МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Институт Информационных Технологий

Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА №2

по дисциплине
«Проектирование интеллектуальных систем (часть 1/2)»

Студент группы:ИКБО-04-22 __Кликушин В.И._
(Ф. И.О. студента)

Преподаватель __Холмогоров В.В.__
(Ф.И.О. преподавателя)

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В условиях роста объема информации и ее разнообразия особую актуальность приобретает задача автоматического анализа данных. Методы интеллектуального анализа позволяют находить скрытые зависимости, выявлять структуру данных и принимать на их основе обоснованные решения. Кластеризация представляет собой важнейший инструмент разведочного анализа, позволяющий разделить данные на группы по сходству признаков без предварительных знаний о метках классов. Применение алгоритмов кластеризации эффективно в задачах маркетинга, биоинформатики, обработки изображений, распознавания образов и других областях, где требуется выявить внутреннюю структуру данных. Современные методы анализа данных позволяют не только визуализировать результаты кластеризации, но и количественно оценить качество полученного разбиения с помощью различных метрик.

1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: приобрести навыки кластеризации как инструмента исследовательского/разведочного анализа данных.

Задачи: определить предметную область решаемой задачи, найти или сгенерировать набор данных для выбранной задачи, проведя предварительную предобработку и подготовку данных, визуализировать подготовленные данные и провести предиктивную аналитику количества и качества кластеров, изучить алгоритмы кластеризации, написать программный код для реализации указанных алгоритмов, сравнить основные показатели производительности алгоритмов: качество результатов, скорость работы и требуемое количество памяти.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Кластеризация – одна из задач Data Mining, а кластер – группа похожих объектов.

Кластеризация – группировка объектов на основе близости их свойств; каждый кластер состоит из схожих объектов, а объекты разных кластеров существенно отличаются. Кластеризацию используют, когда отсутствуют априорные сведения относительно классов, к которым можно отнести объекты исследуемого набора данных, либо кода число объектов велико, что затрудняет их ручной анализ. Постановка задачи кластеризации сложна и неоднозначна, так как оптимальное количество кластеров в общем случае неизвестно и выбор меры «похожести» или близости свойств объектов между собой, как и критерия качества кластеризации, часто носит субъективный характер.

Объекты внутри кластера должны быть похожими друг на друга и отличаться от других, которые вошли в другие кластеры. В задачах кластеризации не требуется указание выходной переменной, т.е. имени кластера, а число кластеров, в которые необходимо сгруппировать все множество данных, может быть неизвестным. Выходом кластеризации является не готовый ответ, а группы похожих объектов — кластеры. Кластеризация указывает только на схожесть объектов, и не более того. Для объяснения образовавшихся кластеров необходима их дополнительная интерпретация.

2.1 Метод локтя

Метод локтя (Elbow method) — инструмент анализа данных, направленный на оптимизацию числа кластеров в алгоритмах кластеризации. Впервые был предложен Робертом Л. Торндайком в 1953 году.

Правильно подобранное количество кластеров в алгоритмах позволяет найти баланс между погрешностью вычисляемой дисперсии и сложностью

модели. Использование метода позволяет избежать недообучения или переобучения алгоритма кластеризации.

Метод применим к алгоритму k-средних и заключается в неоднократном повторении сценария. При использовании метода для каждого натурального числа k из некоторого диапазона строится значение целевой функции, равной сумме внутрикластерных расстояний. Количество кластеров — гиперпараметр, то есть он будет определен перед запуском модели.

Использование метода локтя подразумевает прохождение трех этапов.

На первом этапе для различных значений числа кластеров k вычисляется сумма квадратов расстояний каждой точки данных до их центроида (центра тяжести) (WCSS) по Формуле 2.1.1.

$$WCSS = \sum_{j=1}^{k} \sum_{i=1}^{n} \min(\left\| x_i^{(j)} - c_j \right\|)^2, \quad (2.1.1)$$

где k – число кластеров;

n – количество наблюдений;

$x_i^{(j)}$ – i-ое наблюдение в j-ом кластере;

c_j – центроид j-того кластера.

Второй этап содержит построение графика зависимости WCSS от количества кластеров, где по оси X откладывается число кластеров k, а по оси Y — соответствующая сумма квадратов расстояний.

Третий этап заключается в поиске точки излома («локтя») на графике, которая указывает на оптимальное число кластеров. Оптимальным k будет то, при котором ошибка перестает существенно уменьшаться, т.е. начинает сглаживаться.

2.2 Силуэтный анализ

Силуэт кластера — метод графического представления результатов кластеризации, с помощью которого можно визуально оценить качество построенной кластерной модели.

В основе идеи метода лежит вычисление коэффициентов кластерных силуэтов. На диаграмме для каждого объекта коэффициент силуэта отображается прямоугольником соответствующей длины. Прямоугольники группируются по кластерам (которые обычно выделяются цветом) и в каждом кластере дополнительно ранжируются в порядке убывания.

Таким образом, на диаграмме становится виден «силуэт» каждого кластера, откуда и название метода. По форме силуэтов аналитик оперативно может оценить качество кластеризации. Чем форма силуэтов ближе к прямоугольной, а площадь (средний коэффициент силуэта) ближе к 1, тем лучше кластеризация. Внутри силуэта каждого кластера объекты расположены в порядке убывания их коэффициента силуэта, поэтому легко увидеть, какие именно объекты лучше соответствуют кластеру, а какие хуже.

Напротив, чем больше в кластере объектов с низким коэффициентом силуэта, которые порождают «узкие» силуэты, тем хуже кластеризация.

Таким образом, диаграммы силуэтов и средние значения коэффициентов могут использоваться для определения естественного числа кластеров в наборе данных.

2.3 Алгоритм k-means

Алгоритм K-Means (метод k-средних) — один из наиболее популярных и широко используемых алгоритмов кластеризации. Он позволяет разбить набор объектов на заданное число k кластеров $C_1, C_2, \dots, C_k$ так, чтобы внутри каждого кластера объекты были как можно ближе друг к другу, а между кластерами — как можно дальше друг от друга.

Основная цель алгоритма — минимизировать внутрикластерную дисперсию, то есть расстояние объектов до центра своего кластера.

K-Means относится к центроидным алгоритмам, где каждый кластер описывается своим центром (центроидом), а каждый объект относится к ближайшему центру.

Функция, которую минимизирует алгоритм, называется WSS (Within-Cluster Sum of Squares) или WCSS (Within-Cluster Sum of Squared distances) (Формула 2.1.1).

Алгоритм K-Means работает итеративно и включает следующие шаги:

1. Инициализация центров кластеров.
2. Назначение точек кластерам. Каждая точка x_i назначается ближайшему центру кластера (Формула 2.3.1).

$$cluster(x_i) = \arg\min \left\| x_i - c_j \right\|^2 \quad (2.3.1)$$

3. Обновление центров кластеров. После перераспределения точек, для каждого кластера пересчитывается центр по Формуле 2.3.2.

$$c_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i \quad (2.3.2)$$

4. Проверка на сходимость. Если центры не изменились или изменения меньше заданного порога — завершить. В противном случае вернуться к шагу 2.

2.4 Оценка качества кластеризации

Метрики оценки качества кластеризации делятся на две группы: внутренние и внешние.

Внутренние метрики оценивают качество кластеризации на основе только входных данных и найденных кластеров. Они не требуют знания истинных меток классов.

Наиболее известные следующие внутренние метрики:

1. Silhouette Score (коэффициент силуэта)

Метрика силуэта измеряет, насколько объект похож на объекты своего кластера по сравнению с ближайшим другим кластером.

Для каждого объекта i рассчитываются две величины: $a(i)$ – среднее расстояние до всех точек своего кластера и $b(i)$ – минимальное среднее расстояние до точек другого (соседнего) кластера. Коэффициент силуэта вычисляется по Формуле 2.4.1.

$$s_i = \frac{b(i)-a(i)}{\max\{a(i),b(i)\}} \in [-1;1] \quad (2.4.1)$$

При $s_i \approx 1$ объект находится ближе к своему кластеру, чем к другим, при $s_i \approx 0$ – объект на границе кластеров, а при $s_i < 0$ объект может быть отнесен не к своему кластеру. Среднее значение по всем объектам используется как итоговая метрика.

2. Calinski-Harabasz Index

Также называется индексом дисперсионного отношения. Отражает соотношение между межкластерной и внутрикластерной дисперсиями. Чем выше значение, тем лучше кластеризация. Вычисляется по Формуле 2.4.2.

$$CH = \frac{Tr(B_k)}{Tr(W_k)} * \frac{n-k}{k-1}, \quad (2.4.2)$$

где $Tr(B_k)$ – межкластерная дисперсия (разброс центров кластеров);

$Tr(W_k)$ – внутрикластерная дисперсия (разброс точек внутри кластеров);

n – общее число объектов;

k – количество кластеров.

3. Davies-Bouldin Index

Индекс Дэвиса-Болдина оценивает, насколько похожи кластеры друг на друга. Чем меньше значение, тем лучше кластеризация (меньше перекрытие между кластерами). Вычисляется по Формуле 2.4.3.

$$DB = \frac{1}{k}\sum_{i=1}^{k} \max\left(\frac{\sigma_i+\sigma_j}{d(c_i,c_j)}\right) \text{при } i \neq j \,, \tag{2.4.3}$$

где σ_i – среднее расстояние от точек в кластере i до его центра c_i;

$d(c_i, c_j)$ – расстояние между центрами кластеров i и j.

Внешние метрики применяются только если известны истинные метки классов. Они сравнивают полученные кластеры с эталонной разметкой и определяют, насколько хорошо кластеры соответствуют «настоящим» группам. Наиболее популярные из них:

1. Adjusted Rand Index (ARI)

Скорректированный индекс Рэнда оценивает совпадения пар объектов, отнесённых к одним и тем же или разным кластерам и классам. Значение нормировано от –1 до 1: при ARI = 1 – случайное совпадение, при ARI = 0 – случайное разбиение. ARI учитывает количество:

- True Positive (одинаковый кластер и класс);
- True Negative (разные кластеры и классы);

и корректирует результат относительно случайного совпадения.

2. Rand Index (RI)

Оригинальный (нескорректированный) индекс Рэнда, вычисляет долю совпадающих пар между кластеризацией и эталоном (Формула 2.4.4).

$$RI = \frac{TP+TN}{0.5n} \,, \tag{2.4.4}$$

где TP – количество пар, находящихся в одном и том же кластере и классе;

TN – количество пар, находящихся в разных кластерах и разных классах.

3. Fowlkes-Mallows Index (FMI)

Метрика, основанная на геометрическом среднем между точностью и полнотой парной кластеризации (Формула 2.4.5).

$$FMI = \sqrt{\frac{TP}{TP+FP} * \frac{TP}{TP+FN}}, \quad (2.4.5)$$

где TP – число пар, правильно отнесённых в один кластер и класс;

FP – количество ложноположительных пар;

FN – количество ложноотрицательных пар.

Значение лежит в диапазоне от 0 до 1.

4. Mutual Information (MI)

Взаимная информация измеряет, насколько знание одного распределения (классов) уменьшает неопределённость другого (кластеров). Метрика рассчитывается по Формуле 2.4.6.

$$MI(U,V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} P(i,j) \log \frac{P(i,j)}{P(i)P(j)} \quad (2.4.6)$$

5. Homogeneity

Кластер однороден, если все его элементы относятся к одному истинному классу. Метрика принимает значение от 0 до 1: 1 - идеальная однородность, 0 - полное смешение классов внутри кластеров.

6. Completeness

Кластеризация полна, если все элементы одного класса попали в один кластер. Аналогично Homogeneity, измеряется от 0 до 1.

7. V-measure

Гармоническое среднее между Homogeneity и Completeness (Формула 2.4.7).

$$V = 2 * \frac{homogeneity * completeness}{homogeneity + completeness} \quad (2.4.7)$$

2.5 Алгоритм DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) — это алгоритм кластеризации, основанный на плотности точек. В отличие от алгоритма k-means, DBSCAN не требует заранее задавать число кластеров и способен обнаруживать кластеры произвольной формы, а также выделять выбросы (шум) как отдельную категорию данных.

Главная идея: если вокруг точки находится достаточно других точек, она считается центром плотной области, и эта область расширяется в кластер. Таким образом, кластеры формируются как группы плотных точек, разделённые областями с низкой плотностью.

Алгоритм опирается на два параметра: ε – радиус окрестности (epsilon neighborhood), $MinPts$ – минимальное количество точек в радиусе ε, чтобы область считалась плотной.

