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 136М

Студент гр. 1303 _____ Чубан Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Изучить способы кластеризации предоставленных данных с помощью алгоритмов K-means, DBSCAN и иерархической кластеризации. Научиться проводить кластеризацию данных.

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. Проведем подготовку наборов данных

1.1. Загрузим данные из файлов как Pandas DataFrame (*read_csv*) (см. листинг 1.1).

Листинг 1.1 – Загрузка датасета

```
df_blobs = pd.read_csv("lab2_blobs.csv")
df_checker = pd.read_csv("lab2_checker.csv")
df_circles = pd.read_csv("lab2_circles.csv")
```

1.2. Вызовем у датафреймов метод head и проверим корректность загруженных данных (см. листинг 1.2). Команда выведет первые 5 строк датафреймов. Вывод метода см. в таблице 1.1

Листинг 1.2 – Вызов *head*

```
df_blobs.head()
df_checker.head()
df_circles.head()
```

Таблица 1.1 – Вывод *head*

	df_blobs		df_checker		df_circles	
	x	y	x	y	x	y
0	-8.0267	-4.9731	4.0510	0.9697	0.3400	0.3297
1	-7.0422	-2.6454	7.5581	5.1224	0.6849	0.7212
2	8.9214	9.5679	2.8765	7.0870	0.0085	0.2924
3	1.0887	-0.2884	3.8366	0.8614	-0.8343	-0.3787
4	0.4739	-0.0737	4.2159	0.7742	0.1230	-1.0068

1.3. Построим диаграммы рассеяния наборов данных (листинг 1.3, диаграммы – рис. 1.1, 1.2, 1.3)

Листинг 1.3 – Рисование диаграмм

```
sns.scatterplot(data=df_blobs, x="x", y="y")
sns.scatterplot(data=df_checker, x="x", y="y")
sns.scatterplot(data=df_circles, x="x", y="y")
```

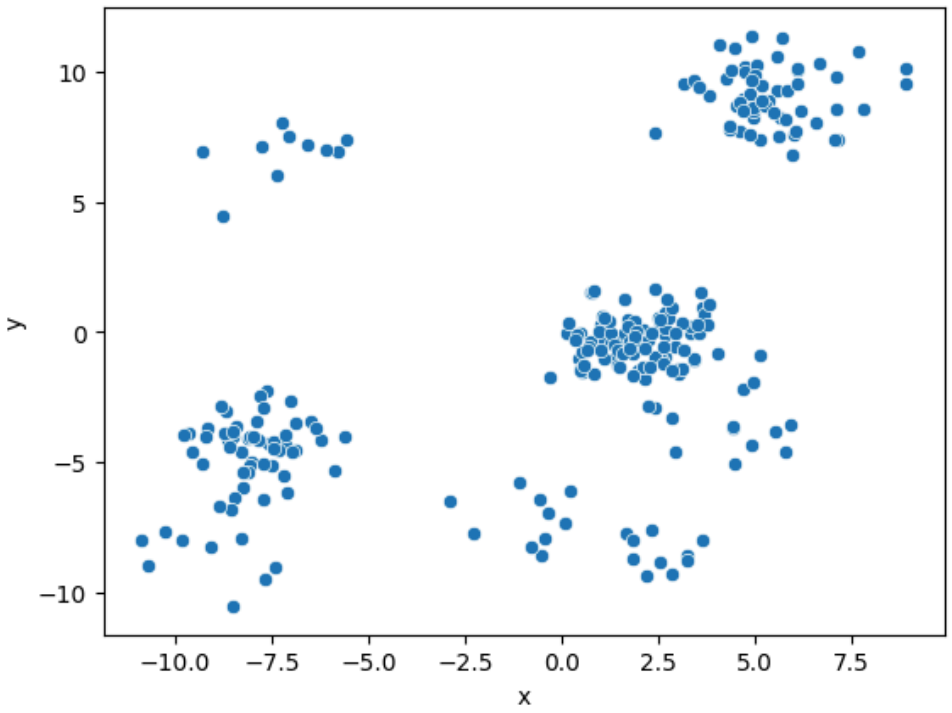

Рисунок 1.1 – Диаграмма рассеяния данных df_blobs

Из диаграммы рассеяния для df_blobs (рис 1.1) можно увидеть 5 скоплений точек с большим количеством выбросов, в основном, расположенные на равном удалении от центральной точки (0, 0).

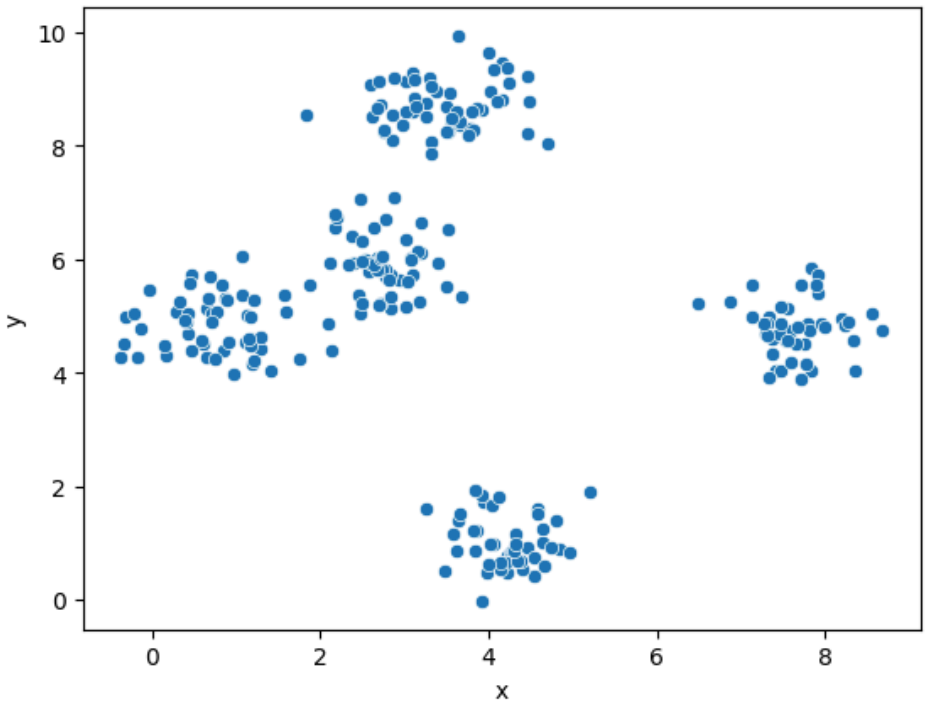

Рисунок 1.2 – Диаграмма рассеяния данных df_checker

Из диаграммы рассеяния для df_checker (рис. 1.2) можно увидеть 5 скоплений точек с небольшими выбросами, в основном, расположенные на равном удалении от точки (4, 5). Также можно заметить смешение скоплений в районе точки (2, 5).

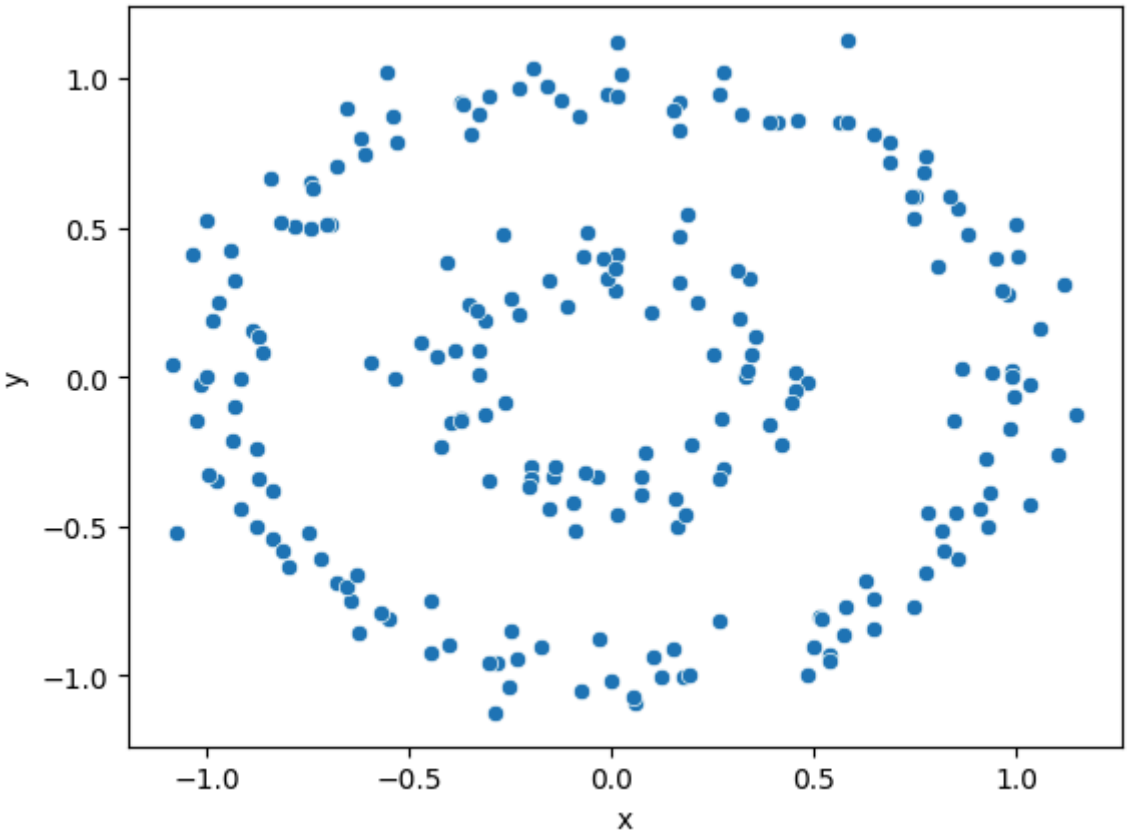

Рисунок 1.3 – Диаграмма рассеяния данных df_circles

Из диаграммы рассеяния для df_circles (рис. 1.3) можно увидеть 2 скопления точек в виде окружностей. Также видны точечные выбросы данных на краях скоплений.

1.4. Подготовим наборы данных, проведя стандартизацию и нормировку данных (листинг 1.4).

Листинг 1.4 – стандартизация и нормировка данных

```
std_scaler = StandardScaler()
robust_scaler = RobustScaler()
min_max_scaler = MinMaxScaler()
df_blobs = robust_scaler.fit_transform(df_blobs)
```

```
df_checker = min_max_scaler.fit_transform(df_checker)
df_circles = std_scaler.fit_transform(df_circles)
```

Для df_blobs выбрана устойчивая к выбросам нормировка RobustScaler, так как данные содержат много одиночных точек, которые могут являться выбросами данных.

Для df_checker выбрана нормировка MinMaxScaler, так как данные различаются по порядку величин.

Для df_circles выбрана стандартизация StandartScaler, так как распределение данных похоже на нормальное.

2. Изучение работы K-Means

2.1. Проведем исследование оптимального количества кластеров методом локтя

Построим график зависимости инерции от количества кластеров для df_blobs (листинг 2.1.1, рис. 2.1.1)

Листинг 2.1.1 – график зависимости инерции от количества кластеров для df_blobs

```
inertia_list = []
clusters_list = []
for i in range(10):
  temp_kmeans = KMeans(n_clusters=i+1, n_init=5)
  temp_kmeans.fit(df_blobs)
  inertia_list.append(temp_kmeans.inertia_)
  clusters_list.append(i+1)
plt.plot(clusters_list, inertia_list)
plt.ylabel('Инерция')
plt.xlabel('Количество кластеров')
plt.xticks(clusters_list)
plt.grid()
```

```
plt.show()
```

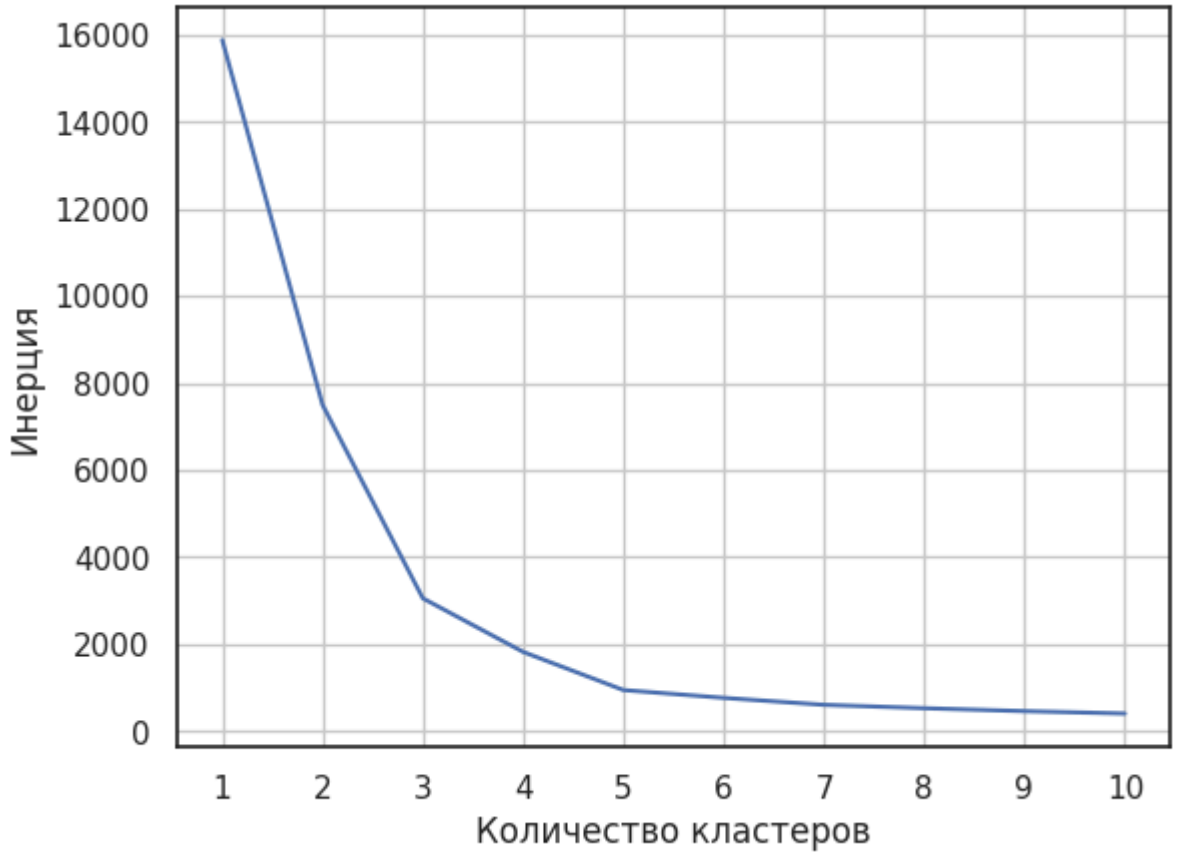

Рисунок 2.1.1 – график зависимости инерции от количества кластеров для df_blobs

Из графика видно, что скорость убывания графика уменьшается при значении количества кластеров равному 4-6. Рассмотрим разбиение данных на данные количества кластеров (листинг 2.1.2, рисунок 2.1.2).

