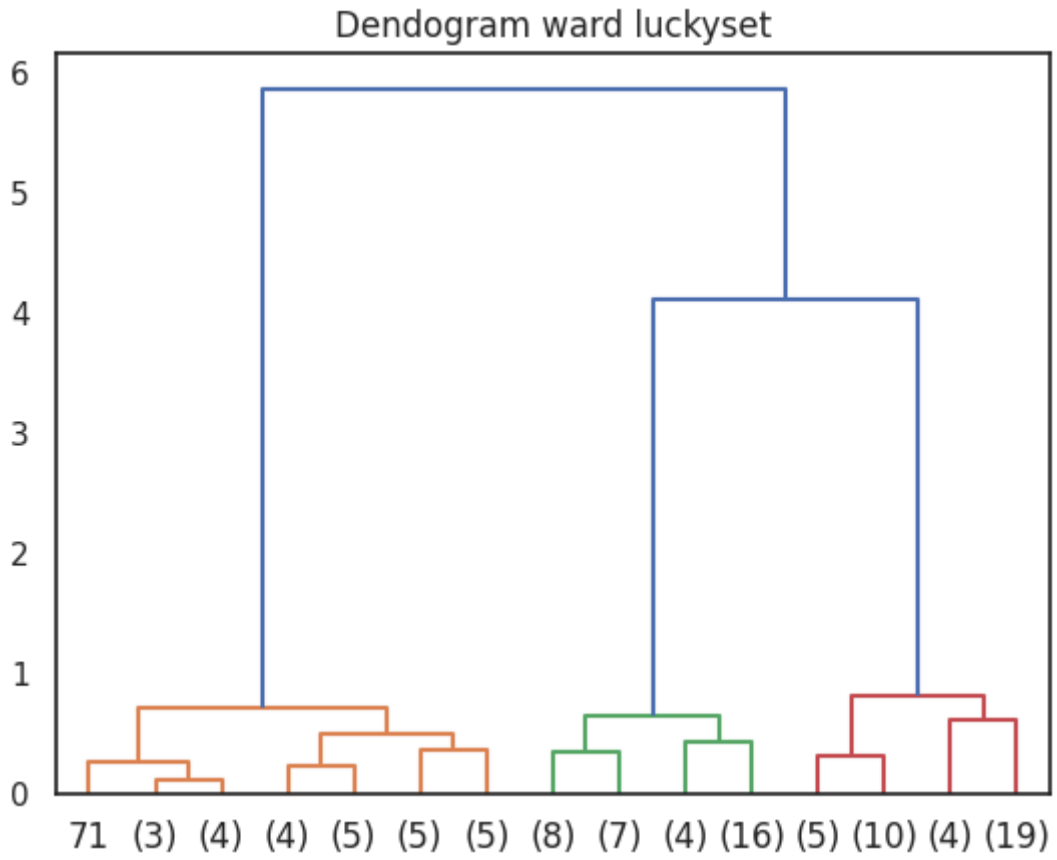

Рисунок 4.1.5 - Дендрограмма для luckyset при linkage=ward

Анализируя рисунок 4.1.5 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 3, то получится 3 кластера (оптимальное количество кластеров для luckyset, найденное в пункте 2).