На их основе формируются следующие понятия:

- ε – окрестность точки p (Формула 2.4.8);

$$N_\varepsilon(p) = \{q \in D \mid dist(p, q) \leq \varepsilon\} \quad (2.4.8)$$

- точка q плотностно-достижима из p, если выполняются следующие условия (Формула 2.4.9-2.4.10);

$$q \in N_\varepsilon(p) \quad (2.4.9)$$

$$|N_\varepsilon(p)| \geq MinPts \quad (2.4.10)$$

То есть, p – ядро кластера.

- точки p и q связаны по плотности, если найдётся цепочка точек, каждая из которых плотностно-достижима из предыдущей;
- точка, которая не входит ни в одну плотную область, считается выбросом.

Алгоритм DBSCAN включает следующие шаги:

1. Для каждой точки p в данных D найти ε-окрестность $N_\varepsilon(p)$.
2. Если размер $N_\varepsilon(p) < MinPts$, пометить p как шум.
3. Иначе создать новый кластер, добавить p и все точки, плотностно-достижимые из p в этот кластер.
4. Рекурсивно применить процесс ко всем новым точкам в кластере.
5. Если обработаны все точки, завершить работу алгоритма, иначе вернуться на шаг 1.

3 ДОКУМЕНТАЦИЯ К ДАННЫМ

3.1 Описание предметной области

В качестве набора данных выбран широко известный Wine Dataset (данные о сортах итальянского вина, полученных в результате химического анализа образцов), хранящийся в открытом репозитории UCI Machine Learning Repository и доступный в библиотеке scikit-learn. Датасет содержит результаты аналитических измерений для красных и белых вин, произведённых тремя различными винодельческими культурами из региона северо-западной Италии. Целью сбора таких данных является изучение химических и физических свойств вина, которые позволяют не только классифицировать образцы по сорту, но и делать выводы о качестве продукта и возможных дефектах при его изготовлении.

3.2 Анализ данных

В исходных данных представлено 178 образцов вин. Каждый образец соответствует конкретной бутылке вина одного из трех сортов (классов), обозначенных как «Class 0», «Class 1» и «Class 2». Эти сорта соответствуют коммерческой классификации винодельческих культур:

- class 0 – сорт «Barolo»;
- class 1 – сорт «Grignolino»;
- class 2 – сорт «Barbera».

Для предобработки и анализа датасета в файле dataset_manager.py написан класс DatasetManager. Содержание файла dataset_manager.py представлено в Приложении А.

Исходные данные представлены в виде таблицы с 14 колонками (13 признаков и целевая метка класса). Каждый объект содержит 13 числовых признаков, характеризующих физико-химическое состояние образца, и целевое значение «target» – метку сорта (0, 1 или 2). Колонки с признаками включают:

1. Alcohol (спиртовая крепость): концентрация этанола в вине.
2. Malic acid (яблочная кислота, г/дм^3): остаточное содержание яблочной кислоты после ферментации.
3. Ash (зола, г/дм^3): количество минеральных веществ, оставшихся после сжигания пробы.
4. Alcalinity of ash (щелочность золы): показатель щелочности зольных составляющих.
5. Magnesium (магний, мг/дм^3): концентрация ионов магния.
6. Total phenols (общие фенолы, г/дм^3): суммарное содержание фенольных соединений, влияющих на цвет и вкус.
7. Flavanoids (флавоноиды, оптическая плотность): концентрация флавоноидных соединений, отвечающих за танинность и антиоксидантные свойства.
8. Nonflavanoid phenols (нефлавоноидные фенолы, оптическая плотность): другие фенольные соединения, не относящиеся к флавоноидам.
9. Proanthocyanins (проантоцианидины, оптическая плотность): полифенолы, влияющие на структуру и долговечность вина.
10. Color intensity (интенсивность цвета, оптическая плотность): показатель насыщенности цвета, получаемый спектрофотометрически.
11. Hue (оттенок): отношение определённых спектральных поглощений, характеризующее оттенок красного/фиолетового.
12. OD280/OD315 of diluted wines (отношение оптической плотности при 280 нм и 315 нм): индикатор содержания фенольных соединений при разведении.
13. Proline (пролин, мг/дм^3): аминокислота, одна из наиболее представленных в вине, влияющая на вкус и аромат.

Описательная статистика признаков отображена на Рисунке 3.2.1.

	count	mean	std	min	25%	50%	75%	max
alcohol	178.0	13.000618	0.811827	11.03	12.3625	13.050	13.6775	14.83
malic_acid	178.0	2.336348	1.117146	0.74	1.6025	1.865	3.0825	5.80
ash	178.0	2.366517	0.274344	1.36	2.2100	2.360	2.5575	3.23
alcalinity_of_ash	178.0	19.494944	3.339564	10.60	17.2000	19.500	21.5000	30.00
magnesium	178.0	99.741573	14.282484	70.00	88.0000	98.000	107.0000	162.00
total_phenols	178.0	2.295112	0.625851	0.98	1.7425	2.355	2.8000	3.88
flavanoids	178.0	2.029270	0.998859	0.34	1.2050	2.135	2.8750	5.08
nonflavanoid_phenols	178.0	0.361854	0.124453	0.13	0.2700	0.340	0.4375	0.66
proanthocyanins	178.0	1.590899	0.572359	0.41	1.2500	1.555	1.9500	3.58
color_intensity	178.0	5.058090	2.318286	1.28	3.2200	4.690	6.2000	13.00
hue	178.0	0.957449	0.228572	0.48	0.7825	0.965	1.1200	1.71
od280/od315_of_diluted_wines	178.0	2.611685	0.709990	1.27	1.9375	2.780	3.1700	4.00
proline	178.0	746.893258	314.907474	278.00	500.5000	673.500	985.0000	1680.00

Рисунок 3.2.1 – Описательная статистика признаков

В таблице приведены стандартные метрики для каждого из 13 признаков: среднее значение (mean), стандартное отклонение (std), минимум (min), первые и третьи квартили (25% и 75%), медиана (50%) и максимум (max).

Описательные статистики показывают, что часть признаков (например, Alcohol, Magnesium) распределены относительно компактно, в то время как другие признаки (Malic acid, Proline, Proanthocyanins) обладают более широким разбросом и выраженной скошенностью. Для корректной кластеризации и визуализации потребуется стандартизация и, возможно, дополнительная обработка выбросов.

Распределение вин по классам представлена на Рисунке 3.2.2.

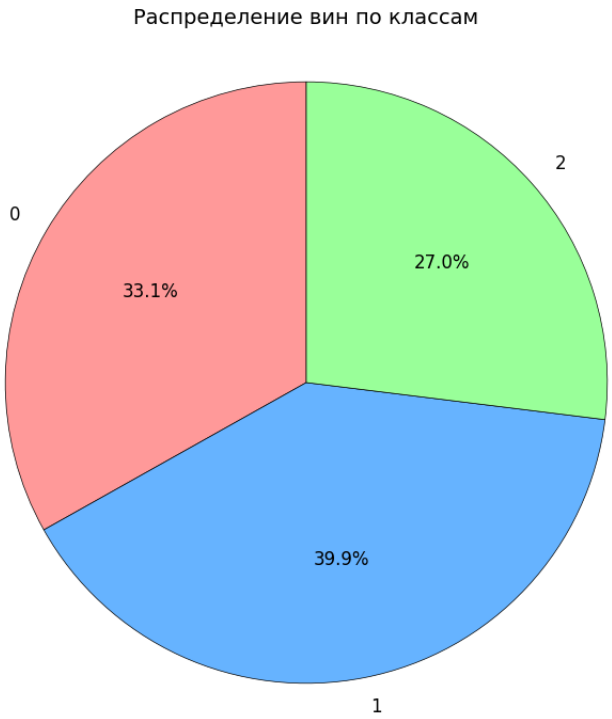

Рисунок 3.2.2 – Распределение вин по классам

Таким образом, класс 1 представлен наиболее обильно, а класс 2 – наименее. Несмотря на небольшую дисбалансировку, соотношение классов достаточно близко к равномерному.

Гистограммы распределений признаков представлены на Рисунке 3.2.3.

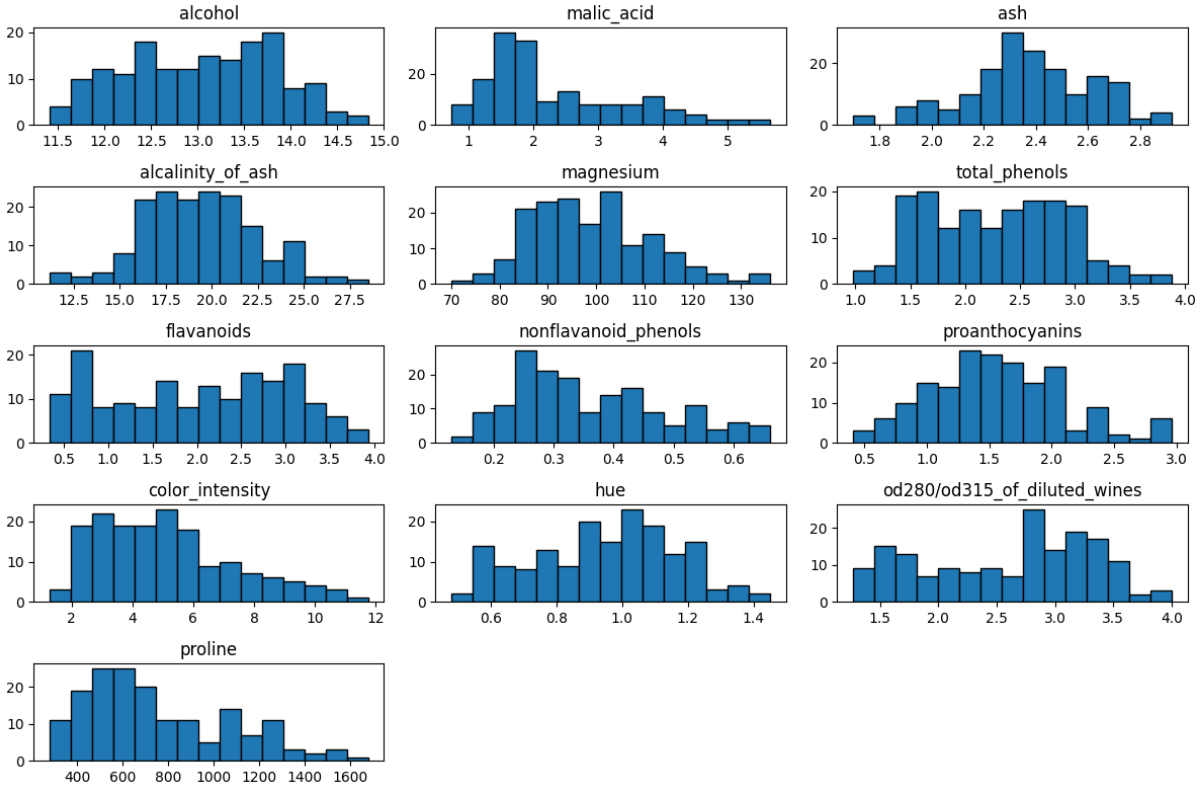

Рисунок 3.2.3 - Гистограммы распределений признаков

Практически все признаки имеют выраженную скошенность и отдельные выбросы. Это подчёркивает необходимость обработки экстремумов и обязательное масштабирование перед кластеризацией.

Матрица рассеяния для первых семи признаков представлена на Рисунке 3.2.4.

