Изображение выглядит как эмблема, символ, герб, нашивка

Автоматически созданное описание

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

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №1**

**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 3](#_Toc198247144)

[1 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc198247145)

[2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc198247146)

[2.1 Алгоритм APriori 7](#_Toc198247147)

[2.2 Алгоритм Eclat 8](#_Toc198247148)

[3 ДОКУМЕНТАЦИЯ К ДАННЫМ 10](#_Toc198247149)

[3.1 Описание предметной области 10](#_Toc198247150)

[3.2 Генерация данных 12](#_Toc198247151)

[3.3 Анализ полученных данных 14](#_Toc198247152)

[4 ПРАКТИЧЕСКАЯ ЧАСТЬ 17](#_Toc198247153)

[4.1 Алгоритм APriori 17](#_Toc198247154)

[4.2 Алгоритм Eclat 21](#_Toc198247155)

[ЗАКЛЮЧЕНИЕ 22](#_Toc198247156)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 23](#_Toc198247157)

[ПРИЛОЖЕНИЯ 24](#_Toc198247158)

# ВВЕДЕНИЕ

Современные интеллектуальные системы всё чаще применяются для анализа больших объёмов данных с целью выявления скрытых закономерностей. Одним из ключевых инструментов в этой области являются ассоциативные правила, позволяющие обнаруживать взаимосвязи между событиями или объектами. Такие правила нашли широкое применение в рекомендательных системах, анализе потребительского поведения, управлении запасами и других сферах, где критически важно понимать структуру взаимодействий в данных.

Актуальность работы обусловлена необходимостью оптимизации бизнес-процессов в условиях высокой конкуренции. Для локальной сети быстрого питания UNIfood, расположенной в кампусе РТУ МИРЭА, анализ ассоциативных правил может стать основой для улучшения ассортимента, персонализации предложений и повышения лояльности клиентов. Однако успешное применение этих методов требует не только теоретического понимания алгоритмов, но и умения работать с реальными данными, учитывая их особенности.

# 1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: приобрести навыки поиска ассоциативных правил при анализе наборов информационных исторических данных.

Задачи: определить предметную область решаемой задачи, выбрать или сгенерировать соответствующий набор данных, включающий списки с унифицированным названиями или группами, проанализировать полученный датасет, найти ассоциативные правила с количеством объектов не меньше двух и рассчитать для них метрики «поддержки» (support), «доверия, уверенности» (confidence) и «убеждённости» (conviction), пояснить суть каждой метрики в целом и смысл в данной задаче предметной области, изучить алгоритмы поиска ассоциативных правил (Apriori, Eclat, FP-Growth), написать программный код для реализации указанных алгоритмов, сравнить основные показатели производительности алгоритмов: качество результатов, скорость работы и требуемое количество памяти.

# 2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Базовым понятием в теории ассоциативных правил является транзакция – некоторое множество событий, происходящих совместно. Примером типичной транзакции является приобретение клиентом товара в супермаркете. В подавляющем большинстве случаев клиент покупает не один товар, а набор товаров, который называется рыночной корзиной.

Следующее важное понятие – предметный набор. Это непустое множество предметов, появившихся в одной транзакции.

Ассоциативное правило состоит из двух наборов предметов, называемых условием и следствием, записываемых в виде 𝑋 → 𝑌, что читается так: «Из 𝑋 следует 𝑌». Таким образом, ассоциативное правило формулируется в виде: «Если условие, то следствие».

Условие может ограничиваться только одним предметом. Правила обычно отображаются с помощью стрелок, направленных от условия к следствию, например, помидоры → салат. Условие и следствие часто называются соответственно: левосторонним и правосторонним компонентами ассоциативного правила. Ассоциативные правила описывают связь между наборами предметов, соответствующие условию и следствию. Обозначим базу данных транзакций как 𝐷, а число транзакций в этой базе – как *T*. Каждая транзакция представляет собой некоторый набор предметов. Для вывода правил используются следующие ключевые показатели:

1. **Поддержка (support)**

Показатель частотности данного предметного набора во всех анализируемых транзакциях или число транзакций, которые содержат как условие, так и следствие (Формула 2.1).

(2.1)

1. **Достоверность (confidence)**

Представляет собой меру точности правила и определяется как отношение количества транзакций, содержащих и условие, и следствие, к количеству транзакций, содержащих только условие. Показатель является условной вероятностью, отражающей наличие множества Y в наборе при наличии множества X. Достоверность вычисляется по Формуле 2.2.

(2.2)

1. **Убежденность, убедительность (conviction)**

Метрика оценивает, насколько правило 𝑋 → 𝑌 чаще ошибается, чем это происходило бы при независимости 𝑋 и 𝑌. Чем выше значение, тем сильнее зависимость между условием и следствием. Можно сказать, что убежденность показывает, насколько X и Y отклоняются от независимости в контексте ассоциативного правила 𝑋 → 𝑌 (Формула 2.3).

(2.3)

1. **Лифт, интерес (lift)**

Отношение частоты появления условия в транзакциях, которые также содержат и следствие к частоте появления следствия в целом. Значения лифта большие 1 показывают, что условие чаще появляется в транзакциях, содержащих следствие, чем в остальных. Можно утверждать, что лифт является обобщенной мерой связи двух предметных наборов: при значениях лифта больше 1 связь положительная, при 1 она отсутствует, а при значениях меньше 1 – отрицательная. Лифт помогает понять, есть ли между X и Y значимая зависимость, или их совместное появление случайно. Метрика вычисляется по Формуле 2.4.

(2.4)

1. **Рычаг (leverage)**

Отражает разность между наблюдаемой частотой, с которой условие и следствие появляются совместно (то есть поддержкой ассоциации), и произведением частот появления (поддержек) условия и следствия по отдельности. Вычисляется по Формуле 2.5.

(2.5)

## 2.1 Алгоритм APriori

Алгоритм Apriori — это классический метод ассоциативного обучения, разработанный для поиска частых наборов элементов (itemsets) в транзакционных данных. Он широко применяется в задачах анализа рыночных корзин, рекомендательных систем, обнаружения паттернов поведения пользователей и биоинформатики. Основная идея алгоритма заключается в выявлении ассоциативных правил вида 𝑋 → 𝑌, где 𝑋 и 𝑌 — наборы товаров или признаков, часто встречающихся вместе.

Apriori опирается на свойство анти-монотонности: если набор 𝑋 не является частым, то ни одно из его расширений не может быть частым. Это позволяет отсекать большую часть кандидатов на каждом этапе. Алгоритм состоит из следующих шагов:

1. Вычислить поддержку для каждого одиночного элемента.
2. Отобрать все элементарные наборы с поддержкой больше min\_support.
3. Объединить попарно все наборы из , чтобы получить кандидатов размера .
4. Отсеять кандидатов, у которых хотя бы одно *(k-1)* подмножество не содержится в .
5. Вычислить support и оставить тех кандидатов, поддержка которых больше min\_support.
6. Повторить шаги 3–5 для k = 2, 3, … n.
7. Продолжать генерировать наборы , пока .
8. Объединить все наборы в результирующий набор частых предметов (items).
9. Для каждого частого набора с размером больше двух сгенерировать все возможные непустые подмножества 𝑋, 𝑌.
10. Для каждого правила 𝑋 → 𝑌 рассчитать метрики.

## 2.2 Алгоритм Eclat

Алгоритм Eclat — это альтернативный метод поиска частых наборов элементов в транзакционных данных, использующий «вертикальное» представление. В отличие от Apriori, который многократно сканирует все транзакции, Eclat хранит для каждого элемента список идентификаторов транзакций (TID-list), в которых он встречается. Это позволяет быстро вычислять поддержку объединённых наборов через пересечение множеств TID.

Основные идеи алгоритма:

* вертикальное хранение: вместо двухмерной матрицы «транзакция × товар» строятся TID-списки: для каждого товара — множество индексов транзакций, где он есть;
* пересечения TID-списков дают поддержку любых объединённых наборов без полного прохода по всем строкам исходных данных;
* анти-монотонное свойство: если пересечение для некоторого набора пусто (или слишком мало), то расширять его нет смысла — и все «потомки» также не пройдут по порогу поддержки.

Алгоритм состоит из следующих шагов:

1. Сформировать TID список для каждого одиночного предмета.
2. Вычислить поддержку и оставить только те предметы, для которых поддержка больше заданного порогового значения.

Для каждого частого набора d и каждого последующего элемента j выполнить шаги 3–5:

1. Построить пересечение: .
2. Вычислить поддержку нового набора, если она превышает min\_support, зафиксировать набор как частый.
3. Рекурсивно расширять набор, добавляя только тех кандидатов, которые следуют после j в исходном порядке (чтобы избежать повторов).
4. Объединить все наборы в результирующий набор частых предметов (items).
5. Построить ассоциативные правила (как в алгоритме APriori).

# 3 ДОКУМЕНТАЦИЯ К ДАННЫМ

## 3.1 Описание предметной области

UNIfood — это локальная сеть быстрого питания, расположенная на территории кампуса Российского технологического университета (РТУ МИРЭА) на Проспекте Вернадского, 78. Основной аудиторией заведения являются студенты, преподаватели и сотрудники университета.

Сеть специализируется на продаже бургеров, пиццы, кофе, выпечки и кондитерских изделий. Основной ассортимент UNIfood представлен на Рисунках 3.1.1–3.1.3.



Рисунок 3.1.1 – Напитки в продаже

Стоит отметить, что баннер напитков содержит не всю информацию о позициях, продаваемых в университете.



Рисунок 3.1.2 – Фастфуд в продаже

Фастфуд – основной и самый продаваемый раздел рассматриваемой сети питания.



Рисунок 3.1.3 – Десерты и выпечка в продаже

Для поиска ассоциативных правил необходимы сведения о покупках (чеки) за длительный период, потому что ассортимент содержит не менее 50 позиций, которые должны быть рассмотрены в рамках ассоциативного анализа.

В открытом доступе требуемую информацию получить невозможно, поэтому принято решение о договорнячке с администрацией компании. Содержание письма с просьбой о предоставлении исторических данных о деятельности компании опубликовано в Приложении А.

Администрацией сети питания UNIfood любезно предоставлена статистика продаж за один месяц работы в РТУ МИРЭА в распечатанном на листах А4 виде. Несмотря на то, что не удалось заполучить готовый архив чеков компании, можно сгенерировать правдоподобный набор транзакций, обладая уникальными сведениями об объемах продаж отдельных позиций.

Отчет о продажах продукции перенесен в JSON формат для дальнейшей генерации вымышленных чеков (транзакций). Содержание JSON файла вынесено в Приложение Б.

Пример записи одной позиции из меню представлен в Листинге 3.1.1.

Листинг 3.1.1 – Запись позиции меню

{

    "Позиция": "Хот-дог куриный",

    "Цена": 168,

    "Масса": 220,

    "Продано": 3789

}

Для каждой позиции указаны ее название, цена в рублях, масса (граммы для еды и миллилитры для напитков), число продаж за месяц.