Листинг 2.1.2 – разбиение данных df_blobs на 4, 5, 6 кластеров

```
clusters_range = np.arange(4, 7, 1)
fig, axs = plt.subplots(1, len(clusters_range),
figsize=(5*(len(clusters_range)+1), 5))
for i, cluster in enumerate(clusters_range):
  kmeans = KMeans(n_clusters=cluster, n_init=5)
  kmeans_df_blobs = kmeans.fit_predict(df_blobs)
  new_df_blobs = pd.DataFrame(data=df_blobs,
columns=["x", "y"])
```

```
  new_df_blobs["cluster"] = kmeans_df_blobs
  sns.scatterplot(data=new_df_blobs, x="x", y="y",
hue="cluster", ax=axs[i],
palette="tab10").set(title=f"Кластеров: {cluster}")
```

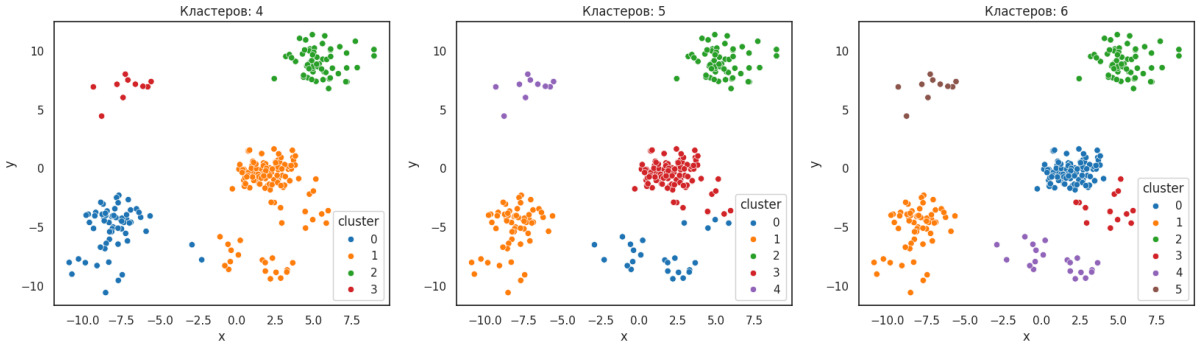

Рисунок 2.1.2 – разбиение данных df_blobs на 4, 5, 6 кластеров

Из графиков видно, что наиболее оптимальным числом кластеров является 5, т.к. в этом варианте кластеры получаются наиболее сгруппированными и менее пересекающимися.

Построим график зависимости инерции от количества кластеров для df_checker (листинг 2.1.3, рис. 2.1.3)

Листинг 2.1.3 – график зависимости инерции от количества кластеров для df_checker

```
inertia_list = []
clusters_list = []
for i in range(10):
  temp_kmeans = KMeans(n_clusters=i+1, n_init=5)
  temp_kmeans.fit(df_checker)
  inertia_list.append(temp_kmeans.inertia_)
  clusters_list.append(i+1)
plt.plot(clusters_list, inertia_list)
plt.ylabel('Инерция')
```

```
plt.xlabel('Количество кластеров')
plt.xticks(clusters_list)
plt.grid()
plt.show()
```

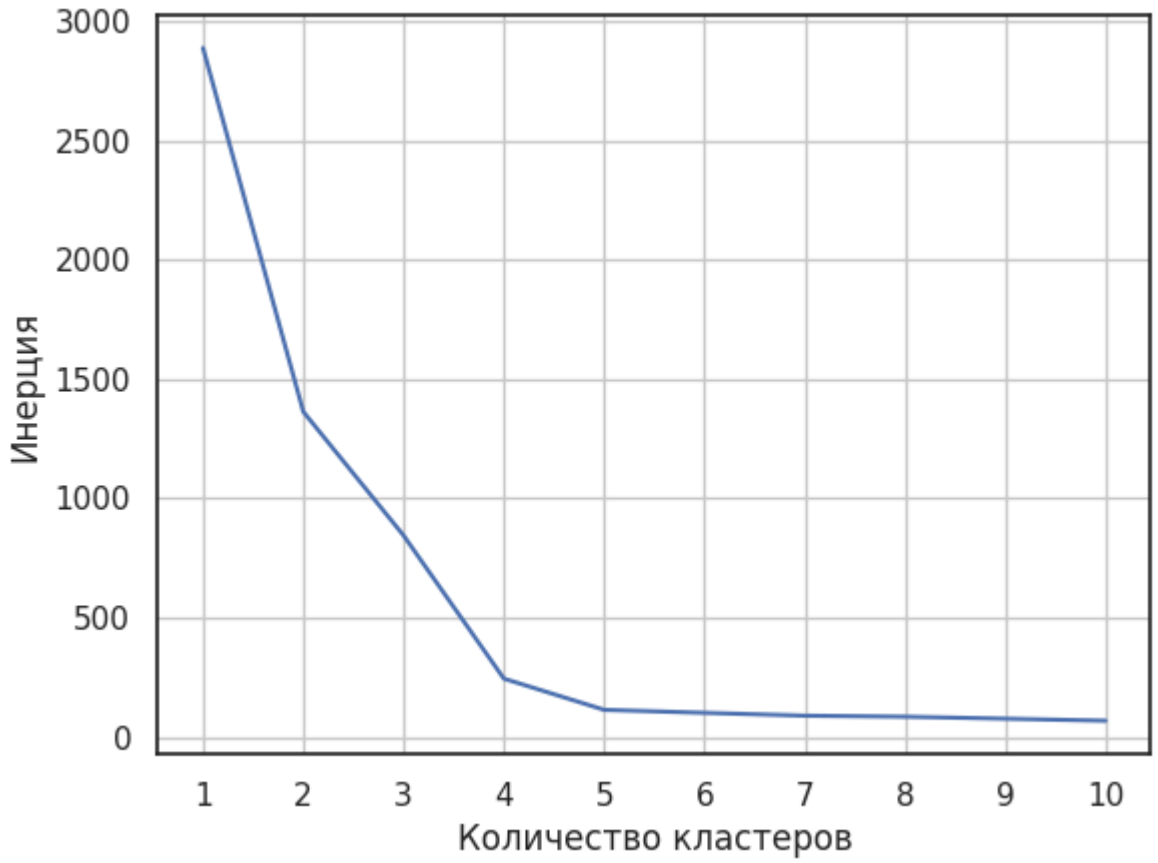

Рисунок 2.1.3 – график зависимости инерции от количества кластеров для df_checker

Из графика видно, что скорость убывания графика уменьшается при значении количества кластеров равному 4-6. Рассмотрим разбиение данных на данные количества кластеров (листинг 2.1.4, рисунок 2.1.4).

Листинг 2.1.4 – разбиение данных df_checker на 4, 5, 6 кластеров

```
clusters_range = np.arange(4, 7, 1)
fig, axs = plt.subplots(1, len(clusters_range),
figsize=(5*(len(clusters_range)+1), 5))
for i, cluster in enumerate(clusters_range):
  kmeans = KMeans(n_clusters=cluster, n_init=5)
```

```
  kmeans_df_checker = kmeans.fit_predict(df_checker)
  new_df_checker = pd.DataFrame(data=df_checker,
columns=["x", "y"])
  new_df_checker["cluster"] = kmeans_df_checker
  sns.scatterplot(data=new_df_checker, x="x", y="y",
hue="cluster", ax=axs[i],
palette="tab10").set(title=f"Кластеров: {cluster}")
```

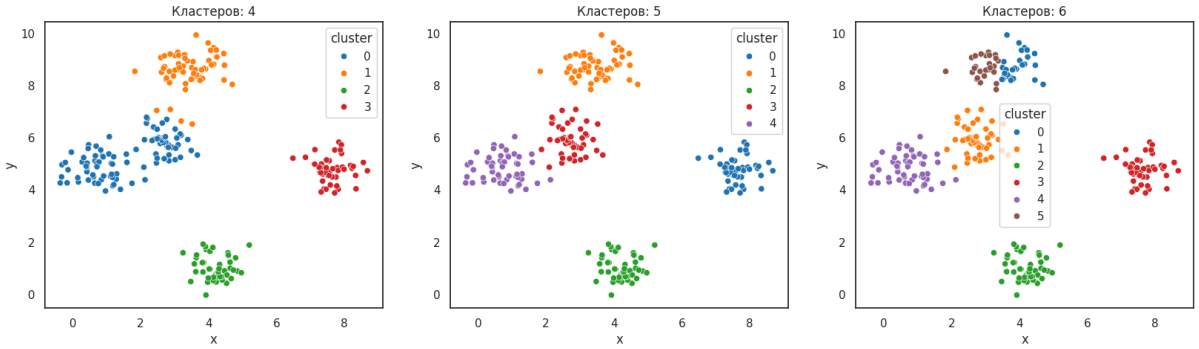

Рисунок 2.1.4 – разбиение данных df_checker на 4, 5, 6 кластеров

Из графиков видно, что наиболее оптимальным числом кластеров является 5, т.к. в этом варианте кластеры получаются наиболее сгруппированными и менее пересекающимися.

Построим график зависимости инерции от количества кластеров для df_circles (листинг 2.1.5, рис. 2.1.5)

Листинг 2.1.5 – график зависимости инерции от количества кластеров для df_circles

```
inertia_list = []
clusters_list = []
for i in range(20):
  temp_kmeans = KMeans(n_clusters=i+1, n_init=5)
  temp_kmeans.fit(df_circles)
  inertia_list.append(temp_kmeans.inertia_)
  clusters_list.append(i+1)
```

```
plt.plot(clusters_list, inertia_list)
plt.ylabel('Инерция')
plt.xlabel('Количество кластеров')
plt.xticks(clusters_list)
plt.grid()
plt.show()
```

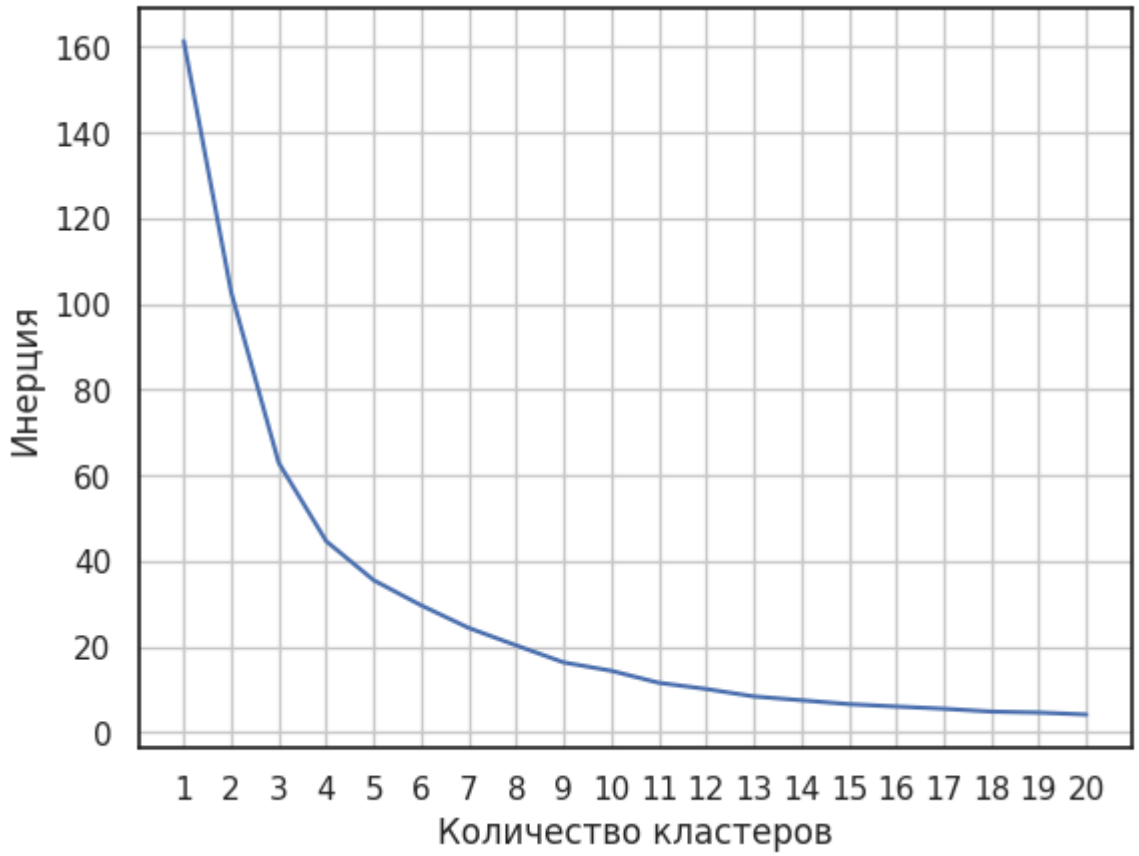

Рисунок 2.1.5 – график зависимости инерции от количества кластеров для df_circles

Из графика видно, что скорость убывания графика уменьшается при значении количества кластеров равному 15-18. Рассмотрим разбиение данных на данные количества кластеров (листинг 2.1.6, рисунок 2.1.6).

Листинг 2.1.6 – разбиение данных df_circles на 15, 16, 17, 18 кластеров

```
clusters_range = np.arange(15, 19, 1)
fig, axs = plt.subplots(1, len(clusters_range),
figsize=(5*(len(clusters_range)+1), 5))
for i, cluster in enumerate(clusters_range):
```

```
  kmeans = KMeans(n_clusters=cluster, n_init=5)
  kmeans_df_circles = kmeans.fit_predict(df_circles)
  new_df_circles = pd.DataFrame(data=df_circles,
columns=["x", "y"])
  new_df_circles["cluster"] = kmeans_df_circles
  sns.scatterplot(data=new_df_circles, x="x", y="y",
hue="cluster", ax=axs[i],
palette="tab10").set(title=f"Кластеров: {cluster}")
```

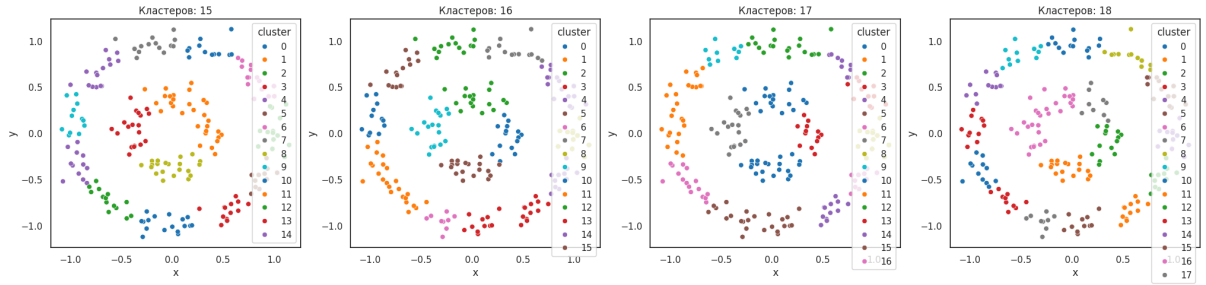

Рисунок 2.1.6 – разбиение данных df_checker на 15, 16, 17, 18 кластеров

Из графиков видно, что наиболее оптимальным числом кластеров является 15, т.к. в этом варианте кластеры получаются наиболее сгруппированными и менее пересекающимися.