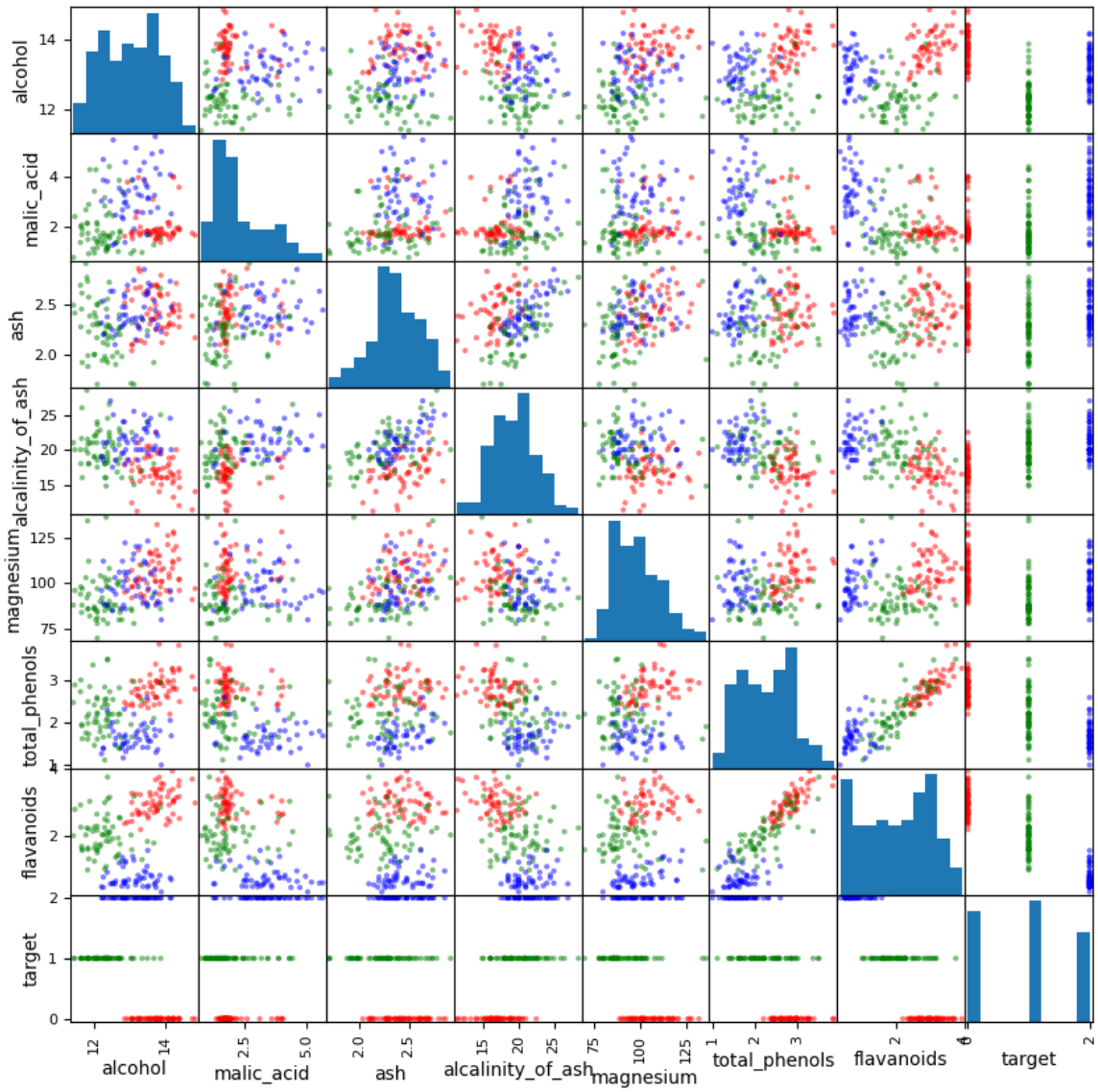

Рисунок 3.2.4 – Матрица рассеяния для первых семи признаков

На диагонали расположены гистограммы отдельных признаков (те же, что частично были на Рисунке 3.2.3, но только для первых семи). В ячейках показаны облака точек для пар признаков. Некоторые пары признаков обеспечивают достаточно чёткое различие классов. Между признаками Total phenols и Flavanoids прослеживается сильная корреляция.

Матрица корреляции признаков представлена на Рисунке 3.2.5.

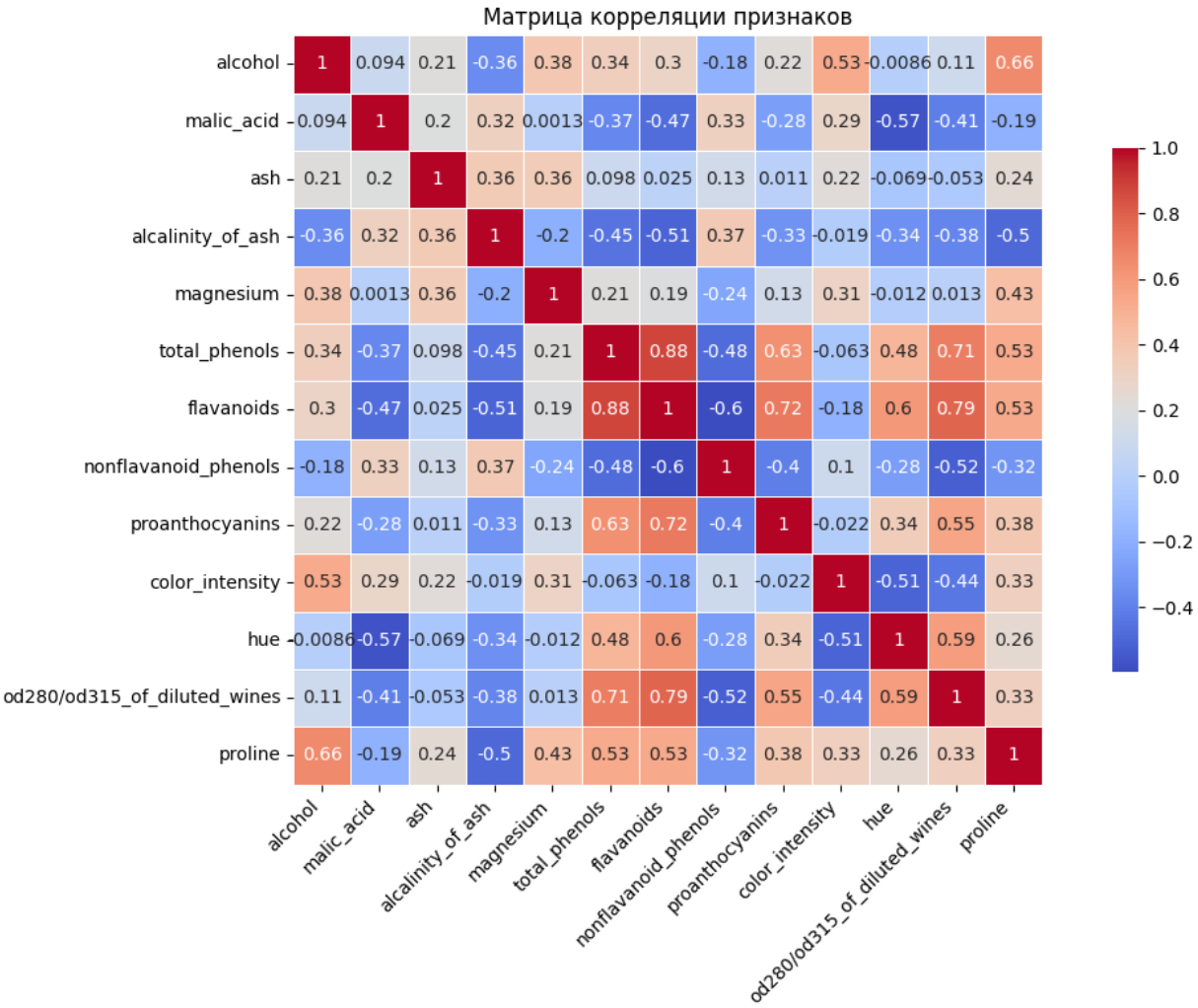

Рисунок 3.2.5 – Матрица корреляции признаков

Высокая корреляция между некоторыми признаками (Flavanoids и Total phenols) позволяет сократить размерность данных, исключив дублирующиеся признаки.

3.3 Предобработка данных

Реализованы следующие этапы предобработки данных:

- удаление дубликатов строк;
- удаление выбросов по Z-оценке;
- масштабирование признаков (StandardScaler);
- выбор и удаление избыточных признаков.

Дублированные строки искажают распределение признаков и искусственно увеличивают число объектов в кластерах. Удаление дубликатов

строк подразумевает обнаружение и удаление полностью идентичных записей (строк) в исходном датасете. Под «идентичностью» понимается совпадение всех значений по всем признакам. В рассматриваемом датасете дубликатов не обнаружено.

Выбросы (аномальные значения) — это отдельные объекты, сильно отклоняющиеся от общей «массы» точек. Чаще всего они встречаются в признаках с широким диапазоном. Один из способов формального выявления выбросов — использовать Z-оценку.

Для каждого значения x_{ij} признака j в образце i рассчитывается величина z_{ij} по Формуле 3.3.1.

$$z_{ij} = \frac{x_{ij} - \mu_i}{\sigma_i}, \tag{3.3.1}$$

где μ_i – среднее отклонение признака j по всем образцам;

σ_i – стандартное отклонение признака j по всем образцам.

Образец i считается выбросом, если хотя бы один признак имеет $|z_{ij}| > z_{threshold}$.

В качестве порогового значения выбран $z_{threshold}$ = 3. Таким образом, удалены все образцы, в которых хотя бы один признак отклонён от среднего более чем на три стандартных отклонения. В результате удаления выбросов по Z-оценке было исключено десять строк из исходного набора данных.

Скалирование признаков — это приведение всех измеряемых величин в единый единичный масштаб. В Wine Dataset признаки измеряются в разных физических и химических единицах. Без масштабирования признаки с большим диапазоном «будут весить» значительно больше при кластеризации по евклидову расстоянию, чем признаки с узким диапазоном.

StandardScaler — один из наиболее распространённых способов стандартизации. Для каждого признака j вычисляется среднее μ_j и стандартное отклонение σ_j. Каждое значение x_{ij} преобразуется по Формуле 3.3.2.

$$x'_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}, \quad (3.3.2)$$

В результате стандартизированный признак x'_{ij} имеет среднее 0 и стандартное отклонение 1.

На Рисунке 3.2.5 показано, что коэффициент корреляции между «Total phenols» и «Flavanoids» составляет примерно 0.86. Это значит, что эти два признака фактически несут очень близкую информацию о составе вина. Когда признаки столь сильно коррелированы, они считаются практически линейно зависимыми: наличие одного позволяет почти однозначно восстановить второй. Поэтому признак «Total phenols» исключен из набора признаков, а соответствующий столбец в датасете удален.

Предобработанные данные представлены на Рисунке 3.2.6.

```
Удалено дубликатов: 0
Удалено выбросов: 10
Предобработка завершена: дубликаты и выбросы (если указано) удалены, признаки масштабированы.
Признак 'total_phenols' успешно удалён из набора данных.
      alcohol  malic_acid       ash  alcalinity_of_ash  magnesium  flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity       hue  od280/od315_of_diluted_wines   proline
0    1.513539   -0.578842  0.250637          -1.209653   2.253466    1.068668             -0.669081         1.395565         0.233962  0.403766                      1.857453  0.980537
1    0.210929   -0.514888 -0.945351          -2.624711   0.097439    0.763248             -0.830933        -0.517348        -0.329398  0.449249                      1.127605  0.933414
2    0.160342    0.015009  1.240421          -0.244840   0.177292    1.251920             -0.507230         2.380432         0.251846  0.358284                      0.804788  1.357515
3    1.690593   -0.359574  0.539324          -0.823728   1.135526    1.506437             -0.992784         1.187228         1.199721 -0.414920                      1.197783  2.284254
4    0.261516    0.225141  2.065240           0.527010   1.534790    0.691983              0.221100         0.505398        -0.356225  0.403766                      0.467935 -0.056154
..        ...         ...       ...                ...        ...         ...                   ...              ...              ...       ...                           ...       ...
163  0.855910    3.020809  0.333119           0.366208  -0.301825   -1.425598              1.273133        -0.934022         1.155010 -1.415537                     -1.202294 -0.040447
164  0.463863    1.431116  0.456842           1.170218   0.257145   -1.283068              0.544803        -0.271131         0.976166 -1.142641                     -1.454934 -0.009032
165  0.299456    1.769154 -0.450459           0.205406   1.694496   -1.344152              0.544803        -0.384770         2.272788 -1.642949                     -1.454934  0.257995
166  0.172989    0.225141  0.003191           0.205406   1.694496   -1.354333              1.354059        -0.176433         1.870388 -1.597467                     -1.370721  0.273702
167  1.387072    1.604703  1.529107           1.652624  -0.221972   -1.272888              1.596836        -0.384770         1.825677 -1.551984                     -1.398792 -0.605914
```

Рисунок 3.2.6 – Предобработанные данные

На данном этапе полученный очищенный набор признаков может использоваться в алгоритмах кластеризации.

4 ПРАКТИЧЕСКАЯ ЧАСТЬ

Описание процесса предобработки данных содержится в разделе 3.3.

Оптимальное число кластеров определено с помощью метода локтя и силуэтного анализа.

4.1 Метод локтя

Метод локтя реализован как метод экземпляра класса DatasetManager.

Заданный диапазон числа кластеров: [1, 10].

На Рисунке 4.1.1 представлен график зависимости суммы внутрикластерных квадратов расстояний от количества кластеров k.