## 3.2 Генерация данных

Для генерации транзакций написан класс DatasetGenerator, который способен генерировать данные в двух режимах: заданное число строк для датасета и генерация, пока не будут распроданы все позиции (Поле «Продано» в JSON отвечает за количество). Стоит отметить, что класс способен работать с любым JSON файлом нужного формата, поэтому его легко можно использовать для генерации транзакций любой другой сети питания.

Класс позволяет произвести очень тонкую настройку для генерируемых транзакций: пользователь может указать минимальную и максимальную стоимость транзакции, минимальное и максимальное число позиций (items) в одной транзакции, а также разрешить или запретить повторяющиеся предметы (продукты) в одной транзакции.

Каждый параметр проверяется на корректность, класс обрабатывает множество исключений. Пример обработки исключения представлен на Рисунке 3.2.1.

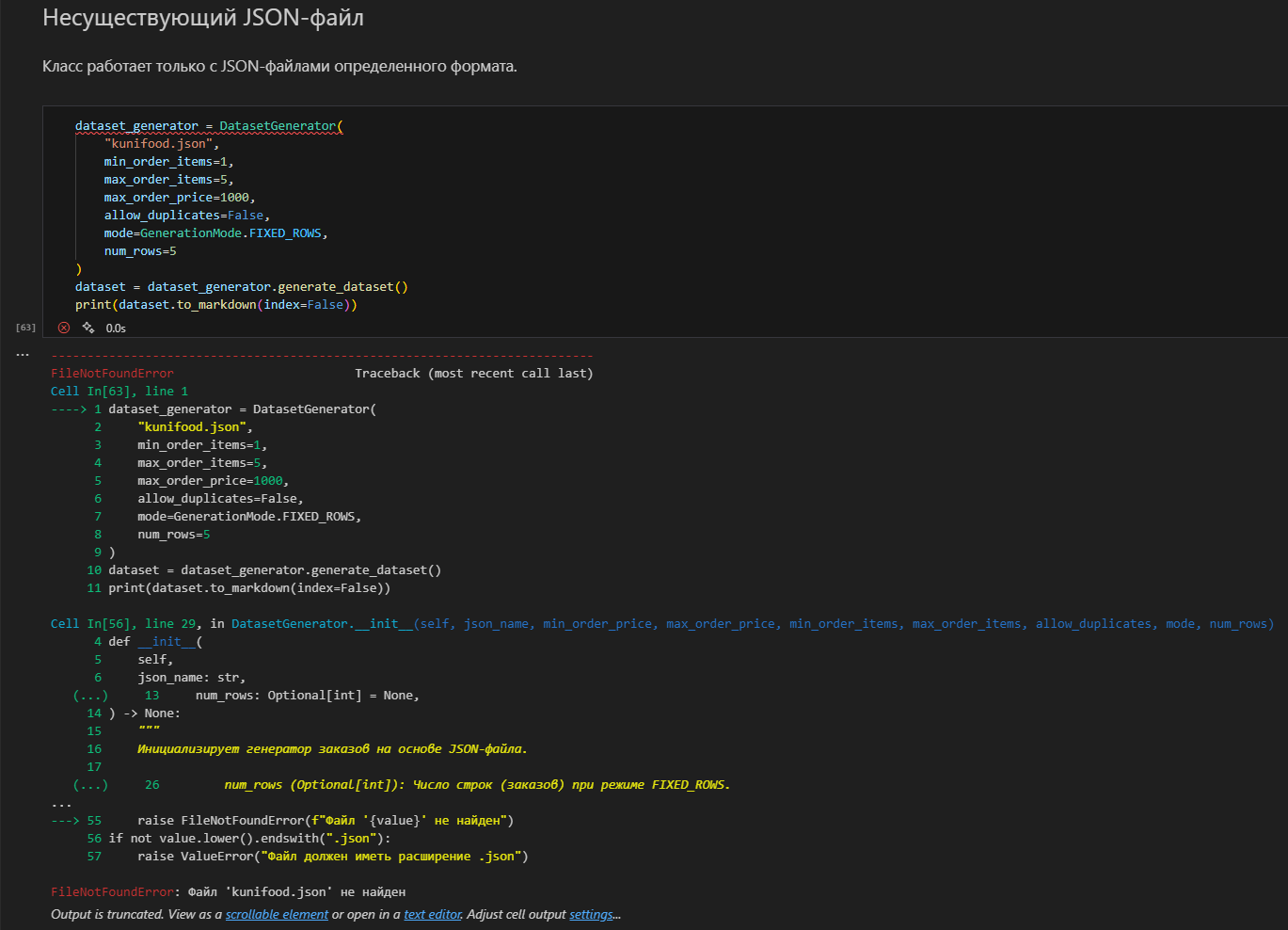


Рисунок 3.2.1 – Обработка исключения, связанного с указанием несуществующего названия файла

Идея генерации данных довольно проста. Случайно выбирается число позиций N в заказе из заданного диапазона, затем случайно выбирается N продуктов. Если созданная транзакция удовлетворяет критериям стоимости, она добавляется в набор данных, в противном случае попытка повторяется 10000 раз, после чего выбрасывается исключение. Исключение говорит о том, что критерии слишком жесткие, невозможно создать данные в заданных условиях.

Кроме того, для режима «UNTIL SOLD» (распродать все) могут оставаться остатки, которые не удалось распределить в заказы (транзакции).

Созданный набор данных отображен на Рисунке 3.2.2.

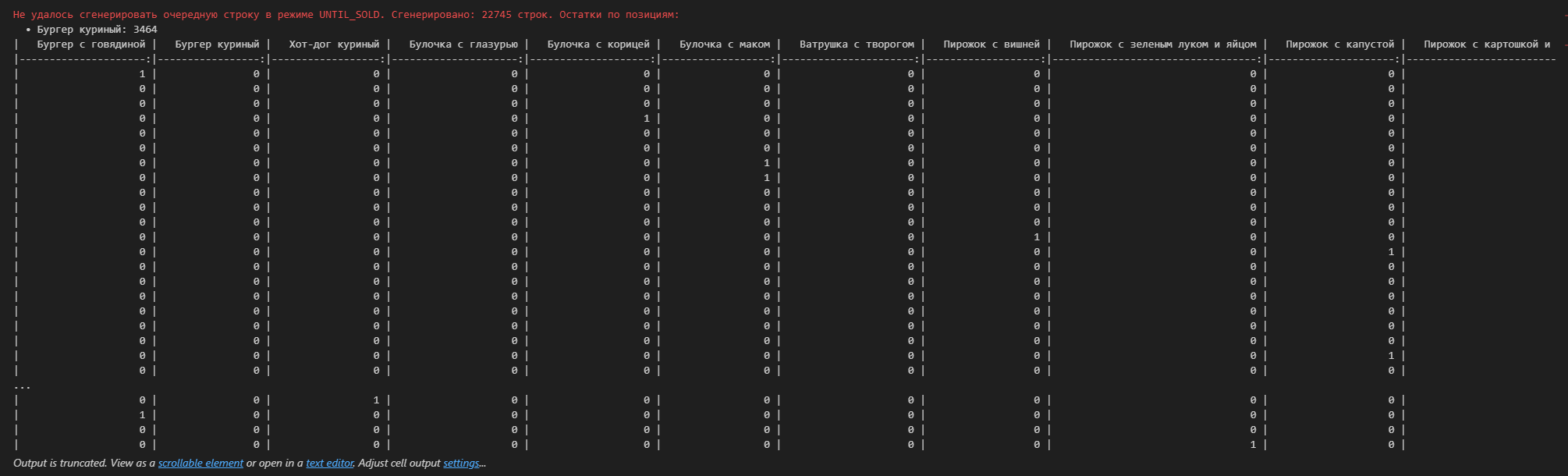


Рисунок 3.2.2 – Созданный набор данных

Сообщение «Не удалось сгенерировать очередную строку в режиме UNTIL\_SOLD. Сгенерировано: 22745 строк. Остатки по позициям:» говорит о том, что распроданы все позиции и остались только бургеры. Так как минимальное число позиций в заказе – две, а дубликаты запрещены, выходит, что оставшиеся бургеры просто нельзя больше распределить по транзакциям.

Полный код файла dataset\_generator.py с классом DatasetGenerator, а также перечислением для режимов генерации и классами кастомных исключений представлен в Приложении В.

## 3.3 Анализ полученных данных

Для анализа и просмотра различной статистики полученного датасета написан класс DatasetManager. Полный код файла dataset\_manager.py представлен в Приложении Г.

Общая информация о датасете отражена на Рисунке 3.3.1.

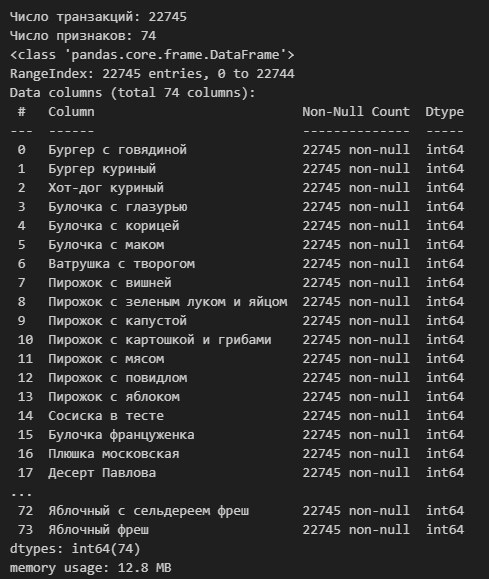


Рисунок 3.3.1 – Общая информация о наборе данных

Число транзакций, созданных в режиме UNTIL SOLD, равно 22745. Рассматривается 74 позиции из меню.

Топ 20 самых популярных товаров в транзакциях изображен на Рисунке 3.3.2.

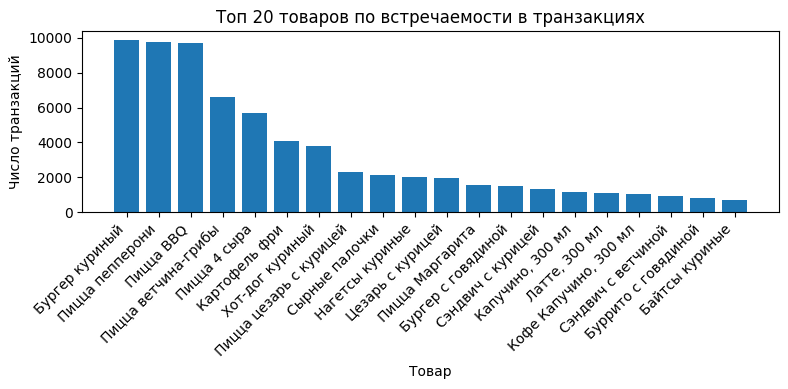


Рисунок 3.3.2 – Топ 20 самых популярных товаров по встречаемости в транзакциях

По диаграмме видно, что несколько позиций составляют основной объем продаж UNIfood. Чтобы выявить ассоциативные правила между непопулярными позициями необходимо будет установить маленькое значение support.

Распределение размеров корзин изображено на Рисунке 3.3.3.