2.2. Проведем исследование оптимального количества кластеров методом силуэта.

Построим график зависимости среднего значения коэффициента силуэта от количества кластеров для df_blobs (листинг 2.2.1, рис. 2.2.1)

Листинг 2.2.1 – построение графика зависимости среднего значения коэффициента силуэта от количества кластеров для df_blobs

```
silhouette_list = []
clusters_list = []
for i in range(1, 10):
```

```
  temp_kmeans = KMeans(n_clusters=i+1, n_init=5)
  temp_clusters = temp_kmeans.fit_predict(df_blobs)
  silhouette_list.append(silhouette_score(df_blobs,
temp_clusters))
  clusters_list.append(i+1)
plt.plot(clusters_list, silhouette_list)
plt.ylabel('Среднее значение коэффицента силуэта')
plt.xlabel('Количество кластеров')
plt.xticks(clusters_list)
plt.grid()
plt.show()
```

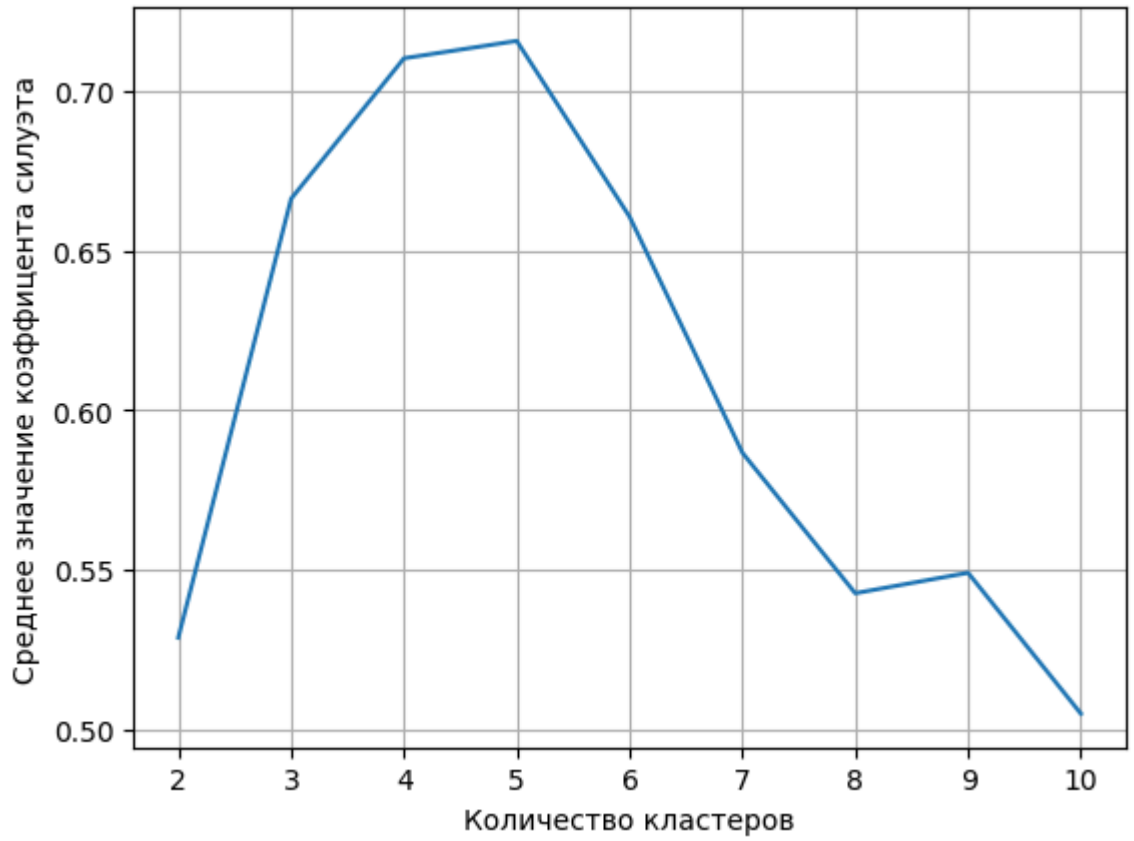

Рисунок 2.2.1 – график зависимости среднего значения коэффициента силуэта от количества кластеров для df_blobs

Коэффициент силуэта максимален при 5 кластерах, значит далее будем рассматривать это количество.

Построим график зависимости среднего значения коэффициента силуэта от количества кластеров для df_checker (листинг 2.2.2, рис. 2.2.2)

Листинг 2.2.2 – построение графика зависимости среднего значения коэффициента силуэта от количества кластеров для df_checker

```
silhouette_list = []
clusters_list = []
for i in range(1, 10):
  temp_kmeans = KMeans(n_clusters=i+1, n_init=5)
  temp_clusters = temp_kmeans.fit_predict(df_checker)
  silhouette_list.append(silhouette_score(df_checker,
temp_clusters))
  clusters_list.append(i+1)
plt.plot(clusters_list, silhouette_list)
plt.ylabel('Среднее значение коэффицента силуэта')
plt.xlabel('Количество кластеров')
plt.xticks(clusters_list)
plt.grid()
plt.show()
```

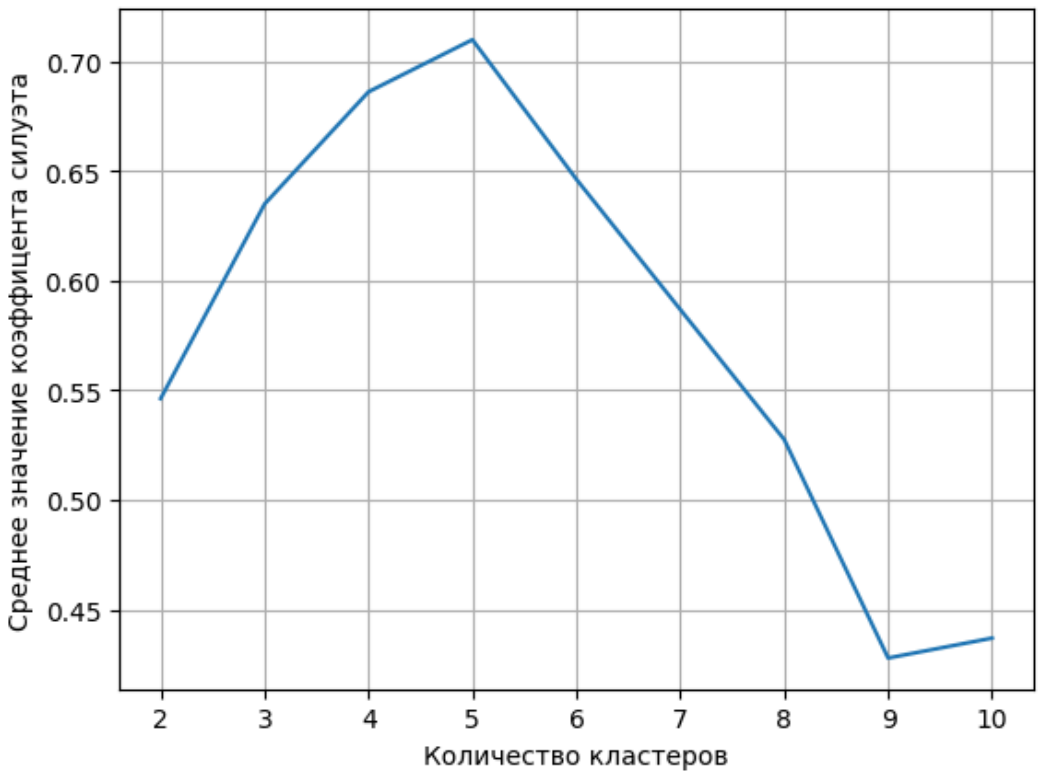

Рисунок 2.2.2 – график зависимости среднего значения коэффициента силуэта от количества кластеров для df_checker

Коэффициент силуэта максимален при 5 кластерах, значит далее будем рассматривать это количество.

Построим график зависимости среднего значения коэффициента силуэта от количества кластеров для df_circles (листинг 2.2.3, рис. 2.2.3)

Листинг 2.2.3 – построение графика зависимости среднего значения коэффициента силуэта от количества кластеров для df_circles

```
silhouette_list = []
clusters_list = []
for i in range(1, 20):
  temp_kmeans = KMeans(n_clusters=i+1, n_init=5)
  temp_clusters = temp_kmeans.fit_predict(df_circles)
  silhouette_list.append(silhouette_score(df_circles,
temp_clusters))
  clusters_list.append(i+1)
plt.plot(clusters_list, silhouette_list)
plt.ylabel('Среднее значение коэффицента силуэта')
plt.xlabel('Количество кластеров')
plt.xticks(clusters_list)
plt.grid()
plt.show()
```

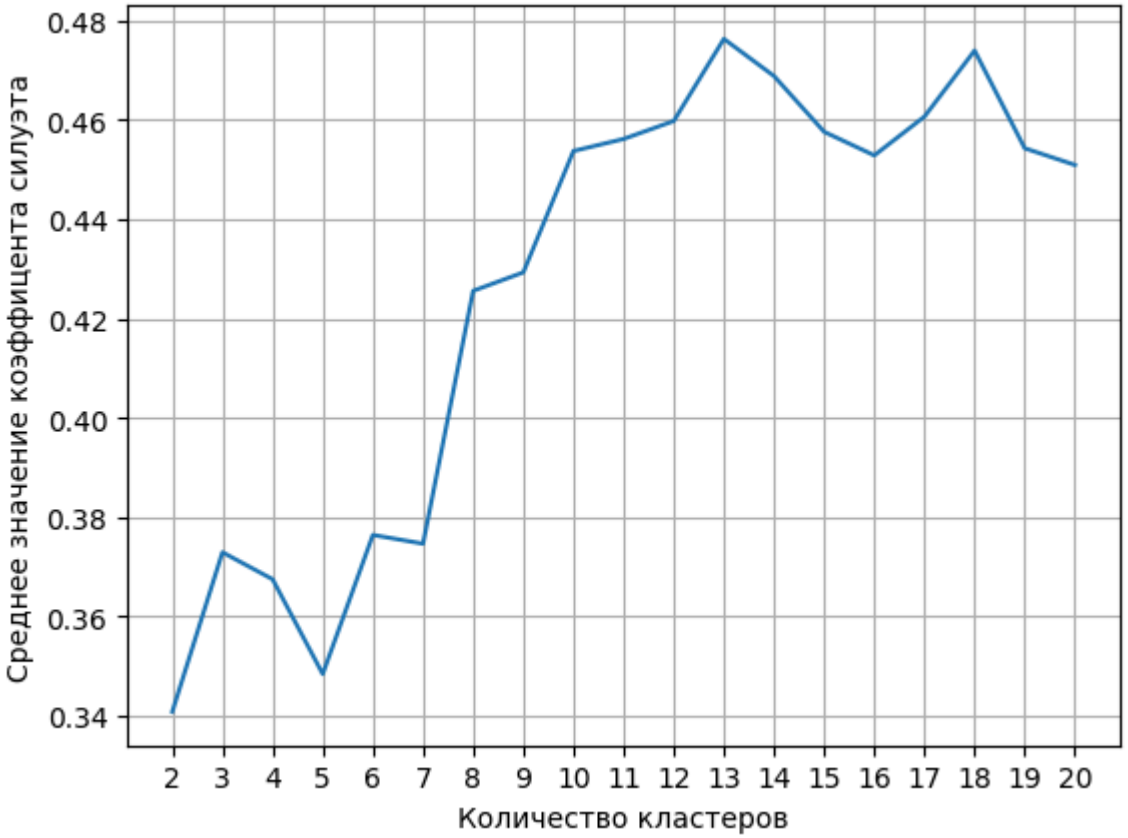

Рисунок 2.2.2 – график зависимости среднего значения коэффициента силуэта от количества кластеров для df_circles

Коэффициент силуэта максимален при 13 кластерах, значит далее будем рассматривать это количество.

2.3. Проведем кластеризацию алгоритмом K-means с выбранными оптимальными количествами кластеров

Для df_blobs с 5 кластерами (листинг 2.3.1, таблица 2.3.1):

Листинг 2.3.1 – кластеризация df_blobs алгоритмом K-means с выбранными оптимальными количествами кластеров

```
clusters_df_blobs = 5
kmeans_df_blobs = KMeans(n_clusters=clusters_df_blobs,
n_init=5)
clusters_list = kmeans_df_blobs.fit_predict(df_blobs)
print(f"Инерция: {kmeans_df_blobs.inertia_}")
new_df_blobs = pd.DataFrame(data=df_blobs, columns=["x",
```

```
"y"])
new_df_blobs["cluster"] = clusters_list
new_df_blobs.head()
```

Таблица 2.3.1 – результат кластеризации df_blobs с помощью K-means

	x	y	cluster
0	-0.978166	-0.398948	1
1	-0.881326	-0.186405	1
2	0.688925	0.928794	2
3	-0.081534	0.028813	0
4	-0.142009	0.048417	0

Для df_checker с 5 кластерами (листинг 2.3.2, таблица 2.3.2):

Листинг 2.3.2 – кластеризация df_checker алгоритмом K-means с выбранными оптимальными количествами кластеров

```
clusters_df_checker = 5
kmeans_df_checker =
KMeans(n_clusters=clusters_df_checker, n_init=5)
clusters_list = kmeans_df_checker.fit_predict(df_checker)
print(f"Инерция: {kmeans_df_checker.inertia_}")
new_df_checker = pd.DataFrame(data=df_checker,
columns=["x", "y"])
new_df_checker["cluster"] = clusters_list
new_df_checker.head()
```

Таблица 2.3.2 – результат кластеризации df_checker с помощью K-means

	x	y	cluster
0	0.488188	0.099736	3
1	0.875721	0.516494	0
2	0.358406	0.713658	1

3	0.464496	0.088867	3
4	0.506409	0.080116	3

Для df_circles с 13 кластерами (листинг 2.3.3, таблица 2.3.3):

Листинг 2.3.3 – кластеризация df_circles алгоритмом K-means с выбранными оптимальными количествами кластеров

```
clusters_df_circles = 13
kmeans_df_circles =
KMeans(n_clusters=clusters_df_circles, n_init=5)
clusters_list = kmeans_df_circles.fit_predict(df_circles)
print(f"Инерция: {kmeans_df_circles.inertia_}")
new_df_circles = pd.DataFrame(data=df_circles,
columns=["x", "y"])
new_df_circles["cluster"] = clusters_list
new_df_circles.head()
```

Таблица 2.3.3 – результат кластеризации df_circles с помощью K-means

	x	y	cluster
0	0.3400	0.3297	12
1	0.6849	0.7212	5
2	0.0085	0.2924	12
3	-0.8343	-0.3787	9
4	0.1230	-1.0068	2

2.4. Построим диаграмму рассеяния результатов кластеризации.