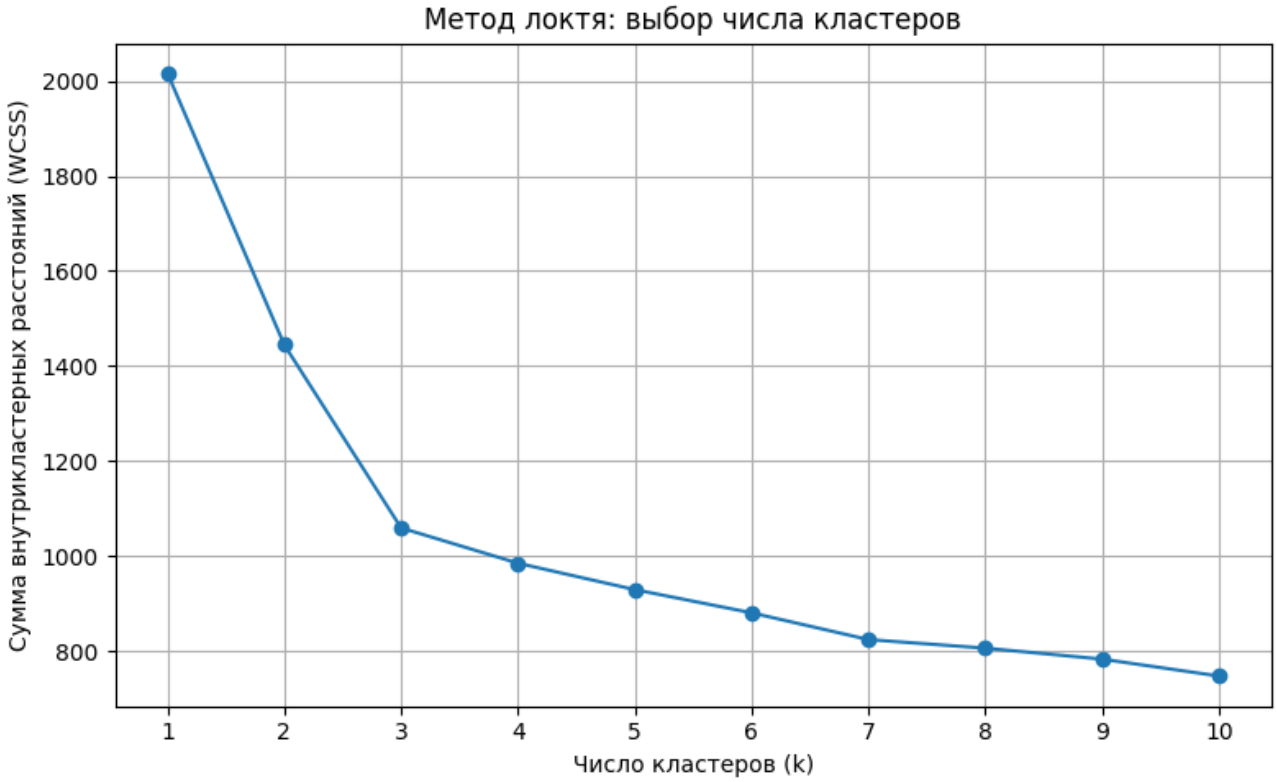

Рисунок 4.1.1 – График зависимости суммы внутрикластерных квадратов расстояний от количества кластеров k

По графику видно, что при k=3 кривая резко замедляет спад, формируя характерный «локоть». Это указывает на то, что увеличение числа кластеров свыше 3 не приводит к значимому улучшению компактности кластеров. Таким образом, оптимальное количество кластеров для данного набора данных — 3,

что соответствует исходному числу классов в датасете Wine. Это подтверждает, что алгоритм k-means способен выделить естественную структуру данных, связанную с тремя сортами вин.

4.2 Силуэтный анализ

Силуэтная диаграмма для двух кластеров представлена на Рисунке 4.2.1.

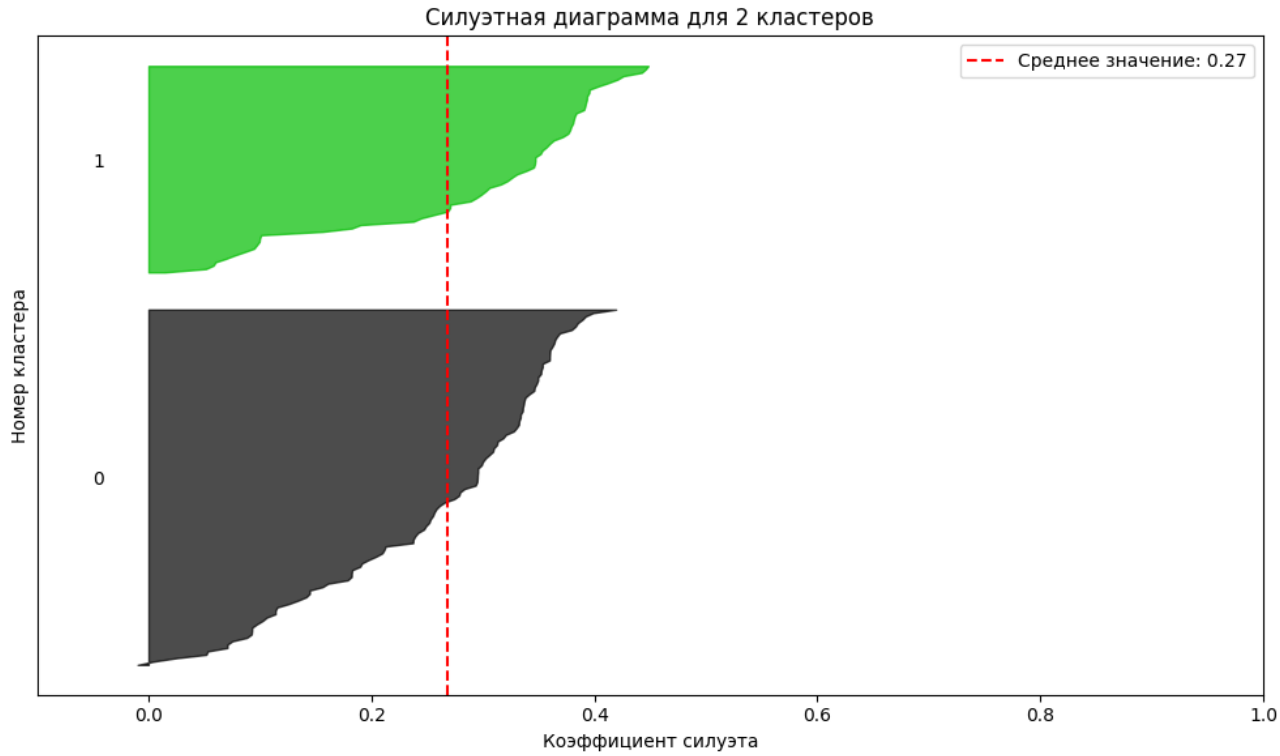

Рисунок 4.2.1 – Силуэтная диаграмма для двух кластеров

Средний коэффициент силуэта составляет 0.27, что говорит о допустимом, но не идеальном качестве кластеризации. Два кластера дают общее разделение, но структура данных предполагает наличие более тонкой кластерной структуры.

Силуэтная диаграмма для трех кластеров представлена на Рисунке 4.2.2.

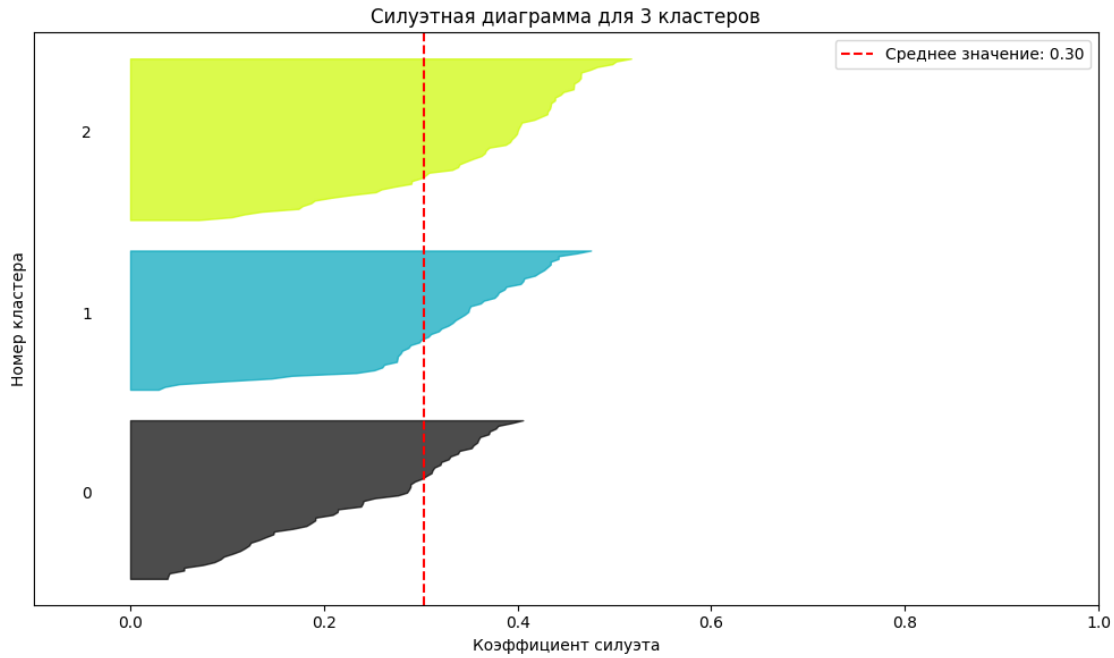

Рисунок 4.2.2 - Силуэтная диаграмма для трех кластеров

Средний коэффициент силуэта увеличивается до 0.30, что свидетельствует о лучшем качестве кластеризации по сравнению с вариантом из двух кластеров. При трёх кластерах достигается наилучший баланс между внутрикластерной плотностью и межкластерной отделённостью. Это подтверждает вывод, сделанный на основе метода локтя.

Силуэтная диаграмма для четырех кластеров представлена на Рисунке 4.2.3.

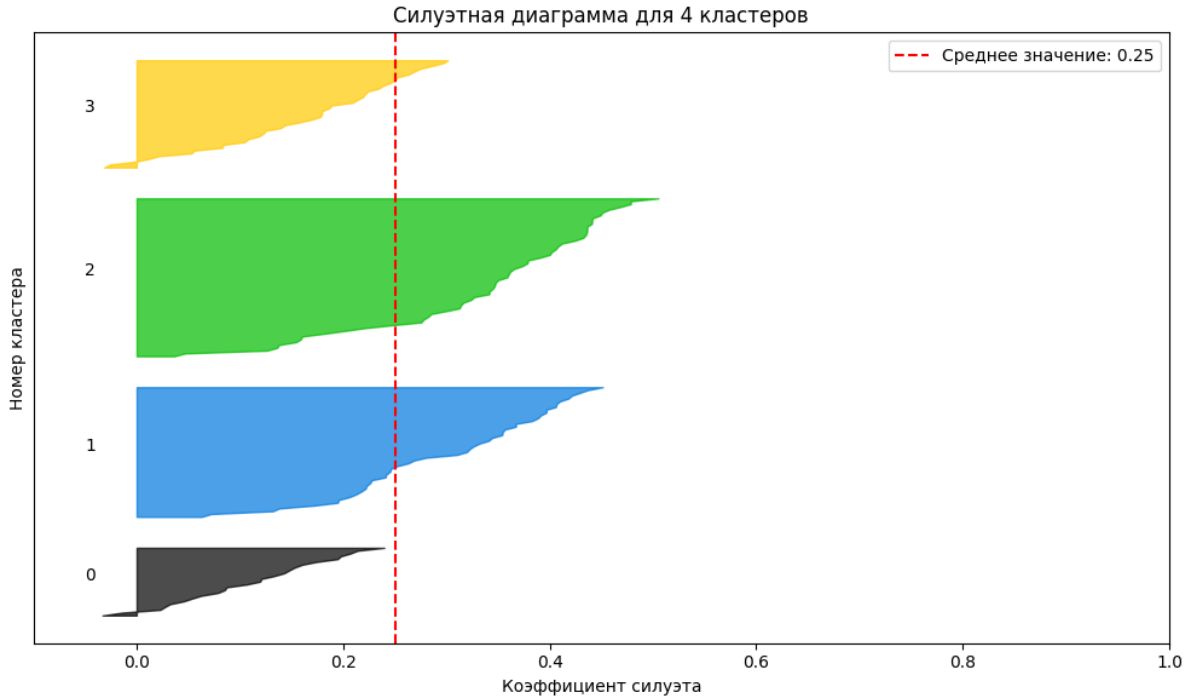

Рисунок 4.2.3 - Силуэтная диаграмма для четырех кластеров

Средний коэффициент силуэта незначительно снижается. Несложно увидеть, что 0-й кластер имеет силуэт, в котором коэффициент силуэта ни для одного объекта не превышает среднее значение коэффициента силуэта, равное 0.25.

