

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

**ОТЧЁТ
по лабораторной работе № 2
по дисциплине «Основы машинного обучения»
Тема: «Кластеризация».**

Студент гр.1304

Поршнев Р.А.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2024

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 проводя стандартизацию или нормировку данных. Обоснуйте выбор операции.

2. K-Means:

2.1. Проведите исследование оптимального количества кластеров методом 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 количеством 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.

Вариант 235К.

1. Подготовка наборов данных:

1.1. Загрузите наборы.

Для загрузки наборов необходимо импортировать библиотеку для работы с датасетами pandas и вызвать метод `read_csv` с параметрами путей нахождения файлов. Реализация данного фрагмента кода представлена в [Листинге 1.1.1](#).

Листинг 1.1.1 – Загрузка данных из файла как Pandas DataFrame

```
import pandas as pd
```

```
scurve_df = pd.read_csv('sample_data/lab2_scurve.csv')
checker_df = pd.read_csv('sample_data/lab2_checker.csv')
luckyset_df = pd.read_csv('sample_data/lab2_luckyset.csv')
```

1.2. Проверьте корректность загрузки.

Для проверки загрузки достаточно посмотреть, например, первые пять записей датасетов и сравнить их с исходными данными в файле. Для просмотра первых пяти записей использован метод `head` без аргументов для каждого датасета. Данный метод должен вывести первые пять записей датасета. Реализация данного фрагмента кода представлена в [Листинге 1.2.1](#), результат выполнения – в [Листинге 1.2.2](#), [Листинге 1.2.3](#) и [Листинге 1.2.4](#) для датасетов `scurve`, `checker` и `luckyset` соответственно.

Листинг 1.2.1 – Проверка загрузки данных

```
scurve_df.head()
checker_df.head()
luckyset_df.head()
```

Листинг 1.2.2 – Результат вывода первых пяти записей датафрейма

	# x	y
0	0.7769	-0.0745
1	1.0976	1.0850
2	-0.1853	1.7859
3	-0.9032	0.3314
4	0.3942	-2.1040

Листинг 1.2.3 – Результат вывода первых пяти записей датафрейма checker

	# x	y
0	4.0510	0.9697
1	7.5581	5.1224
2	2.8765	7.0870
3	3.8366	0.8614
4	4.2159	0.7742

Листинг 1.2.4 – Результат вывода первых пяти записей датафрейма 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

1.3. Постройте диаграмму рассеяния набора данных. Опишите форму данных.

Для реализации данного задания необходимо импортировать библиотеки seaborn и matplotlib, а затем вызвать метод scatterplot для отрисовки диаграммы рассеяния. За счёт использования метода subplots реализована отрисовка всех трёх графиков на одной области. Реализация отрисовки диаграммы рассеяния представлена в [Листинге 1.3.1](#), демонстрация результата – на [Рисунке 1.3.1](#).

Листинг 1.3.1 – Реализация отрисовки диаграммы рассеяния для каждого датасета

```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(12, 4))
df_s = [scurve_df, checker_df, luckyset_df]
titles = ['scurve', 'checker', 'luckyset']
for i, df in enumerate(df_s):
    x, y = df.columns[0], df.columns[1]
    sns.scatterplot(x=df[x], y=df[y], ax=axes[i]).set_title(
        'Диаграмма рассеяния ' + titles[i])
plt.tight_layout(pad=0.5)
plt.show()
```

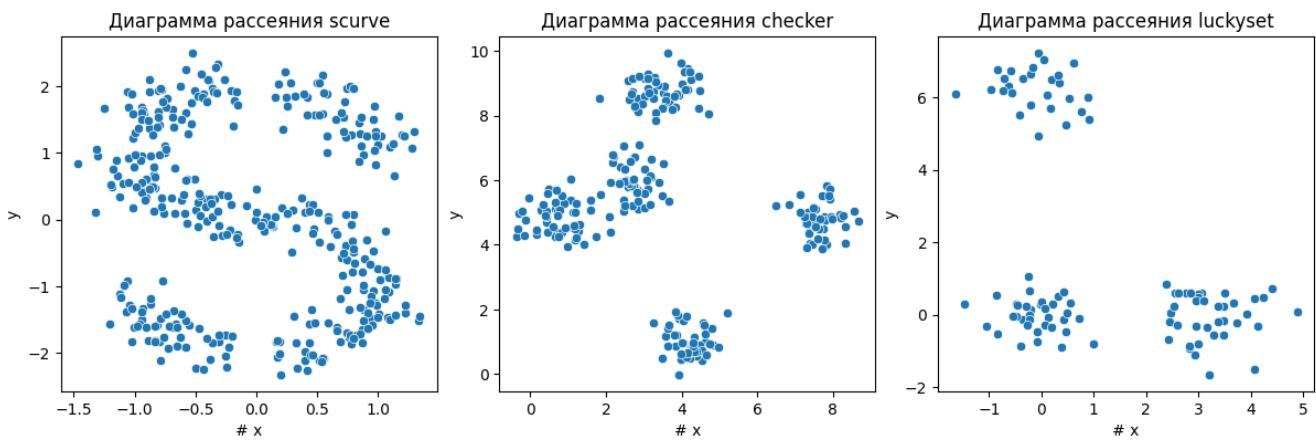


Рисунок 1.3.1 – Диаграмма рассеяния для scurve, checker и luckyset

Датасет scurve имеет форму буквы «S», остальные – форму капель.

1.4. Подготовьте наборы данных проводя стандартизацию или нормировку данных. Обоснуйте выбор операции.

Для проверки датасетов на нормальность использовался тест Шапиро-Уилка. Так как для каждого признака каждого датасета $p\text{value}$ оказалось меньше $\alpha = 0.05$, можно сделать вывод, что стандартизация в данном случае не подходит. Для каждого датасета вызван метод `describe`, исходя из которого сделан вывод, что можно применить неотрицательную нормализацию, ведь отношение минимального значения к максимальному по каждому признаку не оказалось слишком малым (выбросы

если и есть, то не имеют экстремально больших значений) и знак для каждого признака датасетов не имеет значения, что даёт право на использование MinMaxScaler. Реализация теста Шапиро-Уилка в виде функции представлена в [Листинге 1.4.1](#), результат работы – в [Листинге 1.4.2](#), реализация описательной статистики представлена в [Листинге 1.4.3](#), результат – в [Листинге 1.4.4](#), [Листинге 1.4.5](#), [Листинге 1.4.6](#) для датасетов scurve, checker и luckyset соответственно.

Листинг 1.4.1 – Реализация теста Шапиро-Уилка

```
from scipy.stats import shapiro

def shapiro_test(data):
    for column in data.columns:
        print('column:', column)
        print(f"\t\tscurve pvalue: {shapiro(data[column]).pvalue}")
    print('scurve:')
    shapiro_test(scurve_df)
    print('checker:')
    shapiro_test(checker_df)
    print('luckyset:')
    shapiro_test(luckyset_df)
```

Листинг 1.4.2 – Результат теста Шапиро-Уилка

```
scurve:
    column: # x
        pvalue: 1.6214001712436255e-12
    column: y
        pvalue: 2.0171576214922382e-11
checker:
    column: # x
        pvalue: 1.3900943729439064e-09
    column: y
        pvalue: 2.064556303693621e-09
luckyset:
    column: # x
        pvalue: 3.866394990836852e-07
    column: y
        pvalue: 3.5504099660244037e-12
```

Листинг 1.4.3 – Реализация описательной статистики для датасетов scurve, checker и luckyset

```
scurve_df.describe()  
checker_df.describe()  
luckyset_df.describe()
```

Листинг 1.4.4 – Описательная статистика для scurve

	# x	y
count	368.000000	368.000000
mean	-0.040508	0.022287
std	0.754779	1.356283
min	-1.465100	-2.330200
25%	-0.761400	-1.359450
50%	-0.196300	0.104100
75%	0.707850	1.279725
max	1.343800	2.500500

Листинг 1.4.5 – Описательная статистика для checker

	# x	y
count	250.000000	250.000000
mean	3.646908	5.183671
std	2.286382	2.522758
min	-0.367000	-0.024100
25%	2.408200	4.282800
50%	3.322550	5.123200
75%	4.439800	6.553200
max	8.682800	9.940200

Листинг 1.4.6 – Описательная статистика для luckyset

	# x	y
count	100.000000	100.000000
mean	1.165321	1.658658
std	1.724204	2.843817
min	-1.650700	-1.662500
25%	-0.230475	-0.198625
50%	0.431050	0.304450
75%	2.931975	5.286075
max	4.885700	7.233300

Реализация нормализации произведена с помощью MinMaxScaler из sklearn.preprocessing (см. [Листинге 1.4.7](#)).

Листинг 1.4.7 – Реализация нормализации для датасетов scurve, checker и luckyset

```
from sklearn.preprocessing import MinMaxScaler
```

```
scurve_np = MinMaxScaler().fit_transform(scurve_df)
checker_np = MinMaxScaler().fit_transform(checker_df)
luckyset_np = MinMaxScaler().fit_transform(luckyset_df)
```

2. K-Means:

2.1. Проведите исследование оптимального количества кластеров методом локтя. Сделайте выводы, о наиболее подходящем количестве кластеров

Метод локтя заключается в запуске методов кластеризации с последовательным увеличением задаваемого числа кластеров и нахождения такого максимально-го числа кластеров, при котором всё ещё наблюдаются заметные перегибы функции инерции, зависящей от числа кластеров. Также стоит отметить, что для большей точности для каждого датасета и кластера KMeans делает 10 прогонов. Реализация отрисовки графика зависимости инерции от числа кластеров представлена в [Листинге 2.1.1](#), результат отрисовки – на [Рисунке 2.1.1](#).

Листинг 2.1.1 – Реализация отрисовки графика зависимости инерции от числа кластеров

```
from sklearn.cluster import KMeans
import numpy as np

def elbow_method(data, max_n_clusters):
    inertia_list = []
    for i in range(1, max_n_clusters):
        k_means = KMeans(n_clusters=i, n_init=10)
        k_means.fit(data)
        inertia_list.append(k_means.inertia_)
    return inertia_list

data_nps = [scurve_np, checker_np, luckyset_np]
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(12, 4))
for i, data_np in enumerate(data_nps):
    inertia_list = elbow_method(data_np, 11)
    sns.lineplot(x=list(np.arange(1, 11)), y=inertia_list,
                 ax=axes[i]).set_title(
        'Зависимость инерции от количества \nкластеров для ' +
        titles[i])
    axes[i].set_ylabel('Значение инерции')
    axes[i].set_xlabel('Количество кластеров')
    axes[i].set_xticks(list(np.arange(1, 11)))
    axes[i].grid()
plt.tight_layout(pad=1.5)
plt.show()
```

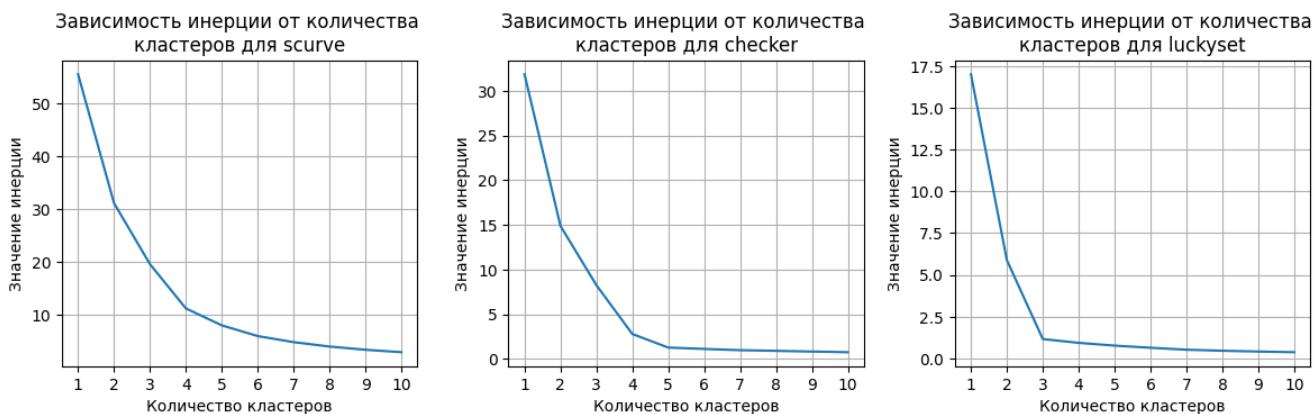


Рисунок 2.1.1 – Графики зависимости инерции от числа кластеров для scurve, checker и luckyset

Исходя из графиков можно сделать вывод, что для scurve и checker опти-

мальное число кластеров равно 5, а для luckyset равно 3.

2.2. Проведите исследование оптимального количества кластеров методом силуэта. Сделайте выводы, о наиболее подходящем количестве кластеров.

Для расчёта значения коэффициента силуэта необходимо импортировать соответствующую метрику из `sklearn.metrics`. Реализация практически аналогична предыдущему пункту, за исключением того, что используется коэффициент силуэта вместо инерции. Для нахождения оптимального числа кластеров нужно найти максимум на графике, ведь чем больше коэффициент силуэта, тем более элементы одного кластера похожи на друг друга и не похожи на элементы из других кластеров. Реализация представлена в [Листинге 2.2.1](#), результат отрисовки – на [Рисунке 2.2.1](#).

Листинг 2.2.1 – Реализация отрисовки графика зависимости среднего значения коэффициента силуэта от числа кластеров

```
from sklearn.metrics import silhouette_score

def silhouette_method(data, max_n_clusters):
    silhouette_list = []
    for i in range(2, max_n_clusters):
        k_means = KMeans(n_clusters=i, n_init=10)
        labels = k_means.fit_predict(data_np)
        silhouette_list.append(silhouette_score(data_np, labels))
    return silhouette_list

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(12, 4))
for i, data_np in enumerate(data_nps):
    silhouette_list = silhouette_method(data_np, 16)
    sns.lineplot(x=list(np.arange(2, 16)), y=silhouette_list,
                 ax=axes[i]).set_title(
        'Зависимость коэффициента силуэта от количества кластеров для ' +
        titles[i])
    axes[i].set_ylabel('Среднее значение коэффициента силуэта')
    axes[i].set_xlabel('Количество кластеров')
    axes[i].set_xticks(list(np.arange(2, 16)))
    axes[i].grid()
plt.tight_layout(pad=1.5)
plt.show()
```



Рисунок 2.2.1 – Графики зависимости среднего значения коэффициента силуэта от числа кластеров для scurve, checker и luckyset

Исходя из графиков можно сделать вывод, что для scurve и checker оптимальное количество кластеров равно 5, а для luckyset – 3.

2.3. Проведите кластеризацию алгоритмом K-means, с выбранным оптимальным количеством кластеров.

Для выполнения данного задания необходимо воспользоваться функционалом из `sklearn.cluster`. Реализация представлена в [Листинге 2.3.1](#), демонстрация результатов кластеризации будет представлена на диаграммах рассеяния в следующем пункте.

Листинг 2.3.1 – Реализация кластеризации scurve, checker и luckyset

```
k_means_scurve = KMeans(n_clusters=5, n_init=10)
labels_scurve = k_means_scurve.fit_predict(scurve_np)
k_means_checker = KMeans(n_clusters=5, n_init=10)
labels_checker = k_means_checker.fit_predict(checker_np)
k_means_luckyset = KMeans(n_clusters=3, n_init=10)
labels_luckyset = k_means_luckyset.fit_predict(luckyset_np)
```

2.4. Постройте диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров.