Для df_blobs (листинг 2.4.1, рис. 2.4.1):

Листинг 2.4.1 – построение диаграммы рассеяния результатов кластеризации для df_blobs

```
sns.scatterplot(data=new_df_blobs, x="x", y="y",
hue="cluster", palette="tab10").set(title="Кластеров: 5")
```

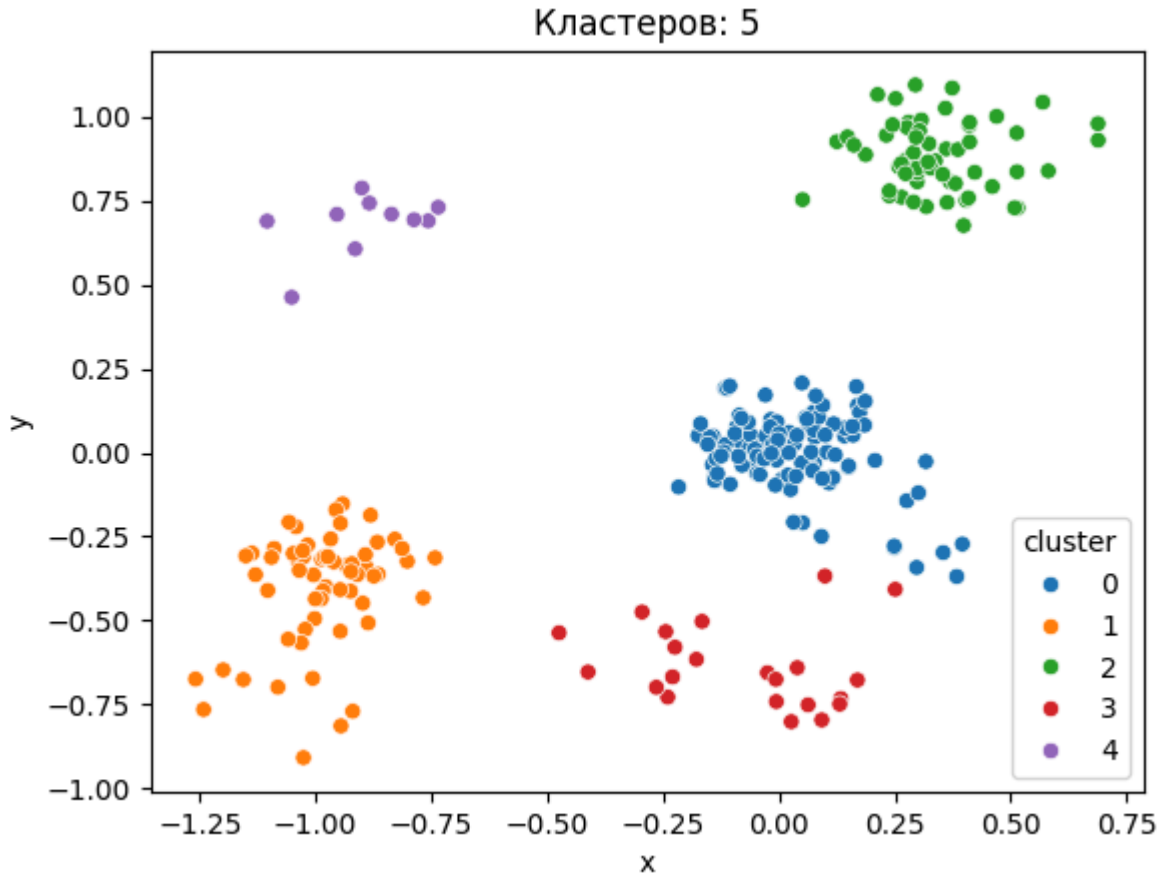

Рисунок 2.4.1 – диаграмма рассеяния результатов кластеризации для df_blobs

Для df_checker (листинг 2.4.2, рис. 2.4.2):

Листинг 2.4.2 – построение диаграммы рассеяния результатов кластеризации для df_blobs

```
sns.scatterplot(data=new_df_blobs, x="x", y="y",
hue="cluster", palette="tab10").set(title="Кластеров: 5")
```

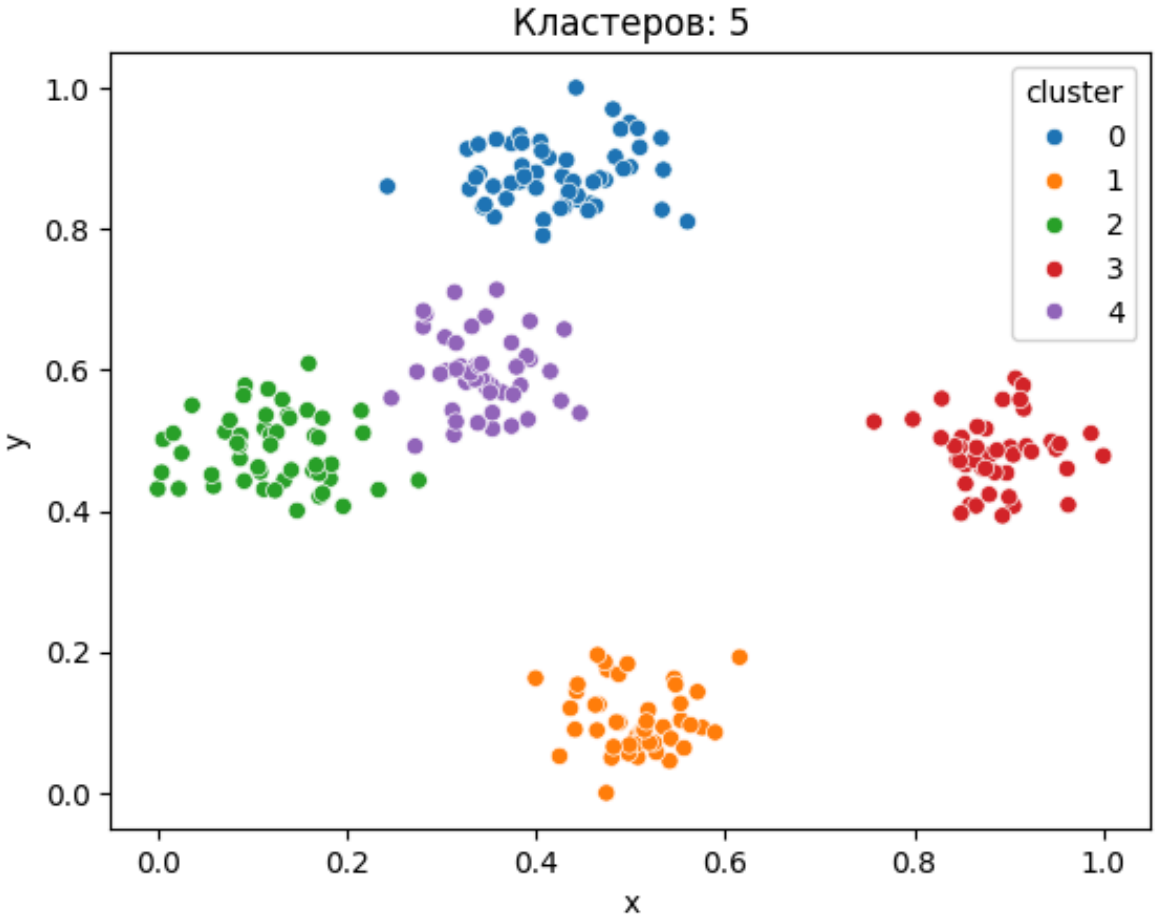

Рисунок 2.4.2 – диаграмма рассеяния результатов кластеризации для df_checker

Для df_circles (листинг 2.4.3, рис. 2.4.3):

Листинг 2.4.3 – построение диаграммы рассеяния результатов кластеризации для df_blobs

```
sns.scatterplot(data=new_df_circles, x="x", y="y",
hue="cluster", palette="tab10").set(title="Кластеров:
13")
```

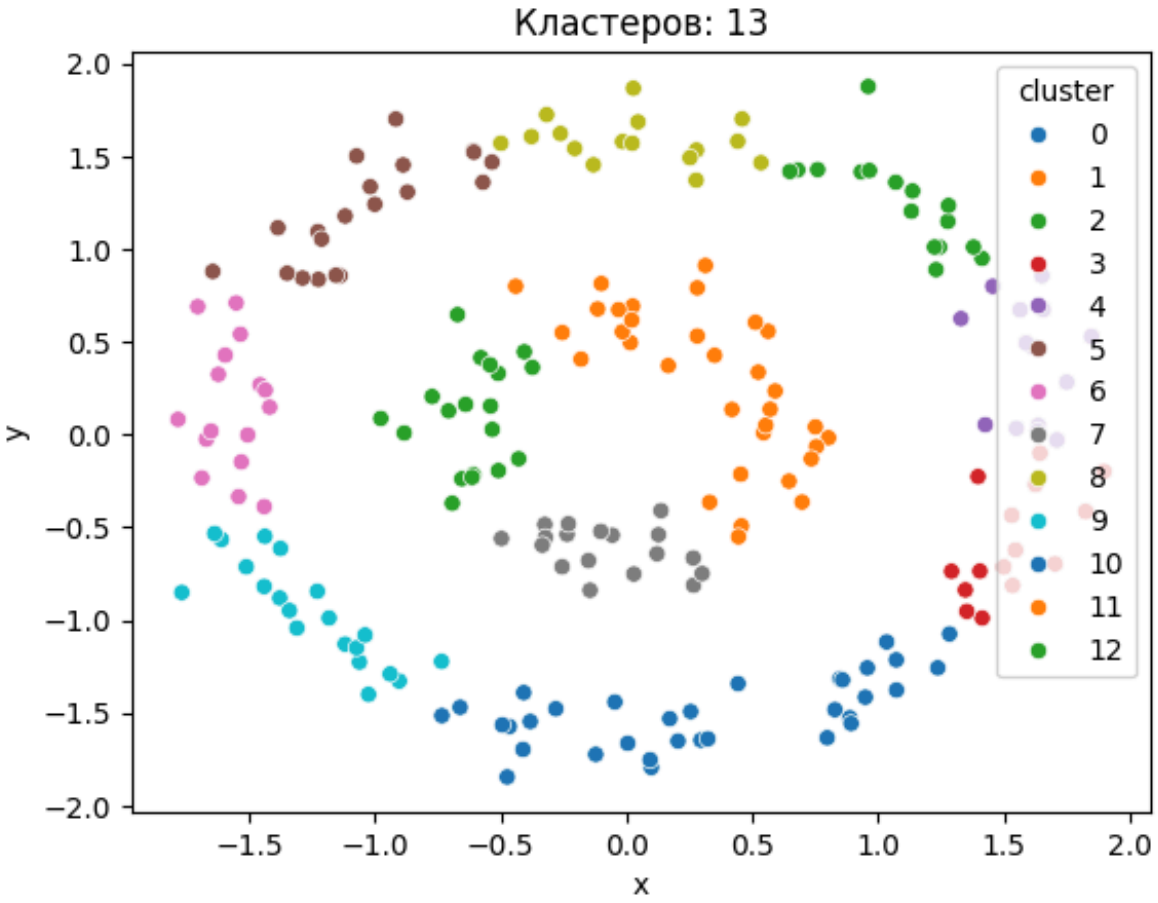

Рисунок 2.4.3 – диаграмма рассеяния результатов кластеризации для df_circles

2.5. Построим диаграммы Вороного для результатов кластеризации.

Для df_blobs (листинг 2.5.1, рис. 2.5.1):

Листинг 2.5.1 – построение диаграммы Вороного для результатов кластеризации df_blobs

```
h = 0.01
x_min, x_max = df_blobs[:, 0].min() - 0.5, df_blobs[:,
0].max() + 0.5
y_min, y_max = df_blobs[:, 1].min() - 0.5, df_blobs[:,
1].max() + 0.5
X, Y = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))

centroids = kmeans_df_blobs.cluster_centers_
```

```
clusters_list_im =
kmeans_df_blobs.predict(np.c_[X.ravel(), Y.ravel()])
clusters_list_im = clusters_list_im.reshape(X.shape)

plt.imshow(
    clusters_list_im,
    interpolation="nearest",
    extent=(X.min(), X.max(), Y.min(), Y.max()),
    cmap=plt.cm.Paired,
    aspect="auto",
    origin="lower"
)
plt.scatter(
    df_blobs[:, 0],
    df_blobs[:, 1],
    s=15,
    color="black"
)
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    marker="x",
    s=169,
    linewidths=3,
    color="white"
)
```

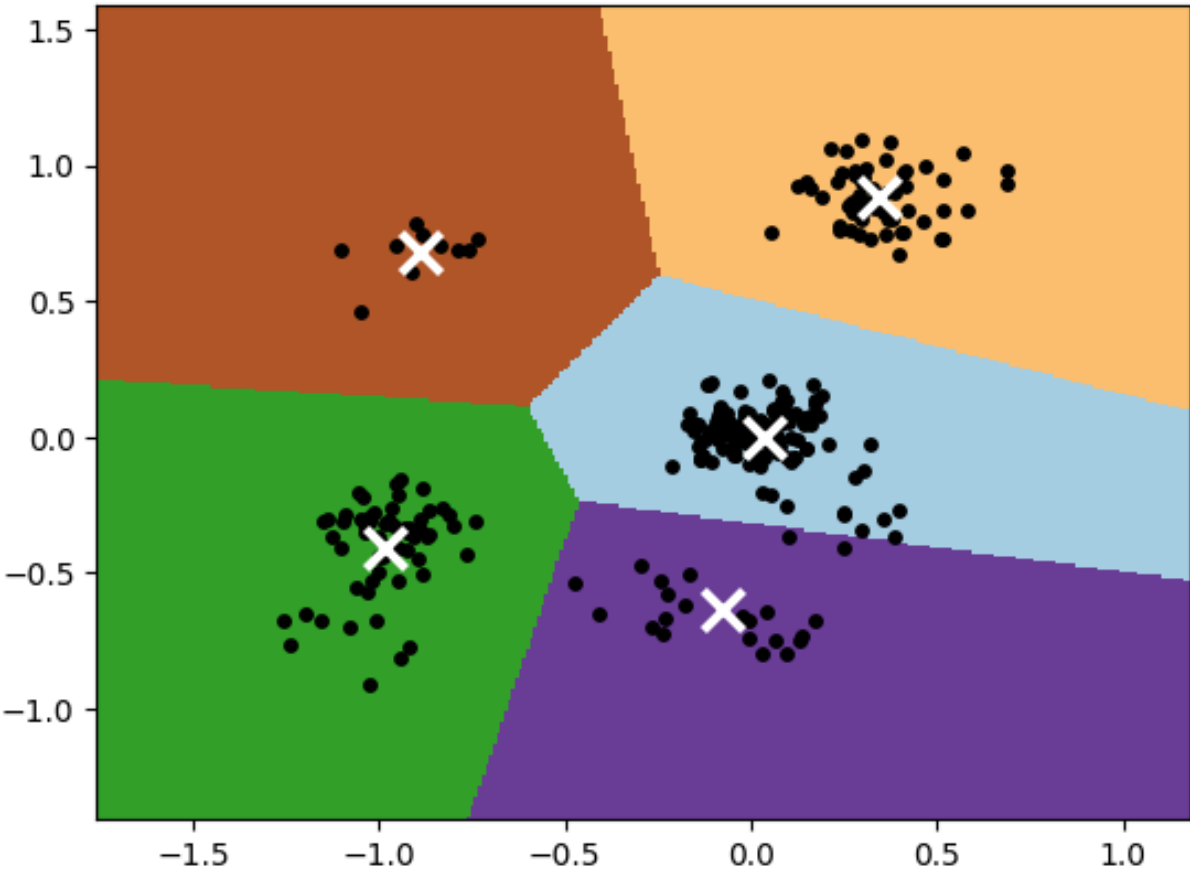

Рисунок 2.5.1 – диаграмма Вороного для результатов кластеризации df_blobs

Для df_checker (листинг 2.5.2, рис. 2.5.2):