4.3 Алгоритм k-means

Алгоритм KMeans применён к масштабированным данным с числом кластеров k=3. Кластеризация производилась после удаления высоко коррелированного признака total_phenols. В результате каждый объект отнесён к одному из трёх кластеров, что соответствует предполагаемому числу сортов вина. Содержание файла KMeans.py представлено в Приложении Б.

Для оценки результата построена визуализация кластеров в пространстве первых двух главных компонент, на которой видно чёткое разделение объектов на 3 группы (Рисунок 4.3.1).

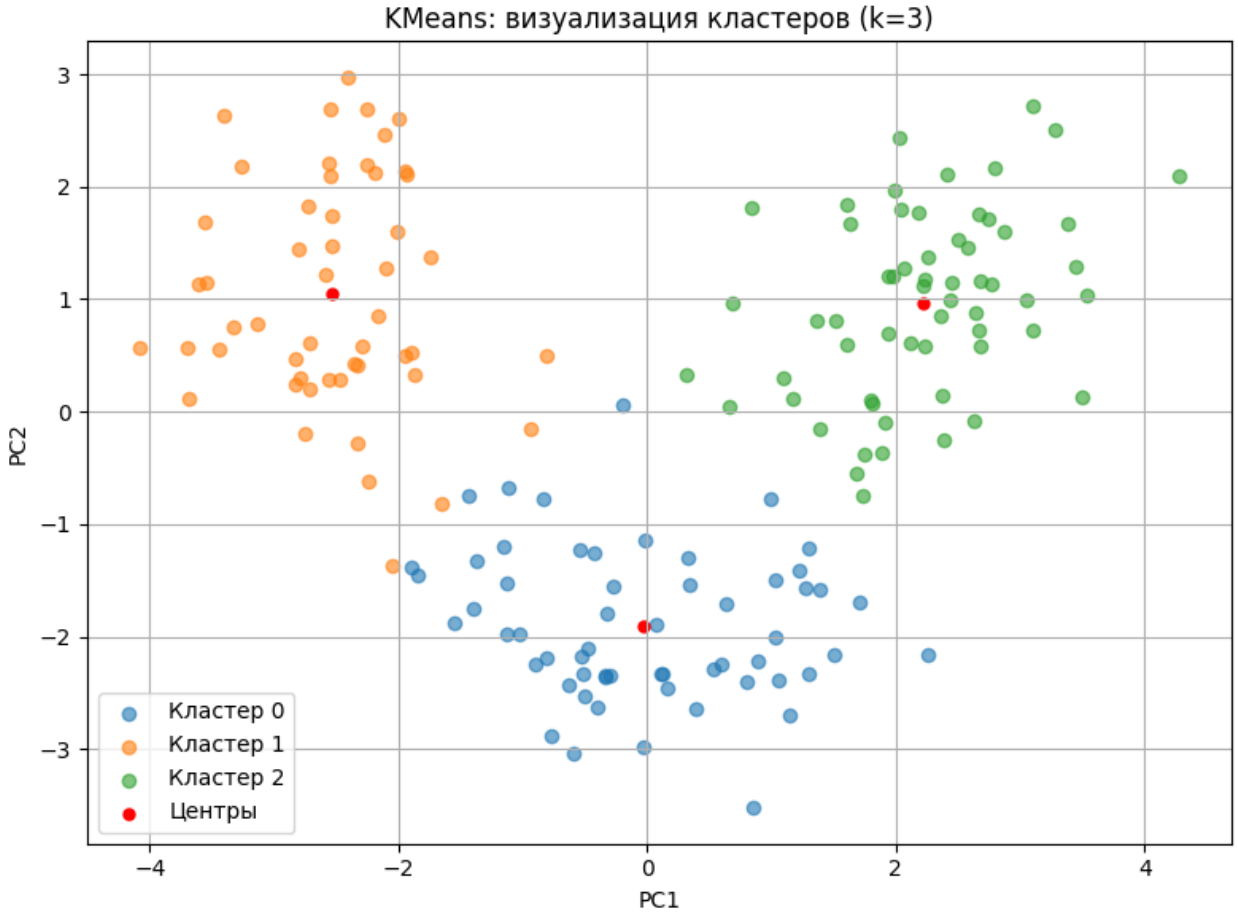

Рисунок 4.3.1 – Визуализация кластеров

Для более полной оценки рассчитаны внутренние и внешние метрики кластеризации. Внутренние метрики описаны в Таблице 4.3.1.

Таблица 4.3.1 – Внутренние метрики

Метрика	Значение	Интерпретация
Silhouette Score	0.304	Среднее качество кластеризации. Значение от 0.3 до 0.5 указывает на частичное перекрытие между кластерами и наличие объектов, которые могут быть отнесены к другому кластеру
Calinski-Harabasz Index	74.48	Умеренное значение индекса. Чем выше — тем чётче различие между кластерами. Показатель ниже 100 характерен для данных, где кластеры частично пересекаются
Davies-Bouldin Index	1.291	Значение выше 1 говорит о том, что межкластерное расстояние не слишком велико по сравнению с внутрикластерной плотностью

Кластеры в целом различимы, но присутствуют пограничные или перекрывающиеся группы объектов. Это может быть связано с наличием похожих по характеристикам наблюдений.

Внешние метрики описаны в Таблице 4.3.2.

Таблица 4.3.2 – Внешние метрики

Метрика	Значение	Интерпретация
Adjusted Rand Index (ARI)	0.913	Очень высокая согласованность кластеризации с реальными метками
Rand Index (RI)	0.962	96.2% пар объектов классифицированы одинаково, как в эталоне

Продолжение Таблицы 4.3.2

Fowlkes-Mallows Index	0.942	Сильное согласование кластеров с истинными метками. Показывает высокий баланс между точностью и полнотой
Mutual Information	0.978	Почти полная информация об истинных метках содержится в кластерах
Homogeneity	0.896	Почти все объекты в одном кластере принадлежат к одному классу
Completeness	0.892	Почти все объекты одного класса собраны в один кластер
V-measure	0.894	Гармоничное сочетание однородности и полноты

Кластеры почти идеально соответствуют реальным классам. Несмотря на средние внутренние метрики, модель правильно разделила данные по классам, что говорит об успешной кластеризации относительно эталона.

Для более глубокого понимания принципов работы алгоритма K-Means была реализована его собственная (самописная) версия, без использования готовых реализаций из библиотек машинного обучения.

Самописный алгоритм KMeans написан в файле KMeans_custom.py, содержание которого представлено в Приложении В.

Начальные центры выбираются случайно из точек исходного набора данных. Для воспроизводимости используется параметр random_state. На каждой итерации для каждой точки рассчитываются расстояния до всех центров кластеров (с помощью евклидовой метрики). Точка назначается в ближайший кластер, затем для каждого кластера заново вычисляется центр как среднее всех точек, отнесённых к нему.

Визуализация кластеров представлена на Рисунке 4.3.2.

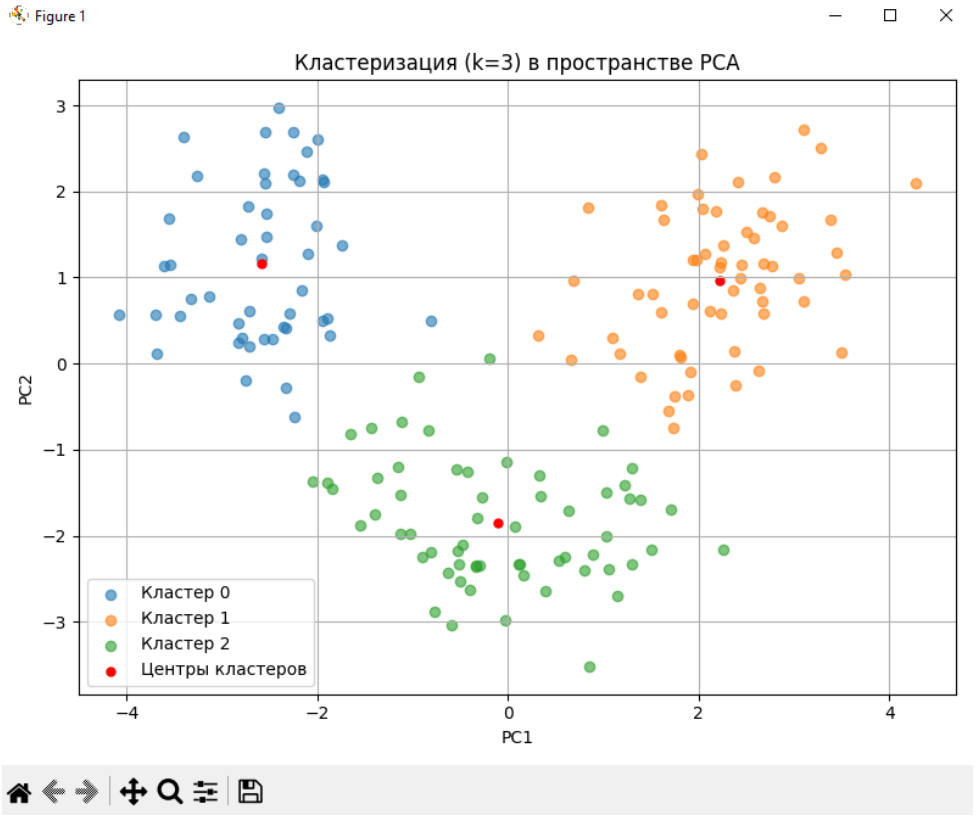

Рисунок 4.3.2 – Визуализация кластеров (самописная реализация алгоритма)

Значения внутренних и внешних представлены на Рисунке 4.3.3.

```
Silhouette Score: 0.303
Calinski-Harabasz Index: 74.319
Davies-Bouldin Index: 1.300
Adjusted Rand Index: 0.963
Rand Index: 0.984
Fowlkes-Mallows Index: 0.976
Mutual Information: 1.032
Homogeneity: 0.946
Completeness: 0.944
V-measure: 0.945
```

Рисунок 4.3.3 – Внутренние и внешние метрики для самописной реализации алгоритма

Полученные значения оказались сопоставимы с результатами библиотечной реализации, что подтверждает корректность собственной реализации алгоритма.

4.4 Алгоритм DBSCAN

Для реализации алгоритма DBSCAN использовалась библиотека scikit-learn. Алгоритм реализован в файле DBSCAN.py, содержание которого представлено в Приложении Г.

Основными параметрами алгоритма являются радиус окрестности ε и минимальное количество точек MinPts. Подбор параметров осуществлялся экспериментальным путем. Оптимальными оказались ε = 2.405 и min_samples = 15.

Для визуализации использовалось пространство первых двух главных компонент (PCA). На Рисунке 4.4.1 видно, что алгоритм выделил три плотных кластера, соответствующих сортам вин, а также отметил часть точек как шум (синим цветом).