Для более удобной реализации отрисовки кластеров на диаграмме рассеяния каждый датасет преобразован из numpy-массива в pandas DataFrame. Для этого реализована вспомогательная функция `get_data_dict`, которая преобразует данные

в словарь для более удобного преобразования данных из одного формата в другой. Реализация представлена в [Листинге 2.4.1](#).

Листинг 2.4.1 – Реализация преобразования данных из формата пимпру-массив в pandas DataFrame

```
def get_data_dict(data, labels):
    d = {'# x': data[:, 0],
          'y': data[:, 1],
          'target': labels}
    return d

scurve_clustered_df = pd.DataFrame(data=get_data_dict(scurve_np,
                                                       labels_scurve))
checker_clustered_df = pd.DataFrame(data=get_data_dict(checker_np,
                                                       labels_checker))
luckyset_clustered_df = pd.DataFrame(data=get_data_dict(luckyset_np,
                                                       labels_luckyset))
```

Для отрисовки диаграмм рассеяния реализована функция plot_scatter, которая принимает на вход список датасетов и названия графиков, которые нужно отобразить на холсте. Реализация и вызов функции представлены в [Листинге 2.4.2](#), результат выполнения – на [Листинге 2.4.1](#).

Листинг 2.4.2 – Реализация отрисовки диаграмм рассеяния с выделением разным цветом разных кластеров

```
def plot_scatter(data_clustered, titles):
    n = len(data_clustered)
    fig, axes = plt.subplots(nrows=1, ncols=len(data_clustered),
                           figsize=(n*5 , 5))
    for i in range(len(data_clustered)):
        sns.scatterplot(data=data_clustered[i], x='# x', y='y',
                        hue='target', palette='tab10', ax=axes[i]).set_title(
                        titles[i])
    plt.show()

data_clustered = [scurve_clustered_df, checker_clustered_df,
                  luckyset_clustered_df]
plot_scatter(data_clustered, ['Кластеризация ' + title for title in titles])
```

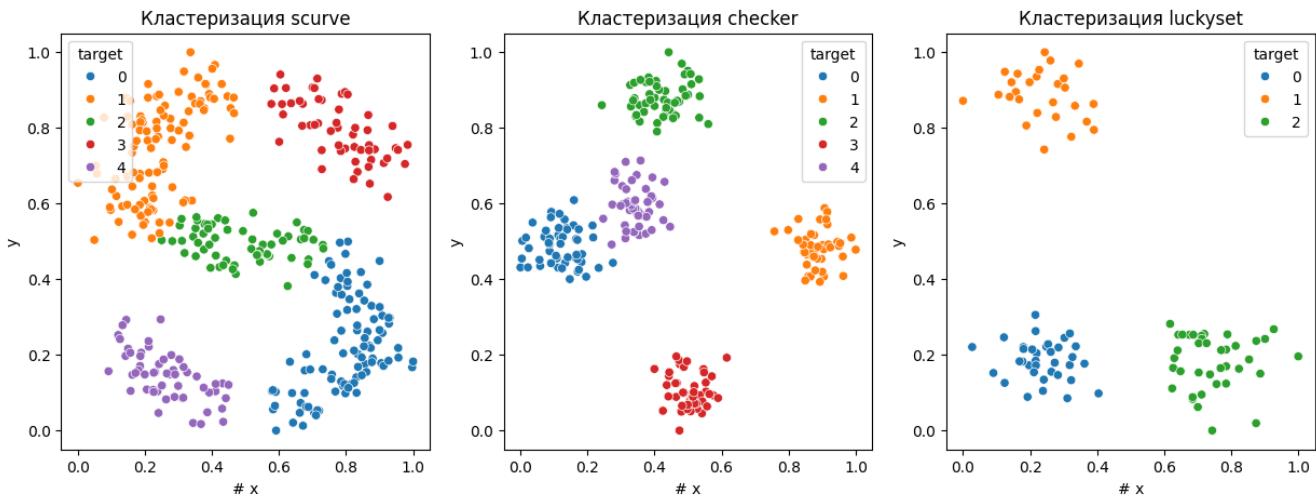


Рисунок 2.4.1 – Результат отрисовки диаграмм рассеяния с выделением разным цветом разных кластеров

2.5. Постройте диаграмму Вороного для результатов кластеризации. На диаграмме отметьте центроиды полученных кластеров.

Для реализации данного задания необходимо импортировать из `scipy.spatial.Voronoi` и `voronoi_plot_2d` для создания необходимых компонент диаграммы и создания фигуры соответственно. Далее необходимо получить центроиды каждого кластера для каждого датасета, создать холст, а затем для каждой группы центроидов нужно добавить фиктивные точки на график для корректного отображения раскрашенных областей, создать окружение диаграммы, создать фигуру диаграммы Вороного, закрасить регионы разными цветами, нанести на диаграмму кластеры и точки и, наконец, добавить оформление диаграммы. Реализация представлена в [Листинге 2.5.1](#), визуализация диаграммы – на [Рисунке 2.5.1](#).

Листинг 2.5.1 – Реализация отрисовки диаграммы Вороного для датасетов scurve, checker и luckyset

```
from scipy.spatial import Voronoi, voronoi_plot_2d

centroids_list = [k_means_scurve.cluster_centers_,
                  k_means_checker.cluster_centers_,
                  k_means_luckyset.cluster_centers_]
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
legends = ['upper left', 'upper right', 'upper right']
for i, centroids in enumerate(centroids_list):
    points = centroids
    max_x, max_y = data_nps[i][:, 0].max(), data_nps[i][:, 1].max()
    min_x, min_y = data_nps[i][:, 0].min(), data_nps[i][:, 1].min()
    points = np.vstack((points, np.array([max_x+10, max_y+10])))
    points = np.vstack((points, np.array([max_x+10, min_y-10])))
    points = np.vstack((points, np.array([min_x-10, max_y+10])))
    points = np.vstack((points, np.array([min_x-10, min_y-10])))

    vor = Voronoi(points)
    voronoi_plot_2d(vor, show_vertices=False, point_size=0, ax=axes[i])
    for region in vor.regions:
        if not -1 in region:
            polygon = [vor.vertices[i] for i in region]
            axes[i].fill(*zip(*polygon))
    axes[i].scatter(data_nps[i][:, 0], data_nps[i][:, 1],
                    s=5, c='black', label='Объекты')
    axes[i].scatter(points[:, 0], points[:, 1], s=200, marker='x',
                    c='white', linewidth=3, label='Центроиды')
    axes[i].set_xticks(list(np.arange(0, 1.1, 0.1)))
    axes[i].set_yticks(list(np.arange(0, 1.1, 0.1)))
    axes[i].set_xlim([min_x-0.1, max_x+0.1])
    axes[i].set_ylim([min_y-0.1, max_y+0.1])
    axes[i].set_title('Диаграмма Вороновского \nдля ' + titles[i])
    axes[i].set_xlabel('# x')
    axes[i].set_ylabel('y')
    axes[i].legend(loc=legends[i])
plt.tight_layout(pad=1.5)
plt.show()
```

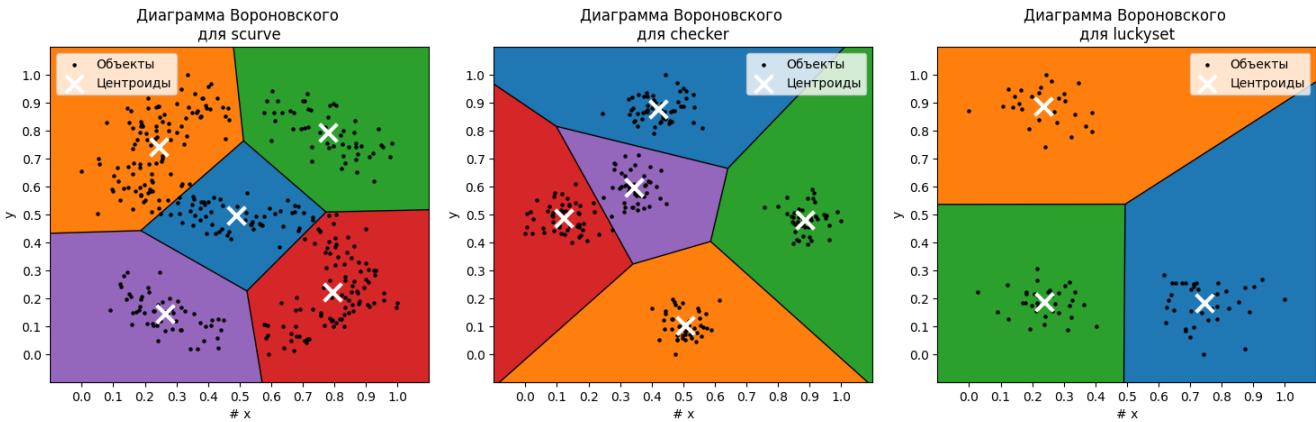


Рисунок 2.5.1 – Диаграмма Вороного для датасетов scurve, checker и luckyset

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.

Для каждого признака каждого датасета построена диаграмма “box-plot”. Для этого создан холст, на котором должно быть расположено 6 диаграмм, ведь у всех датасетов по два признака, а всего исследуется 3 датасета. Для каждой диаграммы добавлена сетка и шкала деления в диапазоне от 0 до 1 с шагом 0.1. Реализация представлена в [Листинге 2.6.1](#), визуализация диаграмм – на [Рисунке 2.6.1](#).

Листинг 2.6.1 – Реализация отрисовки диаграмм “box-plot” для каждого признака каждого датасета

```
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
features = scurve_df.columns
for i in range(len(features)):
    for j in range(len(data_clustered)):
        sns.boxplot(data=data_clustered[j], x=features[i], hue='target',
                    palette='tab10', ax=axes[i][j]).set_title(
                    titles[j])
        axes[i][j].grid()
        axes[i][j].set_xticks(list(np.arange(0, 1.1, 0.1)))
```

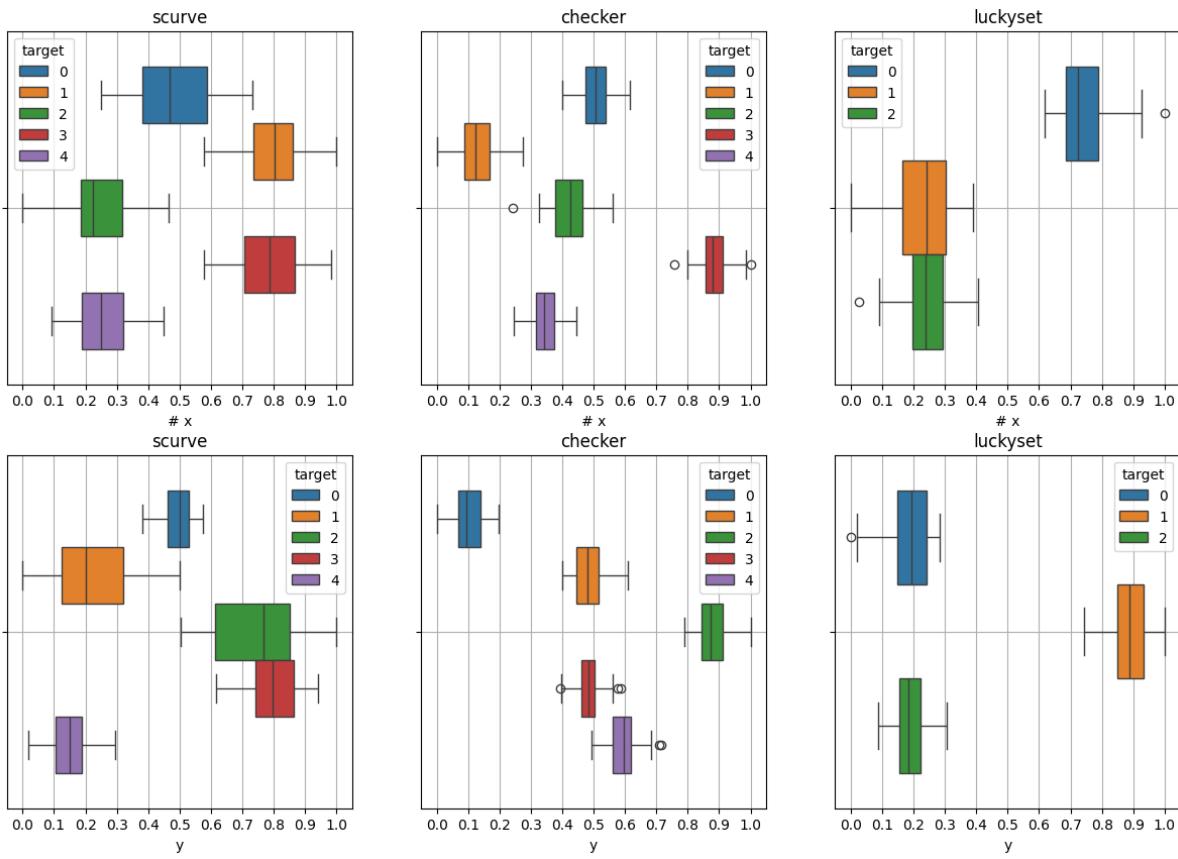


Рисунок 2.6.1 – Отрисовка диаграмм “box-plot” для каждого признака датасетов scurve, checker и luckyset

Исходя из полученных данных можно сделать следующие выводы:

- в scurve отсутствуют выбросы;
- в checker выбросы имеют зелёный и красный кластеры по признаку "# x";
- в luckyset выбросы имеют синий и зелёный кластеры по признаку "# x";
- в checker выбросы имеют красный и фиолетовый кластеры по признаку "y";
- в luckyset выбросы имеет синий кластер по признаку "y".

Кластеризацию можно назвать успешной для датасетов checker и luckyset. С датасетом scurve алгоритм K-means справился плохо, так как он имеет вытянутую форму, но K-means не отслеживает такую информацию ввиду того, что алгоритм объединяет объекты в кластеры в некоторой окрестности центроид. Алгоритм K-means может хорошо кластеризовать датасеты в форме капель, что подтвердилось на практике.

2.7. Рассчитайте для каждого кластера кол-во точек, среднее, СКО, минимум и максимум. Сопоставьте результаты с построеннымми графиками.

Для выполнения данного пункта была реализована функция compute_cluster_data, которая принимает на вход датасет, а возвращает описательную статистику данного датасета, за исключением первой, второй и третьей квантилий, что сделано в соответствии с условием. За основу взят результат выполнения метода describe для сгруппированного по значению целевой переменной датасета, из которого удаляется соответствующие признаки с помощью метода drop. Реализация данной функции и её вызовов представлены в [Листинге 2.7.1](#), результат выполнения – в [Листинге 2.7.2](#), [Листинге 2.7.3](#), [Листинге 2.7.4](#) для датасетов scurve, checker и luckyset соответственно.