Листинг 2.5.2 – построение диаграммы Вороного для результатов кластеризации df_checker

```
h = 0.01

x_min, x_max = df_checker[:, 0].min() - 0.2,
df_checker[:, 0].max() + 0.2
y_min, y_max = df_checker[:, 1].min() - 0.2,
df_checker[:, 1].max() + 0.2
X, Y = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))

centroids = kmeans_df_checker.cluster_centers_
clusters_list_im =
```

```
kmeans_df_checker.predict(np.c_[X.ravel(), Y.ravel()])
clusters_list_im = clusters_list_im.reshape(X.shape)

plt.imshow(
    clusters_list_im,
    interpolation="nearest",
    extent=(X.min(), X.max(), Y.min(), Y.max()),
    cmap=plt.cm.Paired,
    aspect="auto",
    origin="lower"
)
plt.scatter(
    df_checker[:, 0],
    df_checker[:, 1],
    s=15,
    color="black"
)
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    marker="x",
    s=169,
    linewidths=3,
    color="white"
)
```

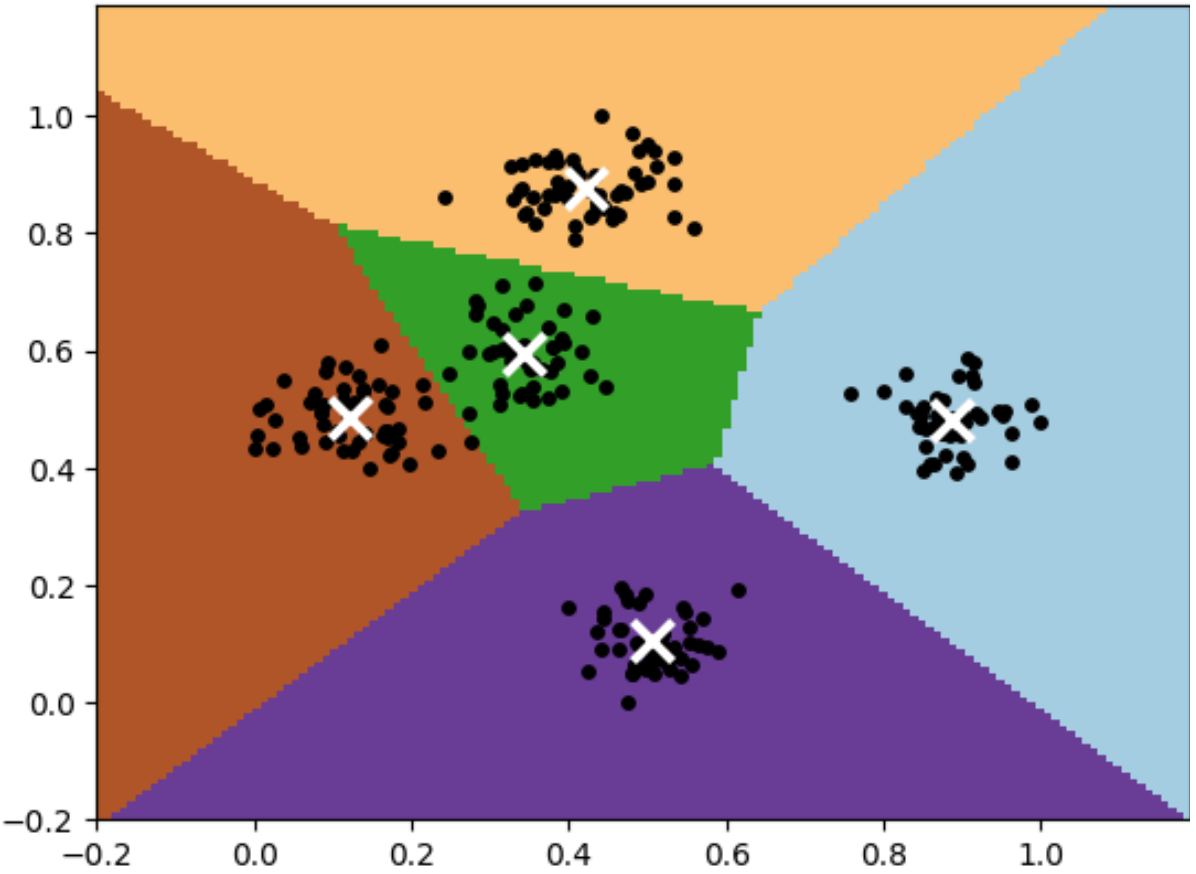

Рисунок 2.5.2 – диаграмма Вороного для результатов кластеризации df_checker

Для df_circles (листинг 2.5.3, рис. 2.5.3):

Листинг 2.5.3 – построение диаграммы Вороного для результатов кластеризации df_circles

```
h = 0.01
x_min, x_max = df_circles[:, 0].min() - 0.2,
df_circles[:, 0].max() + 0.2
y_min, y_max = df_circles[:, 1].min() - 0.2,
df_circles[:, 1].max() + 0.2
X, Y = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
centroids = kmeans_df_circles.cluster_centers_
clusters_list_im =
kmeans_df_circles.predict(np.c_[X.ravel(), Y.ravel()])
clusters_list_im = clusters_list_im.reshape(X.shape)
```

```
plt.imshow(
    clusters_list_im,
    interpolation="nearest",
    extent=(X.min(), X.max(), Y.min(), Y.max()),
    cmap=plt.cm.Paired,
    aspect="auto",
    origin="lower"
)
plt.scatter(
    df_circles[:, 0],
    df_circles[:, 1],
    s=15,
    color="black"
)
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    marker="x",
    s=169,
    linewidths=3,
    color="white"
)
```

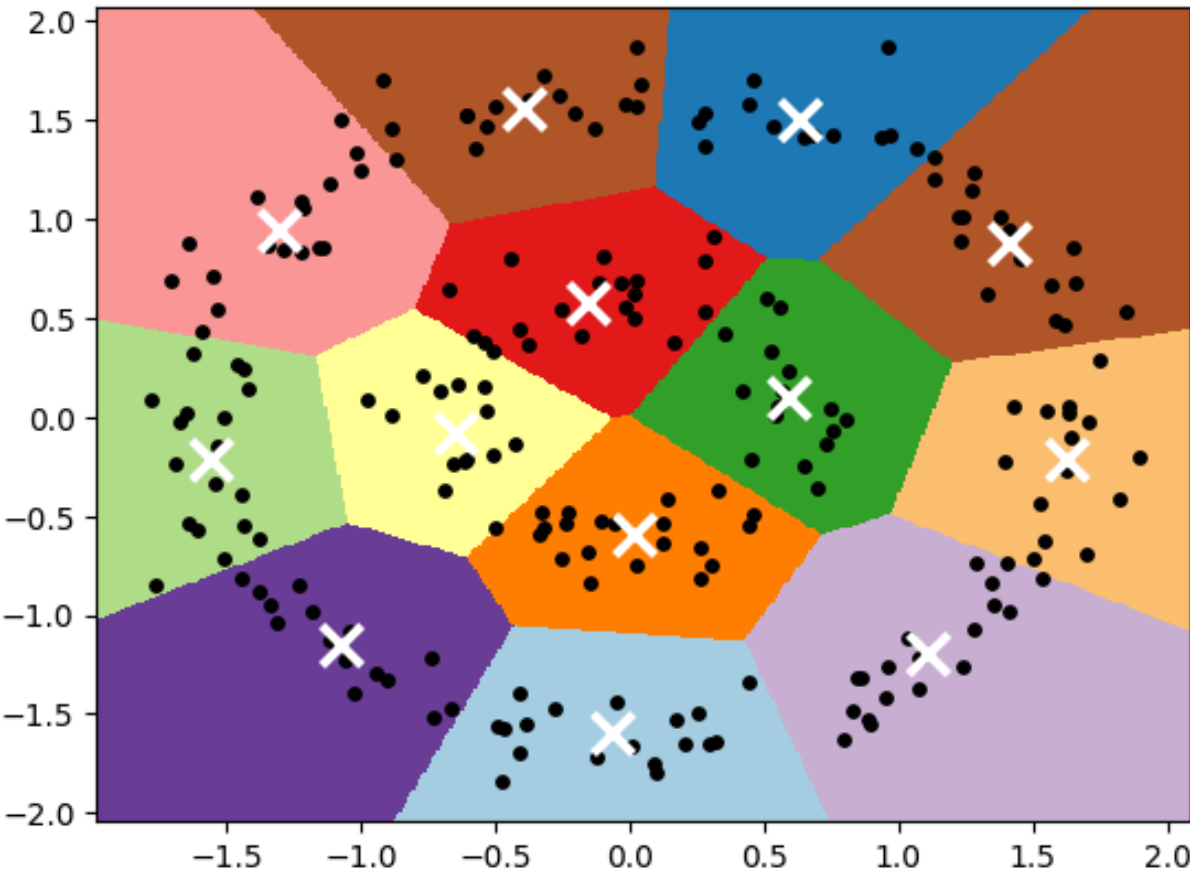

Рисунок 2.5.3 – диаграмма Вороного для результатов кластеризации df_circles

Из рисунков 2.5.1-2.5.3 можно сделать вывод, о разбиении плоскости на части множеств разных классов. На диаграммах, можно увидеть, почему те ил иные точки принадлежат конкретному классу, а также расположение центроидов.

2.6. Построим для каждого признака диаграмму “box-plot”, с разделением по кластерам. Сделаем выводы о разделении кластеров и успешности применения кластеризации K-means к набору данных.

Для df_blobs (листинг 2.6.1, рис. 2.6.1):

Листинг 2.6.1 – построение диаграммы box-plot для df_blobs

```
fig, axs = plt.subplots(1,
len(new_df_blobs.columns[:-1]), figsize=(5 *
```

```
(len(new_df_blobs.columns)), 5))
for i, column in enumerate(new_df_blobs.columns[:-1]):
  sns.boxplot(data=new_df_blobs, x="cluster", y=column,
ax=axs[i], hue="cluster", palette="tab10")
```

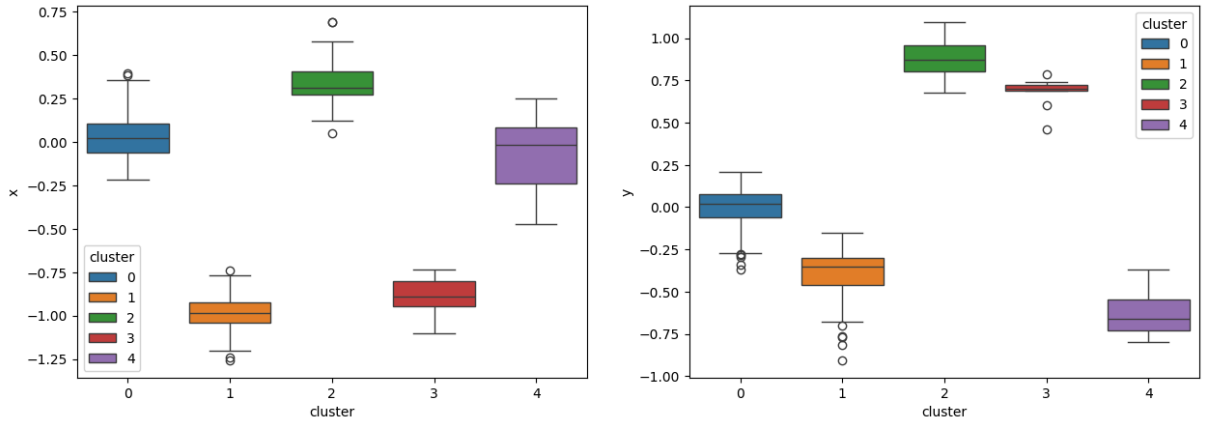

Рисунок 2.6.1 – диаграмма box-plot для df_blobs

Из рисунка 2.6.1 можно сделать следующие выводы:

- На левой диаграмме для кластеров 0, 1, 2 замечены выбросы и малый интерквартильный размах, также для кластера 4 большая часть данных находится в нижней квартили, а в кластере 2 – в верхней.
- На правой диаграмме для кластеров 0, 1, 3 обнаружены выбросы и крайне малый интерквартильный размах для кластера 3. Для кластера 4 большая часть данных находится в верхней квартили, а для кластера 1 – в нижней.

Для df_checker (листинг 2.6.2, рис. 2.6.2):

Листинг 2.6.2 – построение диаграммы box-plot для df_checker

```
fig, axs = plt.subplots(1,
len(new_df_checker.columns[:-1]), figsize=(5 *
(len(new_df_checker.columns)), 5))
for i, column in enumerate(new_df_checker.columns[:-1]):
  sns.boxplot(data=new_df_checker, x="cluster", y=column,
```

```
ax=axs[i], hue="cluster", palette="tab10")
```

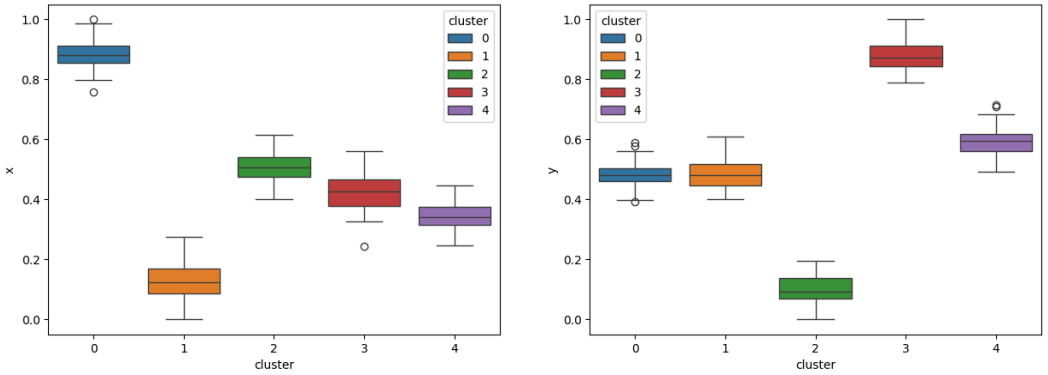

Рисунок 2.6.2 – диаграмма box-plot для df_checker

Из рисунка 2.6.2 можно сделать следующие выводы:

- На левой диаграмме для кластеров 0, 3 замечены выбросы и малый интерквартильный размах у всех кластеров, также для кластера 3 большая часть данных находится в нижней квартили.
- На правой диаграмме для кластеров 0, 4 обнаружены выбросы и крайне малый интерквартильный размах для кластера 0. Для кластера 2 большая часть данных находится в верхней квартили.