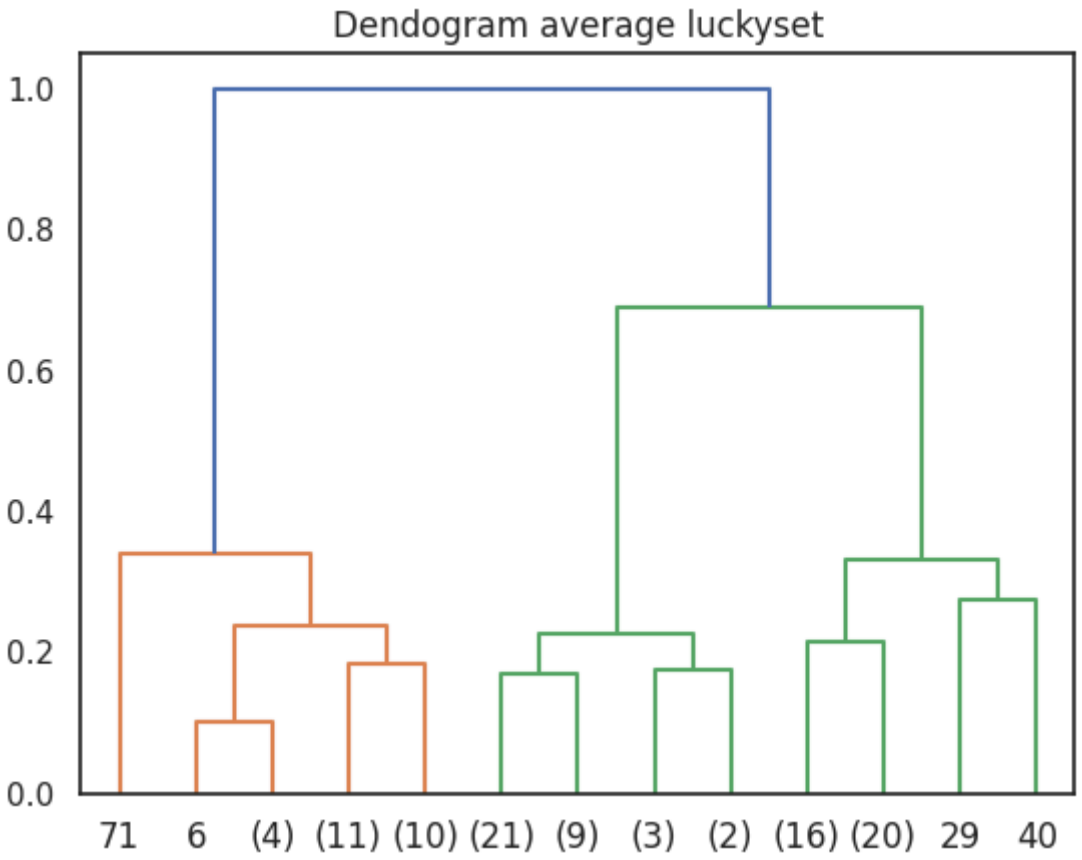

Рисунок 4.1.6 - Дендрограмма для luckyset при linkage=average

Анализируя рисунок 4.1.6 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 0.5, то получится 3 кластера (оптимальное количество кластеров для luckyset, найденное в пункте 2).

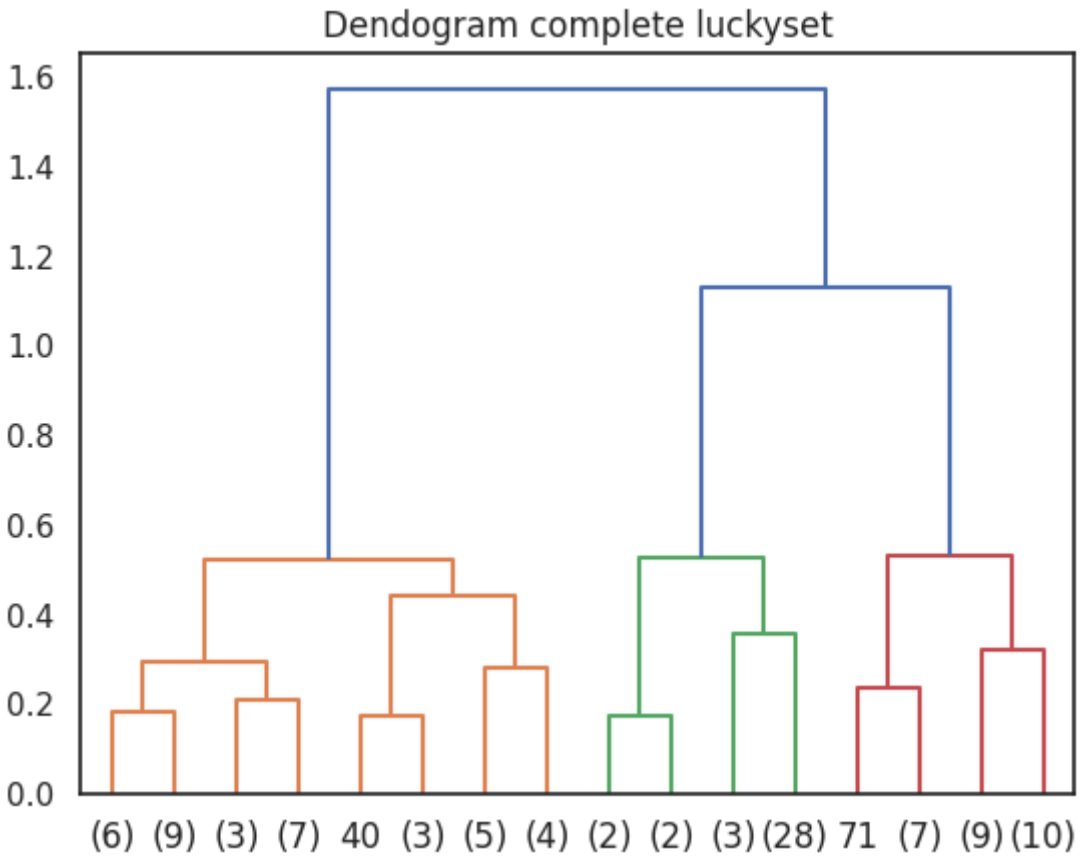

Рисунок 4.1.7 - Дендрограмма для luckyset при linkage=complete

Анализируя рисунок 4.1.7 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 0.8, то получится 3 кластера (оптимальное количество кластеров для luckyset, найденное в пункте 2).