Листинг 2.7.1 – Реализация отрисовки диаграмм “box-plot” для каждого признака каждого датасета

```
def compute_cluster_data(X):
    outer_features, inner_features = ['# x', 'y'], ['25%', '50%', '75%']
    columns_to_drop = []
    for outer_feature in outer_features:
        for inner_feature in inner_features:
            columns_to_drop.append((outer_feature, inner_feature))
    return X.groupby('target').describe().drop(columns_to_drop, axis=1)

compute_cluster_data(scurve_clustered_df)
compute_cluster_data(checker_clustered_df)
compute_cluster_data(luckyset_clustered_df)
```

Листинг 2.7.2 – Результат выполнения функции compute_cluster_data для датасета scurve

	#x				
target	count	mean	std	min	max
0	57.0	0.496013	0.126882	0.281569	0.732066
1	107.0	0.244789	0.100266	0.000000	0.465948
2	55.0	0.264489	0.095704	0.091424	0.449393
3	95.0	0.794597	0.098340	0.578910	1.000000
4	54.0	0.780372	0.109621	0.577094	0.983695
	y				
target	count	mean	std	min	max
0	57.0	0.493253	0.043254	0.381767	0.575486
1	107.0	0.738704	0.132306	0.503012	1.000000
2	55.0	0.145307	0.065403	0.017182	0.293808
3	95.0	0.223218	0.127172	0.000000	0.499472
4	54.0	0.793644	0.078654	0.617405	0.941044

Листинг 2.7.3 – Результат выполнения функции compute_cluster_data для датасета checker

	#x				
target	count	mean	std	min	max
0	46.0	0.504848	0.045788	0.399567	0.615417
1	50.0	0.343477	0.042936	0.246900	0.446452
2	46.0	0.886668	0.046826	0.757464	1.000000
3	55.0	0.421779	0.064685	0.242856	0.560222
4	53.0	0.122688	0.061366	0.000000	0.275951
	y				
target	count	mean	std	min	max
0	46.0	0.103930	0.046222	0.000000	0.195799
1	50.0	0.594685	0.053326	0.491675	0.713658
2	46.0	0.481167	0.047662	0.392983	0.588039
3	55.0	0.877348	0.043846	0.790301	1.000000
4	53.0	0.485998	0.049018	0.400259	0.608984

Листинг 2.7.4 – Результат выполнения функции `compute_cluster_data` для датасета `luckyset`

	#x				
target	count	mean	std	min	max
0	35.0	0.237620	0.079065	0.027355	0.404305
1	27.0	0.236701	0.094429	0.000000	0.390490
2	38.0	0.746698	0.094273	0.617679	1.000000

	y				
target	count	mean	std	min	max
0	35.0	0.185355	0.051775	0.085344	0.305672
1	27.0	0.886214	0.064409	0.742665	1.000000
2	38.0	0.182074	0.071532	0.000000	0.281897

Для сравнения количества рассчитанных точек в каждом кластере и координат каждой центроиды достаточно обратить внимание на диаграмму рассеяния и диаграмму Вороного соответственно, а для сравнения рассчитанного минимума и максимума каждого кластера – на диаграмму рассеяния и ящик с усами соответственно. Рассчитанные значения совпали с их графическими интерпретациями.

3. DBSCAN

3.1. Подберите параметры алгоритма DBSCAN, которые на ваш взгляд дают наилучшие результаты. Опишите процесс (почему и как изменяли параметры) подбора параметров.

Для подбора наилучших гиперпараметров для DBSCAN реализован перебор следующим образом:

- гиперпараметр окрестности поиска принимает значения от 0.01 до 1.43 с шагом 0.01;
- гиперпараметр минимального количества соседей для объекта в датасете принимает значения от двух до количества точек в датасете;
- происходит перебор всевозможных комбинаций гиперпараметров;
- для каждой комбинации гиперпараметров вычисляется оценка качества кластеризации и в словарь по данному ключу записывается данный экземпляр DBSCAN;

- если для текущего датасета данная оценка качества кластеризации была получена ранее, то в словарь по данному ключу записывается новый экземпляр DBSCAN;
- при оценке качества кластеризации выбросы не учитываются, так как в данной реализации они имеют метку одного кластера, но в то же время могут не иметь явной структуры;
- при оценке качества кластеризации количество выбросов не должно превышать того количества, которое определила модифицированная zscore-оценка;
- нельзя рассматривать только те объекты, которые модифицированная zscore-оценка не посчитала выбросами, так как DBSCAN и данная оценка могут по-разному их идентифицировать, поэтому проверяется только то, чтобы количество выбросов в DBSCAN не превышало количества выбросов, которое определила модифицированная zscore-оценка;
- для идентификации выбросов использовалась модифицированная zscore-оценка, которая рассчитывается по следующей формуле:

$$zmod = \frac{0.6745(X_i - \tilde{X})}{MAD} \quad (1)$$

$$MAD = \text{median}(|X_i - \tilde{X}|) \quad (2)$$

- если для X_i значение $|zmod| < 3.5$, то данный объект не является выбросом;
- по окончанию итерации для данного датасета выводится словарь, в котором ключами являются уникальные коэффициенты силуэта, отсортированные по убыванию, а значениями – первые лучшие оценщики.

Для выполнения данного задания, а также заданий в п.5 реализованы следующие функции:

- compute_zmod – данная функция возвращает модифицированную z-score оценку для полученного на вход датасета.
- assign_score – данная функция предназначена для расчёта внутренней

оценки кластеризации в зависимости от выбранной метрики: коэффициент силуэта, коэффициент Калинского-Харабаша, коэффициент Дэвиса-Болдуина. После расчёта оценки кластеризации она добавляется в словарь в качестве ключа, а значением является экземпляр некоторого кластеризатора. Функция принимает на вход датасет, метки объектов, словарь коэффициентов, метрику и экземпляр кластеризатора, а возвращает обновлённый словарь.

- `get_sorted_scores_dict` – данная функция, в зависимости от используемой метрики оценивания кластеризации, сортирует словарь по ключам либо в порядке возрастания, либо в порядке убывания. Существуют следующие закономерности:

- чем больше коэффициент силуэта, тем лучше кластеризация;
- чем больше коэффициент Калинского-Харабаша, тем лучше кластеризация;
- чем меньше коэффициент Дэвиса-Болдуина, тем лучше кластеризация.

Функция получает на вход словарь и метрику, возвращает отсортированный словарь.

- `get_dbscan_scores_dict` – данная функция перебирает гиперпараметры алгоритма DBSCAN, а также запускает вышеописанные функции. Принимает на вход датасет, список радиусов окрестностей, список минимального числа соседей, метрику, а возвращает отсортированный словарь вида {значение коэффициента: экземпляр DBSCAN}.

- `print_scores_estimators` – данная функция предназначена для поэлементного вывода элементов словаря, а также для вывода данных о кластерах: число кластеров и элементов в каждом. Получает на вход датасет, словарь, название кластеризатора, дополнительную информацию (используется для иерархической кластеризации в п.5), число выводимых ключей словаря. Функция ничего не возвращает.

Реализация вышеописанных функций представлена в [Листинге 3.1.1](#), [Листинге 3.1.2](#), [Листинге 3.1.3](#), [Листинге 3.1.4](#), [Листинге 3.1.5](#) соответственно, реализация запуска функции для подбора гиперпараметров алгоритма DBSCAN с использованием коэффициента силуэта – в [Листинге 3.1.6](#), результат – в [Листин-](#)

ге 3.1.7, с использованием коэффициента Дэвиса-Болдуина – в [Листинге 3.1.8](#), результат – в [Листинге 3.1.9](#), с использованием коэффициента Калинского-Харабаша – в [Листинге 3.1.10](#), результат – в [Листинге 3.1.11](#).

Листинг 3.1.1 – Реализация модифицированный z-score оценки

```
def compute_zmod(X):
    MAD = np.median(np.abs(X - np.median(X, axis=0)), axis=0)
    zmod = (0.6745 * (X - np.median(X, axis=0))) / (MAD + 1e-5)
    return zmod
```

Листинг 3.1.2 – Реализация вариативности в расчёте оценок

```
def assign_score(data, labels, scores_dict, metric, estimator):
    mask = (labels != -1)
    match metric:
        case 'ss':
            try:
                score = silhouette_score(data[mask], labels[mask])
            except ValueError:
                score = -1
        else:
            scores_dict[score] = estimator
    case 'dbs':
        try:
            score = davies_bouldin_score(data[mask], labels[mask])
        except ValueError:
            score = -1
        else:
            scores_dict[score] = estimator
    case 'chs':
        try:
            score = calinski_harabasz_score(data[mask], labels[mask])
        except ValueError:
            score = -1
        else:
            scores_dict[score] = estimator
    return scores_dict
```

Листинг 3.1.3 – Реализация сортировки словаря в зависимости от используемой метрики оценивания качества кластеризации

```
def get_sorted_scores_dict(scores_dict, metric):
    match metric:
        case 'ss':
            scores_dict = {key: value for key, value in sorted(
                scores_dict.items(), reverse=True)}
        case 'dbs':
            scores_dict = {key: value for key, value in sorted(
                scores_dict.items())}
        case 'chs':
            scores_dict = {key: value for key, value in sorted(
                scores_dict.items(), reverse=True)}
    return scores_dict
```

Листинг 3.1.4 – Реализация перебора гиперпараметров для DBSCAN

```
from sklearn.cluster import DBSCAN
```

```
def get_dbSCAN_scores_dict(data, eps_s, mins_samples, metric):
    max_score = -1
    scores_dict = {}
    for eps in tqdm(eps_s):
        for min_samples in mins_samples:
            dbSCAN = DBSCAN(eps=eps, min_samples=min_samples)
            zmod = compute_zmod(data)
            zmod_mask = (np.abs(zmod) < 3.5).all(axis=1)
            n_outliers = len(data) - len(data[zmod_mask])
            labels = dbSCAN.fit_predict(data)
            if len(labels[labels == -1]) <= n_outliers:
                scores_dict = assign_score(data, labels,
                                           scores_dict, metric, dbSCAN)
    scores_dict = get_sorted_scores_dict(scores_dict, metric)
    return scores_dict
```

Листинг 3.1.5 – Реализация вывода оценок кластеризации и информации о кластерах

```
def print_scores_estimators(data, scores_dict, name, extra_info=' ', n_keys=None):
    if n_keys is None:
        n_keys = len(scores_dict)
    else:
        n_keys = min(n_keys, len(scores_dict))
    sliced_keys = list(scores_dict)[n_keys:]
    if extra_info != ' ':
        extra_info = f'{extra_info}, n_clusters = '
    print(f'{name}:')
    for key in sliced_keys:
        print(f'\t score = {key}: \n\t {extra_info}{scores_dict[key]}')
        estimator = scores_dict[key]
        if extra_info == ' ':
            labels = estimator.fit_predict(data)
            max_label = max(labels)
            min_label = min(labels)
        else:
            Z = linkage(wineset_normal, extra_info.split(', ')[0])
            labels = fcluster(Z, estimator, 'maxclust')
            max_label = estimator
            min_label = 1
        if min_label in [-1, 0]:
            n = max_label + 1
        else:
            n = max_label
        print(f'\t\t number of clusters = {n}')
        for i in range(min_label, max_label+1):
            power = len(labels[labels == i])
            print(f'\t\t power of cluster {i} = {power}' )
```

Листинг 3.1.6 – Реализация вызова функции для подбора наилучших гиперпараметров DBSCAN при использовании коэффициента силуэтов и вывода словаря очков

```
n_keys=[3, 1, 1]
for i, title in enumerate(titles):
    scores_dict = get_dbSCAN_scores_dict(data_nps[i],
                                          np.arange(0.01, 1.43, 0.01),
                                          np.arange(2, len(data_nps[i])), 'ss')
    print_scores_estimators(data_nps[i], scores_dict, titles[i], n_keys=n_keys[i])
```

Листинг 3.1.7 – Словари коэффициентов силуэта для каждого датасета при использовании коэффициента силуэта

scurve:

```
score = 0.27143355102852385:  
    DBSCAN(eps=0.13, min_samples=15)  
        number of clusters = 2  
        power of cluster 0 = 313  
        power of cluster 1 = 55  
score = 0.2660597367718648:  
    DBSCAN(eps=0.14, min_samples=18)  
        number of clusters = 2  
        power of cluster 0 = 315  
        power of cluster 1 = 53  
score = 0.23472838993201214:  
    DBSCAN(eps=0.14, min_samples=28)  
        number of clusters = 3  
        power of cluster 0 = 259  
        power of cluster 1 = 55  
        power of cluster 2 = 54
```

checker:

```
score = 0.7095475215193976:  
    DBSCAN(eps=0.09, min_samples=22)  
        number of clusters = 5  
        power of cluster -1 = 1  
        power of cluster 0 = 46  
        power of cluster 1 = 46  
        power of cluster 2 = 53  
        power of cluster 3 = 50  
        power of cluster 4 = 54
```

luckyset:

```
score = 0.7925841748145548:  
    DBSCAN(eps=0.08, min_samples=10)  
        number of clusters = 3  
        power of cluster -1 = 23  
        power of cluster 0 = 32  
        power of cluster 1 = 30  
        power of cluster 2 = 15
```

Листинг 3.1.8 – Реализация вызова функции для подбора наилучших гиперпараметров DBSCAN при использовании коэффициента Дэвиса-Болдуина и вывода словаря очков

```
n_keys=[3, 1, 1]
for i, title in enumerate(titles):
    scores_dict = get_dbSCAN_scores_dict(data_nps[i],
                                          np.arange(0.01, 1.43, 0.01),
                                          np.arange(2, len(data_nps[i])), 'dbs')
    print_scores_estimators(data_nps[i], scores_dict, titles[i], n_keys=n_keys[i])
```

Листинг 3.1.9 – Словари коэффициентов силуэта для каждого датасета при использовании коэффициента Дэвиса-Болдуина

scurve:

```
score = 0.9106826540392716:  
    DBSCAN(eps=0.14, min_samples=18)  
        number of clusters = 2  
        power of cluster 0 = 315  
        power of cluster 1 = 53  
score = 0.9167746866262408:  
    DBSCAN(eps=0.13, min_samples=15)  
        number of clusters = 2  
        power of cluster 0 = 313  
        power of cluster 1 = 55  
score = 1.0866426193817549:  
    DBSCAN(eps=0.14, min_samples=20)  
        number of clusters = 3  
        power of cluster 0 = 263  
        power of cluster 1 = 52  
        power of cluster 2 = 53
```

checker:

```
score = 0.38778714679218085:  
    DBSCAN(eps=0.09, min_samples=22)  
        number of clusters = 5  
        power of cluster -1 = 1  
        power of cluster 0 = 46  
        power of cluster 1 = 46  
        power of cluster 2 = 53  
        power of cluster 3 = 50  
        power of cluster 4 = 54
```

luckyset:

```
score = 0.2772848104193372:  
    DBSCAN(eps=0.08, min_samples=10)  
        number of clusters = 3  
        power of cluster -1 = 23  
        power of cluster 0 = 32  
        power of cluster 1 = 30  
        power of cluster 2 = 15
```

Листинг 3.1.10 – Реализация вызова функции для подбора наилучших гиперпараметров DBSCAN при использовании коэффициента Калинского-Харабаша и вывода словаря очков

```
n_keys=[3, 1, 1]
for i, title in enumerate(titles):
    scores_dict = get_dbSCAN_scores_dict(data_nps[i],
                                          np.arange(0.01, 1.43, 0.01),
                                          np.arange(2, len(data_nps[i])), 'chs')
    print_scores_estimators(data_nps[i], scores_dict, titles[i], n_keys=n_keys[i])
```