Для df_circles (листинг 2.6.3, рис. 2.6.3):

Листинг 2.6.3 – построение диаграммы box-plot для df_circles

```
fig, axs = plt.subplots(1,
len(new_df_circles.columns[:-1]), figsize=(5 *
(len(new_df_circles.columns)), 5))
plt.subplots_adjust(wspace=0.28)
for i, column in enumerate(new_df_circles.columns[:-1]):
  sns.boxplot(data=new_df_circles, x="cluster", y=column,
ax=axs[i], hue="cluster", palette="tab10")
  axs[i].legend(bbox_to_anchor=(1.02, 1), loc='upper
left', borderaxespad=0, title="cluster")
```

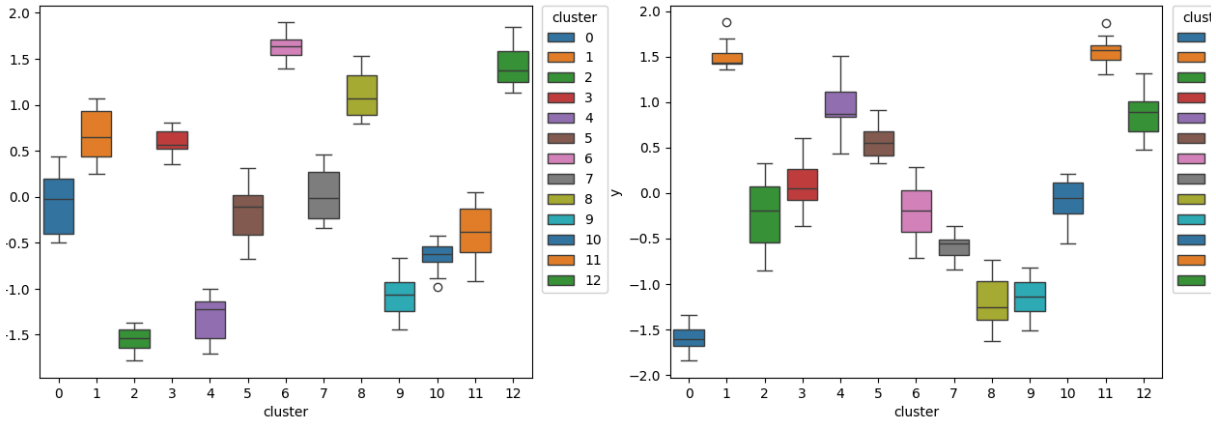

Рисунок 2.6.3 – диаграмма box-plot для df_circles

Из рисунка 2.6.3 можно сделать следующие выводы:

- На левой диаграмме мы видим выброс у кластера 10, большие интерквартильные размахи у кластеров 0, 1, 7, для кластера 3 большая часть данных находится в верхней квартили, у кластера 12 – в нижней.
- На правой диаграмме мы видим выбросы у 1 и 11 кластера и большой интерквартильный размах у кластера 2. Для кластеров 1, 4 большая часть данных находится в верхней квартили, для кластеров 7 и 11 – в нижней.

2.7. Рассчитаем для каждого кластера кол-во точек, среднее, СКО, минимум и максимум. Сопоставим результаты с построенными графиками.

Для df_blobs (листинг 2.7.1, таблица 2.7.1):

Листинг 2.7.1 – расчет кол-ва точек, среднего, СКО, минимума и максимума для каждого кластера df_blobs

```
stats = new_df_blobs.groupby('cluster').agg(['count',
'mean', 'std', 'min', 'max'])
stats
```

Таблица 2.7.1 – кол-во точек, среднее, СКО, минимум и максимум для каждого кластера df_blobs

	x	y

cluster	count	mean	std	min	max	count	mean	std	min	max
0	108	0.033117	0.130049	-0.216461	0.395828	108	0.001065	0.116065	-0.368633	0.206640
1	60	-0.985492	0.107505	-1.259026	-0.742091	60	-0.404092	0.168927	-0.907573	-0.152502
2	60	0.341517	0.123038	0.051061	0.688925	60	0.881711	0.100603	0.675308	1.093235
3	10	-0.892482	0.121068	-1.104643	-0.735323	10	0.681247	0.089950	0.461660	0.786807
4	22	-0.079961	0.200390	-0.474481	0.251056	22	-0.635102	0.121166	-0.800494	-0.367181

Из таблицы 2.7.1 можно понять, что большинство элементов находится в кластере 0, а наименьшее в 3. Сопоставив полученные данные из таблицы и полученные графики, можно предположить, что данные из кластера 3 лучше рассматривать как выбросы или шум, а не как отдельный кластер. Значения среднего, СКО, максимум и минимума варьируются в промежутке интерквартильного размаха из-за преобразования данных с помощью RobustScaler.

Для df_checker (листинг 2.7.2, таблица 2.7.2):

Листинг 2.7.2 – расчет кол-ва точек, среднего, СКО, минимума и максимума для каждого кластера df_checker

```
stats = new_df_checker.groupby('cluster').agg(['count',
'mean', 'std', 'min', 'max'])
stats
```

Таблица 2.7.2 – кол-во точек, среднее, СКО, минимум и максимум для каждого кластера df_checker

	x					y				
cluster	count	mean	std	min	max	count	mean	std	min	max

0	46	0.886668	0.046826	0.757464	1.000000	46	0.481167	0.047662	0.392983	0.588039
1	53	0.122688	0.061366	0.000000	0.275951	53	0.485998	0.049018	0.400259	0.608984
2	46	0.504848	0.045788	0.399567	0.615417	46	0.103930	0.046222	0.000000	0.195799
3	55	0.421779	0.064685	0.242856	0.560222	55	0.877348	0.043846	0.790301	1.000000
4	50	0.343477	0.042936	0.246900	0.446452	50	0.594685	0.053326	0.491675	0.713658

Из таблицы 2.7.2 можно сказать, что данные распределены по кластерам примерно равномерно. Значения среднего, СКО, максимум и минимума варьируются в промежутке от 0 до 1 из-за преобразования данных с помощью MinMaxScaler . Сопоставив полученные данные с таблицы и полученные графики, можно говорить, об успешной кластеризации данных.

Для df_circles (листинг 2.7.3, таблица 2.7.3):

Листинг 2.7.3 – расчет кол-ва точек, среднего, СКО, минимума и максимума для каждого кластера df_circles

```
stats = new_df_circles.groupby('cluster').agg(['count',
'mean', 'std', 'min', 'max'])
stats
```

Таблица 2.7.3 – кол-во точек, среднее, СКО, минимум и максимум для каждого кластера df_circles

	x					y				
cluster	count	mean	std	min	max	count	mean	std	min	max
0	18	-0.066180	0.318719	-0.493170	0.443384	18	-1.598161	0.138855	-1.844962	-1.341925
1	13	0.635523	0.28732	0.25	1.07	13	1.499	0.14	1.359770	1.875692

			0	2476	0063		238	6391		
2	18	-1.559489	0.123162	-1.782003	-1.374778	18	-0.214980	0.366403	-0.850618	0.324621
3	16	0.591762	0.130037	0.350323	0.802594	16	0.093443	0.281746	-0.363440	0.604967
4	17	-1.306360	0.225787	-1.703627	-1.000057	17	0.942968	0.281166	0.428367	1.501347
5	20	-0.155765	0.304288	-0.671372	0.313527	20	0.572984	0.173868	0.329742	0.911415
6	15	1.624724	0.139354	1.396933	1.897220	15	-0.217733	0.303627	-0.713336	0.282330
7	20	0.016110	0.266159	-0.334602	0.456749	20	-0.595969	0.130529	-0.838558	-0.364597
8	19	1.108413	0.237612	0.797644	1.534215	19	-1.189940	0.284014	-1.633670	-0.735473
9	16	-1.072565	0.236924	-1.439623	-0.660977	16	-1.145522	0.218926	-1.514230	-0.818404
10	14	-0.646669	0.152429	-0.976297	-0.428489	14	-0.083201	0.228980	-0.559864	0.205181
11	17	-0.394527	0.327115	-0.916401	0.045893	17	1.562258	0.137076	1.307236	1.866606
12	17	1.413171	0.209077	1.131939	1.847389	17	0.875897	0.267301	0.470989	1.312853

Из таблицы 2.7.3 можно сказать, что данные распределены по кластерам примерно равномерно. Значения среднего, СКО, максимум и минимума центрируются вокруг нуля из-за стандартизации данных с помощью StandardScaler. Сопоставив полученные данные с таблицы и полученные графики, можно сказать, что кластеризацию данных сложно назвать оптимальной.

3. Изучение работы DBSCAN

3.1-3.3. Подберем параметры алгоритма DBSCAN, которые дают наилучшие результаты. Построим диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров.

Для df_blobs (листинг 3.1.1, рис. 3.1.1):

Листинг 3.1.1 – кластеризация df_blobs с помощью DBSCAN

```
dbscan = DBSCAN(eps = 0.2, min_samples = 1)
dbscan_clust = dbscan.fit_predict(df_blobs)
print(set(dbscan_clust))
plt.scatter(df_blobs[dbscan_clust == 0,0],
df_blobs[dbscan_clust == 0,1], c = 'r')
plt.scatter(df_blobs[dbscan_clust == 1,0],
df_blobs[dbscan_clust == 1,1], c = 'b')
plt.scatter(df_blobs[dbscan_clust == 2,0],
df_blobs[dbscan_clust == 2,1], c = 'g')
plt.scatter(df_blobs[dbscan_clust == 3,0],
df_blobs[dbscan_clust == 3,1], c = 'y')
plt.scatter(df_blobs[dbscan_clust == 4,0],
df_blobs[dbscan_clust == 4,1], c = 'c')
plt.scatter(df_blobs[dbscan_clust == -1,0],
df_blobs[dbscan_clust == -1,1], c ='k', marker = 'x')
plt.show()
```

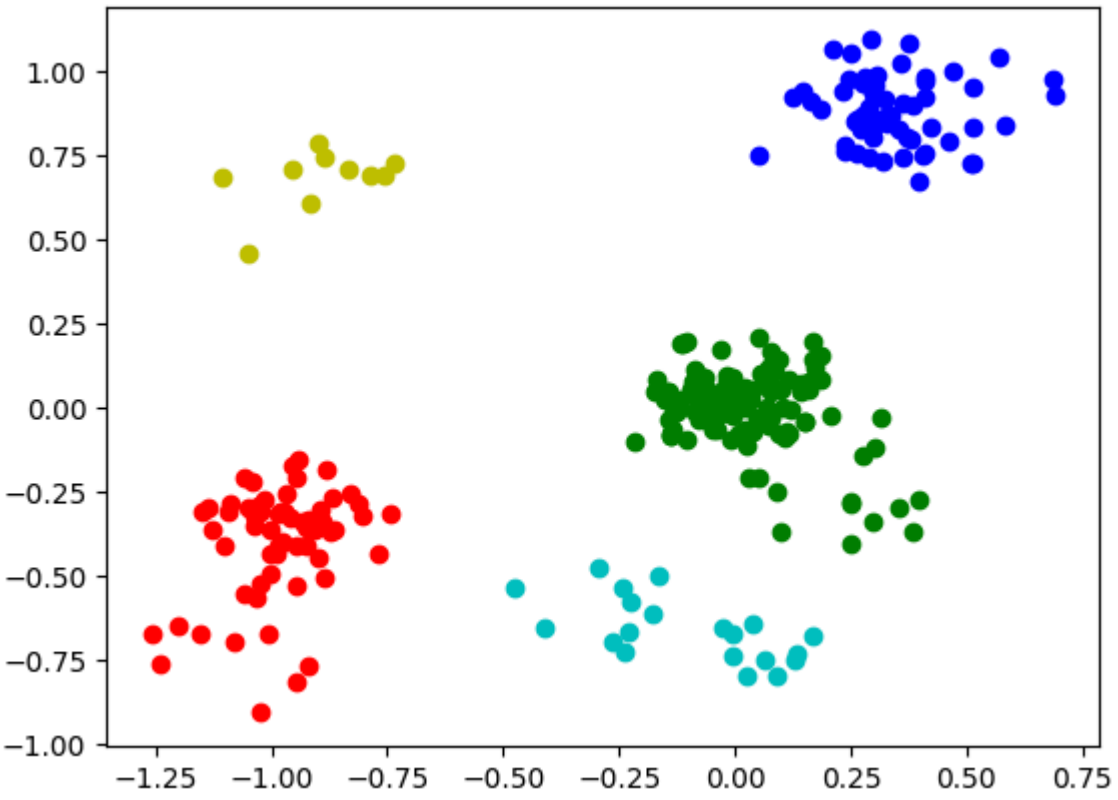

Рисунок 3.1.1 – диаграмма рассеяния результатов кластеризации df_blobs

Параметры eps = 0,2 и min_samples = 3 являются оптимальными, т.к. изменение min_samples от 1 до 6 не меняет результата, т.к. точки в кластере расположены близко друг к другу, а уменьшение eps до 0,1 увеличивает количество точек, не входящих в кластеры и порождает дополнительные кластеры, расположенные слишком близко к друг другу, а увеличение до 0,3 объединяет достаточно удаленные голубой и зеленый кластеры (рис. 3.1.1) в один.

Полученный результат говорит об успешности кластеризации.