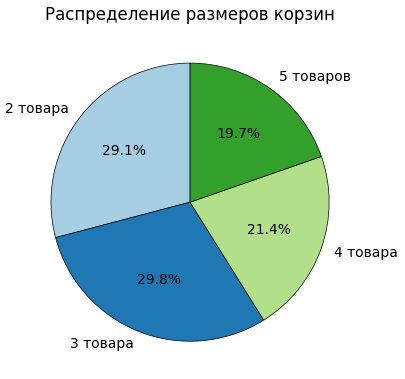


Рисунок 3.3.3 – Распределение размеров корзин

Большинство заказов содержит три позиции.

Стоит отметить, что датасет содержит 15493 дублирующихся транзакций. Связано это с тем, что продается слишком много пиццы и бургеров по сравнению с десертами, выпечкой и кофе. После определенной строки бургеры и пицца будут встречаться в рамках одной транзакции до тех пор, пока не будут проданы до конца, потому что другие категории товаров уже будут распроданы.

# 4 ПРАКТИЧЕСКАЯ ЧАСТЬ

## 4.1 Алгоритм APriori

Изучена и протестирована существующая реализация алгоритма APriori из библиотеки mlxtend.

Для применения алгоритма достаточно подключить две функции из модуля mlxtend.frequent\_patterns: apriori, association\_rules. Первая функция отвечает за генерацию частых наборов (support больше заданного порогового значения), а вторая – за генерацию ассоциативных правил из полученного частого набора. Для использования готовой реализации написан файл APriori.py, содержание которого представлено в Приложении Д.

Функции работают строго со значениями True/False для транзакций, поэтому датасет предварительно приведен к булевым значениям (Рисунок 4.1.1).

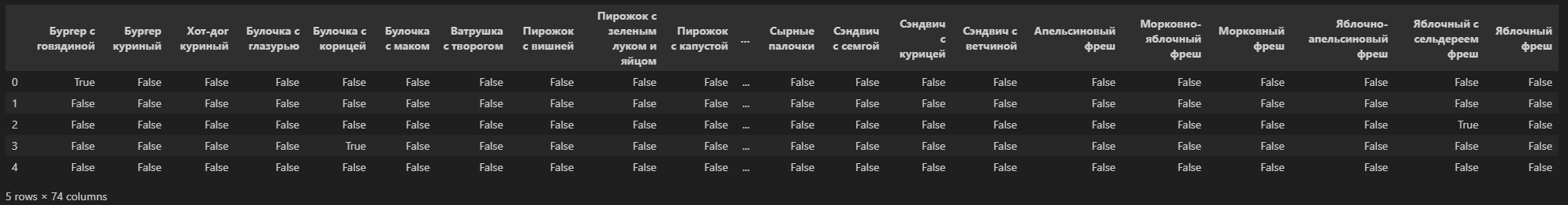


Рисунок 4.1.1 – Обработанный датасет

Значение support принято равным 0.01. Найденные частые наборы данных представлены на Рисунке 4.1.2.

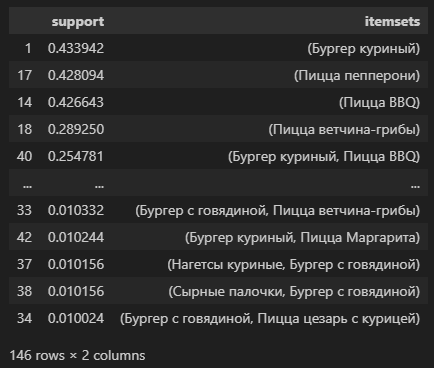


Рисунок 4.1.2 – Частые наборы данных

Найдено 146 наборов, которые включают в себя преимущественно только бургеры и пиццу даже при значении support, равным 0.01.

Определены ассоциативные правила на основе полученного частого набора (Рисунок 4.1.3).

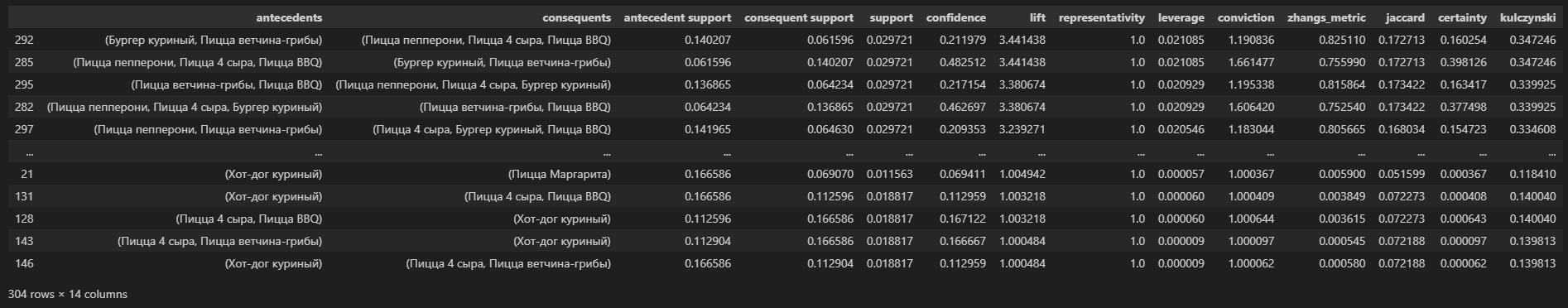


Рисунок 4.1.3 – Полученные ассоциативные правила

Правила отсортированы по значению lift > 1, так как в этом случае два набора положительно коррелируют друг с другом и их склонны покупать вместе.

Для случайной выборки из десяти ассоциативных правил визуализированы метрики lift и confidence (Рисунок 4.1.4).

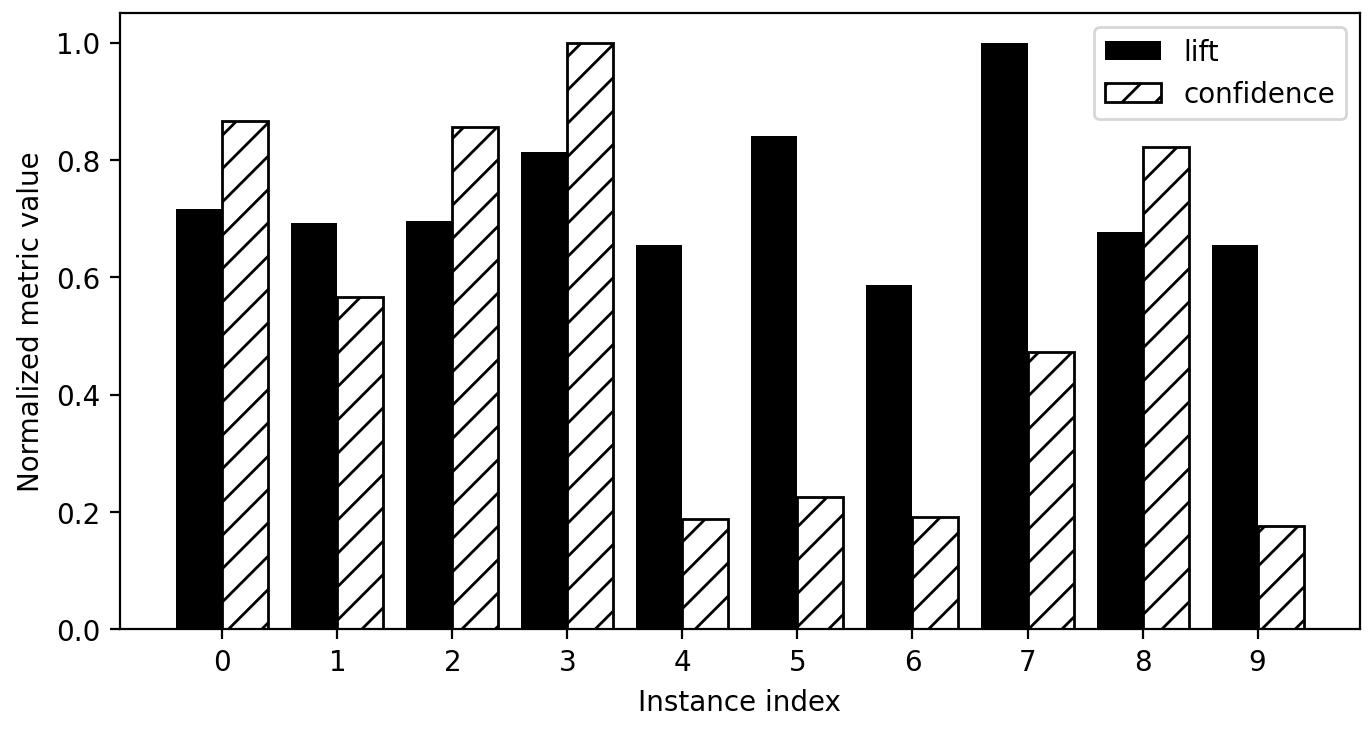


Рисунок 4.1.4 – Визуализация метрик lift, confidence для случайных десяти правил

Высокая confidence и высокий lift говорят о надёжных и значимых правилах. Низкая confidence и низкий lift говорят о том, что правило слабое. Высокая confidence, но низкий lift говорят о том, что правило 𝑋 → 𝑌 часто выполняется, но 𝑌 в целом встречается настолько часто, что сама по себе высокая confidence не значит, что связь с 𝑋 действительно значима. Низкая confidence, но высокий lift говорят о том, что 𝑌 редко появляется в корзинах с 𝑋 (confidence низок), но по отношению к общей частоте 𝑌 на удивление часто встречается именно вместе с 𝑋 (lift высок).

Для собственной реализации алгоритма APriori сохраним сигнатуры функций из библиотеки mlxtend и структуры данных, с которыми работают эти функции. Самописная реализация алгоритма находится в файле APriori\_custom.py, содержание которого представлено в Приложении Е.

Выявленные частые наборы представлены на Рисунке 4.1.5.

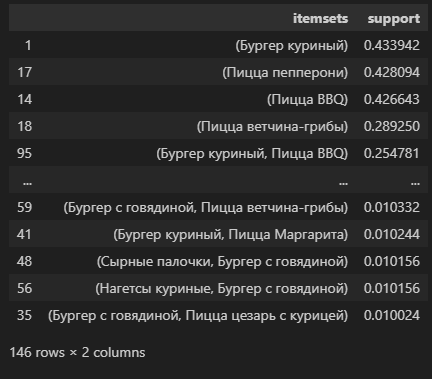


Рисунок 4.1.5 – Выявленные частые наборы

Можно заметить, что частые наборы, которые вернула написанная функция apriori, совпадают с частыми наборами библиотечной функции apriori, что говорит о корректности алгоритма генерации частых наборов. Извлеченные ассоциативные правила с тем же фильтром lift > 1 отображены на Рисунке 4.1.6.