Листинг 3.1.11 – Словари коэффициентов силуэта для каждого датасета при использовании коэффициента Калинского-Харабаша

scurve:

```
score = 94.3215849891565:  
    DBSCAN(eps=0.13, min_samples=15)  
        number of clusters = 2  
        power of cluster 0 = 313  
        power of cluster 1 = 55  
score = 93.6356740518537:  
    DBSCAN(eps=0.14, min_samples=28)  
        number of clusters = 3  
        power of cluster 0 = 259  
        power of cluster 1 = 55  
        power of cluster 2 = 54  
score = 92.93422065768722:  
    DBSCAN(eps=0.12, min_samples=20)  
        number of clusters = 3  
        power of cluster 0 = 260  
        power of cluster 1 = 53  
        power of cluster 2 = 55
```

checker:

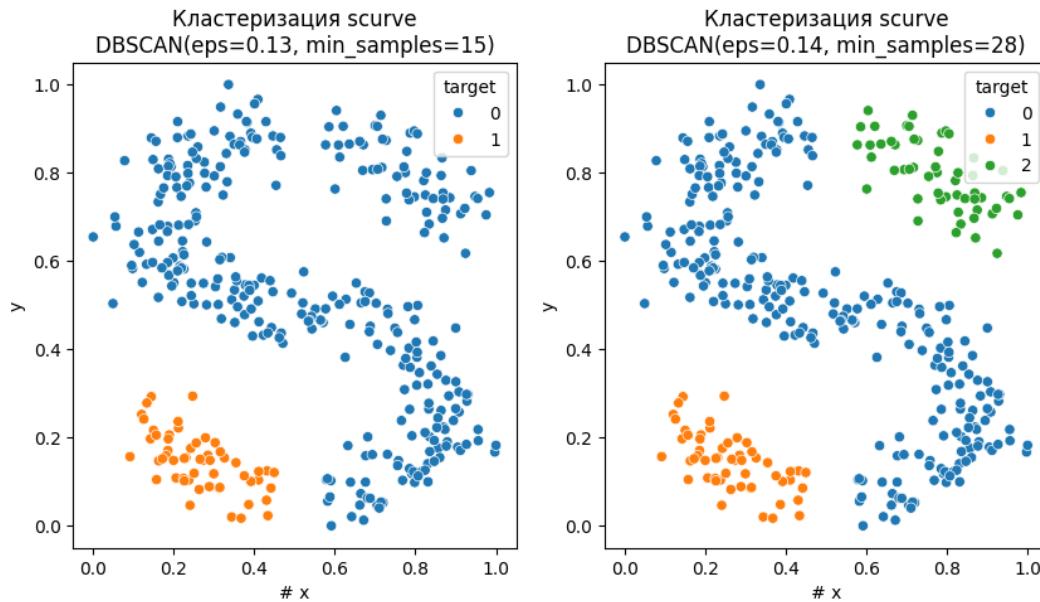
```
score = 1481.941110324499:  
    DBSCAN(eps=0.11, min_samples=38)  
        number of clusters = 5  
        power of cluster -1 = 1  
        power of cluster 0 = 46  
        power of cluster 1 = 46  
        power of cluster 2 = 55  
        power of cluster 3 = 48  
        power of cluster 4 = 54
```

luckyset:

```
score = 827.8403912358822:  
    DBSCAN(eps=0.0699999999999999, min_samples=7)  
        number of clusters = 3  
        power of cluster -1 = 20  
        power of cluster 0 = 31  
        power of cluster 1 = 17  
        power of cluster 2 = 32
```

Внутренние оценки кластеризации имеют рекомендательный характер, то есть, если у некоторого алгоритма кластеризации самое лучшее значение некоторой метрики, то это не значит, что предлагаемый им вариант кластеризации самый лучший. К примеру, если взять наилучшие гиперпараметры, которые предлагает

взять коэффициент силуэтов для датасета scurve, то кластеризация не окажется высокого качества, но если взять третий из лучших результатов, то кластеризация окажется намного лучше (см. [Рисунок 3.1.1](#)). Также стоит отметить, что для данного датасета все метрики предсказали неверные наилучшие параметры для датасета scurve.



[Рисунок 3.1.1 – Диаграммы рассеяния для scurve при разных гиперпараметрах](#)

Реализация отрисовки диаграмм на [Рисунке 3.1.1](#) включает в себя создание экземпляров DBSCAN с двумя наилучшими гиперпараметрами и их отрисовки с помощью функции `plot_scatter` и представлена в [Листинге 3.1.12](#).

[Листинг 3.1.12 – Реализация отрисовки двух диаграмм рассеяния при разных гиперпараметрах для scurve](#)

```
dbscan_1 = DBSCAN(eps=0.13, min_samples=15)
dbscan_labels_scurve_1 = dbscan_1.fit_predict(scurve_np)
dbscan_2 = DBSCAN(eps=0.14, min_samples=28)
dbscan_labels_scurve_2 = dbscan_2.fit_predict(scurve_np)
scurve_clustered_df1 = pd.DataFrame(data=get_data_dict(scurve_np,
                                                       dbscan_labels_scurve_1))
scurve_clustered_df2 = pd.DataFrame(data=get_data_dict(scurve_np,
                                                       dbscan_labels_scurve_2))
data_clustered = [scurve_clustered_df1, scurve_clustered_df2]
dbscans = [dbscan_1, dbscan_2]
plot_scatter(data_clustered, ['Кластеризация scurve\n' + str(dbscan)
                             for dbscan in dbscans])
```

3.2. Постройте диаграмму рассеяния результатов кластеризации с выделением разным цветом разных кластеров.

Для выделения каждого кластера разным цветом необходимо запустить DBSCAN с гиперпараметрами, найденными в предыдущем пункте, получить метки, затем с помощью определённой ранее функцией get_data_dict преобразовать данные в словарь, преобразовать каждый датасет в pandas DataFrame и воспользоваться функцией plot_scatter для отрисовки диаграмм рассеяния для каждого датасета. Реализация данных шагов представлена в [Листинге 3.2.1](#), диаграммы рассеяния каждого датасета – на [Рисунке 3.2.1](#).

Листинг 3.2.1 – Реализация получения меток для объектов каждого датасета, преобразование в pandas DataFrame и отрисовка диаграмм рассеяния

```
dbscan = DBSCAN(eps=0.14, min_samples=28)
dbscan_labels_scurve = dbscan.fit_predict(scurve_np)
dbscan = DBSCAN(eps=0.09, min_samples=22)
dbscan_labels_checker = dbscan.fit_predict(checker_np)
dbscan = DBSCAN(eps=0.08, min_samples=10)
dbscan_labels_luckyset = dbscan.fit_predict(luckyset_np)

scurve_clustered_df = pd.DataFrame(data=get_data_dict(scurve_np,
                                                       dbscan_labels_scurve))
checker_clustered_df = pd.DataFrame(data=get_data_dict(checker_np,
                                                       dbscan_labels_checker))
luckyset_clustered_df = pd.DataFrame(data=get_data_dict(luckyset_np,
                                                       dbscan_labels_luckyset))
data_clustered = [scurve_clustered_df, checker_clustered_df,
                  luckyset_clustered_df]
plot_scatter(data_clustered, ['Кластеризация ' + title for title in titles])
```

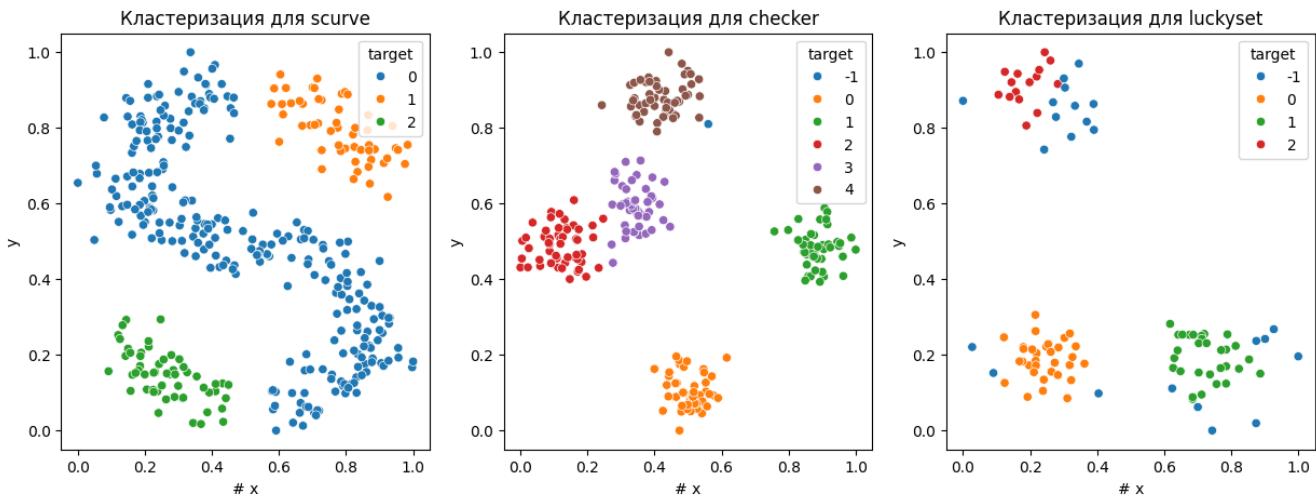


Рисунок 3.2.1 – Диаграмма рассеяния для scurve, checker и luckyset

3.3. Сделайте выводы об успешности кластеризации.

Исходя из полученных результатов, можно сделать следующие выводы:

- подбор гиперпараметров для DBSCAN может быть не так очевиден;
- DBSCAN лучше кластеризует данные вытянутой формы по сравнению с K-means;
- DBSCAN может идентифицировать выбросы;
- DBSCAN хорошо кластеризует данные в форме капель.

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).

Для выполнения иерархической кластеризации достаточно импортировать из `scipy.cluster.hierarchy` функционал для формирования дендрограмм, матриц связей и меток для каждого объекта. Затем с помощью функции `get_clustering_info` для каждого `linkage` каждого датасета нужно сформировать матрицу связей и

метки объектов, для формирования которых нужна полученная матрица связей и количество кластеров. Количество кластеров было взято из п.3. Реализация вышеописанных шагов представлена в [Листинге 4.1.1](#).

Листинг 4.1.1 – Реализация получения меток для объектов каждого датасета, преобразование в pandas DataFrame и отрисовка диаграмм рассеяния

```
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

linkages = [ 'ward', 'complete', 'average', 'single']

def get_clustering_info(data, n_clusters):
    labels_in_linkages = {}
    Z_in_linkages = {}
    for str_linkage in linkages:
        Z_in_linkages[str_linkage] = linkage(data, method=str_linkage)
        labels_in_linkages[str_linkage] = fcluster(Z_in_linkages[str_linkage],
                                                    n_clusters,
                                                    criterion='maxclust')
    return labels_in_linkages, Z_in_linkages

scurve_labels_in_linkages, Z_scurve_in_linkages = get_clustering_info(scurve_np, 3)
checker_labels_in_linkages, Z_checker_in_linkages = get_clustering_info(checker_np, 5)
luckyset_labels_in_linkages, Z_luckyset_in_linkages = get_clustering_info(luckyset_np, 3)
```

Для отрисовки дендрограмм каждого датасета создан холст, на которым будут отображаться все графики. При отрисовке каждой дендрограммы среди необязательных параметров указывается соответствующая ось, а также важный для визуализации кластеров параметр color_threshold. В матрице связей столбец по индексу 2 соответствует расстоянию между кластерами, а строка с индексом n соответствует номеру горизонтальной линии на графике дендрограммы, начиная снизу вверх. Значение с координатами [n, 2] в матрице связей устанавливается как значение для color_threshold, что позволяет увидеть на дендрограмме разделение объектов на кластеры в виде разных цветов. Реализация отрисовки дендрограмм датасетов scurve, checker и luckyset представлена в [Листинге 4.1.2](#), дендрограммы – на [Рисунке 4.1.1](#).

Листинг 4.1.2 – Реализация отрисовки дендрограмм датасетов scurve, checker и luckyset

```

datasets_labels_in_linkages = [scurve_labels_in_linkages,
                               checker_labels_in_linkages,
                               luckyset_labels_in_linkages]
Zs_in_linkages = [Z_scurve_in_linkages, Z_checker_in_linkages,
                  Z_luckyset_in_linkages]
n_clusters = [3, 5, 3]
fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(16, 12))
for i, dataset_labels_in_linkages in enumerate(datasets_labels_in_linkages):
    n = -n_clusters[i] + 1
    for j, str_linkage in enumerate(linkages):
        dendrogram(Zs_in_linkages[i][str_linkage], ax=axes[i][j],
                   color_threshold=Zs_in_linkages[i][str_linkage][n, 2],
                   no_labels=True)
        axes[i][j].set_ylabel('Расстояние между кластерами')
        axes[i][j].set_title(f"Дендрограмма {titles[i]}\nlinkage: {str_linkage}")
    plt.tight_layout(pad=1.5)
plt.show()

```

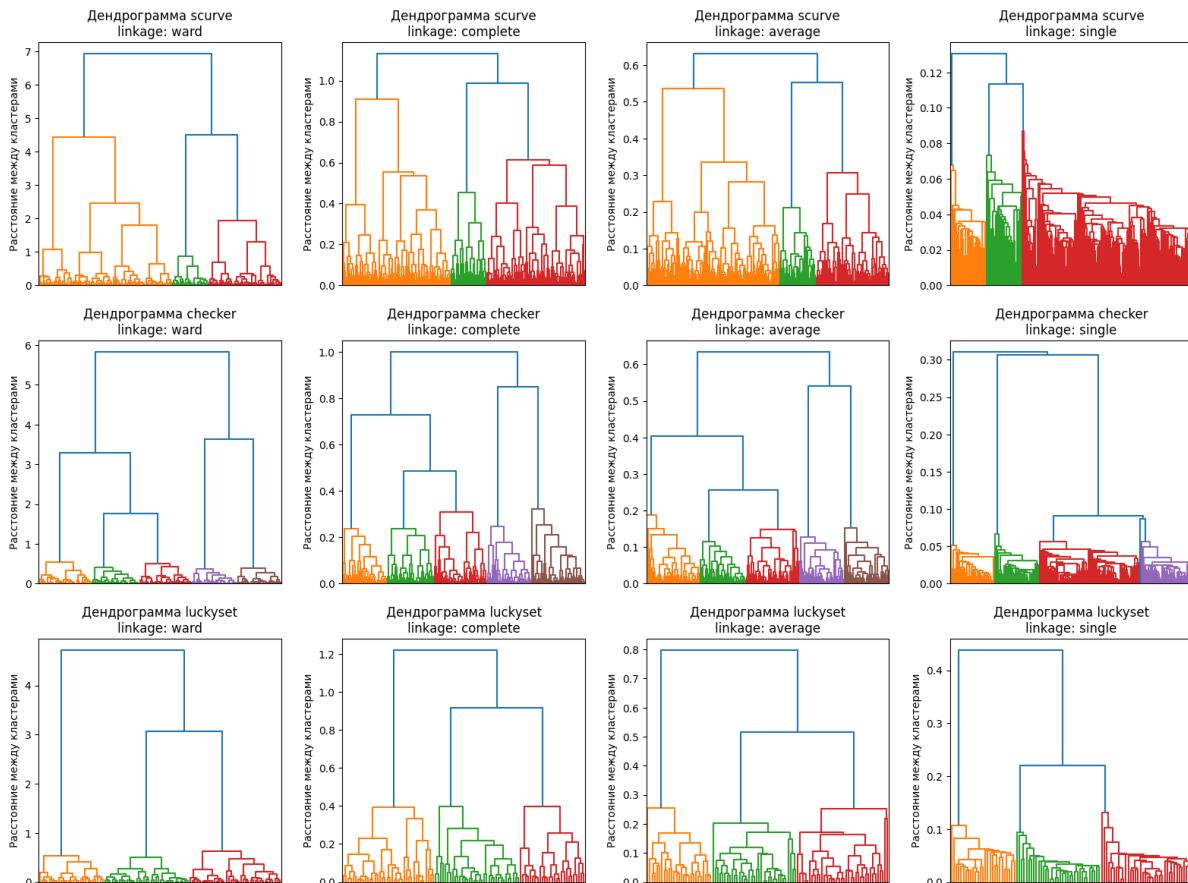


Рисунок 4.1.1 – Дендрограммы датасетов scurve, checker и luckyset

Исходя из полученных дендрограмм можно сделать вывод, что для датасетов checker и luckyset кластеризация выполнена довольно успешно, что нельзя сказать о кластеризации scurve: линия уровня деления на разные цвета должна быть ниже, что соответствует большему числу кластеров.