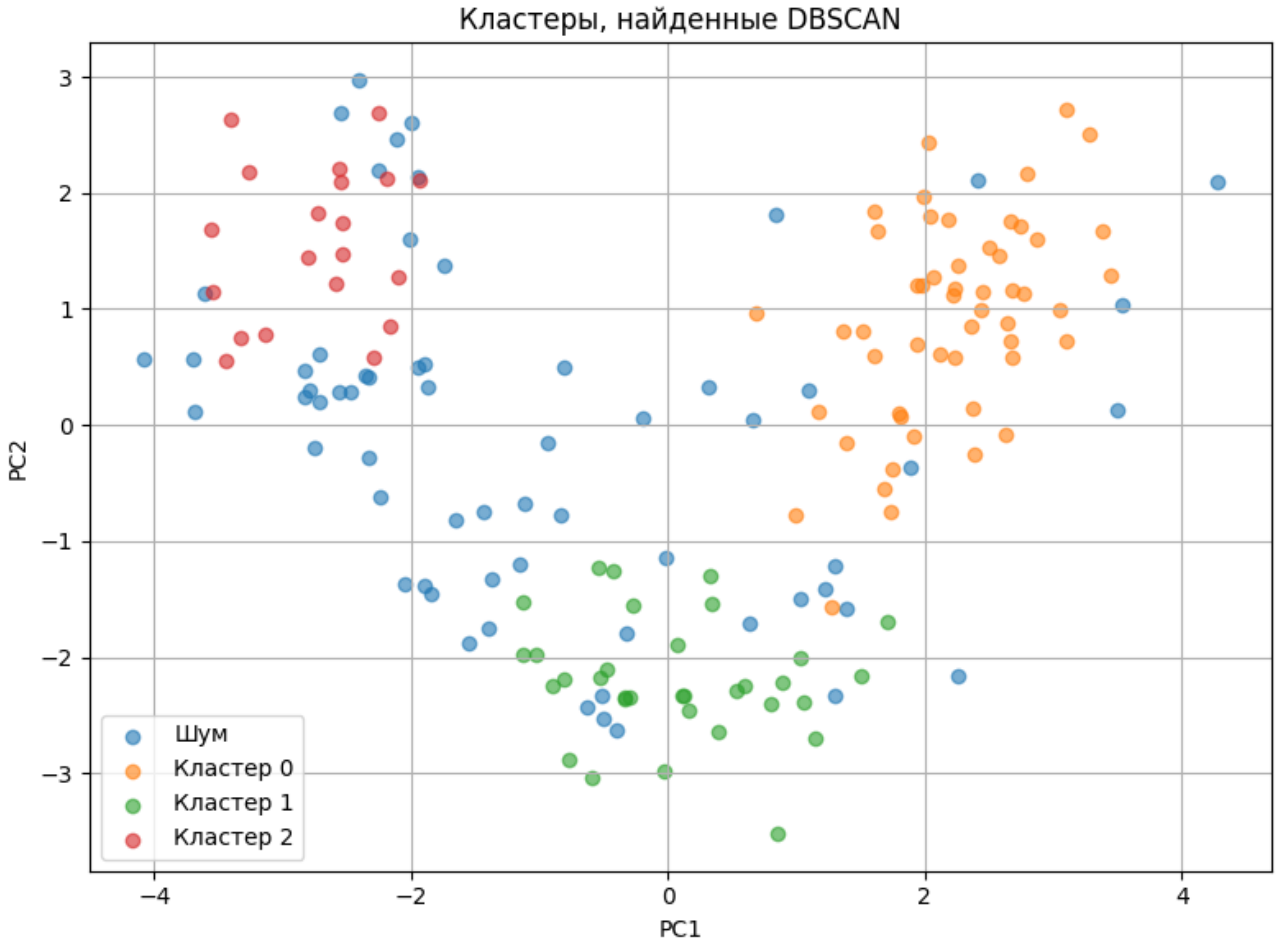

Рисунок 4.4.1 – Результат работы алгоритма DBSCAN

Для оценки качества кластеризации методом DBSCAN рассчитаны как внутренние, так и внешние метрики. Полученные результаты представлены на Рисунке 4.4.2.

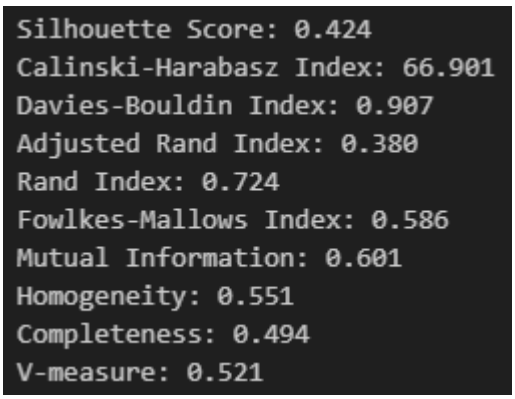

Рисунок 4.4.2 – Метрики качества кластеризации алгоритмом DBSCAN

Метод DBSCAN показал умеренное качество кластеризации. Его преимущество заключается в способности выявлять шум (выбросы), что невозможно в алгоритме k-means. Однако из-за плотностного подхода он чувствителен к параметрам ε и min_samples, и при неудачном подборе параметров может либо объединить несколько кластеров, либо пометить значительную часть точек как шум. Несмотря на это, DBSCAN сумел выделить осмысленную структуру в данных, приближенную к реальным классам.

ЗАКЛЮЧЕНИЕ

В ходе работы проведена комплексная обработка и кластерный анализ набора данных, содержащего химические характеристики вин. На первом этапе выполнены предварительные преобразования: удаление дубликатов и выбросов, масштабирование признаков, а также визуальный и статистический анализ данных. Кроме того, проведена оценка взаимной корреляции признаков, на основе которой один из признаков был исключён как избыточный.

Для решения задачи кластеризации были реализованы два подхода: алгоритм K-Means, как классический метод, ориентированный на разделение точек вокруг центров масс кластеров и алгоритм DBSCAN, основанный на плотностной структуре данных и способный выделять выбросы.

Кроме визуализации, для оценки качества кластеризации применялись как внутренние метрики, так и внешние, основанные на сравнении с реальными метками классов.

Результаты показали, что K-Means обеспечивает более высокое качество кластеризации в условиях хорошо разделённых классов, но не может обнаруживать выбросы. В то же время, DBSCAN способен выявлять шум и кластеры произвольной формы, однако требует тщательной настройки параметров ε и min_samples.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
2. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
3. Кластеризация в ML: от теоретических основ популярных алгоритмов к их реализации с нуля на Python [Электронный ресурс]: Habr. URL: https://habr.com/ru/articles/798331/#dbscan (Дата обращения: 15.05.2025).
4. Машинное обучение: Кластеризация методом K-means. Теория и реализация. С нуля [Электронный ресурс]: Habr. URL: https://habr.com/ru/articles/868542/ (Дата обращения: 15.05.2025).

ПРИЛОЖЕНИЯ

Приложение А — Файл dataset_manager.py для предобработки и анализа датасета.

Приложение Б — Файл KMeans.py с использованием готовой реализации алгоритма KMeans.

Приложение В — Файл KMeans_custom.py с собственной реализацией алгоритма KMeans.

Приложение Г — Файл DBSCAN.py с использованием готовой реализации алгоритма DBSCAN.

Приложение А

Файл dataset_manager.py для предобработки и анализа датасета

Листинг А – Содержание файла dataset_manager.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns
from pandas.plotting import scatter_matrix
from sklearn.datasets import load_wine
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_samples, silhouette_score
from typing import Optional, Dict, Tuple, List
from pandas import DataFrame, Series

class DatasetManager:
    def __init__(
        self,
        source: str = "sklearn",
        csv_path: Optional[str] = None,
    ) -> None:
        """
        Инициализирует менеджер датасета для загрузки, анализа, предобработки и
визуализации.

        Параметры:
            source (str): Источник данных.
                - "sklearn": загружаем встроенный датасет Wine из sklearn.
                - "csv": читаем CSV-файл по пути csv_path.
            csv_path (Optional[str]): Путь к CSV-файлу при source="csv".
                Если source="sklearn", игнорируется.
        """
        self.source: str = source
        self.csv_path: Optional[str] = csv_path
        self.df: Optional[DataFrame] = None
        self.features: Optional[DataFrame] = None
        self.target: Optional[Series] = None
        self.scaled_features: Optional[DataFrame] = None
        self.stats: Dict[str, DataFrame] = {}

        self._load_data()
        self._extract_features_target()

    def _load_data(self) -> None:
        """
        Загружает исходный датасет в self.df.

        При source="sklearn" загружается Wine-датасет из sklearn.
        При source="csv" загружается CSV-файл по пути csv_path.

        Выбрасывает:
            ValueError: если source="csv" и csv_path не указан или source не
равен "sklearn"/"csv".
        """
        if self.source == "sklearn":
            raw = load_wine(as_frame=True)
            df0 = raw.frame.copy()
            self.df = df0
```

Продолжение Листинга А

```
        elif self.source == "csv":
            if self.csv_path is None:
                raise ValueError("При source='csv' необходимо указать путь
csv_path")
            self.df = pd.read_csv(self.csv_path)
        else:
            raise ValueError("source должен быть 'sklearn' или 'csv'")

        print(
            f"Данные загружены: {self.df.shape[0]} строк, {self.df.shape[1]}
столбцов"
        )

    def _extract_features_target(self) -> None:
        """
        Разделяет DataFrame на признаки и метку (если столбец 'target'
присутствует).

        После выполнения:
            - self.features будет содержать DataFrame только с признаками.
            - self.target будет содержать Series с метками классов (или None,
если 'target' отсутствует).
        """
        if self.df is None:
            raise RuntimeError("Данные не загружены. Сначала вызовите
_load_data().")

        if "target" in self.df.columns:
            self.target = self.df["target"].copy()
            self.features = self.df.drop(columns=["target"]).copy()
        else:
            self.target = None
            self.features = self.df.copy()

    def compute_basic_statistics(self) -> Dict[str, DataFrame]:
        """
        Вычисляет базовые статистики по признакам и сохраняет их в self.stats.

        Сохраняются:
            - "describe": описательные статистики (mean, std, min, max,
квартили) для каждого признака.
            - "correlation_matrix": матрица корреляций между признаками.
            - "class_distribution": распределение по классам (если есть
self.target).

        Возвращает:
            Dict[str, DataFrame]: Словарь с DataFrame-статистиками.
        """
        if self.features is None:
            raise RuntimeError(
                "Признаки не выделены. Сначала вызовите
_extract_features_target()."
            )

        desc = self.features.describe().T
        self.stats["describe"] = desc

        corr = self.features.corr()
        self.stats["correlation_matrix"] = corr

        if self.target is not None:
```

Продолжение Листинга А

```
            class_counts: Series = self.target.value_counts().sort_index()
            self.stats["class_distribution"] =
class_counts.to_frame(name="count")

        return self.stats

    def preprocess(
        self,
        drop_duplicates: bool = True,
        drop_outliers: bool = True,
        z_thresh: float = 3.0,
    ) -> None:
        """
        Полная предобработка данных:
            1. Удаление дубликатов.
            2. Удаление выбросов по Z-оценке (если drop_outliers=True).
            3. Масштабирование признаков StandardScaler.

        Параметры:
            drop_duplicates (bool): Удалять ли полные дубликаты строк
(True/False).
            drop_outliers (bool): Удалять ли выбросы по Z-оценке (True/False).
            z_thresh (float): Порог Z-оценки; объекты, у которых хотя бы один
признак
                              имеет |z_score| > z_thresh, считаются выбросами.

        После выполнения:
            - self.df обновляется без дубликатов и выбросов.
            - self.features обновляются (признаки из очищенного DataFrame).
            - self.target обновляется (метки из очищенного DataFrame).
            - self.scaled_features заполняется DataFrame-ом масштабированных
признаков.
        """
        if self.df is None:
            raise RuntimeError("Данные не загружены. Сначала вызовите
_load_data().")

        df_proc: DataFrame = self.df.copy()

        if drop_duplicates:
            before = df_proc.shape[0]
            df_proc = df_proc.drop_duplicates().reset_index(drop=True)
            after = df_proc.shape[0]
            print(f"Удалено дубликатов: {before - after}")

        if drop_outliers:
            df_no_target = df_proc.drop(columns=["target"], errors="ignore")
            means = df_no_target.mean()
            stds = df_no_target.std(ddof=0)
            z_scores = (df_no_target - means) / stds
            mask = (z_scores.abs() <= z_thresh).all(axis=1)
            before_out = df_proc.shape[0]
            df_proc = df_proc[mask].reset_index(drop=True)
            after_out = df_proc.shape[0]
            print(f"Удалено выбросов: {before_out - after_out}")

        scaler = StandardScaler()
        feat: DataFrame = df_proc.drop(columns=["target"], errors="ignore")
        scaled_array = scaler.fit_transform(feat)
        scaled_df = pd.DataFrame(scaled_array, columns=feat.columns,
index=feat.index)
```

Продолжение Листинга А

```
        self.df = df_proc
        if "target" in df_proc.columns:
            self.target = df_proc["target"].copy()
            self.features = df_proc.drop(columns=["target"]).copy()
        else:
            self.target = None
            self.features = df_proc.copy()

        self.scaled_features = scaled_df
        print(
            "Предобработка завершена: дубликаты и выбросы (если указано)
удалены, признаки масштабированы."
        )

    def visualize_distributions(self, figsize: Tuple[int, int] = (12, 8)) ->
None:
        """
        Строит гистограммы распределений каждого признака (до масштабирования).

        Параметры:
            figsize (Tuple[int, int]): Размер фигуры (ширина, высота) в дюймах.
        """
        if self.features is None:
            raise RuntimeError(
                "Признаки не выделены. Сначала вызовите
_extract_features_target()."
            )

        n = len(self.features.columns)
        cols = 3
        rows = (n + cols - 1) // cols
        fig, axes = plt.subplots(rows, cols, figsize=figsize)
        axes = axes.flatten()

        for i, col in enumerate(self.features.columns):
            axes[i].hist(self.features[col], bins=15, edgecolor="black")
            axes[i].set_title(col)
        for j in range(n, len(axes)):
            axes[j].axis("off")

        plt.tight_layout()
        plt.show()

    def visualize_scatter_matrix(
        self,
        with_target: bool = True,
        figsize: Tuple[int, int] = (10, 10),
    ) -> None:
        """
        Строит матрицу рассеяния (pairplot) для первых 5–7 признаков.