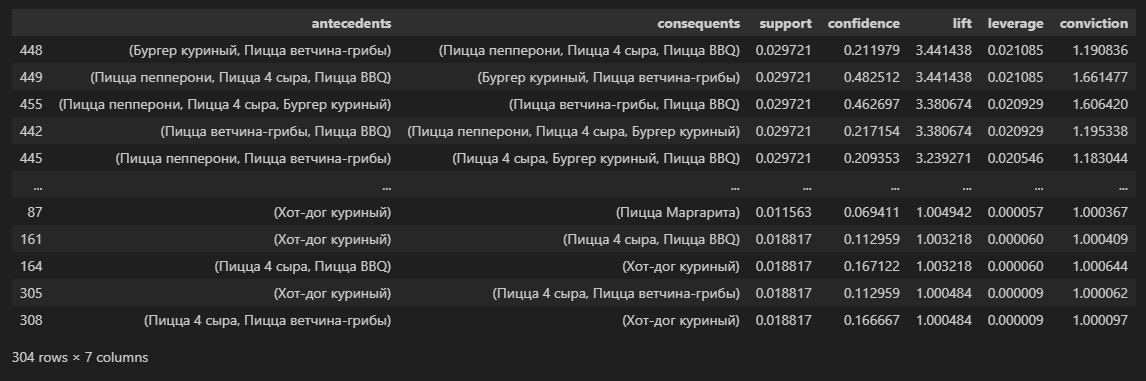


Рисунок 4.1.6 – Извлеченные ассоциативные правила

Полученные правила и метрики для них совпадают с теми правилами и метриками, которые были возвращены в результате отработки библиотечной функции association\_rules.

Значимое отличие самописной реализации – возможность фильтрации по нескольким метрикам сразу. Такой функционал реализован благодаря классу RuleFilter, который в качестве атрибутов экземпляра класса содержит элемент перечисления для метрики, его нижнюю и верхнюю границы (Рисунок 4.1.7).

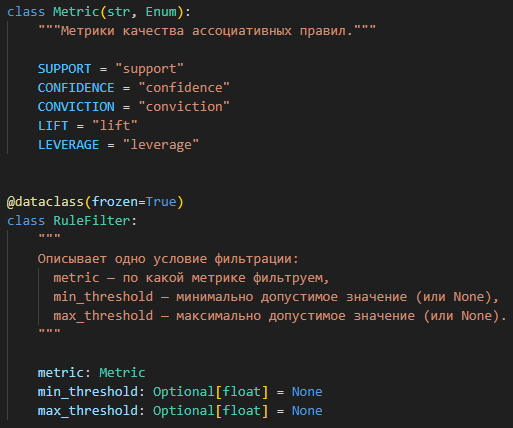


Рисунок 4.1.7 – Класс для фильтрации правил

На Рисунке 4.1.8 представлен результат извлечения правил с фильтрацией по нескольким метрикам.

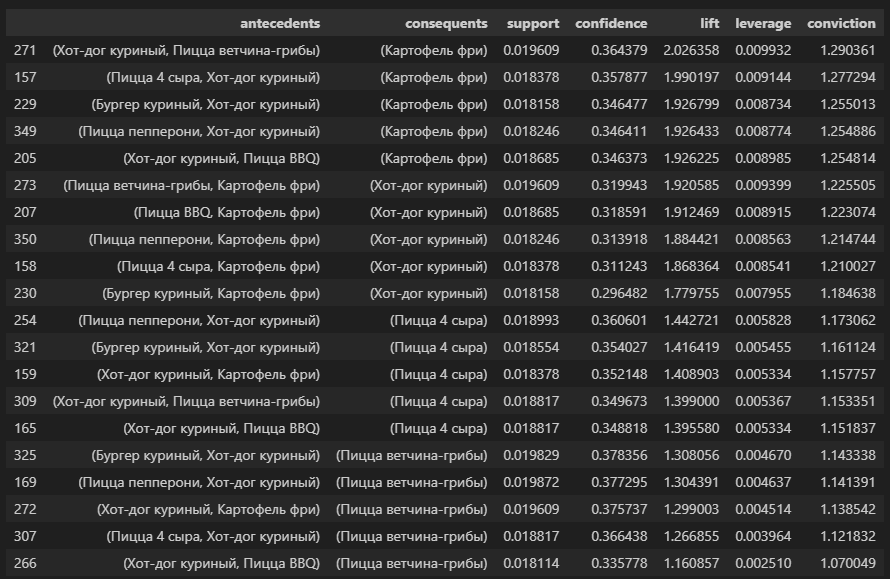


Рисунок 4.1.8 – Результат фильтрации по нескольким метрикам

Заданы следующие фильтры: lift > 1, conf > 0.25, support < 0.02.

## 4.2 Алгоритм Eclat

Не удалось найти популярную библиотеку, в которой реализован алгоритм Eclat.

Для реализации алгоритма Eclat переписан файл Apriori\_custom.py. Достаточно изменить лишь класс Apriori, даже только функцию генерации частых наборов, ведь способ извлечения правил остается прежним. Содержание файла Eclat.py представлено в Приложении Ж.

Найденные частые наборы представлены на Рисунке 4.2.1.

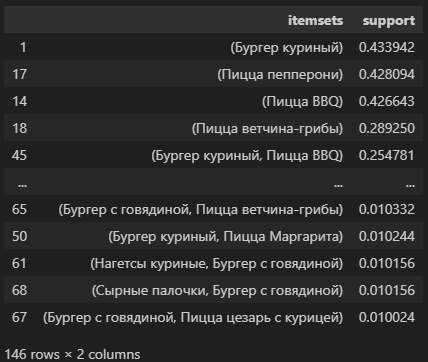


Рисунок 4.2.1 – Найденные частые наборы

Извлеченные ассоциативные правила представлены на Рисунке 4.2.2.

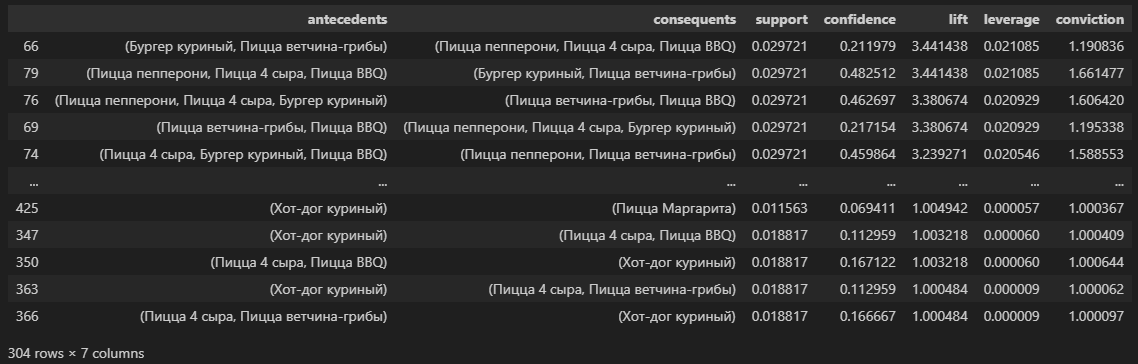


Рисунок 4.2.2 - Извлеченные ассоциативные правила

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы реализованы и протестированы два классических алгоритма поиска частых наборов и ассоциативных правил — Apriori и Eclat — на сгенерированном датасете транзакций сети быстрого питания UNIfood.

Apriori и Eclat дают идентичные множества частых наборов при одинаковых параметрах min\_support и max\_len. Проверка на примере библиотечной функции apriori из mlxtend и собственной реализации подтвердила полное совпадение полученных itemsets и ассоциативных правил (метрики support, confidence, lift, conviction). Ассоциативные правила, построенные из частых наборов обоими алгоритмами и отфильтрованные по lift > 1 и confidence > 0.25, также идентичны, что свидетельствует об эквивалентности самописных реализаций.

Apriori сканирует весь датасет на каждом уровне k, что приводит к множеству последовательных операций фильтрации и пересчёта support. Eclat сначала строит TID‐списки (односканирование), а затем рекурсивно пересекает эти списки, избегая повторных проходов по строкам.

Eclat работает примерно в 2× быстрее на данном наборе транзакций, особенно при низких уровнях поддержки, когда число частых одиночек невелико, а пересечения списков остаются компактными.

Apriori хранит только булеву матрицу размера (транзакции × товары) и промежуточные списки кандидатов. Память растёт с числом кандидатов, но не требует дополнительных структур.

Eclat помимо булевой матрицы формирует для каждого частого элемента и каждого частого набора TID‐list — множество индексов транзакций. В худшем случае (при низких порогах) их количество и размеры могут превысить объём самих данных.

# 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. Ассоциативные правила, или пиво с подгузниками [Электронный ресурс]: Habr. URL: <https://habr.com/ru/companies/ods/articles/353502/> (Дата обращения: 13.05.2025).

# ПРИЛОЖЕНИЯ

Приложение А — Письмо с просьбой о предоставлении исторических данных о деятельности компании UNIfood.

Приложение Б — Файл для генерации вымышленных чеков, содержащий сведения о позициях сети быстрого питания UNIfood.

Приложение В — Файл dataset\_generator.py для генерации набора данных.

Приложение Г — Файл dataset\_manager.py для просмотра статистики датасета.

Приложение Д — Файл APriori.py с использованием готовой реализации алгоритма APriori.

Приложение Е — Файл APriori\_custom.py с самописной реализацией алгоритма APriori.

Приложение Ж — Файл Eclat.py с самописной реализацией алгоритма Eclat.

### Приложение А

Письмо с просьбой о предоставлении исторических данных о деятельности компании UNIfood

Листинг А – Содержание письма

Уважаемая команда сети быстрого питания «UNIfood»!

Меня зовут Кликушин Владислав Игоревич, я студент 3 курса Московского университета МИРЭА, обучающийся по направлению 09.03.04 «Программная инженерия», профиль «Интеллектуальные системы поддержки принятия решений». В рамках моей исследовательской работы, посвященной анализу ассоциативных правил (выявление зависимостей между заказами), мне необходимо собрать данные о заказах пользователей.

Прошу вас рассмотреть возможность предоставления обезличенных статистических данных, которые могли бы помочь в моем исследовании. Меня интересует следующая информация:

- Список заказов, где каждый заказ представлен перечнем входящих в него позиций за любой доступный период.

Хочу подчеркнуть, что исследование носит исключительно академический характер. Все полученные данные будут использованы в обобщенном виде, без указания коммерческой или персональной информации. Готов подписать соглашение о конфиденциальности, если это потребуется.

Если для обработки запроса необходимо официальное письмо от университета или куратора проекта, готов его предоставить.

Благодарю за внимание к моей просьбе! Надеюсь, ваш опыт и информация помогут провести исследование в рамках моей работы.