Для каждого типа linkage датасета scurve нужно вручную подбирать число кластеров. Для каждого типа linkage оптимальное число кластеров оказалось следующим:

- для «ward» – 7;
- для «complete» – 8;
- для «average» – 8;
- для «single» – 3.

Реализация отрисовки дендрограммы для scurve представлена в [Листинге 4.1.3](#), результат отрисовки – на [Рисунке 4.1.2](#).

Листинг 4.1.3 – Реализация отрисовки дендрограмм датасета scurve для разных linkage

```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(16, 4))
ns_clusters = [7, 8, 8, 3]
for i, n_clusters in enumerate(ns_clusters):
    n = -n_clusters + 1
    scurve_labels_in_linkages, Z_scurve_in_linkages = get_clustering_info(
        scurve_np, n_clusters)
    dendrogram(Z_scurve_in_linkages[linkages[i]], ax=axes[i],
               color_threshold=Z_scurve_in_linkages[linkages[i]][n, 2],
               no_labels=True)

    axes[i].set_ylabel('Расстояние между кластерами')
    axes[i].set_title(f"Дендрограмма scurve \nlinkage: {linkages[i]}")
plt.tight_layout(pad=1.5)
plt.show()
```

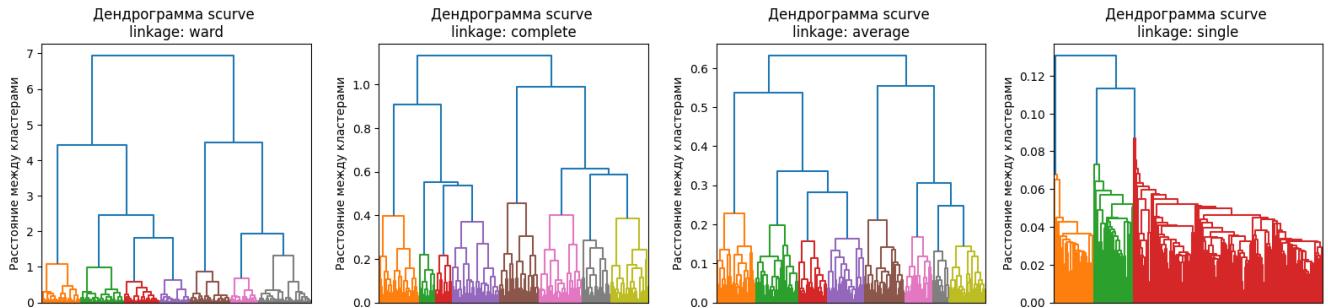


Рисунок 4.1.2 – Дендрограммы датасетов scurve, checker и luckyset после изменения числа кластеров для scurve

На [Рисунке 4.1.2](#) можно заметить, что для датасета scurve высота перепадов расстояния между кластерами уменьшилась, что говорит о более лучшем качестве кластеризации.

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

Для выполнения данного задания была реализована функция `plot_hierarchical_scatter`, которая принимает на вход датасет, его название и список числа кластеров, которые являются наиболее оптимальными для каждого типа `linkage`. Реализация данной функции, а также её вызовы для каждого датасета представлены в [Листинге 4.2.1](#), диаграммы рассеяния представлены на [Рисунках 4.2.1](#), [Рисунках 4.2.2](#), [Рисунках 4.2.3](#) для датасетов scurve, checker и luckyset соответственно.

Листинг 4.2.1 – Реализация отрисовки диаграмм рассеяния для датасетов scurve, checker и luckyset

```

def plot_hierarchical_scatter(data, title, ns_clusters):
    data_clustered = []
    for i, n_cluster in enumerate(ns_clusters):
        labels_in_linkages, Z_in_linkages = get_clustering_info(
            data, n_cluster)
        clustered_df = pd.DataFrame(data=get_data_dict(data),
                                      labels_in_linkages[linkages[i]]))
        data_clustered.append(clustered_df)
    plot_scatter(data_clustered, [f'Кластеризация {title}\nlinkage: {linkage}' for linkage in linkages])

plot_hierarchical_scatter(scurve_np, 'scurve', [7, 8, 8, 3])
plot_hierarchical_scatter(checker_np, 'checker', [5, 5, 5, 5])
plot_hierarchical_scatter(luckyset_np, 'luckyset', [3, 3, 3, 3])

```

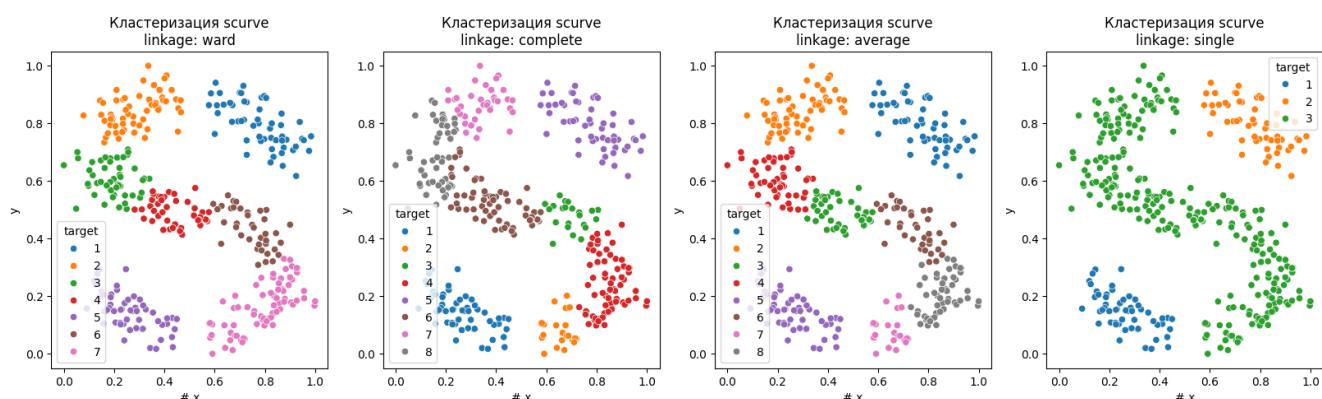


Рисунок 4.2.1 – Диаграммы рассеяния датасета scurve при разных типах linkage и наиболее оптимальном количестве кластеров

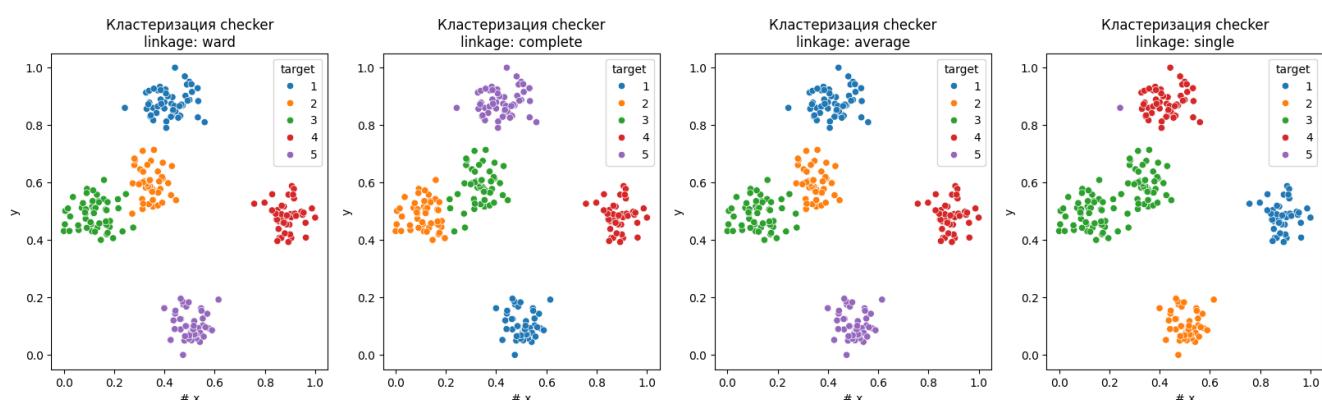


Рисунок 4.2.2 – Диаграммы рассеяния датасета checker при разных типах linkage и наиболее оптимальном количестве кластеров

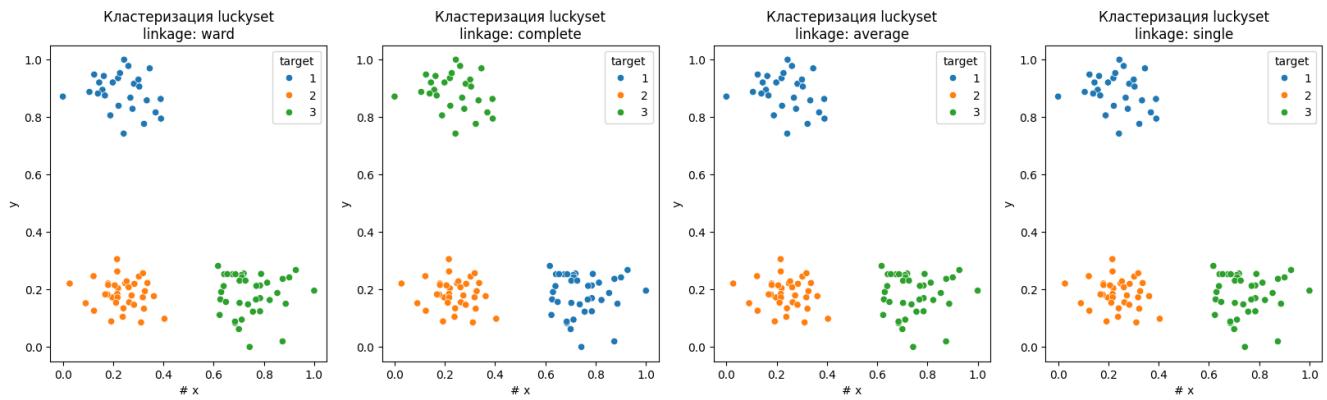


Рисунок 4.2.3 – Диаграммы рассеяния датасета luckyset при разных типах linkage и наиболее оптимальном количестве кластеров

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.

Исходя из результатов иерархической кластеризации можно сделать вывод, что для датасета scurve наилучшими гиперпараметрами является следующая пара: 3 кластера, linkage – «single». Для датасета checker наилучшая пара гиперпараметров выглядит следующим образом: 5 кластеров, linkage –«average». Для датасета luckyset наилучшая пара гиперпараметров следующая: 3 кластера, linkage – любой. Стоит отметить, что, в отличие от K-means, иерархическая кластеризация хорошо проявила себя при работе с датасетом вытянутой формы.

Сравнивая рассмотренные методы кластеризации, можно сделать вывод, что для данных датасетов наилучшее качество кластеризации у DBSCAN, затем следует иерархическая кластеризация, а хуже всех кластеризовал алгоритм K-means. Следует отметить, что для хорошего качества кластеризации при использовании DBSCAN или иерархической кластеризации, нужно правильно подобрать гиперпараметры: eps и min_samples для DBSCAN, linkage и число кластеров для иерархической кластеризации. DBSCAN стоит выше иерархической кластеризации в контексте качества, так как DBSCAN нашёл выбросы в датасетах checker и luckyset. Таким образом, DBSCAN является наилучшим методом кластеризации для датасетов scurve, checker и luckyset.

5. Изучение набора данных с большим количеством 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.

Для кластеризации данных необходимо исследовать данные, а также провести их предобработку. Для выполнения всех вышеперечисленных процедур необходимо загрузить датасет в программу, а затем проверить корректность загрузки, вызвав метод `head` у датафрейма. Вышеописанные шаги представлены в [Листинге 5.1.1](#).

Листинг 5.1.1 – Реализация загрузки датасета в программу и отображения первых пяти записей датафрейма

```
wineset = pd.read_csv('sample_data/lab2_winequality_red.csv')  
wineset.head()
```

После загрузки датасета нужно выбрать предобработку данных. Для этого можно проверить датасет на нормальность с помощью теста Шапиро-Уилка. Для этого нужно воспользоваться функцией `shapiro_test`. Реализация вызова функции представлена в [Листинге 5.1.2](#), результат – в [Листинге 5.1.3](#).

Листинг 5.1.2 – Реализация вызова функции для проверки признаков на нормальность

```
print("wineset:")  
shapiro_test(wineset)
```

Листинг 5.1.3 – Результат теста на нормальность для датасета wineset wineset:

```
column: fixed acidity
    pvalue: 1.5227779442162196e-24
column: volatile acidity
    pvalue: 2.686385373947307e-16
column: citric acid
    pvalue: 1.0233944777578548e-21
column: residual sugar
    scurve pvalue: 0.0
column: chlorides
    pvalue: 0.0
column: free sulfur dioxide
    pvalue: 7.699692533903026e-31
column: total sulfur dioxide
    pvalue: 3.573768919849872e-34
column: density
    pvalue: 1.9199848821926935e-08
column: pH
    pvalue: 1.7218767425219994e-06
column: sulphates
    pvalue: 5.821617678881608e-38
column: alcohol
    pvalue: 6.63998167657323e-27
column: quality
    pvalue: 9.524199756965729e-36
```

Исходя из результатов теста, можно сделать вывод, что стандартизация для данного датасета не подходит, так как данные не подчиняются нормальному распределению. Следовательно, можно рассмотреть вариант нормализации данных. Для этого нужно построить boxplot и оценить данные на наличие выбросов, которые могут заметно ухудшить качество нормализации. Для данной задачи была реализована функция features_boxplot, которая принимает на вход данные, параметр hue, по которому данные могут быть разделены цветом. Реализация данной функции и её вызов представлены в [Листинге 5.1.4](#), результат – на [Рисунке 5.1.1](#).