Для df_checker (листинг 3.1.2, рис. 3.1.2):

Листинг 3.1.2 – кластеризация df_checker с помощью DBSCAN

```
dbscan = DBSCAN(eps = 0.046, min_samples = 3)
dbscan_clust = dbscan.fit_predict(df_checker)
print(set(dbscan_clust))
plt.scatter(df_checker[dbscan_clust == 0,0],
df_checker[dbscan_clust == 0,1], c = 'r')
plt.scatter(df_checker[dbscan_clust == 1,0],
```

```
df_checker[dbscan_clust == 1,1], c = 'b')
plt.scatter(df_checker[dbscan_clust == 2,0],
df_checker[dbscan_clust == 2,1], c = 'g')
plt.scatter(df_checker[dbscan_clust == 3,0],
df_checker[dbscan_clust == 3,1], c = 'y')
plt.scatter(df_checker[dbscan_clust == 4,0],
df_checker[dbscan_clust == 4,1], c = 'c')
plt.scatter(df_checker[dbscan_clust == 5,0],
df_checker[dbscan_clust == 5,1], c = 'gray')
plt.scatter(df_checker[dbscan_clust == -1,0],
df_checker[dbscan_clust == -1,1], c ='k', marker = 'x')
plt.show()
```

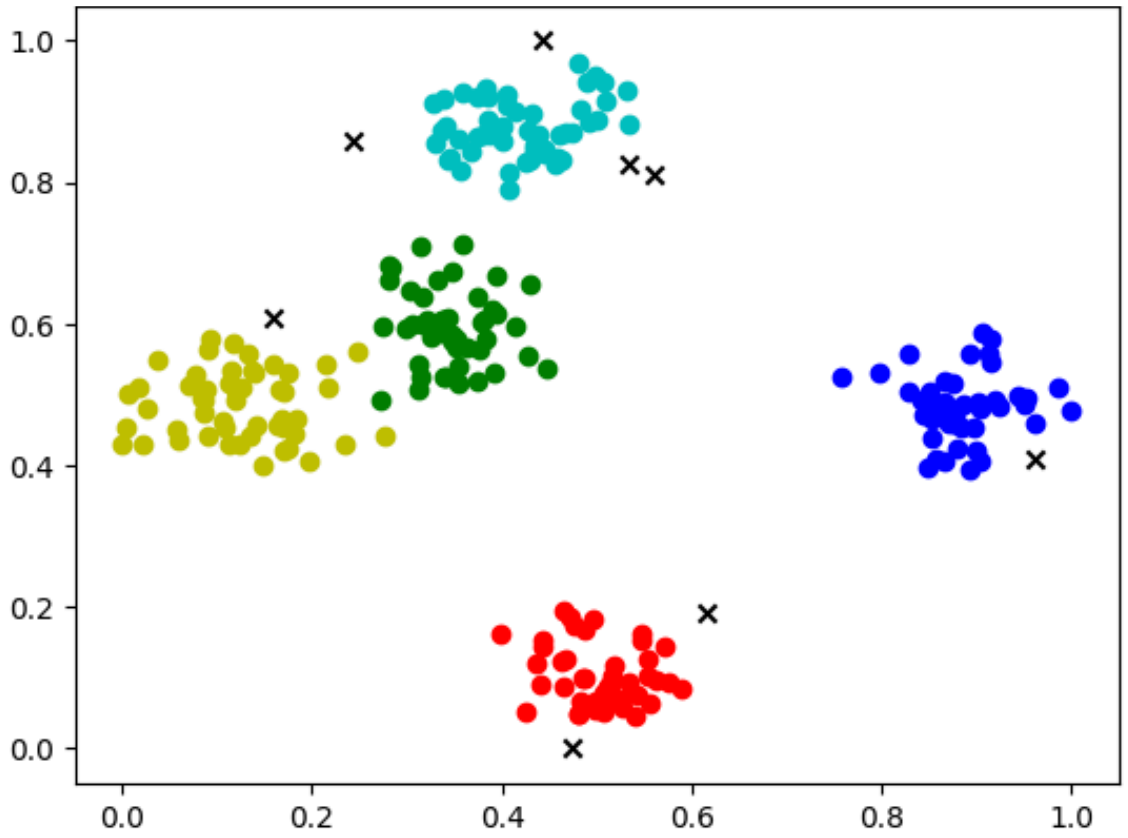

Рисунок 3.1.2 – диаграмма рассеяния результатов кластеризации df_checker

Параметры eps = 0,046 и min_samples = 3 являются оптимальными, т.к. изменение min_samples в меньшую сторону приведет к большому количеству мелких кластеров рядом друг с другом, а уменьшение eps порождает чрезмерно большое количество дополнительных кластеров,

расположенные слишком близко к друг другу, а увеличение до объединяет достаточно удаленные голубой, зеленый и желтый кластеры (рис. 3.1.2) в один.

Полученный результат говорит о не самой успешной кластеризации ввиду соприкосновения желтого и зеленых кластеров

Для df_circles (листинг 3.1.3, рис. 3.1.3):

Листинг 3.1.3 – кластеризация df_circles с помощью DBSCAN

```
dbscan = DBSCAN(eps = 0.27, min_samples = 3)
dbscan_clust = dbscan.fit_predict(df_circles)
print(set(dbscan_clust))
plt.scatter(df_circles[dbscan_clust == 0,0],
df_circles[dbscan_clust == 0,1], c = 'r')
plt.scatter(df_circles[dbscan_clust == 1,0],
df_circles[dbscan_clust == 1,1], c = 'b')
plt.scatter(df_circles[dbscan_clust == 2,0],
df_circles[dbscan_clust == 2,1], c = 'g')
plt.scatter(df_circles[dbscan_clust == -1,0],
df_circles[dbscan_clust == -1,1], c ='k', marker = 'x')
plt.show()
```

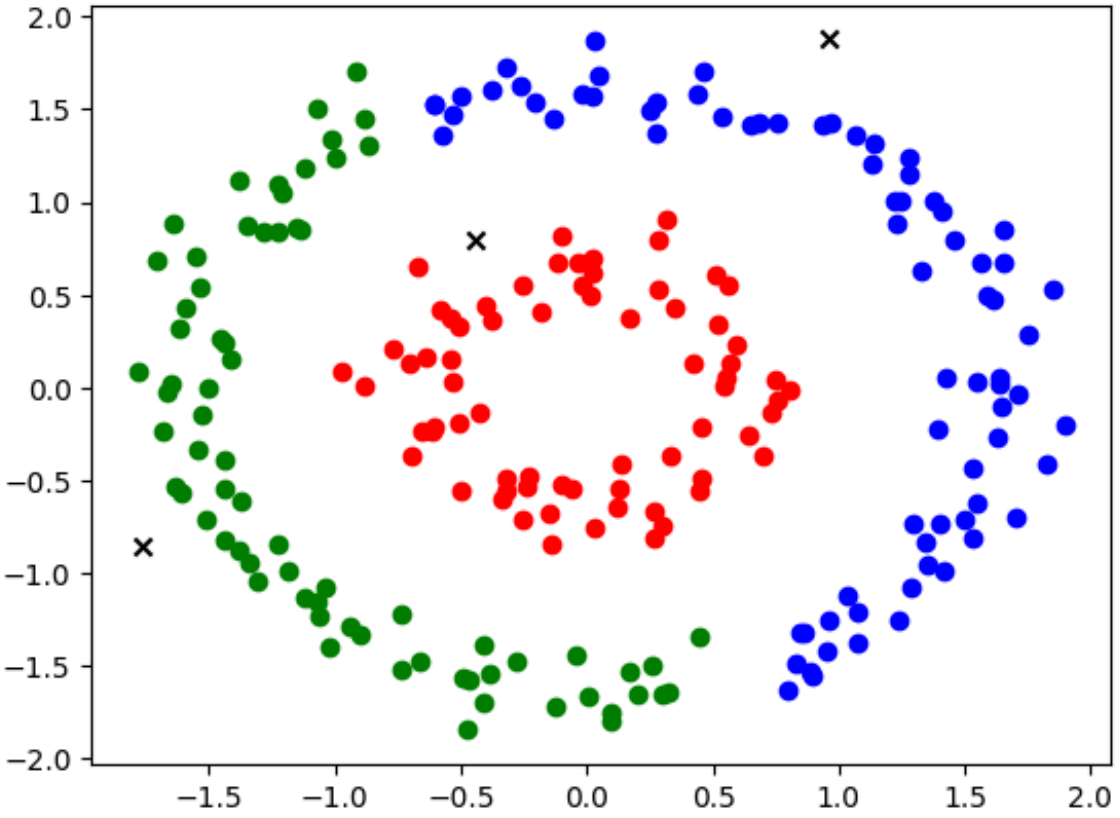

Рисунок 3.1.3 – диаграмма рассеяния результатов кластеризации df_circles

Параметры eps = 0,27 и min_samples = 3 являются оптимальными, т.к. изменение min_samples в меньшую сторону приведет к большому количеству кластеров состоящих из одной точки, а увеличение приводит к росту количества плотно соседствующих кластеров и появлению выбросов. В то время как уменьшение eps порождает большое количество дополнительных кластеров с чрезмерно большим количеством точек, не входящих ни в один кластер, а увеличение до 0,4 объединяет достаточно удаленные (в точках (-0,75, 1,5) и (0,6, -1,5)) зеленый и синий кластеры (рис. 3.1.3) в один. Также с такими данными количество точек, не попавших в кластеры, минимально.

Полученный результат говорит о достаточно успешной кластеризации набора данных.

4. Изучение иерархической кластеризации

4.1. Проведем иерархическую кластеризацию при всех возможных параметрах linkage, используя количество

кластеров полученных в п.2 или п.3. Для каждого из результатов построим дендрограмму.

На построенных дендрограммах будем ориентироваться на то, чтобы графики были примерно на одном уровне по расстоянию.

Для df_blobs (листинг 4.1.1, рис. 4.1.1):

Листинг 4.1.1 – построение дендрограмм для df_blobs с разными linkage

```
fig, axis = plt.subplots(1, 4, figsize = (20, 5))
linkages = ["ward", "complete", "average", "single"]
for i in range(4):
  agc = AgglomerativeClustering(distance_threshold=0,
n_clusters = None, linkage=linkages[i])
  agc = agc.fit(df_blobs)
  plot_dendrogram(agc, truncate_mode="level", p=3, ax =
axis[i])
  axis[i].set_title(linkages[i])
```

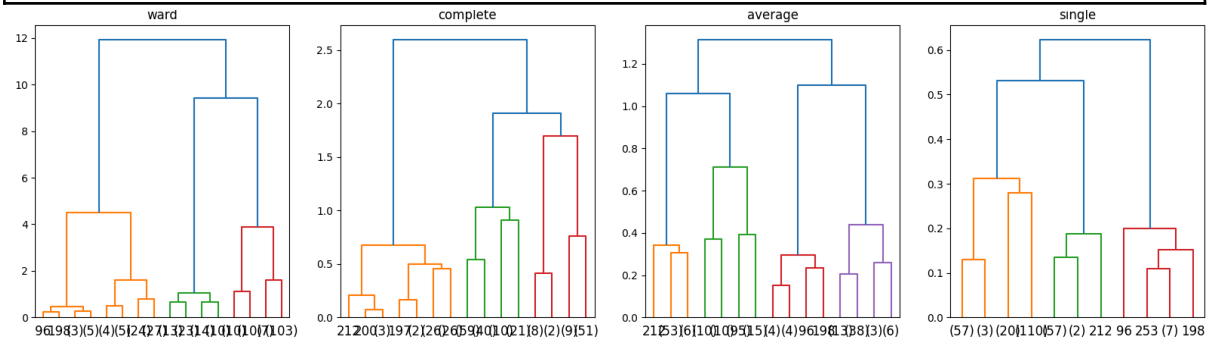

Рисунок 4.1.1 – дендрограммы для df_blobs

Для df_checker (листинг 4.1.2, рис. 4.1.2):

Листинг 4.1.2 – построение дендрограмм для df_checker с разными linkage

```
fig, axis = plt.subplots(1, 4, figsize = (20, 5))
linkages = ["ward", "complete", "average", "single"]
for i in range(4):
  agc = AgglomerativeClustering(distance_threshold=0,
```

```
n_clusters = None, linkage=linkages[i])
  agc = agc.fit(df_checker)
  plot_dendrogram(agc, truncate_mode="level", p=3, ax =
axis[i])
  axis[i].set_title(linkages[i])
```

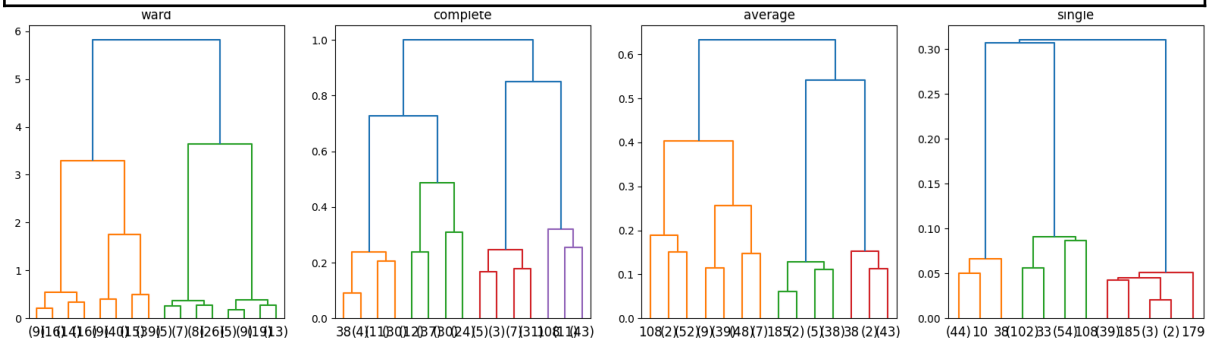

Рисунок 4.1.2 – дендрограммы для df_checker

Для df_circles (листинг 4.1.3, рис. 4.1.3):

Листинг 4.1.3 – построение дендрограмм для df_circles с разными linkage

```
fig, axis = plt.subplots(1, 4, figsize = (20, 5))
linkages = ["ward", "complete", "average", "single"]
for i in range(4):
  agc = AgglomerativeClustering(distance_threshold=0,
n_clusters = None, linkage=linkages[i])
  agc = agc.fit(df_circles)
  plot_dendrogram(agc, truncate_mode="level", p=3, ax =
axis[i])
  axis[i].set_title(linkages[i])
```

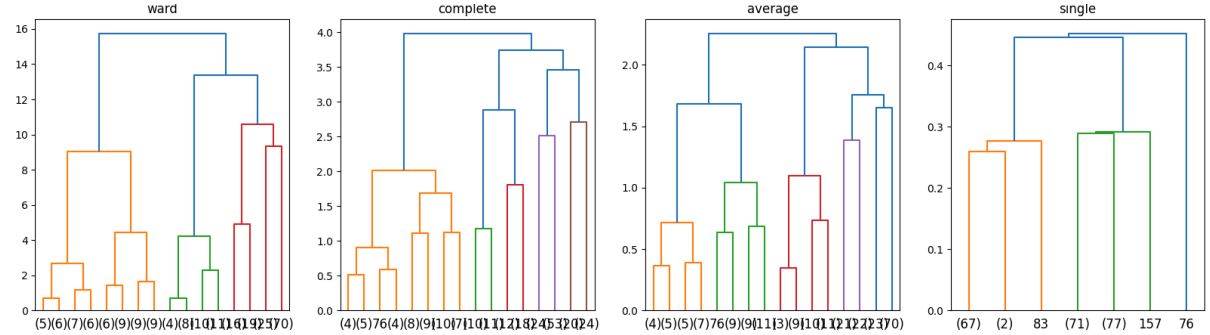

Рисунок 4.1.3 – дендрограммы для df_circles

Выберем оптимальное расстояние при кластеризации для df_blobs (рис. 4.1.1):

- ward: 2, 5 кластеров
- complete: 0.8, 6 кластеров
- average: 0.5, 5 кластеров
- single: 0.2, 5 кластеров

Таким образом, для параметров ward, average и single при оптимальном расстоянии количество кластеров совпадает с результатом K-means и DBSCAN, для complete при оптимальном расстоянии количество кластеров входит в промежуток метода локтя (рис. 2.1.1).