        Параметры:
            with_target (bool): Если True и self.target определён, раскрашивает
точки по классам.
            figsize (Tuple[int, int]): Размер фигуры (ширина, высота) в дюймах.
        """
        if self.features is None:
            raise RuntimeError(
                "Признаки не выделены. Сначала вызовите
_extract_features_target()."
```

Продолжение Листинга А

```
            )

        num_to_plot = min(7, len(self.features.columns))
        df_plot = self.features.iloc[:, :num_to_plot].copy()

        if with_target and self.target is not None:
            df_plot["target"] = self.target.values
            colors = {0: "red", 1: "green", 2: "blue"}
            scatter_matrix(
                df_plot,
                figsize=figsize,
                diagonal="hist",
                color=df_plot["target"].map(colors),
                alpha=0.5,
            )
        else:
            scatter_matrix(df_plot, figsize=figsize, diagonal="hist", alpha=0.5)

        plt.suptitle("Матрица рассеяния признаков", y=1.02)
        plt.show()

    def visualize_correlation_heatmap(self, figsize: Tuple[int, int] = (12, 8))
-> None:
        """
        Строит «приятную» тепловую карту корреляций между признаками с помощью
seaborn.

        Параметры:
            figsize (Tuple[int, int]): Размер фигуры (ширина, высота) в дюймах.
        """
        if self.features is None:
            raise RuntimeError(
                "Признаки не выделены. Сначала вызовите
_extract_features_target()."
            )

        plt.figure(figsize=figsize)
        sns.heatmap(
            self.features.corr(),
            annot=True,
            cmap="coolwarm",
            linewidths=0.5,
            square=True,
            cbar_kws={"shrink": 0.7},
        )
        plt.title("Матрица корреляции признаков")
        plt.xticks(rotation=45, ha="right")
        plt.yticks(rotation=0)
        plt.tight_layout()
        plt.show()

    def get_preprocessed_data(self) -> Tuple[DataFrame, Optional[Series]]:
        """
        Возвращает масштабированные признаки и метки (если есть) для дальнейшего
анализа/кластеризации.

        Возвращает:
            Tuple[DataFrame, Optional[Series]]:
                - DataFrame: self.scaled_features (масштабированные признаки).
                - Series или None: self.target (метки классов, если были
изначально).
```

Продолжение Листинга А

```
        Выбрасывает:
            RuntimeError: если self.scaled_features ещё не вычислены (не вызван
preprocess()).
        """
        if self.scaled_features is None:
            raise RuntimeError(
                "Данные ещё не предобработаны. Сначала вызовите preprocess()."
            )
        return self.scaled_features, self.target

    def visualize_class_distribution(
        self,
        figsize: Tuple[int, int] = (8, 8),
        title: str = "Распределение по классам",
        colors: Optional[List[str]] = None,
        autopct: str = "%1.1f%%",
        startangle: int = 90,
    ) -> None:
        """
        Строит круговую диаграмму распределения объектов по классам.

        Параметры:
            figsize (Tuple[int, int]): Размер фигуры (ширина, высота) в дюймах.
            title (str): Заголовок диаграммы.
            colors (Optional[List[str]]): Список цветов для секторов.
            autopct (str): Формат отображения процентных значений.
            startangle (int): Угол начала первой секции.
        """
        if "class_distribution" not in self.stats:
            raise RuntimeError(
                "Распределение по классам не вычислено. Вызовите
compute_basic_statistics()."
            )

        class_dist = self.stats["class_distribution"]
        labels = class_dist.index.astype(str).tolist()
        sizes = class_dist["count"].tolist()

        if not colors:
            colors = ["#ff9999", "#66b3ff", "#99ff99", "#ffcc99"]

        plt.figure(figsize=figsize)
        plt.pie(
            sizes,
            labels=labels,
            colors=colors,
            autopct=autopct,
            startangle=startangle,
            textprops={"fontsize": 12},
            wedgeprops={"edgecolor": "black", "linewidth": 0.5},
        )
        plt.title(title, fontsize=14, pad=20)
        plt.axis("equal")
        plt.show()

    def remove_feature(self, feature_name: str) -> None:
        """
        Удаляет признак из текущего набора данных по его имени.

        Параметры:
```

Продолжение Листинга А

```
            feature_name (str): Название удаляемого признака.

        Исключения:
            ValueError: если feature_name не является строкой.
            KeyError: если признака с таким именем нет в self.features.
            RuntimeError: если self.features ещё не инициализирован (нет
данных).
        """
        if self.features is None:
            raise RuntimeError(
                "Набор признаков пуст. Сначала выполните загрузку данных и метод
_extract_features_target()."
            )

        if not isinstance(feature_name, str):
            raise ValueError(
                f"Имя признака должно быть строкой, получено
{type(feature_name).__name__}"
            )

        if feature_name not in self.features.columns:
            raise KeyError(
                f"Признак '{feature_name}' отсутствует в текущем наборе
признаков."
            )

        self.features.drop(columns=[feature_name], inplace=True)

        if self.df is not None and feature_name in self.df.columns:
            self.df.drop(columns=[feature_name], inplace=True)

        if (
            self.scaled_features is not None
            and feature_name in self.scaled_features.columns
        ):
            self.scaled_features.drop(columns=[feature_name], inplace=True)

        print(f"Признак '{feature_name}' успешно удалён из набора данных.")

    def visualize_elbow_method(self, k_range: range = range(1, 11)) -> None:
        """
        Реализует метод локтя для определения оптимального числа кластеров.

        Для каждого k в заданном диапазоне вычисляется WCSS (within-cluster sum
of squares).
        Строится график зависимости WCSS от количества кластеров k.

        Параметры:
            k_range (range): Диапазон значений k (число кластеров) для анализа.

        Исключения:
            RuntimeError: если данные ещё не были предобработаны.
        """
        if self.scaled_features is None:
            raise RuntimeError(
                "Сначала вызовите preprocess(), чтобы получить масштабированные
признаки."
            )

        wcss = []
```

Продолжение Листинга А

```
        for k in k_range:
            kmeans = KMeans(n_clusters=k, random_state=42, n_init="auto")
            kmeans.fit(self.scaled_features)
            wcss.append(kmeans.inertia_)

        plt.figure(figsize=(8, 5))
        plt.plot(list(k_range), wcss, marker="o")
        plt.title("Метод локтя: выбор числа кластеров")
        plt.xlabel("Число кластеров (k)")
        plt.ylabel("Сумма внутрикластерных расстояний (WCSS)")
        plt.xticks(list(k_range))
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    def visualize_silhouette_analysis(self, n_clusters: int = 3) -> None:
        """
        Строит силуэтную диаграмму для оценки качества кластеризации методом k-
средних.

        Параметры:
            n_clusters (int): Число кластеров для KMeans.

        Исключения:
            RuntimeError: если не были предобработаны признаки.
        """
        if self.scaled_features is None:
            raise RuntimeError(
                "Сначала вызовите preprocess(), чтобы получить масштабированные
признаки."
            )

        X = self.scaled_features.values

        kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init="auto")
        cluster_labels = kmeans.fit_predict(X)

        silhouette_avg = silhouette_score(X, cluster_labels)
        sample_silhouette_values = silhouette_samples(X, cluster_labels)

        fig, ax1 = plt.subplots(figsize=(10, 6))
        y_lower = 10
        for i in range(n_clusters):
            ith_cluster_silhouette_values = sample_silhouette_values[
                cluster_labels == i
            ]
            ith_cluster_silhouette_values.sort()

            size_cluster_i = ith_cluster_silhouette_values.shape[0]
            y_upper = y_lower + size_cluster_i

            color = cm.nipy_spectral(float(i) / n_clusters)
            ax1.fill_betweenx(
                np.arange(y_lower, y_upper),
                0,
                ith_cluster_silhouette_values,
                facecolor=color,
                edgecolor=color,
                alpha=0.7,
            )
```

Окончание Листинга А

```
            ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
            y_lower = y_upper + 10

        ax1.set_title(f"Силуэтная диаграмма для {n_clusters} кластеров")
        ax1.set_xlabel("Коэффициент силуэта")
        ax1.set_ylabel("Номер кластера")

        ax1.axvline(
            x=silhouette_avg,
            color="red",
            linestyle="--",
            label=f"Среднее значение: {silhouette_avg:.2f}",
        )
        ax1.set_xlim([-0.1, 1.0])
        ax1.set_yticks([])
        ax1.legend()
        plt.tight_layout()
        plt.show()

if __name__ == "__main__":
    manager = DatasetManager(source="sklearn")
    stats = manager.compute_basic_statistics()

    manager.visualize_class_distribution(
        title="Распределение вин по классам",
        colors=["#ff9999", "#66b3ff", "#99ff99"],
        autopct="%1.1f%%",
    )

    print("Описание признаков:")
    print(stats["describe"])
    if "class_distribution" in stats:
        print("\nРаспределение по классам:")
        print(stats["class_distribution"])

    manager.visualize_distributions()
    manager.visualize_scatter_matrix()
    manager.visualize_correlation_heatmap()

    manager.preprocess()

    manager.remove_feature("total_phenols")
    manager.visualize_elbow_method()
    manager.visualize_silhouette_analysis()

    X_scaled, y = manager.get_preprocessed_data()
```

Приложение Б

Файл KMeans.py с использованием готовой реализации алгоритма KMeans

Листинг Б – Файл KMeans.py

```
from sklearn.metrics import (
    silhouette_score,
    calinski_harabasz_score,
    davies_bouldin_score,
    adjusted_rand_score,
    rand_score,
    fowlkes_mallows_score,
    mutual_info_score,
    homogeneity_score,
    completeness_score,
    v_measure_score,
)
from typing import Optional, Union
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
from dataset_manager import DatasetManager

class KMeansClustering:
    def __init__(self, X_scaled: pd.DataFrame, n_clusters: int = 3) -> None:
        """
        Инициализирует кластеризатор KMeans.

        Параметры:
            X_scaled (pd.DataFrame): Масштабированные данные.
            n_clusters (int): Число кластеров.
        """
        self.X = X_scaled
        self.n_clusters = n_clusters
        self.model: Optional[KMeans] = None
        self.labels: Optional[np.ndarray] = None
        self.pca_components: Optional[pd.DataFrame] = None

    def fit(self) -> None:
        """
        Обучает модель KMeans и сохраняет метки кластеров.
        """
        self.model = KMeans(n_clusters=self.n_clusters, random_state=42,
n_init="auto")
        self.labels = self.model.fit_predict(self.X)

    def visualize_clusters(self) -> None:
        """
        Визуализирует результат кластеризации в 2D (через PCA).
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")

        pca = PCA(n_components=2)
        components = pca.fit_transform(self.X)
        df_pca = pd.DataFrame(components, columns=["PC1", "PC2"])
        df_pca["cluster"] = self.labels
        self.pca_components = df_pca
```

Продолжение Листинга Б

```
        plt.figure(figsize=(8, 6))
        for label in np.unique(self.labels):
            subset = df_pca[df_pca["cluster"] == label]
            plt.scatter(subset["PC1"], subset["PC2"], label=f"Кластер {label}",
alpha=0.6)

        centers_2d = pca.transform(self.model.cluster_centers_)
        plt.scatter(
            centers_2d[:, 0], centers_2d[:, 1], c="red", s=25, label="Центры"
        )

        plt.title(f"KMeans: визуализация кластеров (k={self.n_clusters})")
        plt.xlabel("PC1")
        plt.ylabel("PC2")
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    def compute_silhouette(self) -> float:
        """
        Вычисляет среднее значение силуэт-метрики.

        Возвращает:
            float: Silhouette Score.
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return silhouette_score(self.X, self.labels)

    def compute_calinski_harabasz(self) -> float:
        """
        Вычисляет индекс Калински-Харабаза.

        Возвращает:
            float: Calinski-Harabasz Index.
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return calinski_harabasz_score(self.X, self.labels)

    def compute_davies_bouldin(self) -> float:
        """
        Вычисляет индекс Дэвиса-Болдина.

        Возвращает:
            float: Davies-Bouldin Index.
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return davies_bouldin_score(self.X, self.labels)

    def compute_adjusted_rand_index(self, y_true: Union[pd.Series, np.ndarray])
-> float:
        """
        Вычисляет Adjusted Rand Index (ARI).

        Параметры:
            y_true (Series | ndarray): Истинные метки.

        Возвращает:
```

Продолжение Листинга Б

```
            float: Adjusted Rand Index.
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return adjusted_rand_score(y_true, self.labels)

    def compute_rand_index(self, y_true: Union[pd.Series, np.ndarray]) -> float:
        """
        Вычисляет Rand Index.

        Параметры:
            y_true (Series | ndarray): Истинные метки.

        Возвращает:
            float: Rand Index.
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return rand_score(y_true, self.labels)

    def compute_fowlkes_mallows(self, y_true: Union[pd.Series, np.ndarray]) ->
float:
        """
        Вычисляет индекс Фаулкса-Мэллоуза.