Листинг 5.1.4 – Реализация функции для отрисовки boxplot для каждого признака и её вызова

```
def features_boxplot(data, hue=None):
    palette = None
    if hue is not None:
        palette = 'tab10'
    fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(16, 12))
    for i in range(12):
        sns.boxplot(data=data, x=data.columns[i], ax=axes[i//4][i%4],
                    hue=hue, palette=palette)
    fig.suptitle('Распределения признаков на диаграмме boxplot',
                 y=1.02, fontsize=20)
    plt.tight_layout(pad=1.5)
    plt.show()

features_boxplot(wineset)
```

Распределения признаков на диаграмме boxplot

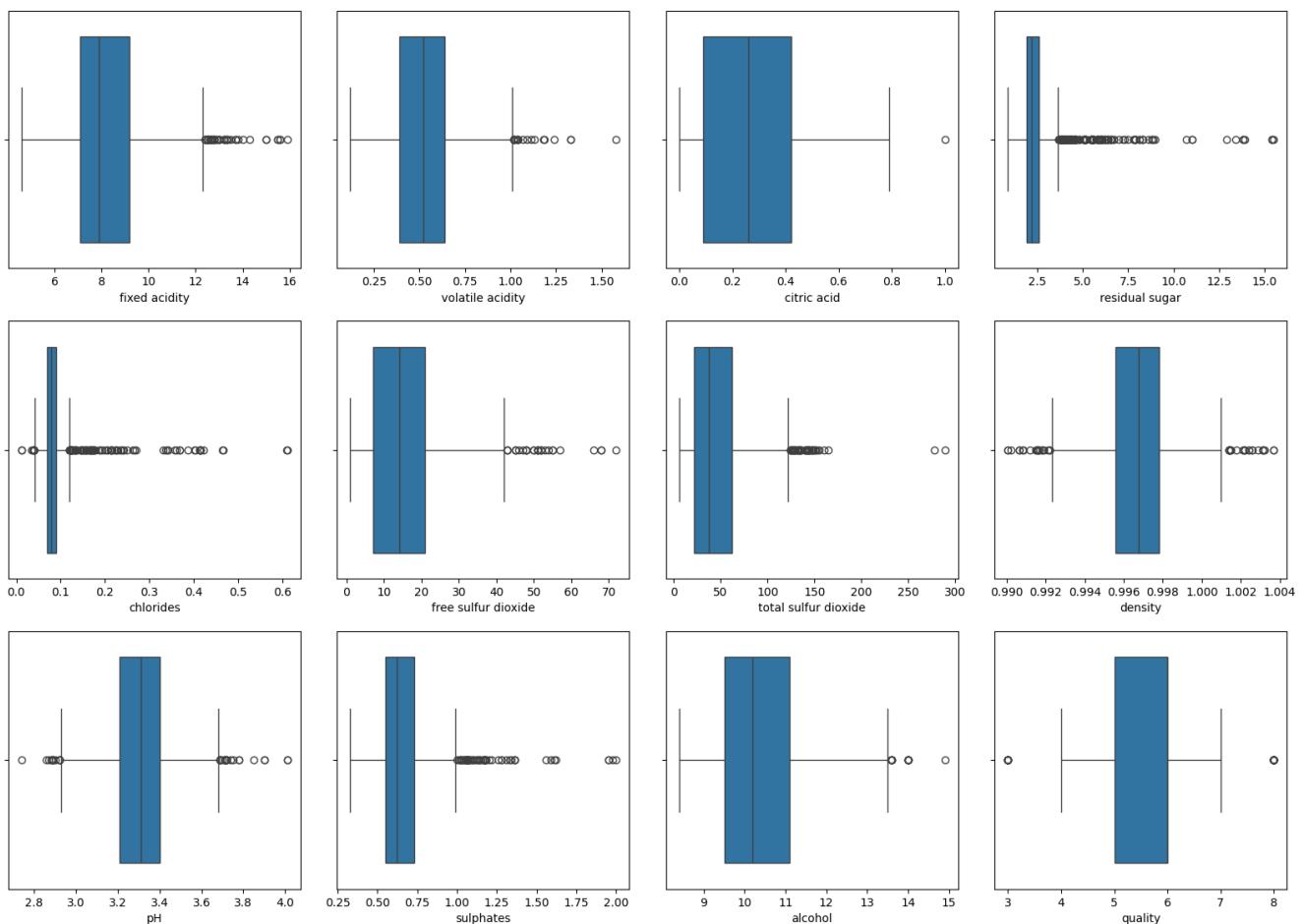


Рисунок 5.1.1 – Распределение признаков в виде диаграммы boxplot

Исходя из результатов, можно сделать вывод, что в датасете есть выбросы, которые нужно устраниТЬ. Для удаления выбросов использована ранее реализованная функция `compute_zmod`, которая принимает на вход датасет, а возвращает рассчитанные значения модифицированной z-score оценки. Затем формируется маска, на основе которой удаляются те объекты, для которых установлено значение `False`. Реализация представлена в [Листинге 5.1.5](#), результат удаления выбросов представлен на [Рисунке 5.1.2](#) в виде диаграммы boxplot.

Листинг 5.1.5 – Реализация создания маски, предназначенной для удаления выбросов из датасета

```
zmod = compute_zmod(wineset)
zmod_mask = (np.abs(zmod) < 3.5).all(axis=1)
```

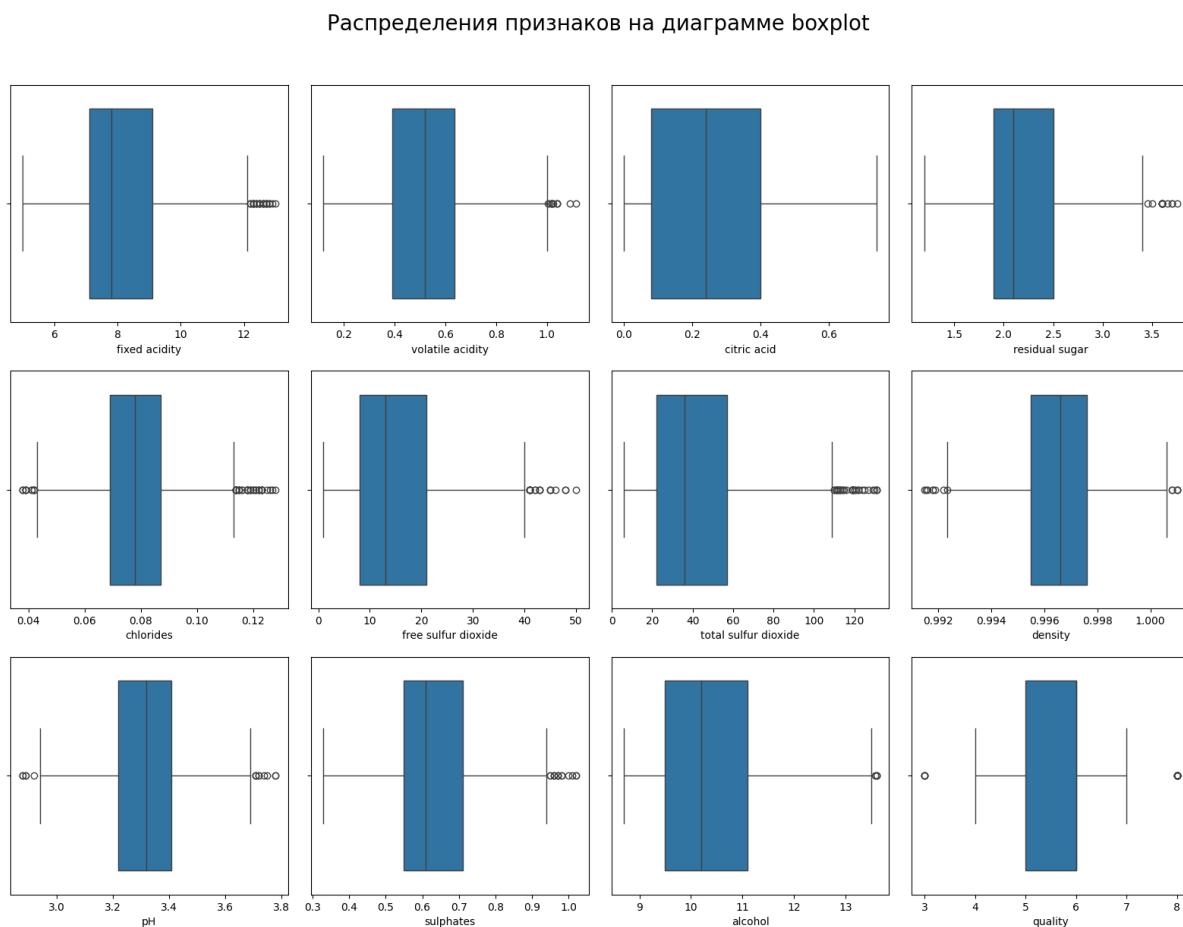


Рисунок 5.1.2 – Распределение признаков в виде диаграммы boxplot после удаления выбросов

Исходя из результатов, можно сделать вывод, выбросы, в соответствии с модифицированной z-score оценкой, успешно удалены.

Для выбора алгоритма кластеризации была изучена документация `sklearn.cluster`. После изучения документации было установлено, что нужно проанализировать следующие алгоритмы и выбрать из них наилучший: k-средних, спектральная кластеризация, иерархическая кластеризация, DBSCAN, HDBSCAN. Алгоритмы Birch и MiniBatchKMeans не будут рассматриваться, так как их использование для данного датасета избыточно; MeanShift и AffinityPropogation – данные алгоритмы формируют большое количество кластеров, что для данного датасета не подходит, так как после очистки от выбросов его мощность равна около 1300 объектов, и в случае применения данных алгоритмов кластеризации будет получено большое количество кластеров с весьма небольшим количеством объектов. Алгоритм Gaussian mixture не подходит, так как он предназначен для поиска кластеров с нормальным распределением, что для данного датасета нельзя гарантировать априори.

Для начала следует проанализировать алгоритм KMeans для данного датасета. Для этого была реализована функция `get_kmeans_scores_dict`, похожая по своему смыслу на ранее описанную `get_dbSCAN_scores_dict`. Отличие заключается в том, что перебирается только 1 гиперпараметр – число кластеров. Данная функция, как и подобные следующие для каждого алгоритма кластеризации, запускается с метрикой в виде коэффициента силуэта. В случае неоднозначности выбора алгоритма кластеризации будет использована реализованная функция `average_n_features_to_sep`. Данная функция вычисляет для каждого кластера число признаков, по которым можно отделить данный кластер от остальных на основе первой и третьей квартилий. Если ящик данного кластера для некоторого признака не перекрывается ни с каким другим ящиком, значит, данный кластер отделим от остальных кластеров по данному признаку. После расчёта такого числа признаков для каждого кластера вычисляется среднее, что является возвращаемым значением функции. Реализация функции `average_n_features_to_sep` представлена в [Листинге 5.1.6](#), реализация функции `get_kmeans_scores_dict`, её вызов и функции для отображения лучших гиперпараметров кластеризации представлены в [Листин-](#)

ге 5.1.7, результат вызовов – в [Листинге 5.1.8](#).

Листинг 5.1.6 – Реализация функции для расчёта среднего числа признаков, по которым каждый кластер можно отделить от остальных

```
def average_n_features_to_sep(data):
    df = data.groupby('target').describe()
    columns = data.columns[:-1]
    counts = np.zeros(len(df))
    for column in columns:
        for i in range(len(df)):
            separated = True
            for j in range(len(df)):
                if i != j:
                    if not ((df[column].iloc[i]['25%'] > df[column].iloc[j]['75%']) \ 
                            or (df[column].iloc[i]['75%'] < df[column].iloc[j]['25%'])):
                        separated = False
            if separated:
                counts[i] += 1
    return counts.mean()
```

Листинг 5.1.7 – Реализация функции перебора числа кластеров для KMeans, её вызов и функции для отображения трёх лучших гиперпараметров кластеризации

```
def get_kmeans_scores_dict(metric):
    rs = 42
    scores_dict = {}
    for i in tqdm(range(2, len(wineset_preproc) // 50)):
        kmeans = KMeans(n_clusters=i, n_init=10, random_state=rs)
        labels = kmeans.fit_predict(wineset_preproc)
        scores_dict = assign_score(wineset_preproc, labels, scores_dict,
                                   metric, kmeans)
    scores_dict = get_sorted_scores_dict(scores_dict, metric)
    return scores_dict

scores_dict = get_kmeans_scores_dict('ss')
print_scores_estimators(wineset_preproc, scores_dict, 'KMeans silhouette_score',
                        n_keys=3)
```

Листинг 5.1.8 – Наилучшие гиперпараметры KMeans и количественные составы кластеров

KMeans silhouette_score:

```
score = 0.21548017558133398:  
    KMeans(n_clusters=2, n_init=10, random_state=42)  
        number of clusters = 2  
        power of cluster 0 = 486  
        power of cluster 1 = 811  
score = 0.1920146493592023:  
    KMeans(n_clusters=3, n_init=10, random_state=42)  
        number of clusters = 3  
        power of cluster 0 = 572  
        power of cluster 1 = 324  
        power of cluster 2 = 401  
score = 0.18557994253473986:  
    KMeans(n_clusters=4, n_init=10, random_state=42)  
        number of clusters = 4  
        power of cluster 0 = 253  
        power of cluster 1 = 451  
        power of cluster 2 = 307  
        power of cluster 3 = 286
```

Результат кластеризации k-средних можно назвать удовлетворительным: коэффициент силуэта довольно небольшой, небольшое число кластеров, что и требовалось, количественный состав кластеров удовлетворителен.

Для анализа иерархической кластеризации с перебором числа кластеров реализована функция `get_hierarchical_scores_dict`, которая принимает на вход значение `linkage` и используемую метрику оценки кластеризации. Реализация данной функции представлена в [Листинге 5.1.9](#), вызов данной функции при использовании «ward» и функции для отображения лучших гиперпараметров кластеризации представлены в [Листинге 5.1.10](#), результат – в [Листинге 5.1.11](#), при использовании «complete» – в [Листинге 5.1.12](#), результат – в [Листинге 5.1.13](#), при использовании «average» – в [Листинге 5.1.14](#), результат – в [Листинге 5.1.15](#), при использовании «single» – в [Листинге 5.1.16](#), результат – [Листинге 5.1.17](#).

Листинг 5.1.9 – Реализация функции перебора числа кластеров для иерархической кластеризации

```
def get_hierarchical_scores_dict(str_linkage, metric):
    scores_dict = {}
    for i in range(2, len(wineset_preproc) // 50):
        Z = linkage(wineset_preproc, method=str_linkage)
        labels = fcluster(Z, i, criterion='maxclust')
        scores_dict = assign_score(wineset_preproc,
                                   labels, scores_dict, metric, i)
    scores_dict = get_sorted_scores_dict(scores_dict, metric)
    return scores_dict
```

Листинг 5.1.10 – Вызов функции перебора числа кластеров для иерархической кластеризации с параметром «ward» и функции для отображения трёх лучших гиперпараметров кластеризации

```
scores_dict = get_hierarchical_scores_dict('ward', 'ss')
print_scores_estimators(wineset_preproc, scores_dict,
                        'hierarchy silhouette score',
                        'ward', n_keys=3)
```

Листинг 5.1.11 – Наилучшие параметры кластеризации при использовании «ward»

```
hierarchy silhouette score:
score = 0.20642308203018633:
    ward, n_clusters = 2
        number of clusters = 2
        power of cluster 1 = 460
        power of cluster 2 = 837
score = 0.14736284157198343:
    ward, n_clusters = 3
        number of clusters = 3
        power of cluster 1 = 460
        power of cluster 2 = 201
        power of cluster 3 = 636
score = 0.14563459125267775:
    ward, n_clusters = 4
        number of clusters = 4
        power of cluster 1 = 460
        power of cluster 2 = 201
        power of cluster 3 = 112
        power of cluster 4 = 524
```

Результат кластеризации при использовании «ward» можно назвать удовле-

творительным: коэффициент силуэта довольно небольшой, небольшое число кластеров, что и требовалось, количественный состав кластеров удовлетворителен.