С уважением,

Кликушин Владислав Игоревич

Студент МИРЭА

Контактные данные: 8902\*\*\*\*\*40, [\*\*\*\*\*\*\*\*\*\*\*\*\*@yandex.ru](mailto:*************@yandex.ru)

### Приложение Б

Файл для генерации вымышленных чеков, содержащий сведения о позициях сети быстрого питания UNIfood

Листинг Б – Содержание JSON файла

{

    "БУРГЕРЫ": [

        {

            "Позиция": "Бургер с говядиной",

            "Цена": 241.5,

            "Масса": 250,

            "Продано": 1534

        },

        {

            "Позиция": "Бургер куриный",

            "Цена": 199.5,

            "Масса": 250,

            "Продано": 13334

        },

        {

            "Позиция": "Хот-дог куриный",

            "Цена": 168,

            "Масса": 220,

            "Продано": 3789

        }

    ],

    "ВЫПЕЧКА ПИРОЖКИ": [

        {

            "Позиция": "Булочка с глазурью",

            "Цена": 73.5,

            "Масса": 130,

            "Продано": 288

        },

        {

            "Позиция": "Булочка с корицей",

            "Цена": 52.5,

            "Масса": 100,

            "Продано": 238

        },

        {

            "Позиция": "Булочка с маком",

            "Цена": 70,

            "Масса": 85,

            "Продано": 2

        },

        {

            "Позиция": "Ватрушка с творогом",

            "Цена": 52.5,

            "Масса": 100,

            "Продано": 146

        },

        {

            "Позиция": "Пирожок с вишней",

            "Цена": 70,

            "Масса": 100,

            "Продано": 148

        },

        {

Продолжение Листинга Б

            "Позиция": "Пирожок с зеленым луком и яйцом",

            "Цена": 52.5,

            "Масса": 100,

            "Продано": 151

        },

        {

            "Позиция": "Пирожок с капустой",

            "Цена": 52.5,

            "Масса": 100,

            "Продано": 139

        },

        {

            "Позиция": "Пирожок с картошкой и грибами",

            "Цена": 52.5,

            "Масса": 100,

            "Продано": 142

        },

        {

            "Позиция": "Пирожок с мясом",

            "Цена": 110,

            "Масса": 100,

            "Продано": 147

        },

        {

            "Позиция": "Пирожок с повидлом",

            "Цена": 52,

            "Масса": 100,

            "Продано": 146

        },

        {

            "Позиция": "Пирожок с яблоком",

            "Цена": 52.5,

            "Масса": 100,

            "Продано": 144

        },

        {

            "Позиция": "Сосиска в тесте",

            "Цена": 110,

            "Масса": 115,

            "Продано": 345

        },

        {

            "Позиция": "Булочка француженка",

            "Цена": 70,

            "Масса": 100,

            "Продано": 250

        },

        {

            "Позиция": "Плюшка московская",

            "Цена": 52.5,

            "Масса": 100,

            "Продано": 213

        }

    ],

    "КОНДИТЕРКА": [

        {

            "Позиция": "Десерт Павлова",

            "Цена": 140,

            "Масса": 65,

            "Продано": 1

Продолжение Листинга Б

        },

        {

            "Позиция": "Кекс морковный",

            "Цена": 120,

            "Масса": 100,

            "Продано": 29

        },

        {

            "Позиция": "Творожное кольцо",

            "Цена": 90,

            "Масса": 75,

            "Продано": 42

        },

        {

            "Позиция": "Красный бархат",

            "Цена": 200,

            "Масса": 125,

            "Продано": 99

        },

        {

            "Позиция": "Лимонный тарт",

            "Цена": 140,

            "Масса": 100,

            "Продано": 17

        },

        {

            "Позиция": "Медовик",

            "Цена": 140,

            "Масса": 100,

            "Продано": 24

        },

        {

            "Позиция": "Миндальный торт",

            "Цена": 200,

            "Масса": 100,

            "Продано": 80

        },

        {

            "Позиция": "Наполеон",

            "Цена": 120,

            "Масса": 100,

            "Продано": 43

        },

        {

            "Позиция": "Песочная полоска",

            "Цена": 90,

            "Масса": 75,

            "Продано": 37

        },

        {

            "Позиция": "Пирожное картошка",

            "Цена": 94,

            "Масса": 60,

            "Продано": 39

        },

        {

            "Позиция": "Прага",

            "Цена": 180,

            "Масса": 100,

            "Продано": 100

        },

Продолжение Листинга Б

        {

            "Позиция": "Сочник с творогом",

            "Цена": 90,

            "Масса": 90,

            "Продано": 28

        },

        {

            "Позиция": "Тирамису",

            "Цена": 200,

            "Масса": 100,

            "Продано": 16

        },

        {

            "Позиция": "Птичье молоко",

            "Цена": 240,

            "Масса": 100,

            "Продано": 64

        },

        {

            "Позиция": "Три шоколада",

            "Цена": 200,

            "Масса": 100,

            "Продано": 77

        },

        {

            "Позиция": "Чизкейк манговое пюре",

            "Цена": 180,

            "Масса": 100,

            "Продано": 95

        },

        {

            "Позиция": "Чизкейк Нью-Йорк",

            "Цена": 180,

            "Масса": 100,

            "Продано": 72

        },

        {

            "Позиция": "Донаты",

            "Цена": 90,

            "Масса": 70,

            "Продано": 443

        }

    ],

    "КОФЕ": [

        {

            "Позиция": "Американо, 200 мл",

            "Цена": 120,

            "Объем": 200,

            "Продано": 110

        },

        {

            "Позиция": "Американо, 300 мл",

            "Цена": 160,

            "Объем": 300,

            "Продано": 129

        },

        {

            "Позиция": "Какао",

            "Цена": 180,

            "Объем": 200,

            "Продано": 57

Продолжение Листинга Б

        },

        {

            "Позиция": "Капучино, 200 мл",

            "Цена": 140,

            "Объем": 200,

            "Продано": 109

        },

        {

            "Позиция": "Капучино, 300 мл",

            "Цена": 190,

            "Объем": 300,

            "Продано": 1178

        },

        {

            "Позиция": "Карамелатте",

            "Цена": 210,

            "Объем": 350,

            "Продано": 182

        },

        {

            "Позиция": "Американский, 200 мл",

            "Цена": 120,

            "Объем": 200,

            "Продано": 181

        },

        {

            "Позиция": "Американский, 300 мл",

            "Цена": 160,

            "Объем": 300,

            "Продано": 141

        },

        {

            "Позиция": "Кофе Капучино, 200 мл",

            "Цена": 140,

            "Объем": 200,

            "Продано": 619

        },

        {

            "Позиция": "Кофе Капучино, 300 мл",

            "Цена": 190,

            "Объем": 300,

            "Продано": 1062

        },

        {

            "Позиция": "Латте, 200 мл",

            "Цена": 150,

            "Объем": 200,

            "Продано": 489

        },

        {

            "Позиция": "Латте, 300 мл",

            "Цена": 200,

            "Объем": 300,

            "Продано": 1086

        },

        {

            "Позиция": "Раф",

            "Цена": 280,

            "Объем": 300,

            "Продано": 120

        },

Продолжение Листинга Б

        {

            "Позиция": "Экспрессо, 35 мл",

            "Цена": 105,

            "Объем": 35,

            "Продано": 83

        },

        {

            "Позиция": "Экспрессо, 65 мл",

            "Цена": 145,

            "Объем": 65,

            "Продано": 99

        }

    ],

    "ПЕРВЫЕ БЛЮДА": [

        {

            "Позиция": "Крем-суп грибной",

            "Цена": 125,

            "Масса": 310,

            "Продано": 543

        }

    ],

    "ПИЦЦА": [

        {

            "Позиция": "Пицца BBQ",

            "Цена": 94.5,

            "Масса": 115,

            "Продано": 9704

        },

        {

            "Позиция": "Пицца 4 сыра",

            "Цена": 94.5,

            "Масса": 110,

            "Продано": 5685

        },

        {

            "Позиция": "Пицца Маргарита",

            "Цена": 94.5,

            "Масса": 110,

            "Продано": 1571

        },

        {

            "Позиция": "Пицца пепперони",

            "Цена": 94.5,

            "Масса": 115,

            "Продано": 9737

        },

        {

            "Позиция": "Пицца ветчина-грибы",

            "Цена": 94.5,

            "Масса": 115,

            "Продано": 6579

        },

        {

            "Позиция": "Пицца цезарь с курицей",

            "Цена": 94.5,

            "Масса": 115,

            "Продано": 2308

        }

    ],

    "САЛАТ": [

        {

Продолжение Листинга Б

            "Позиция": "Цезарь с курицей",

            "Цена": 283.5,

            "Масса": 290,

            "Продано": 1955

        }

    ],

    "ФАСТФУД": [

        {

            "Позиция": "Байтсы куриные",

            "Цена": 130,

            "Масса": 100,

            "Продано": 704

        },

        {

            "Позиция": "Батат сладкий",

            "Цена": 150,

            "Масса": 100,

            "Продано": 114

        },

        {

            "Позиция": "Буррито с говядиной",

            "Цена": 180,

            "Масса": 220,

            "Продано": 838

        },

        {

            "Позиция": "Картофель по-деревенски",

            "Цена": 84,

            "Масса": 100,

            "Продано": 75

        },

        {

            "Позиция": "Картофель фри",

            "Цена": 84,

            "Масса": 100,

            "Продано": 4090

        },

        {

            "Позиция": "Нагетсы куриные",

            "Цена": 115.5,

            "Масса": 100,

            "Продано": 2002

        },

        {

            "Позиция": "Сырные палочки",

            "Цена": 115.5,

            "Масса": 100,

            "Продано": 2161

        },

        {

            "Позиция": "Сэндвич с семгой",

            "Цена": 262.5,

            "Масса": 180,

            "Продано": 170

        },

        {

            "Позиция": "Сэндвич с курицей",

            "Цена": 168,

            "Масса": 180,

            "Продано": 1326

        },

Окончание Листинга Б

        {

            "Позиция": "Сэндвич с ветчиной",

            "Цена": 157.5,

            "Масса": 180,

            "Продано": 916

        }

    ],

    "ФРЕШИ": [

        {

            "Позиция": "Апельсиновый фреш",

            "Цена": 300,

            "Объем": 350,

            "Продано": 79

        },

        {

            "Позиция": "Морковно-яблочный фреш",

            "Цена": 200,

            "Объем": 350,

            "Продано": 37

        },

        {

            "Позиция": "Морковный фреш",

            "Цена": 150,

            "Объем": 350,

            "Продано": 44

        },

        {

            "Позиция": "Яблочно-апельсиновый фреш",

            "Цена": 250,

            "Объем": 350,

            "Продано": 89

        },

        {

            "Позиция": "Яблочный с сельдереем фреш",

            "Цена": 220,

            "Объем": 350,

            "Продано": 11

        },

        {

            "Позиция": "Яблочный фреш",

            "Цена": 230,

            "Объем": 350,

            "Продано": 59

        }

    ]

}

### Приложение В

Код файла dataset\_generator.py для генерации набора данных

Листинг В – Код файла dataset\_generator.py

import json

import random

import pandas as pd

from copy import deepcopy

from pathlib import Path

from enum import Enum, auto

from typing import List, Dict, Optional, Any

class GenerationMode(Enum):

    """Режимы работы генератора."""