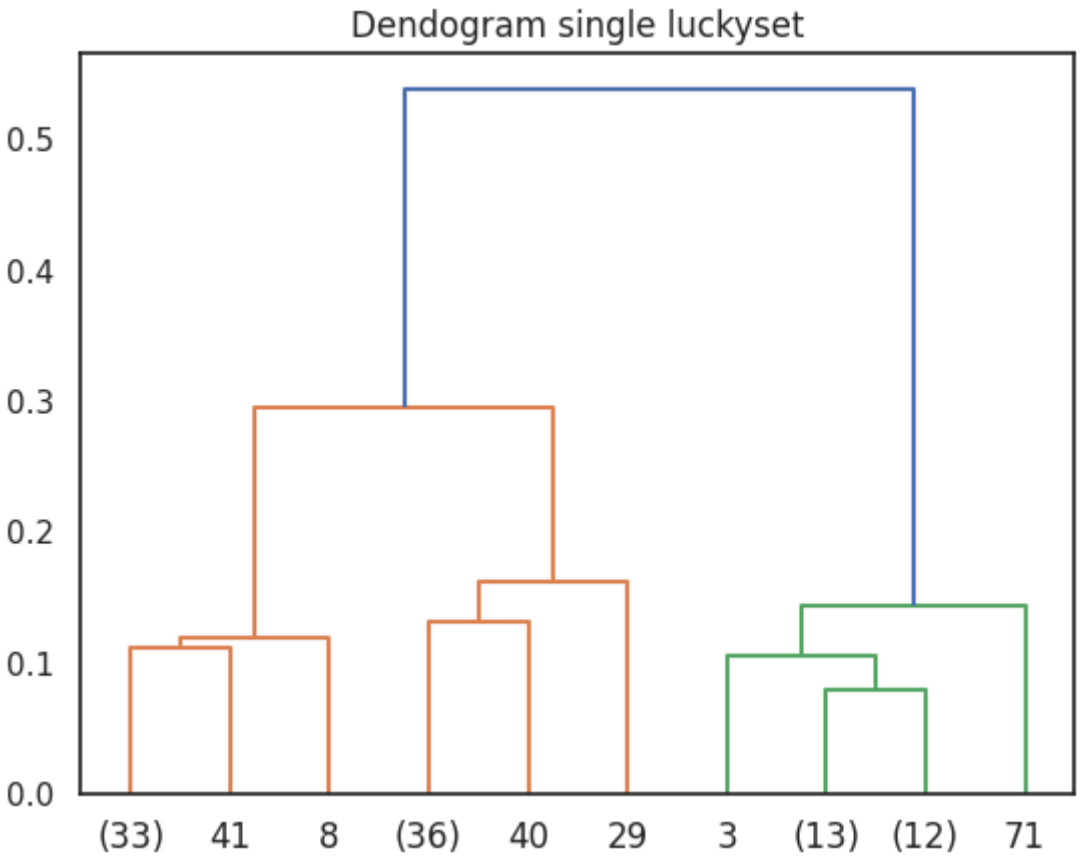

Рисунок 4.1.8 - Дендрограмма для luckyset при linkage=single

Анализируя рисунок 4.1.8 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 0.2, то получится 3 кластера (оптимальное количество кластеров для luckyset, найденное в пункте 2).

Таким образом оптимальное количество кластеров для luckyset равно 3, как и в пункте 2.

Проведем иерархическую кластеризацию при всех возможных параметрах linkage для набора данных circles. Четыре дендрограммы для всех параметров linkage представлены на рисунках 4.1.9-4.1.12.