Листинг 5.1.12 – Вызов функции перебора числа кластеров для иерархической кластеризации с параметром «complete» и функции для отображения трёх лучших гиперпараметров кластеризации

```
scores_dict = get_hierarchical_scores_dict('complete', 'ss')
print_scores_estimators(wineset_preproc, scores_dict,
                        'hierarchy silhouette_score',
                        'complete', n_keys=3)
```

Листинг 5.1.13 – Наилучшие параметры кластеризации при использовании «complete»

```
hierarchy silhouette_score:
score = 0.14220398990256602:
    complete, n_clusters = 3
        number of clusters = 3
        power of cluster 1 = 175
        power of cluster 2 = 271
        power of cluster 3 = 851
score = 0.1417079592960589:
    complete, n_clusters = 2
        number of clusters = 2
        power of cluster 1 = 175
        power of cluster 2 = 1122
score = 0.13807279391384308:
    complete, n_clusters = 4
        number of clusters = 4
        power of cluster 1 = 6
        power of cluster 2 = 169
        power of cluster 3 = 271
        power of cluster 4 = 851
```

Результат кластеризации при использовании «complete» можно назвать удовлетворительным: коэффициент силуэта довольно небольшой, небольшое число кластеров, что и требовалось, количественный состав кластеров удовлетворителен.

Листинг 5.1.14 – Вызов функции перебора числа кластеров для иерархической кластеризации с параметром «average» и функции для отображения трёх лучших гиперпараметров кластеризации

```
scores_dict = get_hierarchical_scores_dict('average', 'ss')
print_scores_estimators(wineset_preproc, scores_dict,
                        'hierarchy silhouette_score',
                        'average', n_keys=3)
```

Листинг 5.1.15 – Наилучшие параметры кластеризации при использовании «average»

```
hierarchy silhouette_score:
score = 0.3754010832454051:
    average, n_clusters = 2
        number of clusters = 2
        power of cluster 1 = 4
        power of cluster 2 = 1293
score = 0.2599520226825635:
    average, n_clusters = 3
        number of clusters = 3
        power of cluster 1 = 4
        power of cluster 2 = 1292
        power of cluster 3 = 1
score = 0.20298128663806136:
    average, n_clusters = 4
        number of clusters = 4
        power of cluster 1 = 4
        power of cluster 2 = 3
        power of cluster 3 = 1289
        power of cluster 4 = 1
```

Результат кластеризации при использовании «average» неудовлетворителен, так как очень сильно нарушен баланс классов. В дальнейшем сравнении иерархическая кластеризация с данным типом linkage рассматриваться не будет.

Листинг 5.1.16 – Вызов функции перебора числа кластеров для иерархической кластеризации с параметром «single» и функции для отображения трёх лучших гиперпараметров кластеризации

```
scores_dict = get_hierarchical_scores_dict('single', 'ss')
print_scores_estimators(wineset_preproc, scores_dict,
                        'hierarchy silhouette_score',
                        'single', n_keys=3)
```

Листинг 5.1.17 – Наилучшие параметры кластеризации при использовании «single»

```
hiearachy silhouette_score:  
    score = 0.4033133950993059:  
        single, n_clusters = 2  
            number of clusters = 2  
            power of cluster 1 = 2  
            power of cluster 2 = 1295  
    score = 0.29832217021876195:  
        single, n_clusters = 3  
            number of clusters = 3  
            power of cluster 1 = 2  
            power of cluster 2 = 1294  
            power of cluster 3 = 1  
    score = 0.29085343609517733:  
        single, n_clusters = 4  
            number of clusters = 4  
            power of cluster 1 = 2  
            power of cluster 2 = 2  
            power of cluster 3 = 1292  
            power of cluster 4 = 1
```

Результат кластеризации при использовании «single» неудовлетворителен, так как очень сильно нарушен баланс классов. В дальнейшем сравнении иерархическая кластеризация с данным типом linkage рассматриваться не будет.

Для анализа спектральной кластеризации с перебором числа кластеров реализована функция `get_spectral_scores_dict`, которая принимает на вход используемую метрику оценки кластеризации. Реализация данной функции представлена в [Листинге 5.1.18](#), её вызов и функции для отображения лучших гиперпараметров кластеризации представлены в [Листинге 5.1.19](#), результат вызова функций – в [Листинге 5.1.20](#).

Листинг 5.1.18 – Реализация функции перебора числа кластеров для спектральной кластеризации

```
from sklearn.cluster import SpectralClustering
```

```
def get_spectral_scores_dict(metric):
    rs = 42
    scores_dict = {}
    for i in tqdm(range(2, len(wineset_preproc) // 50)):
        sc = SpectralClustering(n_clusters=i, random_state=rs)
        labels = sc.fit_predict(wineset_preproc)
        scores_dict = assign_score(wineset_preproc, labels, scores_dict,
                                   metric, sc)
    scores_dict = get_sorted_scores_dict(scores_dict, metric)
    return scores_dict
```

Листинг 5.1.19 – Вызов функции перебора числа кластеров для спектральной кластеризации и функции для отображения трёх лучших гиперпараметров кластеризации

```
score_dict = get_spectral_scores_dict('ss')
print_scores_estimators(wineset_preproc, score_dict, 'spectral clustering',
                        n_keys=3)
```

Листинг 5.1.20 – Наилучшие гиперпараметры спектральной кластеризации

spectral clustering:

```
score = 0.21612514701344024:
    SpectralClustering(n_clusters=2, random_state=42)
        number of clusters = 2
        power of cluster 0 = 456
        power of cluster 1 = 841
score = 0.18428837591751912:
    SpectralClustering(n_clusters=4, random_state=42)
        number of clusters = 4
        power of cluster 0 = 272
        power of cluster 1 = 454
        power of cluster 2 = 302
        power of cluster 3 = 269
score = 0.1704629492466312:
    SpectralClustering(n_clusters=3, random_state=42)
        number of clusters = 3
        power of cluster 0 = 531
        power of cluster 1 = 362
        power of cluster 2 = 404
```

Результат спектральной кластеризации можно назвать удовлетворительным: коэффициент силуэта довольно небольшой, небольшое число кластеров, что и требовалось, количественный состав кластеров удовлетворителен.

Для анализа кластеризации DBSCAN была использована описанная ранее функция `get_dbSCAN_scores_dict`. Реализация вызова данной функции и функции для отображения наилучших гиперпараметров реализации представлены представлены в [Листинге 5.1.21](#), результат вызова представлен в [Листинге 5.1.22](#).

Листинг 5.1.21 – Вызов функции перебора гиперпараметров для DBSCAN и функции для отображения трёх лучших гиперпараметров кластеризации

```
score_dict = get_dbSCAN_scores_dict(wineset_preproc, np.arange(0.1, 3.5, 0.1),
                                      np.arange(2, len(wineset_preproc) // 100),
                                      'ss')
print_scores_estimators(wineset_preproc, score_dict, 'dbSCAN', n_keys=3)
```

Листинг 5.1.22 – Вызов функции перебора гиперпараметров для DBSCAN и функции для отображения трёх лучших гиперпараметров кластеризации

dbSCAN:

```
score = 0.4036924025662953:
    DBSCAN(eps=0.7000000000000001, min_samples=2)
        number of clusters = 2
        power of cluster -1 = 1
        power of cluster 0 = 1294
        power of cluster 1 = 2
score = 0.4033133950993059:
    DBSCAN(eps=0.8, min_samples=2)
        number of clusters = 2
        power of cluster 0 = 1295
        power of cluster 1 = 2
score = 0.34143520578623926:
    DBSCAN(eps=0.6, min_samples=2)
        number of clusters = 3
        power of cluster -1 = 1
        power of cluster 0 = 1292
        power of cluster 1 = 2
        power of cluster 2 = 2
```

Результат кластеризации DBSCAN является неудовлетворительным, так как количественно кластеры сильно не сбалансированы.

Для анализа кластеризации HDBSCAN была реализована функция `get_hdbscan_scores_dict`. Она очень похожа на соответствующую функцию для DBSCAN, но отличается тем, что используется HDBSCAN вместо DBSCAN, поэтому можно перебирать лишь параметр `min_samples`. Реализация данной функции, её вызов и функции для отображения наилучших гиперпараметров представлены в [Листинге 5.1.23](#), результат – в [Листинге 5.1.24](#).

Листинг 5.1.23 – Вызов функции перебора гиперпараметров для HDBSCAN и функции для отображения трёх лучших гиперпараметров кластеризации

```
import hdbscan
```

```
def get_hdbscan_scores_dict(data, mins_samples, metric):
    scores_dict = {}
    for min_samples in mins_samples:
        h_dbSCAN = hdbscan.HDBSCAN(min_samples=min_samples)
        zmod = compute_zmod(data)
        zmod_mask = (np.abs(zmod) < 3.5).all(axis=1)
        n_outliers = len(data) - len(data[zmod_mask])
        labels = dbSCAN.fit_predict(data)
        if len(labels[labels == -1]) <= n_outliers:
            scores_dict = assign_score(data, labels,
                                       scores_dict, metric, h_dbSCAN)
    scores_dict = get_sorted_scores_dict(scores_dict, metric)
    return scores_dict

score_dict = get_hdbscan_scores_dict(wineset_preproc,
                                     np.arange(2, len(wineset_preproc) // 10),
                                     'ss')
print_scores_estimators(wineset_preproc, score_dict, 'hdbscan')
```

Листинг 5.1.24 – Вызов функции перебора гиперпараметров для HDBSCAN и функции для отображения трёх лучших гиперпараметров кластеризации

hdbSCAN:

```
score = 0.319745716631278:  
    HDBSCAN(min_samples=3)  
        number of clusters = 2  
        power of cluster -1 = 28  
        power of cluster 0 = 6  
        power of cluster 1 = 1263  
score = 0.3180167949646541:  
    HDBSCAN(min_samples=4)  
        number of clusters = 2  
        power of cluster -1 = 35  
        power of cluster 0 = 5  
        power of cluster 1 = 1257  
score = 0.3122663027509165:  
    HDBSCAN(min_samples=2)  
        number of clusters = 2  
        power of cluster -1 = 19  
        power of cluster 0 = 8  
        power of cluster 1 = 1270
```

Результат кластеризации HDBSCAN является неудовлетворительным, так как количественно кластеры сильно не сбалансированы.

Исходя из результатов анализа, можно сделать вывод, что наилучшими алгоритмами кластеризации для wineset является k-средних, иерархическая кластеризация «ward» и «complete» и спектральная кластеризация.

Для выбора наилучшего алгоритма кластеризации следует воспользоваться функцией average_n_features_to_sep. Реализация вызова данной функции для наилучших по методу коэффициентов гиперпараметров алгоритма KMeans представлена в [Листинге 5.1.25](#).

Листинг 5.1.25 – Вызов функции для расчёта среднего числа признаков, по которым каждый кластер отличается от остальных

```
kmeans = KMeans(n_clusters=2, n_init=10, random_state=42)  
labels = kmeans.fit_predict(wineset_preproc)  
wineset_preproc_clust = wineset_preproc.copy()  
wineset_preproc_clust['target'] = labels  
print(average_n_features_to_sep(wineset_preproc_clust))
```

Результатом вызова данной функции для алгоритма k-средних является число 3.0.

Вызов данной функции для иерархической кластеризации «ward» представлен в [Листинге 5.1.26](#).

Листинг 5.1.26 – Вызов функции для расчёта среднего числа признаков, по которым каждый кластер отличается от остальных

```
Z = linkage(wineset_preproc, method= 'ward ')
labels = fcluster(Z, 2, criterion= 'maxclust ')
wineset_preproc_clust = wineset_preproc.copy()
wineset_preproc_clust[ 'target ']= labels
print(average_n_features_to_sep(wineset_preproc_clust))
```

Результатом вызова данной функции для иерархической кластеризации «ward» является число 3.0.

Вызов данной функции для иерархической кластеризации «complete» представлен в [Листинге 5.1.27](#).

Листинг 5.1.27 – Вызов функции для расчёта среднего числа признаков, по которым каждый кластер отличается от остальных

```
Z = linkage(wineset_preproc, method= 'complete ')
labels = fcluster(Z, 3, criterion= 'maxclust ')
wineset_preproc_clust = wineset_preproc.copy()
wineset_preproc_clust[ 'target ']= labels
print(average_n_features_to_sep(wineset_preproc_clust))
```

Результатом вызова данной функции для иерархической кластеризации «complete» является число 1.33.

Вызов данной функции для спектральной кластеризации представлен в [Листинге 5.1.28](#).

Листинг 5.1.28 – Вызов функции для расчёта среднего числа признаков, по которым каждый кластер отличается от остальных

```
Z = linkage(wineset_preproc, method= 'complete ')
labels = fcluster(Z, 3, criterion= 'maxclust ')
wineset_preproc_clust = wineset_preproc.copy()
wineset_preproc_clust[ 'target ']= labels
print(average_n_features_to_sep(wineset_preproc_clust))
```

Результатом вызова данной функции для спектральной кластеризации является число 3.0.

Так как у иерархической кластеризации «complete» наименьшее значение функции average_n_features_to_sep, а у остальных кластеризаторов данное значение одинаковое, то иерархическая кластеризация «complete» далее не рассматривается, а оставшиеся кластеризаторы будут оцениваться по значению коэффициента силуэта.

Наибольшее значение коэффициента силуэта у спектральной кластеризации, следовательно, для датасета wineset данный метод кластеризации является наилучшим.

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.

Для отрисовки boxplot достаточно воспользоваться функцией box-plot, описанной в п.5.1 (см. [Рисунок 5.2.1](#)).