    UNTIL\_SOLD = auto()

    FIXED\_ROWS = auto()

class EmptyGenerationSetException(Exception):

    """Нет ни одной комбинации, удовлетворяющей заданным параметрам генерации."""

    pass

class GenerationException(Exception):

    """Ошибка во время процесса генерации."""

    pass

class DatasetGenerator:

    """Генератор случайных «заказов» на основе JSON-файла с товарами."""

    def \_\_init\_\_(

        self,

        json\_name: str,

        min\_order\_price: Optional[float | int] = None,

        max\_order\_price: Optional[float | int] = None,

        min\_order\_items: Optional[int] = None,

        max\_order\_items: Optional[int] = None,

        allow\_duplicates: bool = True,

        mode: GenerationMode = GenerationMode.UNTIL\_SOLD,

        num\_rows: Optional[int] = None,

    ) -> None:

        """

        Инициализирует генератор заказов на основе JSON-файла.

        Параметры:

            json\_name (str): Путь к JSON-файлу с описанием товаров.

            min\_order\_price (Optional[float | int]): Минимальная сумма заказа.

            max\_order\_price (Optional[float | int]): Максимальная сумма одного заказа.

            min\_order\_items (Optional[int]): Минимальное число позиций в одном заказе.

            max\_order\_items (Optional[int]): Максимальное число позиций в одном заказе.

            allow\_duplicates (bool): Разрешить ли несколько единиц одного товара в заказе.

            mode (GenerationMode): Режим генерации (UNTIL\_SOLD или FIXED\_ROWS).

            num\_rows (Optional[int]): Число строк (заказов) при режиме FIXED\_ROWS.

Продолжение Листинга В

        """

        self.json\_name = json\_name

        self.min\_order\_price = min\_order\_price

        self.max\_order\_price = max\_order\_price

        self.min\_order\_items = min\_order\_items

        self.max\_order\_items = max\_order\_items

        self.\_validate\_min\_max()

        self.allow\_duplicates = allow\_duplicates

        self.mode = mode

        self.num\_rows = num\_rows

        self.\_validate\_mode\_and\_num\_rows()

        self.\_load\_data()

    @property

    def json\_name(self) -> str:

        """Путь к JSON-файлу с описанием товаров."""

        return self.\_json\_name

    @json\_name.setter

    def json\_name(self, value: str) -> None:

        if not isinstance(value, str):

            raise TypeError("Имя файла должно быть строкой")

        if not Path(value).exists():

            raise FileNotFoundError(f"Файл '{value}' не найден")

        if not value.lower().endswith(".json"):

            raise ValueError("Файл должен иметь расширение .json")

        self.\_json\_name = value

    @property

    def min\_order\_price(self) -> Optional[float]:

        """Нижняя граница суммы заказа."""

        return self.\_min\_order\_price

    @min\_order\_price.setter

    def min\_order\_price(self, value: Optional[float | int]) -> None:

        if value is not None:

            if not isinstance(value, (int, float)):

                raise TypeError("min\_order\_price должен быть числом")

            if value < 0:

                raise ValueError("min\_order\_price не может быть отрицательным")

        self.\_min\_order\_price = float(value) if value is not None else None

    @property

    def max\_order\_price(self) -> Optional[float]:

        """Верхняя граница суммы заказа."""

        return self.\_max\_order\_price

    @max\_order\_price.setter

    def max\_order\_price(self, value: Optional[float | int]) -> None:

        if value is not None:

            if not isinstance(value, (int, float)):

                raise TypeError("max\_order\_price должен быть числом")

            if value < 0:

                raise ValueError("max\_order\_price не может быть отрицательным")

        self.\_max\_order\_price = float(value) if value is not None else None

Продолжение Листинга В

    @property

    def min\_order\_items(self) -> Optional[int]:

        """Минимальное число товарных позиций в заказе."""

        return self.\_min\_order\_items

    @min\_order\_items.setter

    def min\_order\_items(self, value: Optional[int]) -> None:

        if value is not None:

            if not isinstance(value, int):

                raise TypeError("min\_order\_items должен быть целым числом")

            if value < 0:

                raise ValueError("min\_order\_items не может быть отрицательным")

        self.\_min\_order\_items = int(value) if value is not None else None

    @property

    def max\_order\_items(self) -> Optional[int]:

        """Максимальное число товарных позиций в заказе."""

        return self.\_max\_order\_items

    @max\_order\_items.setter

    def max\_order\_items(self, value: Optional[int]) -> None:

        if value is not None:

            if not isinstance(value, int):

                raise TypeError("max\_order\_items должен быть целым числом")

            if value < 0:

                raise ValueError("max\_order\_items не может быть отрицательным")

        self.\_max\_order\_items = int(value) if value is not None else None

    @property

    def allow\_duplicates(self) -> bool:

        """Флаг: можно ли несколько единиц одного товара в заказе."""

        return self.\_allow\_duplicates

    @allow\_duplicates.setter

    def allow\_duplicates(self, value: bool) -> None:

        if not isinstance(value, bool):

            raise ValueError("allow\_duplicates должен быть булевым")

        self.\_allow\_duplicates = value

    @property

    def mode(self) -> GenerationMode:

        """Режим генерации."""

        return self.\_mode

    @mode.setter

    def mode(self, value: GenerationMode) -> None:

        if not isinstance(value, GenerationMode):

            raise ValueError("mode должен быть элементом GenerationMode")

        self.\_mode = value

    @property

    def num\_rows(self) -> Optional[int]:

        """Число строк для режима FIXED\_ROWS."""

        return self.\_num\_rows

    @num\_rows.setter

    def num\_rows(self, value: Optional[int]) -> None:

        if value is not None:

            if not isinstance(value, int):

                raise TypeError("num\_rows должен быть целым числом")

Продолжение Листинга В

            if value < 0:

                raise ValueError("num\_rows не может быть отрицательным")

        self.\_num\_rows = value

    def \_validate\_min\_max(self) -> None:

        """

        Проверяет, что min\_order\_price < max\_order\_price и

        min\_order\_items < max\_order\_items (если оба заданы).

        Исключения:

            ValueError

        """

        if self.min\_order\_price is not None and self.max\_order\_price is not None:

            if self.min\_order\_price >= self.max\_order\_price:

                raise ValueError(

                    f"min\_order\_price должен быть меньше max\_order\_price: {self.min\_order\_price} < {self.max\_order\_price} - False"

                )

        if self.min\_order\_items is not None and self.max\_order\_items is not None:

            if self.min\_order\_items >= self.max\_order\_items:

                raise ValueError(

                    f"min\_order\_items должен быть меньше max\_order\_items: {self.min\_order\_items} < {self.max\_order\_items} - False"

                )

    def \_validate\_mode\_and\_num\_rows(self) -> None:

        """

        Проверяет согласованность mode и num\_rows:

        - UNTIL\_SOLD требует num\_rows=None

        - FIXED\_ROWS требует num\_rows заданным

        Исключения:

            ValueError

        """

        if self.mode == GenerationMode.UNTIL\_SOLD and self.num\_rows is not None:

            raise ValueError(

                "Для режима генерации UNTIL\_SOLD параметр num\_rows должен опущен"

            )

        if self.mode == GenerationMode.FIXED\_ROWS and self.num\_rows is None:

            raise ValueError(

                "Для режима генерации FIXED\_ROWS параметр num\_rows должен быть задан"

            )

    def \_load\_data(self) -> None:

        """

        Загружает данные из JSON:

        - self.\_original\_data: список всех товаров (List[Dict])

        - self.\_headers: список их названий (List[str])

        Исключения:

            ValueError, RuntimeError

        """

        try:

            with open(self.\_json\_name, "r", encoding="utf-8") as f:

                raw = json.load(f)

        except json.JSONDecodeError as e:

            raise ValueError(f"Ошибка парсинга JSON-файла: {e}")

Продолжение Листинга В

        except Exception as e:

            raise RuntimeError(

                f"Непредвиденная ошибка при чтении {self.\_json\_name}: {e}"

            )

        if not isinstance(raw, dict):

            raise ValueError(

                "Ожидался словарь со строками в ключах и списками словарей в значениях"

            )

        self.\_original\_data = []

        self.\_headers = []

        for items in raw.values():

            if 'teremok' in self.json\_name:

                for item in items:

                    item['Цена'] = float(item['Цена'].rstrip('₽'))

            self.\_original\_data.extend(items)

            for item in items:

                if "Позиция" in item:

                    self.\_headers.append(item["Позиция"])

                elif "Название продукта" in item:

                    self.\_headers.append(item["Название продукта"])

    def generate\_dataset(self) -> pd.DataFrame:

        """

        Формирует DataFrame заказов в зависимости от режима.

        Возвращает:

            pd.DataFrame: строки — заказы, столбцы — позиции.

        """

        if self.mode == GenerationMode.FIXED\_ROWS:

            rows = self.\_generate\_fixed\_rows()

        elif self.mode == GenerationMode.UNTIL\_SOLD:

            rows = self.\_generate\_until\_sold()

        return pd.DataFrame(rows, columns=self.\_headers)

    def \_generate\_fixed\_rows(self) -> List[List[int]]:

        """

        Составляет ровно num\_rows заказов, каждый удовлетворяет параметрам.

        Возвращает:

            List[List[int]]: список векторов количеств по каждому товару.

        Исключения:

            GenerationException

        """

        data\_pool = self.\_validate\_params()

        rows = []

        for \_ in range(self.num\_rows):

            for attempt in range(10000):

                min\_items = self.min\_order\_items or 1

                max\_items = self.max\_order\_items or len(data\_pool)

                num\_items = random.randint(min\_items, max\_items)

                if self.allow\_duplicates:

                    chosen = random.choices(data\_pool, k=num\_items)

                else:

                    chosen = random.sample(data\_pool, k=num\_items)

                total\_price = sum(item["Цена"] for item in chosen)

Продолжение Листинга В

                if (

                    self.min\_order\_price is not None

                    and total\_price < self.min\_order\_price

                ):

                    continue

                if (

                    self.max\_order\_price is not None

                    and total\_price > self.max\_order\_price

                ):

                    continue

                row = [0] \* len(self.\_original\_data)

                for item in chosen:

                    idx = self.\_original\_data.index(item)

                    row[idx] += 1

                rows.append(row)

                break

            else:

                raise GenerationException(

                    f"Не удалось сгенерировать строку {\_ + 1}/{self.num\_rows} "

                    f"за {attempt + 1} попыток"

                )

        return rows

    def \_validate\_params(self) -> List[Dict[str, Any]]:

        """

        Отбирает товары по max\_order\_price и проверяет min\_order\_items

        (при allow\_duplicates=False).

        Возвращает:

            List[Dict]: список отфильтрованных товаров.

        Исключения:

            EmptyGenerationSetException

        """

        data = deepcopy(self.\_original\_data)

        if self.max\_order\_price is not None:

            data = list(

                filter(

                    lambda item: item["Цена"] <= self.max\_order\_price,

                    self.\_original\_data,

                )

            )

            if not data:

                raise EmptyGenerationSetException(

                    f"Невозможно сгенерировать датасет, не найдено позиций, удовлетворяющих параметрам генерации: max\_order\_price={self.max\_order\_price}"

                )

        if self.min\_order\_items is not None and not self.allow\_duplicates:

            total\_unique = len(data)

            if self.min\_order\_items > len(data):

                raise EmptyGenerationSetException(

                    f"Невозможно сгенерировать датасет, не найдено позиций, удовлетворяющих параметрам генерации: min\_order\_items={self.min\_order\_items}, allow\_duplicates={self.allow\_duplicates}. Невозможно выбрать {self.min\_order\_items} уникальных товаров из {total\_unique}"

                )

        return data

Продолжение Листинга В

    def \_generate\_until\_sold(self) -> List[List[int]]:

        """

        Генерирует заказы, пока есть остатки товаров (поле 'Продано').