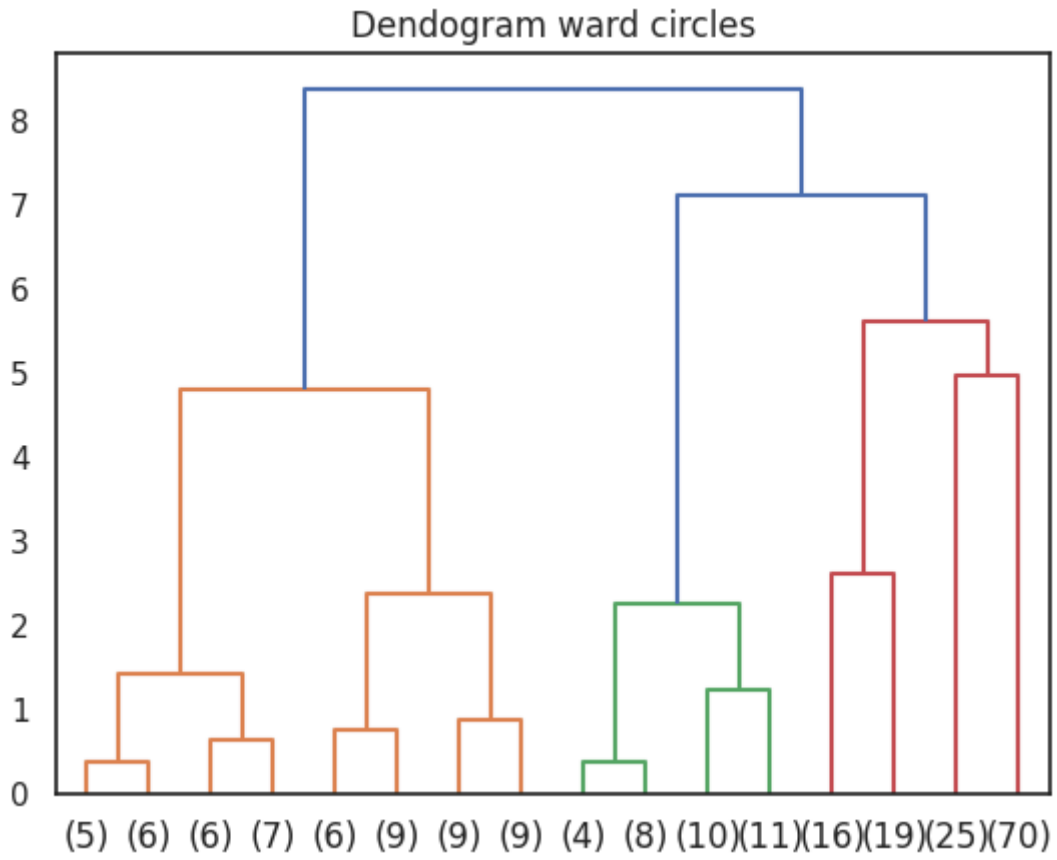

Рисунок 4.1.9 - Дендрограмма для circles при linkage=ward

Анализируя рисунок 4.1.9 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 3, то получится 6 кластеров, если до ~2.5, то получится 8 кластеров (оптимальное количество кластеров для circles, найденное в пункте 2).

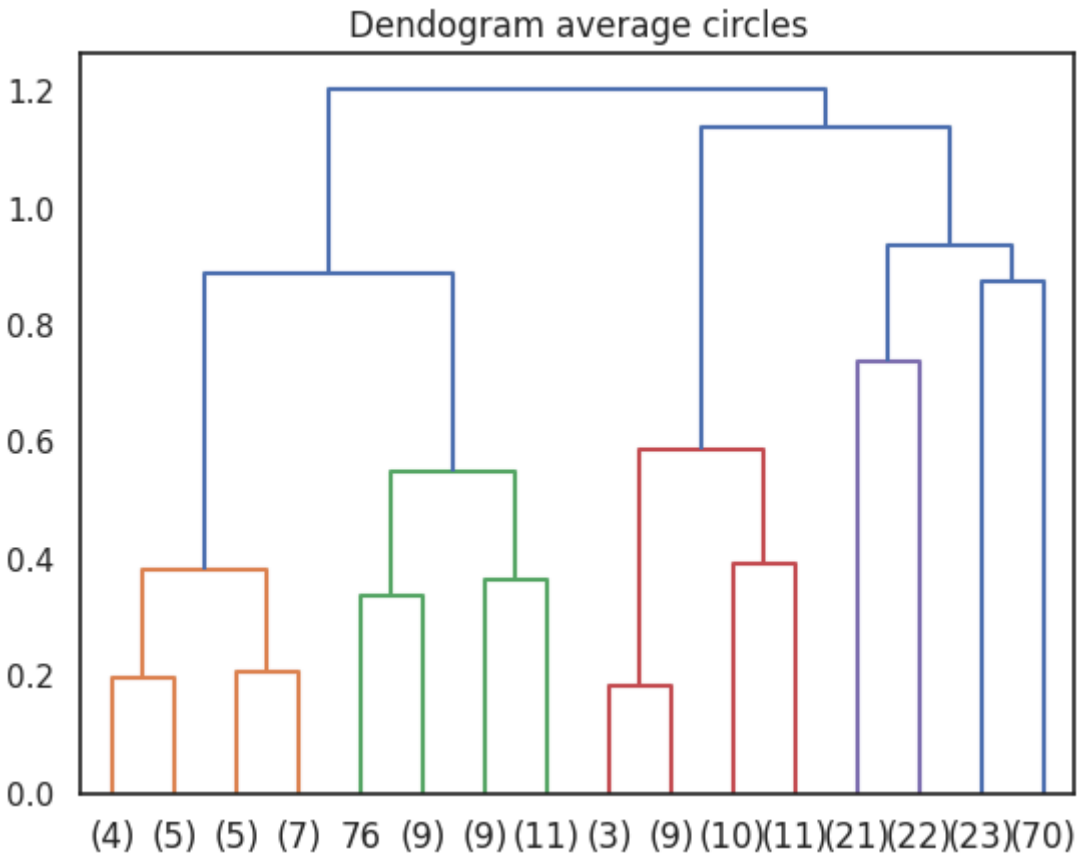

Рисунок 4.1.10 - Дендрограмма для circles при linkage=average

Анализируя рисунок 4.1.10 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 0.7, то получится 7 кластеров, если до ~0.6, то получится 8 кластеров (оптимальное количество кластеров для circles, найденное в пункте 2).