        Параметры:
            y_true (Series | ndarray): Истинные метки.

        Возвращает:
            float: Fowlkes-Mallows Index.
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return fowlkes_mallows_score(y_true, self.labels)

    def compute_mutual_info(self, y_true: Union[pd.Series, np.ndarray]) ->
float:
        """
        Вычисляет взаимную информацию (MI).

        Параметры:
            y_true (Series | ndarray): Истинные метки.

        Возвращает:
            float: Mutual Information.
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return mutual_info_score(y_true, self.labels)

    def compute_homogeneity(self, y_true: Union[pd.Series, np.ndarray]) ->
float:
        """
        Вычисляет метрику однородности (Homogeneity).

        Параметры:
            y_true (Series | ndarray): Истинные метки.

        Возвращает:
            float: Homogeneity Score.
        """
```

Продолжение Листинга Б

```
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return homogeneity_score(y_true, self.labels)

    def compute_completeness(self, y_true: Union[pd.Series, np.ndarray]) ->
float:
        """
        Вычисляет метрику полноты (Completeness).

        Параметры:
            y_true (Series | ndarray): Истинные метки.

        Возвращает:
            float: Completeness Score.
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return completeness_score(y_true, self.labels)

    def compute_v_measure(self, y_true: Union[pd.Series, np.ndarray]) -> float:
        """
        Вычисляет V-меру (среднее между Homogeneity и Completeness).

        Параметры:
            y_true (Series | ndarray): Истинные метки.

        Возвращает:
            float: V-Measure Score.
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")
        return v_measure_score(y_true, self.labels)

manager = DatasetManager(source="sklearn")
manager.preprocess()
manager.remove_feature("total_phenols")
X_scaled, y_true = manager.get_preprocessed_data()

clusterer = KMeansClustering(X_scaled, n_clusters=3)
clusterer.fit()
clusterer.visualize_clusters()

# ВНУТРЕННИЕ МЕТРИКИ
silhouette = clusterer.compute_silhouette()
print(f"Silhouette Score: {silhouette:.3f}")

calinski = clusterer.compute_calinski_harabasz()
print(f"Calinski-Harabasz Index: {calinski:.2f}")

davies = clusterer.compute_davies_bouldin()
print(f"Davies-Bouldin Index: {davies:.3f}")

# ВНЕШНИЕ МЕТРИКИ
ari = clusterer.compute_adjusted_rand_index(y_true)
print(f"Adjusted Rand Index (ARI): {ari:.3f}")

ri = clusterer.compute_rand_index(y_true)
print(f"Rand Index (RI): {ri:.3f}")

fmi = clusterer.compute_fowlkes_mallows(y_true)
print(f"Fowlkes-Mallows Index: {fmi:.3f}")
```

Окончание Листинга Б

```
mi = clusterer.compute_mutual_info(y_true)
print(f"Mutual Information: {mi:.3f}")

homogeneity = clusterer.compute_homogeneity(y_true)
print(f"Homogeneity: {homogeneity:.3f}")

completeness = clusterer.compute_completeness(y_true)
print(f"Completeness: {completeness:.3f}")

v_measure = clusterer.compute_v_measure(y_true)
print(f"V-measure: {v_measure:.3f}")
```

Приложение В

Файл KMeans_custom.py с собственной реализацией алгоритма KMeans

Листинг В – Файл KMeans_custom.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import (
    silhouette_score,
    calinski_harabasz_score,
    davies_bouldin_score,
    adjusted_rand_score,
    rand_score,
    fowlkes_mallows_score,
    mutual_info_score,
    homogeneity_score,
    completeness_score,
    v_measure_score,
)
from dataset_manager import DatasetManager

class CustomKMeans:
    def __init__(
        self,
        n_clusters: int = 3,
        max_iter: int = 300,
        tol: float = 1e-4,
        random_state: int = 42,
    ):
        """
        Собственная реализация алгоритма K-Means.

        Параметры:
            n_clusters (int): Количество кластеров.
            max_iter (int): Максимальное число итераций.
            tol (float): Порог сходимости.
            random_state (int): Фиксация генератора случайных чисел.
        """
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.tol = tol
        self.random_state = random_state

        self.centroids = None
        self.labels = None

    def fit(self, X: pd.DataFrame) -> None:
        """
        Обучает модель на данных X.

        Параметры:
            X (pd.DataFrame): Массив признаков.
        """
        np.random.seed(self.random_state)
        self.X = X.to_numpy()
        n_samples, n_features = self.X.shape

        random_idx = np.random.choice(n_samples, self.n_clusters, replace=False)
        self.centroids = self.X[random_idx]
```

Продолжение Листинга В

```
        for iteration in range(self.max_iter):
            distances = self._euclidean_distance(self.X, self.centroids)
            self.labels = np.argmin(distances, axis=1)

            new_centroids = np.array(
                [
                    self.X[self.labels == i].mean(axis=0)
                    if np.any(self.labels == i)
                    else self.centroids[i]
                    for i in range(self.n_clusters)
                ]
            )

            shift = np.linalg.norm(self.centroids - new_centroids)
            if shift < self.tol:
                break
            self.centroids = new_centroids

    def predict(self, X: pd.DataFrame) -> np.ndarray:
        """
        Присваивает метки кластерам новым объектам.

        Параметры:
            X (pd.DataFrame): Признаки новых объектов.

        Возвращает:
            np.ndarray: Массив меток.
        """
        distances = self._euclidean_distance(X.to_numpy(), self.centroids)
        return np.argmin(distances, axis=1)

    def _euclidean_distance(self, A: np.ndarray, B: np.ndarray) -> np.ndarray:
        """
        Вычисляет матрицу евклидовых расстояний между двумя матрицами точек.
        """
        return np.linalg.norm(A[:, np.newaxis] - B, axis=2)

    def get_labels(self) -> np.ndarray:
        """
        Возвращает метки кластеров после обучения.
        """
        return self.labels

    def get_centroids(self) -> np.ndarray:
        """
        Возвращает центры кластеров.
        """
        return self.centroids

    def compute_all_metrics(self, y_true: np.ndarray) -> dict:
        """
        Вычисляет внутренние и внешние метрики кластеризации.

        Параметры:
            y_true (np.ndarray): Истинные метки классов.

        Возвращает:
            dict: Метрики в виде словаря.
        """
        return {
```

Продолжение Листинга В

```
            "Silhouette Score": silhouette_score(self.X, self.labels),
            "Calinski-Harabasz Index": calinski_harabasz_score(self.X,
self.labels),
            "Davies-Bouldin Index": davies_bouldin_score(self.X, self.labels),
            "Adjusted Rand Index": adjusted_rand_score(y_true, self.labels),
            "Rand Index": rand_score(y_true, self.labels),
            "Fowlkes-Mallows Index": fowlkes_mallows_score(y_true, self.labels),
            "Mutual Information": mutual_info_score(y_true, self.labels),
            "Homogeneity": homogeneity_score(y_true, self.labels),
            "Completeness": completeness_score(y_true, self.labels),
            "V-measure": v_measure_score(y_true, self.labels),
        }

    def visualize_clusters(self) -> None:
        """
        Визуализирует результат кластеризации в пространстве двух главных
компонент (PCA).
        """
        if self.labels is None or self.centroids is None:
            raise RuntimeError("Сначала вызовите fit(), чтобы обучить модель.")

        pca = PCA(n_components=2)
        X_pca = pca.fit_transform(self.X)
        centroids_pca = pca.transform(self.centroids)

        df_plot = pd.DataFrame(X_pca, columns=["PC1", "PC2"])
        df_plot["cluster"] = self.labels

        plt.figure(figsize=(8, 6))
        for cluster_id in np.unique(self.labels):
            cluster_points = df_plot[df_plot["cluster"] == cluster_id]
            plt.scatter(cluster_points["PC1"], cluster_points["PC2"],
label=f"Кластер {cluster_id}", alpha=0.6)

        plt.scatter(
            centroids_pca[:, 0],
            centroids_pca[:, 1],
            c="red",
            s=25,
            label="Центры кластеров"
        )

        plt.title(f"Кластеризация (k={self.n_clusters}) в пространстве PCA")
        plt.xlabel("PC1")
        plt.ylabel("PC2")
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        plt.show()

manager = DatasetManager(source="sklearn")
manager.preprocess()
manager.remove_feature("total_phenols")
X_scaled, y_true = manager.get_preprocessed_data()

model = CustomKMeans(n_clusters=3)
model.fit(X_scaled)
model.visualize_clusters()

labels = model.get_labels()
centroids = model.get_centroids()
```

Окончание Листинга В

```
metrics = model.compute_all_metrics(y_true)
for name, value in metrics.items():
    print(f"{name}: {value:.3f}")
```

Приложение Г

Файл DBSCAN.py с использованием готовой реализации алгоритма DBSCAN

Листинг Г – Файл DBSCAN.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
from sklearn.metrics import (
    silhouette_score,
    calinski_harabasz_score,
    davies_bouldin_score,
    adjusted_rand_score,
    rand_score,
    fowlkes_mallows_score,
    mutual_info_score,
    homogeneity_score,
    completeness_score,
    v_measure_score,
)
from dataset_manager import DatasetManager

class DBSCANClustering:
    def __init__(self, eps: float = 0.5, min_samples: int = 5):
        """
        Кластеризация методом DBSCAN.

        Параметры:
            eps (float): Радиус ε-окрестности.
            min_samples (int): Минимальное количество точек в ε-окрестности.
        """
        self.eps = eps
        self.min_samples = min_samples
        self.model = None
        self.labels = None
        self.X = None

    def fit(self, X: pd.DataFrame) -> None:
        """
        Обучает DBSCAN и сохраняет метки кластеров.
        """
        self.X = X
        self.model = DBSCAN(eps=self.eps, min_samples=self.min_samples)
        self.labels = self.model.fit_predict(X)

    def visualize_clusters(self) -> None:
        """
        Визуализирует кластеры в 2D-пространстве (PCA).
        """
        if self.labels is None:
            raise RuntimeError("Сначала вызовите fit().")

        pca = PCA(n_components=2)
        X_pca = pca.fit_transform(self.X)
        df = pd.DataFrame(X_pca, columns=["PC1", "PC2"])
        df["cluster"] = self.labels

        plt.figure(figsize=(8, 6))
        for label in np.unique(self.labels):
```

Продолжение Листинга Г

```
            subset = df[df["cluster"] == label]
            plt.scatter(
                subset["PC1"],
                subset["PC2"],
                label=f"Кластер {label}" if label != -1 else "Шум",
                alpha=0.6,
            )
        plt.title("Кластеры, найденные DBSCAN")
        plt.xlabel("PC1")
        plt.ylabel("PC2")
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    def compute_metrics(self, y_true: np.ndarray) -> dict:
        """
        Вычисляет внутренние и внешние метрики (если есть метки).

        Параметры:
            y_true (np.ndarray): Истинные метки классов.

        Возвращает:
            dict: Метрики кластеризации.
        """
        labels = self.labels
        valid_mask = labels != -1
        X_valid = self.X[valid_mask]
        labels_valid = labels[valid_mask]

        metrics = {
            "Silhouette Score": silhouette_score(X_valid, labels_valid)
            if len(set(labels_valid)) > 1
            else -1,
            "Calinski-Harabasz Index": calinski_harabasz_score(X_valid,
labels_valid)
            if len(set(labels_valid)) > 1
            else -1,
            "Davies-Bouldin Index": davies_bouldin_score(X_valid, labels_valid)
            if len(set(labels_valid)) > 1
            else -1,
        }

        if y_true is not None:
            metrics.update(
                {
                    "Adjusted Rand Index": adjusted_rand_score(y_true, labels),
                    "Rand Index": rand_score(y_true, labels),
                    "Fowlkes-Mallows Index": fowlkes_mallows_score(y_true,
labels),
                    "Mutual Information": mutual_info_score(y_true, labels),
                    "Homogeneity": homogeneity_score(y_true, labels),
                    "Completeness": completeness_score(y_true, labels),
                    "V-measure": v_measure_score(y_true, labels),
                }
            )

        return metrics

manager = DatasetManager(source="sklearn")
manager.preprocess()
```

Окончание Листинга Г

```
manager.remove_feature('total_phenoles')
X_scaled, y_true = manager.get_preprocessed_data()

dbscan = DBSCANClustering(eps=2.46, min_samples=12)
dbscan.fit(X_scaled)
dbscan.visualize_clusters()

metrics = dbscan.compute_metrics(y_true)

for name, score in metrics.items():
    print(f"{name}: {score:.3f}" if score != -1 else f"{name}: недостаточно
кластеров")
```