Распределения признаков на диаграмме boxplot

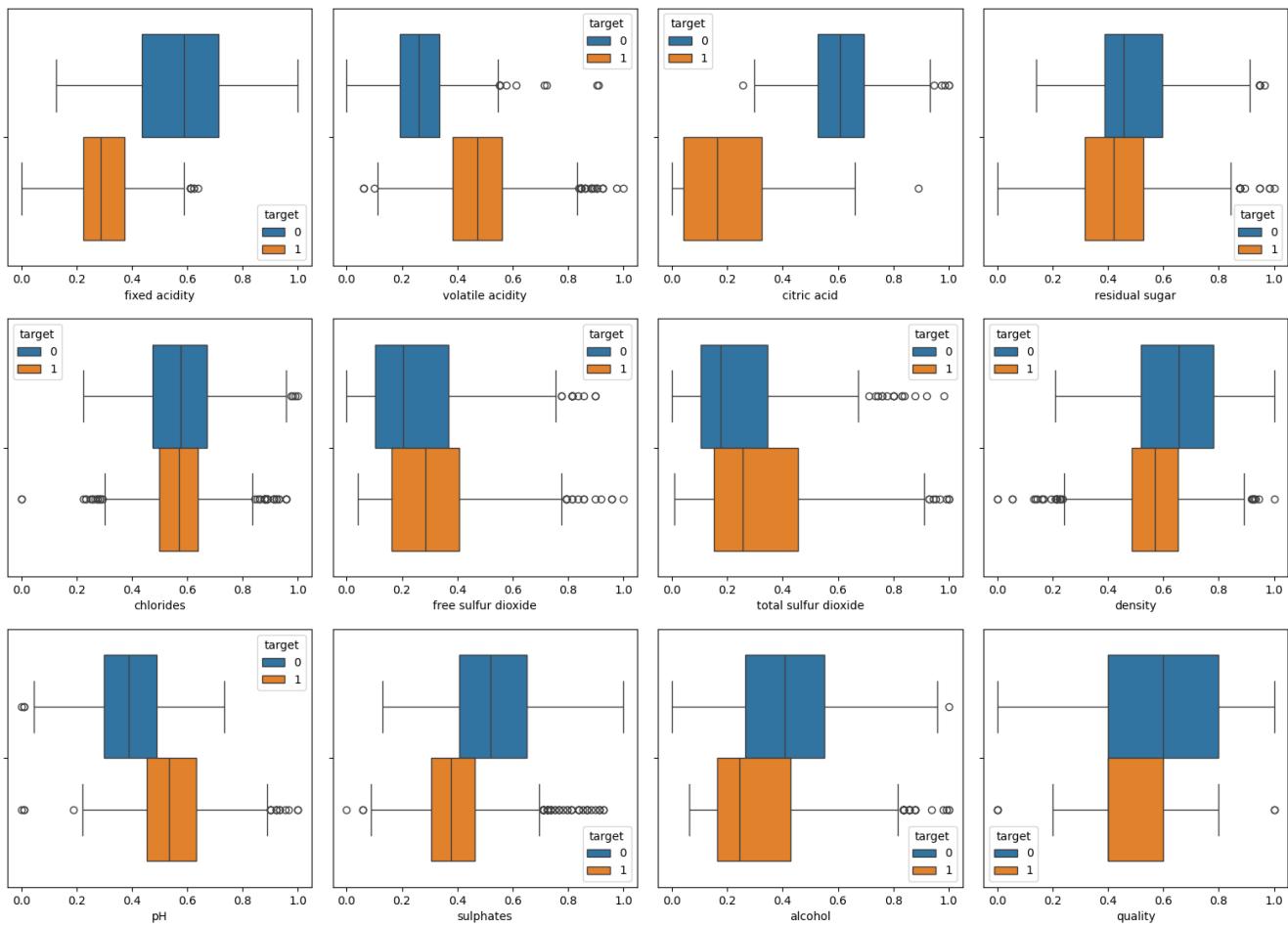


Рисунок 5.2.1 – Box-plot для датасета wineset после кластеризации

Исходя из данных на [Рисунке 5.2.1](#), данные относительно хорошо кластеризуются по признакам «fixed acidity», «valotile acidity» и «citric acid».

Также для визуализации следует построить диаграмму рассеяния. Для понижения размерности будет использован алгоритм РСА. Реализация понижения размерности и отрисовки представлены в [Листинге 5.2.1](#), диаграмма рассеяния – на [Рисунке 5.2.2](#).

Листинг 5.2.1 – Вызов функции для расчёта среднего числа признаков, по которым каждый кластер отличается от остальных

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

pca = PCA(n_components=2)
wineset_pca = pca.fit_transform(
    StandardScaler().fit_transform(wineset[zmod_mask]))
wineset_clustered = pd.DataFrame(data=wineset_pca,
                                   columns=['First axis', 'Second axis'])
wineset_clustered['target'] = labels

sns.scatterplot(data=wineset_clustered, hue='target',
                 x=wineset_clustered.columns[0],
                 y=wineset_clustered.columns[1],
                 palette='tab10').set(title='PCA')
```

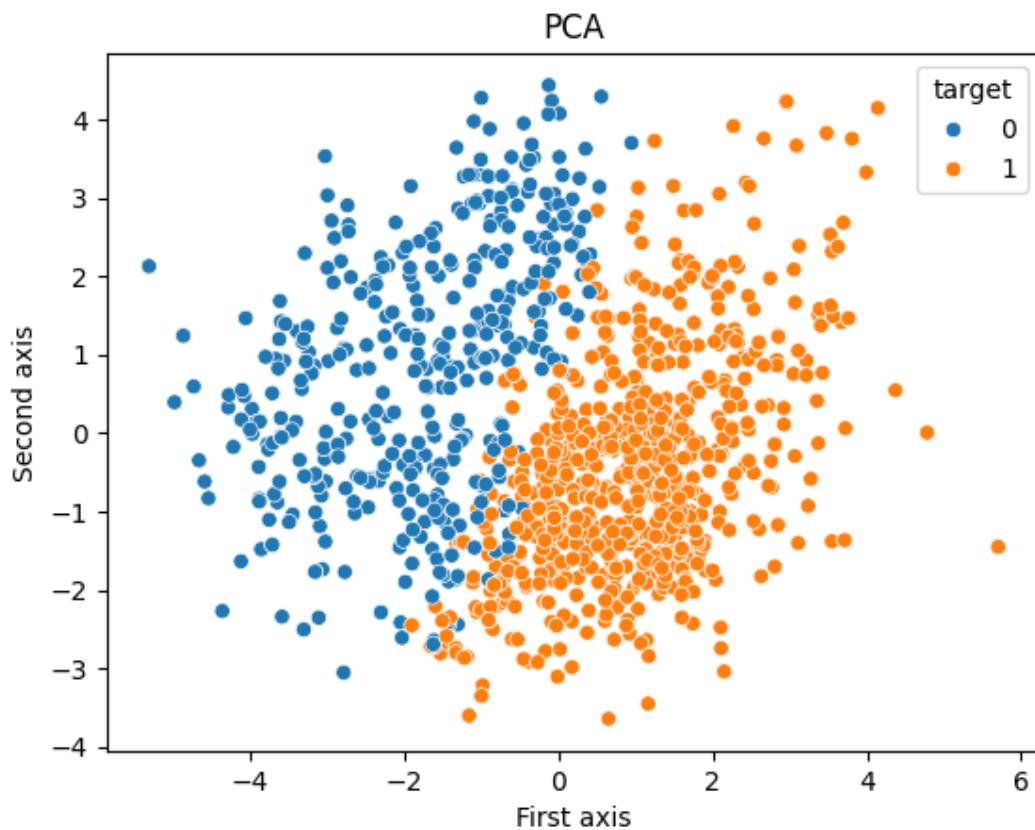


Рисунок 5.2.2 – Диаграмма рассеяния датасета wineset после понижения размерности

Для анализа полученной кластеризации реализованы попарные диаграммы

рассеяния, ядерной оценки плотности, а также одномерная ядерная оценка плотности. Исходный код представлен в [Листинге 5.2.2](#), результат выполнения – на [Рисунке 5.2.3](#).

Листинг 5.2.2 – Реализация отрисовки диаграмм рассеяния, попарных и одномерных ядерных оценок плотностей

```
def pair_grid(dataset, target):
    g = sns.PairGrid(dataset, diag_sharey=False, hue=target, palette='tab10')
    g.map_upper(sns.scatterplot, s=15)
    g.map_lower(sns.kdeplot)
    g.map_diag(sns.kdeplot)
    color_labels = []
    for curr_color in g.palette:
        color_labels.append(plt.Circle((0, 0), 1, color=curr_color))
    g.add_legend({g.hue_names[i]: color_labels[i] for i in range(len(g.palette))})
    g.fig.suptitle("Scatter plots and KDE's", y=1.02)
```

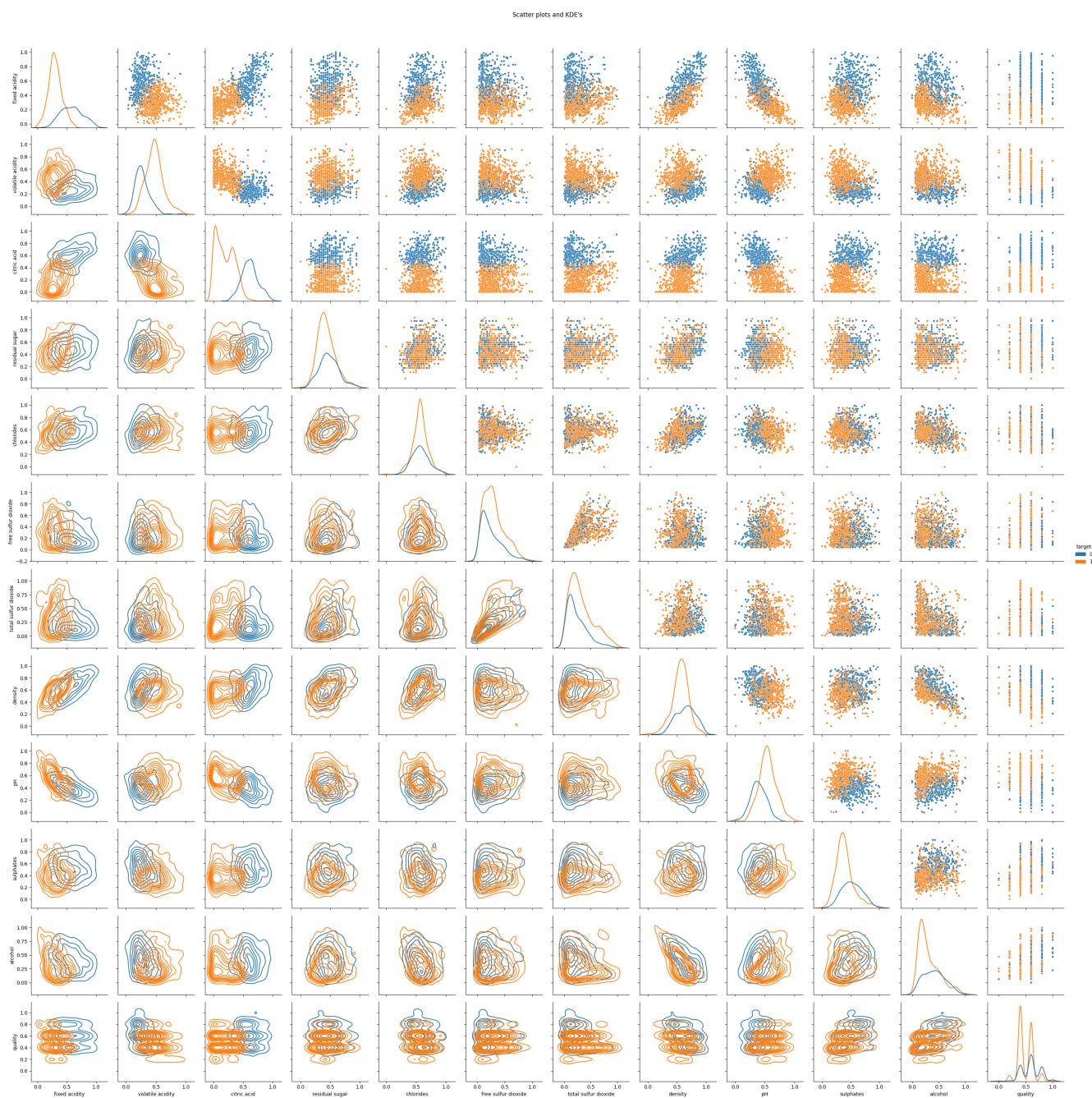


Рисунок 5.2.3 – Диаграммы рассеяния, попарных и одномерных ядерных оценок плотностей кластеризованного датасета wineset

Исходя из [Рисунка 5.2.3](#), можно сделать вывод, что данные относительно хорошо кластеризуются по следующим парам признаков:

- «volatile acidity» - «fixed acidity»;
- «citric acid» - «fixed acidity»;
- «citric acid» - «volatile acidity»;

- «residual sugar» - «citric acid»;
- «chlorides» - «citric acid»;
- «residual sugar» - «citric acid»;
- «free sulfur dioxide» - «citric acid»;
- «total sulfur dioxide» - «citric acid»;
- «density» - «citric acid»;
- «pH» - «citric acid»;
- «sulphates» - «citric acid»;
- «alcohol» - «citric acid»;
- «quality» - «citric acid».

Также был произведён тест на нормальность кластеров. Реализация вызова соответствующей функции представлена в [Листинге 5.2.3](#) и [Листинге 5.2.4](#) для первого и второго кластера соответственно, результат её выполнения – [Листинге 5.2.5](#) и [Листинге 5.2.6](#) соответственно.

Листинг 5.2.3 – Реализация теста на нормальность для первого кластера

```
print("cluster 1:")
shapiro_test(wineset_preproc_clust[wineset_preproc_clust['target'] == 0])
```

Листинг 5.2.4 – Результат теста на нормальность для каждого кластера

cluster 1:

```
column: fixed acidity
    pvalue: 0.00015080512093845755
column: volatile acidity
    pvalue: 8.509798768430166e-12
column: citric acid
    pvalue: 0.0019886502996087074
column: residual sugar
    pvalue: 3.4817901450878708e-06
column: chlorides
    pvalue: 0.000147741666296497
column: free sulfur dioxide
    pvalue: 2.0874936733541345e-17
column: total sulfur dioxide
    pvalue: 2.5309011937352248e-17
column: density
    pvalue: 0.00016346832853741944
column: pH
    pvalue: 0.36351609230041504
column: sulphates
    pvalue: 0.007227932568639517
column: alcohol
    pvalue: 8.164454811776523e-06
column: quality
    pvalue: 4.2006242588307015e-19
column: target
    pvalue: 1.0
```

Листинг 5.2.5 – Реализация теста на нормальность для первого кластера

```
print("cluster 2:")
shapiro_test(wineset_preproc_clust[wineset_preproc_clust['target'] == 1])
```

Листинг 5.2.6 – Результат теста на нормальность для каждого кластера

cluster 2:

```
column: fixed acidity
    pvalue: 6.101507096900605e-05
column: volatile acidity
    pvalue: 1.4675057968815963e-07
column: citric acid
    pvalue: 2.7523222546116723e-19
column: residual sugar
    pvalue: 2.49592013723543e-13
column: chlorides
    pvalue: 1.3281160171629836e-08
column: free sulfur dioxide
    pvalue: 1.2013192632968608e-18
column: total sulfur dioxide
    pvalue: 1.1696359710893954e-20
column: density
    pvalue: 2.250864739039571e-08
column: pH
    pvalue: 0.0006379640544764698
column: sulphates
    pvalue: 2.3500833830363997e-16
column: alcohol
    pvalue: 3.916005324421613e-24
column: quality
    pvalue: 3.967434493347319e-30
column: target
    pvalue: 1.0
```

Исходя из результатов, можно сделать вывод, что распределению Гаусса подчиняется только признак «рН» первого кластера, так как его значение pvalue больше 0.05. Остальные признаки, кроме density и quality имеют колоколообразное распределение.

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

Исходя из полученных результатов, можно сделать вывод, что сорта красных вин можно кластеризовать по характеристике содержания лимонной кислоты в них.

Вывод.

В ходе выполнения лабораторной работы были изучены различные алгоритмы кластеризации: k-средних, DBSCAN, иерархическая кластеризация, спектральная кластеризация, HDBSCAN. Для данных методов кластеризации были подобраны наиболее оптимальные гиперпараметры на основе таких метрик, как коэффициент силуэта, Дэвиса-Болдуина и Калинского-Харабаша. Также была изучена применимость различных методов кластеризации и закреплены различные методы предобработки данных и навыки визуализации результатов.