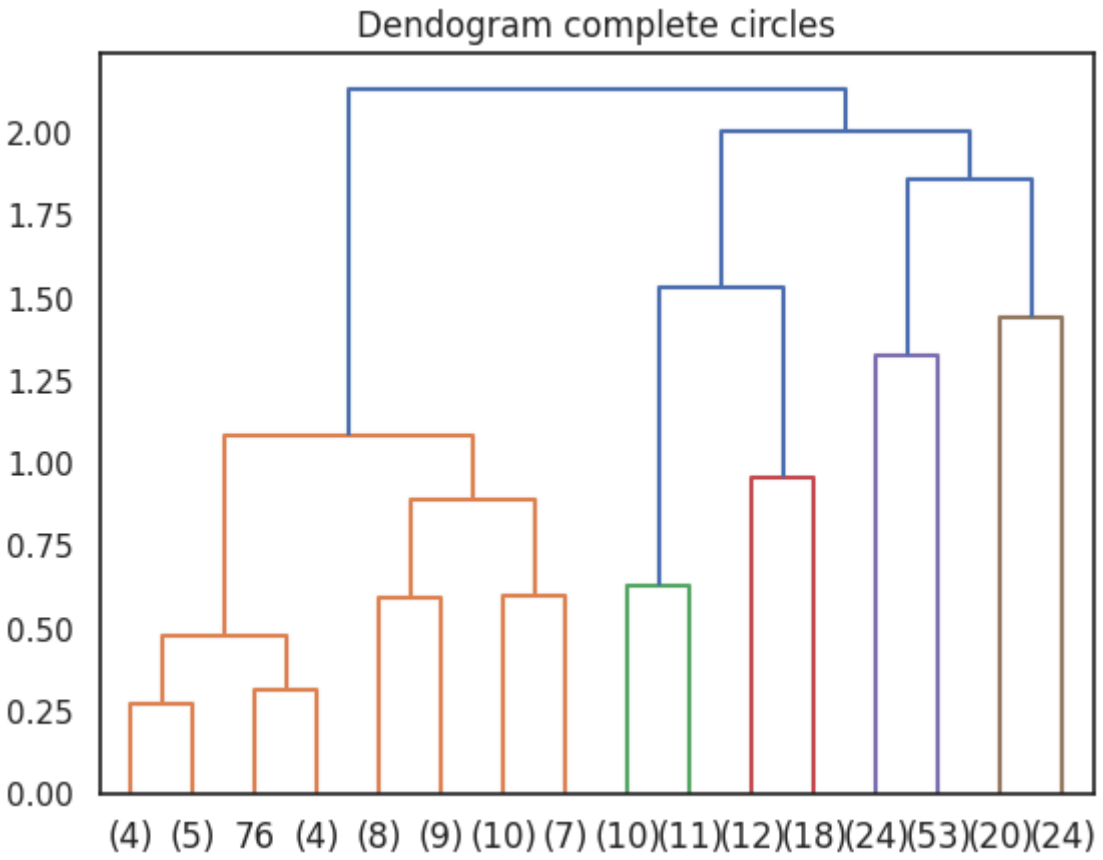

Рисунок 4.1.11 - Дендрограмма для circles при linkage=complete

Анализируя рисунок 4.1.11 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до ~1.15, то получится 8 кластеров (оптимальное количество кластеров для circles, найденное в пункте 2).