Выберем оптимальное расстояние при кластеризации для df_checker (рис. 4.1.2):

- ward: 1, 5 кластеров
- complete: 0.4, 5 кластеров
- average: 0.2, 5 кластеров
- single: 0.07, 5 кластеров

Таким образом, для параметров ward, average, complete и single при оптимальном расстоянии, совпало количество кластеров с результатом алгоритмом K-Means и DBSCAN.

Выберем оптимальное расстояние при кластеризации для df_circles (рис. 4.1.3). Из рисунка сложно сказать, какое расстояние является оптимальным, потому что на дендрограммах слишком разные уровни графиков при разном расстоянии. Тогда возьмём примерное расстояние для каждого параметра linkage:

- ward: 5, 6 кластеров
- complete: 1.3, 14 кластеров
- average: 0.8, 13 кластеров
- single: 0.3, 3 кластера

Таким образом, для параметра average при оптимальном расстоянии, совпало количество кластеров с результатом алгоритма K-means. Для параметров complete, ward и single при оптимальном расстоянии количество кластеров равно 14, 6 и 3, которые до этого рассмотрены не были.

4.2. Построим диаграмму рассеяния результатов кластеризации, используя лучшие результаты, полученные для определенных параметров linkage.

Для df_blobs возьмем ward с расстоянием 2 (листинг 4.2.1, рис. 4.2.1), для df_checker возьмем ward с расстоянием 1 (листинг 4.2.2, рис. 4.2.2), для df_circles возьмем average с расстоянием 0.8 (листинг 4.2.3, рис. 4.2.3)

Листинг 4.2.1 – рисование диаграммы с кластерами для df_blobs

```
agc = AgglomerativeClustering(n_clusters=None,
distance_threshold=2, linkage="ward")
ag_blobs_clust = agc.fit_predict(df_blobs)
new_df_blobs = pd.DataFrame(data=df_blobs,
columns=["x","y"])
new_df_blobs["cluster"] = ag_blobs_clust
sns.scatterplot(data=new_df_blobs, x="x", y="y",
hue="cluster", palette="tab10")
```

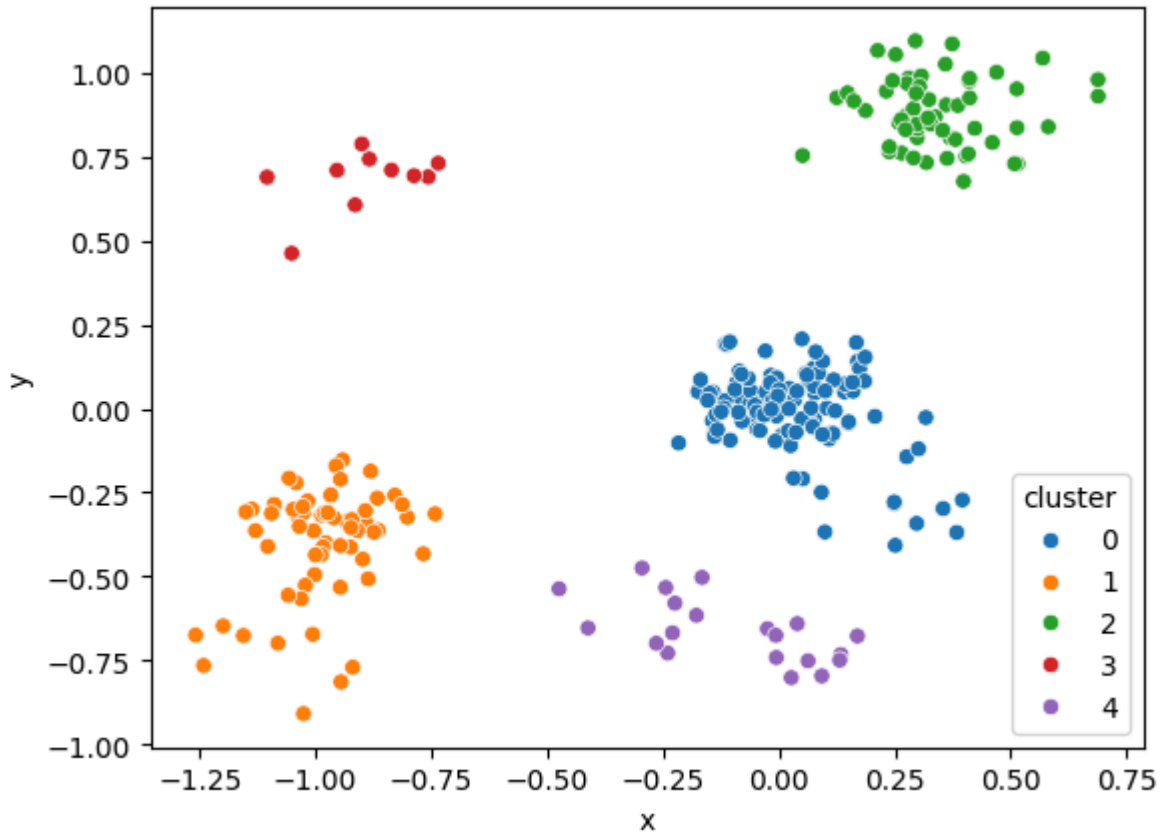

Рисунок 4.2.1 – диаграмма с кластерами для df_blobs

Из рисунка 4.2.1 можно сделать вывод об успешной кластеризации и хорошем разделении данных между кластерами. Результат аналогичен K-means.

Листинг 4.2.2 – рисование диаграммы с кластерами для df_checker

```
agc = AgglomerativeClustering(n_clusters=None,
distance_threshold=1, linkage="ward")
ag_checker_clust = agc.fit_predict(df_checker)
new_df_checker = pd.DataFrame(data=df_checker,
columns=["x","y"])
new_df_checker["cluster"] = ag_checker_clust
sns.scatterplot(data=new_df_checker, x="x", y="y",
hue="cluster", palette="tab10")
```

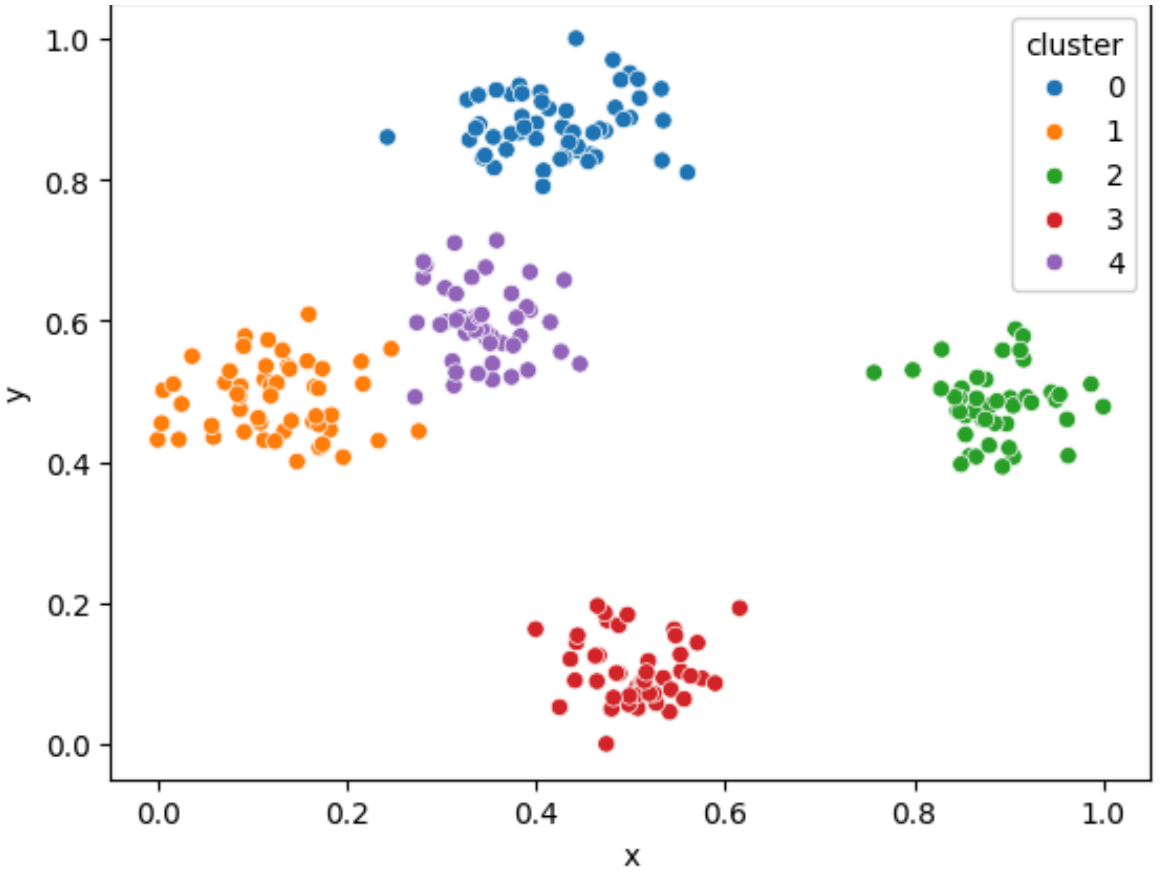

Рисунок 4.2.2 – диаграмма с кластерами для df_checker

Из рисунка 4.2.2 можно сделать вывод об успешной кластеризации и хорошем разделении данных между кластерами, однако кластеры 1 и 4 немного смешиваются. Результат аналогичен K-means.

Листинг 4.2.3 – рисование диаграммы с кластерами для df_circles

```
agc = AgglomerativeClustering(n_clusters=None,
distance_threshold=1.3, linkage="complete")
ag_circles_clust = agc.fit_predict(df_circles)
new_df_circles = pd.DataFrame(data=df_circles,
columns=["x","y"])
new_df_circles["cluster"] = ag_circles_clust
sns.scatterplot(data=new_df_circles, x="x", y="y",
hue="cluster", palette="tab10")
```

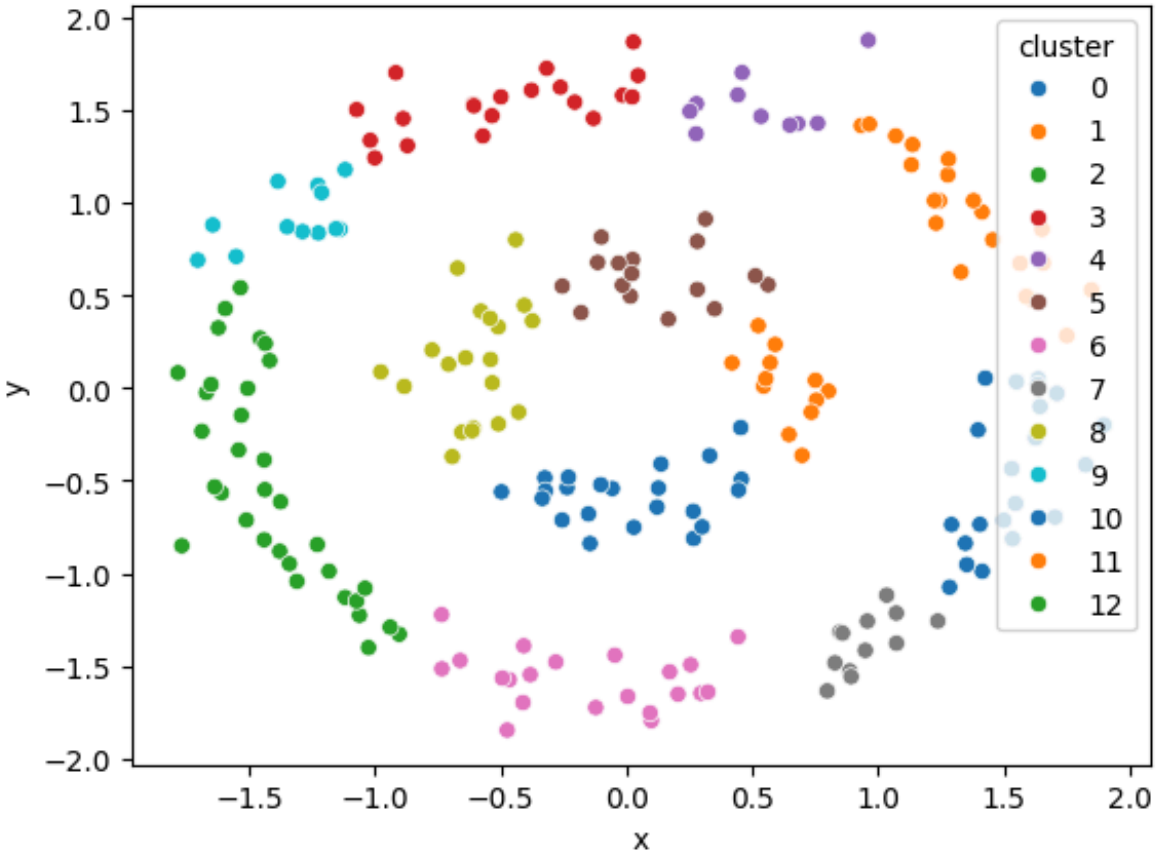

Рисунок 4.2.3 – диаграмма с кластерами для df_circles

Из рисунка 4.2.3 можно сделать вывод о кластеризации и разделении данных между кластерами. Из-за формы данных сложно сказать, что кластеризация была произведена правильно. Полученный результат не похож ни на один из полученных ранее.

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

Для df_blobs можно сказать, что оптимальными методами являются иерархический и DBSCAN. Они четко отделили 5 кластеров (рис. 4.2.1, рис. 3.1.1), в отличие от K-means, где произошло соприкосновение кластеров (рис. 2.4.1).

Для df_checker оптимальными алгоритмами будут являться K-means(рис. 2.4.2) и иерархический (рис. 4.2.2), т.к. в отличие от DBSCAN (рис. 3.1.2) они не дают точек, не отнесенных ни к одному кластеру. Однако и DBSCAN хорошо справился с кластеризацией.

Для df_circles оптимальным будет являться метод DBSCAN, так как он разделяет данные на 3 четко визуально распознаваемых кластера(рис. 3.1.3), в то время как K-means(рис. 2.4.3) и иерархический(рис. 4.2.3) методы разбили данные на чрезмерно большое (13) количество кластеров, сильно пересекающихся между собой.

Вывод.

В ходе лабораторной работы были изучены методы кластеризации данных. Для кластеризации данных были использован алгоритм K-Means, DBSCAN, а также Иерархическая кластеризация. Были изучены методы локтя и силуэта для нахождения лучшего количества кластеров при кластеризации данных