        Возвращает:

            List[List[int]]: список векторов количеств по каждому товару.

        """

        data\_pool = self.\_validate\_params()

        remaining = [item["Продано"] for item in data\_pool]

        rows = []

        while any(rem > 0 for rem in remaining):

            for \_ in range(10000):

                min\_items = self.min\_order\_items or 1

                max\_items = self.max\_order\_items or len(data\_pool)

                num\_items = random.randint(min\_items, max\_items)

                available\_indices = [

                    idx for idx, rem in enumerate(remaining) if rem > 0

                ]

                if not available\_indices:

                    return rows

                if self.allow\_duplicates:

                    chosen\_idxs = random.choices(available\_indices, k=num\_items)

                else:

                    if num\_items > len(available\_indices):

                        continue

                    chosen\_idxs = random.sample(available\_indices, k=num\_items)

                cnt = {}

                for i in chosen\_idxs:

                    cnt[i] = cnt.get(i, 0) + 1

                    if cnt[i] > remaining[i]:

                        break

                else:

                    total\_price = sum(

                        self.\_original\_data[i]["Цена"] \* cnt[i] for i in cnt

                    )

                    if (

                        self.min\_order\_price is not None

                        and total\_price < self.min\_order\_price

                    ):

                        continue

                    if (

                        self.max\_order\_price is not None

                        and total\_price > self.max\_order\_price

                    ):

                        continue

                    for i, q in cnt.items():

                        remaining[i] -= q

                    row = [0] \* len(self.\_original\_data)

                    for i, q in cnt.items():

                        row[i] = q

                    rows.append(row)

                    break

            else:

                remaining\_items = [

                    (self.\_original\_data[i]["Позиция"], rem)

Окончание Листинга В

                    for i, rem in enumerate(remaining)

                    if rem > 0

                ]

                print(

                    "\033[91m"

                    + f"Не удалось сгенерировать очередную строку в режиме UNTIL\_SOLD. "

                    f"Сгенерировано: {len(rows)} строк. Остатки по позициям:"

                    + "\033[0m"

                )

                for name, qty in remaining\_items:

                    print(f"  • {name}: {qty}")

                return rows

        return rows

### Приложение Г

Код файла dataset\_manager.py для просмотра статистики датасета

Листинг Г – Код файла dataset\_manager.py

from dataset\_generator import (

    DatasetGenerator,

    GenerationMode,

    EmptyGenerationSetException,

    GenerationException,

)

import matplotlib.pyplot as plt

import pandas as pd

class DatasetManager:

    def \_\_init\_\_(self, dataset: pd.DataFrame) -> None:

        """

        Инициализирует менеджер статистик по датасету.

        Параметры:

            dataset (pd.DataFrame): DataFrame, где строки — транзакции, столбцы — позиции товаров.

        """

        self.dataset: pd.DataFrame = dataset

    def show\_dataset\_info(self) -> None:

        """

        Выводит основную информацию о датасете:

        число транзакций, число признаков и подробную сводку pandas.

        """

        print(f"Число транзакций: {len(self.dataset)}")

        print(f"Число признаков: {dataset.columns.size}")

        self.dataset.info()

    def show\_top\_n\_items(self, n: int) -> None:

        """

        Строит столбчатую диаграмму для первых N товаров по числу транзакций,

        в которых они встречаются.

        Параметры:

            n (int): количество топ-товаров для отображения.

        """

        freq: pd.Series = (self.dataset > 0).sum(axis=0)

        top: pd.Series = freq.sort\_values(ascending=False).head(n)

        plt.figure(figsize=(8, 4))

        plt.bar(top.index, top.values)

        plt.xticks(rotation=45, ha="right")

        plt.xlabel("Товар")

        plt.ylabel("Число транзакций")

        plt.title(f"Топ {n} товаров по встречаемости в транзакциях")

        plt.tight\_layout()

        plt.show()

        print(top)

    def plot\_transaction\_length\_distribution(self) -> None:

        """

        Строит круговую (pie) диаграмму распределения размеров корзин (сумма по строке).

        """

Окончание Листинга Г

        transaction\_lengths: pd.Series = self.dataset.sum(axis=1).astype(int)

        size\_counts: pd.Series = transaction\_lengths.value\_counts().sort\_index()

        labels = [

            f"{size} товар{'а' if size < 5 else 'ов'}" for size in size\_counts.index

        ]

        plt.figure(figsize=(8, 4))

        plt.pie(

            size\_counts,

            labels=labels,

            autopct="%1.1f%%",

            startangle=90,

            colors=plt.cm.Paired.colors,

            wedgeprops={"edgecolor": "black", "linewidth": 0.5},

        )

        plt.title("Распределение размеров корзин")

        plt.tight\_layout()

        plt.show()

    def show\_basket\_stats(self) -> None:

        """

        Выводит основные статистики по размерам корзин:

        среднее, медиану, моду, минимум и максимум.

        """

        transaction\_lengths: pd.Series = self.dataset.sum(axis=1)

        print(f"Средний размер корзины: {transaction\_lengths.mean():.2f}")

        print(f"Медианный размер: {transaction\_lengths.median()}")

        print(f"Мода: {transaction\_lengths.mode().iat[0]}")

        print(

            f"Минимум/Максимум: {transaction\_lengths.min()}/{transaction\_lengths.max()}"

        )

    def check\_duplicates(self) -> None:

        """

        Подсчитывает и выводит число полностью дублирующихся транзакций в датасете.

        """

        duplicates: int = self.dataset.duplicated().sum()

        print(f"Число дубликатов транзакций: {duplicates}")

dataset\_generator = DatasetGenerator(

    "unifood.json",

    min\_order\_items=2,

    max\_order\_items=5,

    max\_order\_price=1000,

    allow\_duplicates=False,

    mode=GenerationMode.UNTIL\_SOLD,

)

dataset = dataset\_generator.generate\_dataset()

dataset\_manager = DatasetManager(dataset)

dataset\_manager.show\_dataset\_info()

dataset\_manager.show\_top\_n\_items(20)

dataset\_manager.plot\_transaction\_length\_distribution()

dataset\_manager.show\_basket\_stats()

dataset\_manager.check\_duplicates()

### Приложение Д

Файл APriori.py с использованием готовой реализации алгоритма APriori

Листинг Д – Файл dataset\_manager.py

from dataset\_generator import (

    DatasetGenerator,

    GenerationMode,

    EmptyGenerationSetException,

    GenerationException,

)

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

from mlxtend.frequent\_patterns import apriori, association\_rules

def hot\_encode(x):

    if x == 0:

        return False

    return True

dataset\_generator = DatasetGenerator(

    "unifood.json",

    min\_order\_items=2,

    max\_order\_items=5,

    max\_order\_price=1000,

    allow\_duplicates=False,

    mode=GenerationMode.UNTIL\_SOLD,

)

dataset = dataset\_generator.generate\_dataset()

dataset = dataset.map(hot\_encode)

print(dataset.head())

frq\_items = apriori(dataset, min\_support=0.01, use\_colnames=True)

frq\_items.sort\_values(["support"], ascending=[False])

print(frq\_items.sort\_values(["support"], ascending=[False]))

rules = association\_rules(frq\_items, metric="lift", min\_threshold=1)

rules.sort\_values(["lift"], ascending=[False])

print(rules.sort\_values(["lift"], ascending=[False]))

rules\_random = rules.sample(10, random\_state=42)

rules\_lift = rules\_random[["lift"]].to\_numpy()

rules\_lift = (rules\_lift / rules\_lift.max()).transpose()[0]

rules\_conf = rules\_random[["confidence"]].to\_numpy()

rules\_conf = (rules\_conf / rules\_conf.max()).transpose()[0]

width = 0.40

plt.figure(figsize=(8, 4), dpi=200)

plt.bar(np.arange(len(rules\_random)) - 0.2, rules\_lift, width, color="black")

plt.bar(

    np.arange(len(rules\_random)) + 0.2,

    rules\_conf,

    width,

    hatch="//",

    edgecolor="black",

    facecolor="white",

Окончание Листинга Д

)

plt.xlabel("Instance index")

plt.ylabel("Normalized metric value")

plt.legend(["lift", "confidence"])

plt.xticks(range(0, 10))

plt.show()

### Приложение Е

Файл APriori\_custom.py с самописной реализацией алгоритма APriori

Листинг Е – APriori\_custom.py

from dataset\_generator import (

    DatasetGenerator,

    GenerationMode,

    EmptyGenerationSetException,

    GenerationException,

)

from enum import Enum

from math import inf

from dataclasses import dataclass

from itertools import combinations

from typing import Optional, Set, FrozenSet, List

import pandas as pd

class Metric(str, Enum):

    """Метрики качества ассоциативных правил."""

    SUPPORT = "support"

    CONFIDENCE = "confidence"

    CONVICTION = "conviction"

    LIFT = "lift"

    LEVERAGE = "leverage"

@dataclass(frozen=True)

class RuleFilter:

    """

    Описывает одно условие фильтрации:

      metric — по какой метрике фильтруем,

      min\_threshold — минимально допустимое значение (или None),

      max\_threshold — максимально допустимое значение (или None).

    """

    metric: Metric

    min\_threshold: Optional[float] = None

    max\_threshold: Optional[float] = None

class APriori:

    def \_\_init\_\_(self, dataset: pd.DataFrame) -> None:

        """

        Инициализирует датасет для поиска частых наборов и построения правил.

        Параметры:

            dataset (pd.DataFrame): one-hot–кодированный DataFrame транзакций,

                где столбцы — товары, строки — транзакции, значения 0/1 или False/True.

        """

        self.dataset = dataset

    def apriori(

        self, min\_support: float = 0.25, max\_len: Optional[int] = None

    ) -> pd.DataFrame:

        """

        Находит частые наборы элементов методом Apriori.

        Параметры:

            min\_support (float): минимальная поддержка в диапазоне [0.0, 1.0].

            max\_len (Optional[int]): максимальный размер наборов (None — без ограничения).

Продолжение Листинга Е

        Возвращает:

            pd.DataFrame с колонками:

                - itemsets (frozenset): частый набор элементов,

                - support (float): доля транзакций, содержащих этот набор.