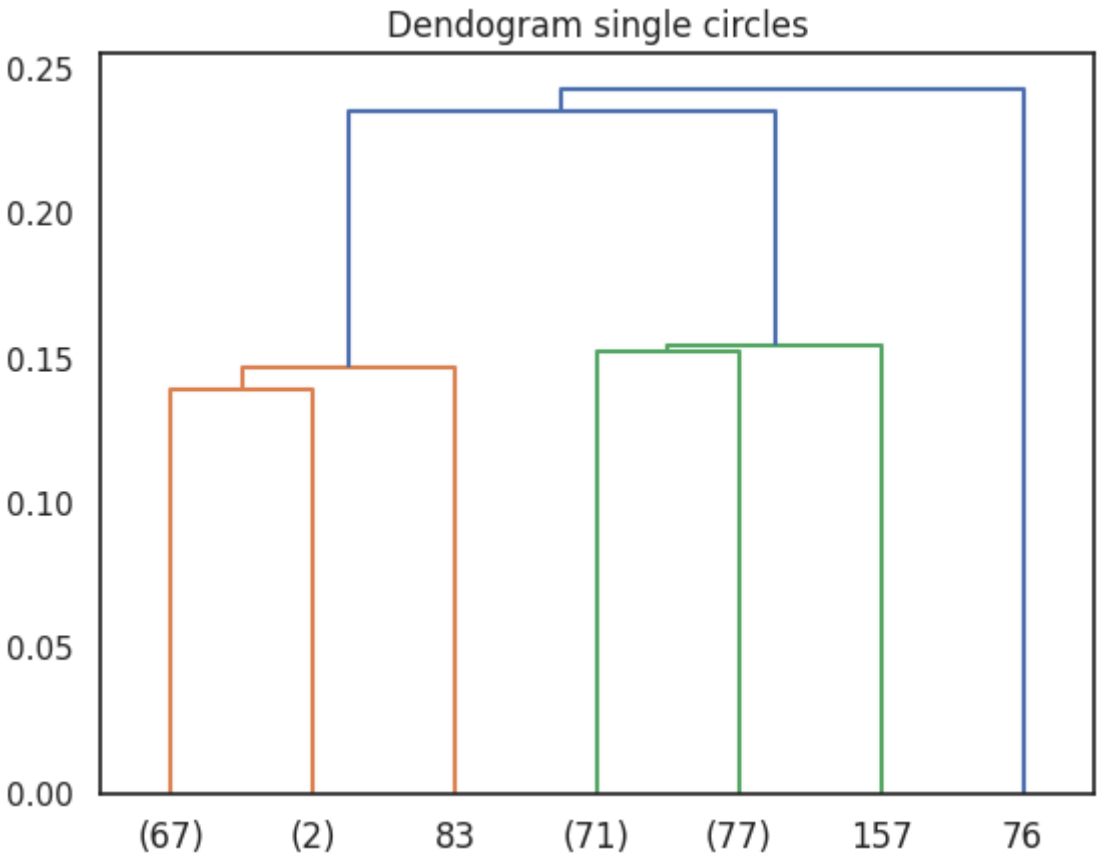

Рисунок 4.1.12 - Дендрограмма для circles при linkage=single

Анализируя рисунок 4.1.12 можно заметить, что если расстояние при котором кластеры объединяются (значение по оси y) будет снижено до 0.1, то получится 7 кластеров (оптимальное количество кластеров для circles, найденное в пункте 2, 8, однако при linkage=single максимальное количество кластеров равно 7).

Таким образом оптимальное количество кластеров для circles может быть равным 8 (6-8), как и в пункте 2.

4.2. Построим диаграммы рассеяния результатов кластеризации с выделением разным цветом разных кластеров. Будем использовать лучшие результаты, полученные для определенного параметра linkage.

Для набора данных blobs при параметре linkage=ward получаются наилучшие результаты. Построим диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров для набора данных blobs (см. листинг 4.2.1 и рис. 4.2.1).

Листинг 4.2.1 - Диаграмма рассеяния для blobs

```
agc = AgglomerativeClustering(n_clusters = 5, linkage = 'ward')
ag_clust = agc.fit_predict(scaled_bl)
plt.scatter(scaled_bl[ag_clust == 0,0], scaled_bl[ag_clust ==
0,1], c = 'r')
plt.scatter(scaled_bl[ag_clust == 1,0], scaled_bl[ag_clust ==
1,1], c = 'b')
plt.scatter(scaled_bl[ag_clust == 2,0], scaled_bl[ag_clust ==
2,1], c = 'g')
plt.scatter(scaled_bl[ag_clust == 3,0], scaled_bl[ag_clust ==
3,1], c = 'c')
plt.scatter(scaled_bl[ag_clust == 4,0], scaled_bl[ag_clust ==
4,1], c = 'm')
plt.show()
```

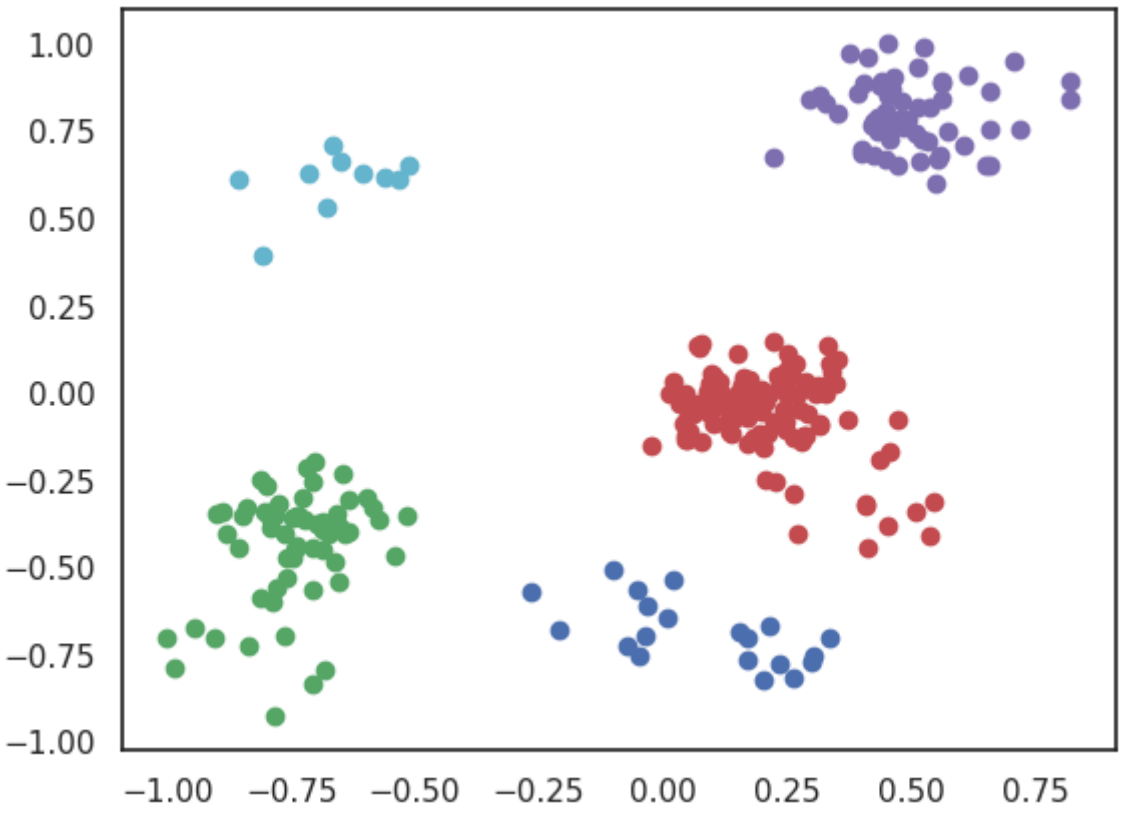

Рисунок 4.2.1 - Диаграмма рассеяния для blobs

Для набора данных luckyset при параметре linkage=ward получаются наилучшие результаты. Построим диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров для набора данных luckyset (см. листинг 4.2.2 и рис. 4.2.2).

Листинг 4.2.2 - Диаграмма рассеяния для luckyset

```
agc = AgglomerativeClustering(n_clusters = 3, linkage = 'ward')
ag_clust = agc.fit_predict(scaled_ls)
plt.scatter(scaled_ls[ag_clust == 0,0], scaled_ls[ag_clust ==
0,1], c = 'r')
plt.scatter(scaled_ls[ag_clust == 1,0], scaled_ls[ag_clust ==
1,1], c = 'b')
plt.scatter(scaled_ls[ag_clust == 2,0], scaled_ls[ag_clust ==
2,1], c = 'g')
plt.show()
```

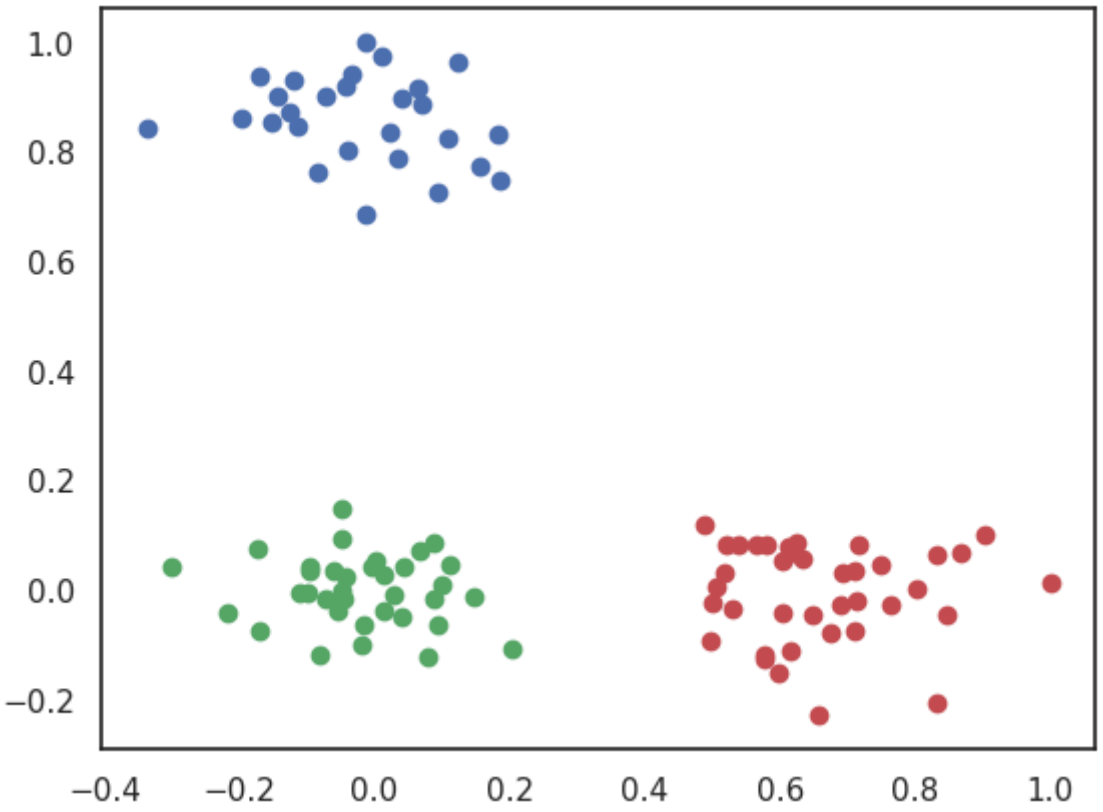

Рисунок 4.2.2 - Диаграмма рассеяния для luckyset

Для набора данных circles при параметре linkage=ward получаются наилучшие результаты (сложно однозначно было сказать, какое значение linkage даст наилучший результат, поэтому были построены диаграммы для всех параметров и наилучшим оказался ward). Построим диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров для набора данных circles (см. листинг 4.2.3 и рис. 4.2.3).