        """

        if not isinstance(min\_support, (float, int)):

            raise TypeError("Минимальное значение support должно быть числом")

        if min\_support < 0 or min\_support > 1:

            raise ValueError(

                "Минимальное значение support должно находиться в диапазоне [0;1]"

            )

        if max\_len is not None:

            if not isinstance(max\_len, int):

                raise TypeError("Значение max\_len должно быть целым числом")

            if max\_len < 0:

                raise ValueError("Значение max\_len должно быть положительным числом")

        df = self.dataset

        n\_transactions = len(df)

        support\_data = {}

        L = []

        for col in df.columns:

            sup = df[col].sum() / n\_transactions

            if sup >= min\_support:

                itemset = frozenset([col])

                support\_data[itemset] = sup

                L.append({col})

        k = 2

        prev\_L = L

        while prev\_L and (max\_len is None or k <= max\_len):

            Ck = set()

            for i in range(len(prev\_L)):

                for j in range(i + 1, len(prev\_L)):

                    union\_set = prev\_L[i] | prev\_L[j]

                    if len(union\_set) == k:

                        subsets = combinations(union\_set, k - 1)

                        if all(

                            frozenset(sub) in map(frozenset, prev\_L) for sub in subsets

                        ):

                            Ck.add(frozenset(union\_set))

            next\_L = []

            for candidate in Ck:

                mask = df[list(candidate)].all(axis=1)

                sup = mask.sum() / n\_transactions

                if sup >= min\_support:

                    support\_data[candidate] = sup

                    next\_L.append(set(candidate))

            prev\_L = next\_L

            k += 1

        result = pd.DataFrame(

            [

Продолжение Листинга Е

                {"itemsets": itemset, "support": support}

                for itemset, support in support\_data.items()

            ]

        )

        return result

    def association\_rules(

        self, frequent\_itemsets: pd.DataFrame, filters: List[RuleFilter]

    ) -> pd.DataFrame:

        """

        Строит ассоциативные правила из частых наборов и фильтрует их.

        Параметры:

            frequent\_itemsets (pd.DataFrame): результат apriori,

                колонки itemsets (frozenset) и support (float).

            filters (List[RuleFilter]): список условий фильтрации по метрикам.

        Возвращает:

            pd.DataFrame с колонками:

                - antecedents (frozenset)

                - consequents (frozenset)

                - support (float)

                - confidence (float)

                - lift (float)

                - leverage (float)

                - conviction (float)

            и только теми строками, которые проходят все фильтры.

        """

        df\_bool = self.dataset

        support\_map = {

            frozenset(row["itemsets"]): float(row["support"])

            for \_, row in frequent\_itemsets.iterrows()

        }

        records = []

        for itemset, support\_AB in support\_map.items():

            if len(itemset) < 2:

                continue

            for r in range(1, len(itemset)):

                for antecedent in combinations(itemset, r):

                    X = frozenset(antecedent)

                    Y = itemset - X

                    support\_A = support\_map.get(X)

                    support\_B = support\_map.get(Y)

                    if support\_B is None:

                        mask\_B = df\_bool[list(Y)].all(axis=1)

                        support\_B = float(mask\_B.sum()) / len(df\_bool)

                    confidence = support\_AB / support\_A

                    lift = confidence / support\_B

                    leverage = support\_AB - support\_A \* support\_B

                    conviction = (

                        (1 - support\_B) / (1 - confidence) if confidence != 1 else inf

                    )

                    records.append(

                        {

                            "antecedents": X,

                            "consequents": Y,

Окончание Листинга Е

                            "support": support\_AB,

                            "confidence": confidence,

                            "lift": lift,

                            "leverage": leverage,

                            "conviction": conviction,

                        }

                    )

        rules\_df = pd.DataFrame(records)

        for f in filters:

            col = f.metric.value

            if f.min\_threshold is not None:

                rules\_df = rules\_df[rules\_df[col] >= f.min\_threshold]

            if f.max\_threshold is not None:

                rules\_df = rules\_df[rules\_df[col] <= f.max\_threshold]

        return rules\_df

dataset\_generator = DatasetGenerator(

    "unifood.json",

    min\_order\_items=2,

    max\_order\_items=5,

    max\_order\_price=1000,

    allow\_duplicates=False,

    mode=GenerationMode.UNTIL\_SOLD,

)

dataset = dataset\_generator.generate\_dataset()

apriori = APriori(dataset)

frequent\_itemsets = apriori.apriori(min\_support=0.01)

frequent\_itemsets.sort\_values(["support"], ascending=[False])

print(frequent\_itemsets.sort\_values(["support"], ascending=[False]))

filters = [RuleFilter(Metric.LIFT, min\_threshold=1)]

rules = apriori.association\_rules(frequent\_itemsets, filters)

print(rules)

### Приложение Ж

Файл Eclat.py с самописной реализацией алгоритма Eclat

Листинг Ж – Eclat.py

from dataclasses import dataclass

from enum import Enum

from itertools import combinations

from math import inf

from typing import Optional, List, Set

import pandas as pd

from dataset\_generator import (

    DatasetGenerator,

    GenerationMode,

    EmptyGenerationSetException,

    GenerationException,

)

class Metric(str, Enum):

    """Метрики качества ассоциативных правил."""

    SUPPORT = "support"

    CONFIDENCE = "confidence"

    CONVICTION = "conviction"

    LIFT = "lift"

    LEVERAGE = "leverage"

@dataclass(frozen=True)

class RuleFilter:

    """

    Описывает одно условие фильтрации:

      metric — по какой метрике фильтруем,

      min\_threshold — минимально допустимое значение (или None),

      max\_threshold — максимально допустимое значение (или None).

    """

    metric: Metric

    min\_threshold: Optional[float] = None

    max\_threshold: Optional[float] = None

class Eclat:

    def \_\_init\_\_(self, dataset: pd.DataFrame) -> None:

        """

        Инициализирует объект для поиска частых наборов методом Eclat.

        Параметры:

            dataset (pd.DataFrame): one-hot–кодированный DataFrame транзакций,

                где столбцы — товары, строки — транзакции, значения 0/1 или False/True.

        """

        self.dataset = dataset > 0

    def eclat(

        self, min\_support: float = 0.25, max\_len: Optional[int] = None

    ) -> pd.DataFrame:

        """

        Находит частые наборы элементов методом Eclat (вертикальное хранение).

        Параметры:

Продолжение Листинга Ж

            min\_support (float): нижняя граница поддержки в [0.0, 1.0].

            max\_len (Optional[int]): максимальная длина наборов (None — без ограничения).

        Возвращает:

            pd.DataFrame с колонками:

                - itemsets (frozenset): частый набор элементов

                - support (float): доля транзакций, содержащих набор

        """

        if not isinstance(min\_support, (float, int)):

            raise TypeError("min\_support должен быть числом")

        if not 0 <= min\_support <= 1:

            raise ValueError("min\_support должен быть в диапазоне [0;1]")

        if max\_len is not None:

            if not isinstance(max\_len, int):

                raise TypeError("max\_len должен быть целым или None")

            if max\_len < 1:

                raise ValueError("max\_len должен быть ≥1 или None")

        n\_transactions = len(self.dataset)

        tid\_lists = {}

        for col in self.dataset.columns:

            tids = set(self.dataset.index[self.dataset[col]].tolist())

            sup = len(tids) / n\_transactions

            if sup >= min\_support:

                tid\_lists[col] = tids

        support\_data = {}

        for item, tids in tid\_lists.items():

            support\_data[frozenset([item])] = len(tids) / n\_transactions

        def dfs(prefix: List[str], prefix\_tids: Set[int], items: List[str]) -> None:

            """

            prefix — текущий набор (список товаров),

            prefix\_tids — пересечённый TID-list,

            items — оставшиеся кандидаты для расширения

            """

            for i, item in enumerate(items):

                new\_prefix = prefix + [item]

                new\_tids = prefix\_tids & tid\_lists[item] if prefix else tid\_lists[item]

                sup = len(new\_tids) / n\_transactions

                if sup < min\_support:

                    continue

                fs = frozenset(new\_prefix)

                support\_data[fs] = sup

                if max\_len is None or len(new\_prefix) < max\_len:

                    dfs(new\_prefix, new\_tids, items[i + 1 :])

        items = sorted(tid\_lists.keys())

        dfs([], set(), items)

        result = pd.DataFrame(

            [

                {"itemsets": itemset, "support": support}

                for itemset, support in support\_data.items()

            ]

        )

        return result

Продолжение Листинга Ж

    def association\_rules(

        self, frequent\_itemsets: pd.DataFrame, filters: List[RuleFilter]

    ) -> pd.DataFrame:

        """

        Строит ассоциативные правила из частых наборов и фильтрует их.

        Параметры:

            frequent\_itemsets (pd.DataFrame): результат apriori,

                колонки itemsets (frozenset) и support (float).

            filters (List[RuleFilter]): список условий фильтрации по метрикам.

        Возвращает:

            pd.DataFrame с колонками:

                - antecedents (frozenset)

                - consequents (frozenset)

                - support (float)

                - confidence (float)

                - lift (float)

                - leverage (float)

                - conviction (float)

            и только теми строками, которые проходят все фильтры.

        """

        df\_bool = self.dataset

        support\_map = {

            frozenset(row["itemsets"]): float(row["support"])

            for \_, row in frequent\_itemsets.iterrows()

        }

        records = []

        for itemset, support\_AB in support\_map.items():

            if len(itemset) < 2:

                continue

            for r in range(1, len(itemset)):

                for antecedent in combinations(itemset, r):

                    X = frozenset(antecedent)

                    Y = itemset - X

                    support\_A = support\_map.get(X)

                    support\_B = support\_map.get(Y)

                    if support\_B is None:

                        mask\_B = df\_bool[list(Y)].all(axis=1)

                        support\_B = float(mask\_B.sum()) / len(df\_bool)

                    confidence = support\_AB / support\_A

                    lift = confidence / support\_B

                    leverage = support\_AB - support\_A \* support\_B

                    conviction = (

                        (1 - support\_B) / (1 - confidence) if confidence != 1 else inf

                    )

                    records.append(

                        {

                            "antecedents": X,

                            "consequents": Y,

                            "support": support\_AB,

                            "confidence": confidence,

                            "lift": lift,

                            "leverage": leverage,

                            "conviction": conviction,

                        }

Окончание Листинга Ж

                    )

        rules\_df = pd.DataFrame(records)

        for f in filters:

            col = f.metric.value

            if f.min\_threshold is not None:

                rules\_df = rules\_df[rules\_df[col] >= f.min\_threshold]

            if f.max\_threshold is not None:

                rules\_df = rules\_df[rules\_df[col] <= f.max\_threshold]

        return rules\_df

dataset\_generator = DatasetGenerator(

    "unifood.json",

    min\_order\_items=2,

    max\_order\_items=5,

    max\_order\_price=1000,

    allow\_duplicates=False,

    mode=GenerationMode.UNTIL\_SOLD,

)

dataset = dataset\_generator.generate\_dataset()

eclat = Eclat(dataset)

frequent\_itemsets = eclat.eclat(min\_support=0.01)

frequent\_itemsets.sort\_values(["support"], ascending=[False])

print(frequent\_itemsets.sort\_values(["support"], ascending=[False]))

filters = [RuleFilter(Metric.LIFT, min\_threshold=1)]

rules = eclat.association\_rules(frequent\_itemsets, filters)

print(rules)