Листинг 4.2.3 - Диаграмма рассеяния для circles

```
agc = AgglomerativeClustering(n_clusters = 8, linkage = 'ward')
ag_clust = agc.fit_predict(scaled_cc)
plt.scatter(scaled_cc[ag_clust == 0,0], scaled_cc[ag_clust ==
0,1], c = 'r')
plt.scatter(scaled_cc[ag_clust == 1,0], scaled_cc[ag_clust ==
1,1], c = 'b')
plt.scatter(scaled_cc[ag_clust == 2,0], scaled_cc[ag_clust ==
2,1], c = 'g')
plt.scatter(scaled_cc[ag_clust == 3,0], scaled_cc[ag_clust ==
3,1], c = 'c')
plt.scatter(scaled_cc[ag_clust == 4,0], scaled_cc[ag_clust ==
4,1], c = 'm')
plt.scatter(scaled_cc[ag_clust == 5,0], scaled_cc[ag_clust ==
5,1], c = 'y')
plt.scatter(scaled_cc[ag_clust == 6,0], scaled_cc[ag_clust ==
6,1], c = 'pink')
plt.scatter(scaled_cc[ag_clust == 7,0], scaled_cc[ag_clust ==
7,1], c = 'purple')
plt.show()
```

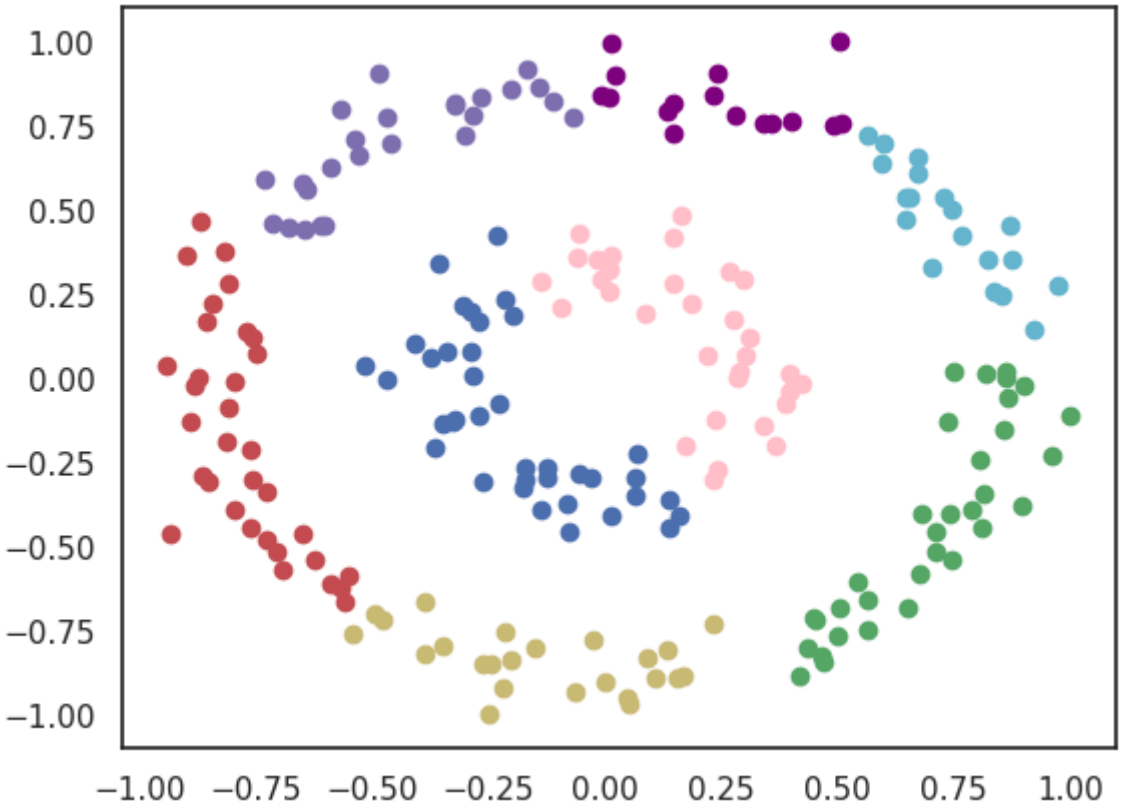

Рисунок 4.2.3 - Диаграмма рассеяния для circles

4.3. Сравним результаты кластеризации с результатами полученными в п.2 и п.3.

Для набора данных blobs все рассмотренные методы кластеризации работают неплохо, так как только два кластера не имеют четкой границы и значения из одного кластера могла попасть в другой (как в случае k-средних). Но все же метод кластеризации DBSCAN и иерархическая кластеризация справились чуть лучше, потому что более правильно разделили те два класса, у которых значения чуть смешались. Таким образом, для набора данных blobs DBSCAN и иерархическая кластеризация справились лучше, чем K-means.

Для набора данных luckyset все рассмотренные методы кластеризации работают хорошо, так все кластеры четко отделены друг от друга и находятся на достаточно далеком расстоянии. Все диаграммы рассеивания из пункта 2, 3 и 4 получились одинаковыми.

Для набора данных circles все не так однозначно. Если считать, что кластеров должно быть два - внутренне и внешнее кольцо/эллипс, то лучше всех справился DBSCAN, так как только у данного метода получилось подобрать такие параметры, чтобы было 2 кластера. Если считать, что кластеров должно быть 6-8, то лучших всех справились K-means и иерархическая кластеризация, а хуже всех DBSCAN.

Вывод.

В ходе выполнения работы были рассмотрены и проанализированы на оптимальное число кластеров различные наборы данных: от самых простых с четко разделенными данными до сложных, где однозначно было трудно сказать, сколько кластеров будет являться оптимальным числом. Были изучены различные методы кластеризации (K-means, DBSCAN, AgglomerativeClustering) и для каждого набора данных выбран лучший.