



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
1 ПОСТАНОВКА ЗАДАЧИ .....	4
2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ .....	5
2.1 Алгоритм APriori .....	7
2.2 Алгоритм Eclat .....	8
3 ДОКУМЕНТАЦИЯ К ДАННЫМ .....	10
3.1 Описание предметной области .....	10
3.2 Генерация данных .....	12
3.3 Анализ полученных данных .....	14
4 ПРАКТИЧЕСКАЯ ЧАСТЬ .....	17
4.1 Алгоритм APriori .....	17
4.2 Алгоритм Eclat .....	21
ЗАКЛЮЧЕНИЕ .....	22
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ .....	23
ПРИЛОЖЕНИЯ .....	24

## **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  $X \rightarrow Y$ , 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  $\rightarrow$  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  $d_i$  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).

$$supp(X \rightarrow Y) = P(X \cap Y) = \frac{\text{количество транзакций, содержащих } X \text{ и } Y}{\text{общее количество транзакций}} = \frac{|X \cap Y|}{|T|} \quad (2.1)$$

## 2. 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.

$$conf(X \rightarrow Y) = \frac{P(X \cap Y)}{P(X)} = \frac{\text{количество транзакций, содержащих } X \text{ и } Y}{\text{количество транзакций, содержащих только } X} = \frac{|X \cap Y|}{|X|} \quad (2.2)$$

## 3. Убежденность, убедительность (conviction)

Метрика оценивает, насколько правило  $X \rightarrow Y$  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  $X \rightarrow Y$  (Формула 2.3).

$$conv(X \rightarrow Y) = \frac{1 - \text{supp}(Y)}{1 - \text{conf}(X \rightarrow Y)} = \frac{\text{supp}(X) * \text{supp}(\neg Y)}{\text{supp}(X \cap \neg Y)} \quad (2.3)$$

## 4. 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.

$$lift(X \rightarrow Y) = \frac{P(X \cap Y)}{P(X) * P(Y)} = \frac{supp(X \rightarrow Y)}{supp(X) * supp(Y)} \quad (2.4)$$

## 5. 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.

$$leverage(X \rightarrow Y) = supp(X \cap Y) - supp(X) * supp(Y) \quad (2.5)$$

## 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  $X \rightarrow Y$ , 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 отсекалть большую часть кандидатов на каждом этапе. Алгоритм состоит из следующих шагов:

1. Вычислитель поддержку для каждого одиночного элемента.
2. Отобравть все элементарные наборы с поддержкой больше  $min\_support$ .
3. Объединить попарно все наборы из  $L_{k-1}$ , чтобы получить кандидатов размера  $k$ .
4. Отсеять кандидатов, у которых хотя бы одно  $(k-1)$  подмножество не

содержится в  $L_{k-1}$ .

5. Вычислить `support` и оставить тех кандидатов, поддержка которых больше `min_support`.
6. Повторить шаги 3–5 для  $k = 2, 3, \dots n$ .
7. Продолжать генерировать наборы  $L_1, L_2$  пока  $L_k$  не пуст.
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  $X \rightarrow Y$  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 двумерной матрицы «транзакция  $\times$  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:

3. Построить пересечение:  $TID(d \cup \{j\}) = TID(d) \cap TID(\{j\})$ .
4. Вычислить поддержку нового набора, если она превышает  $\min\_support$ , зафиксировать набор как частый.
5. Рекурсивно расширять набор, добавляя только тех кандидатов, которые следуют после  $j$  в исходном порядке (чтобы избежать повторов).
6. Объединить все наборы в результирующий набор частых предметов (items).
7. 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.

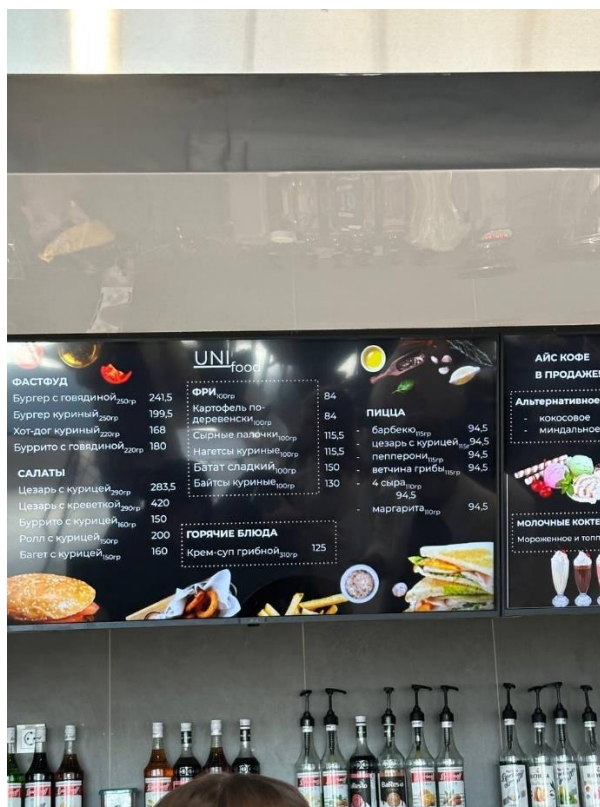


Рисунок 3.1.2 – Фастфуд в продаже

Фастфуд – основной и самый продаваемый раздел рассматриваемой сети питания.

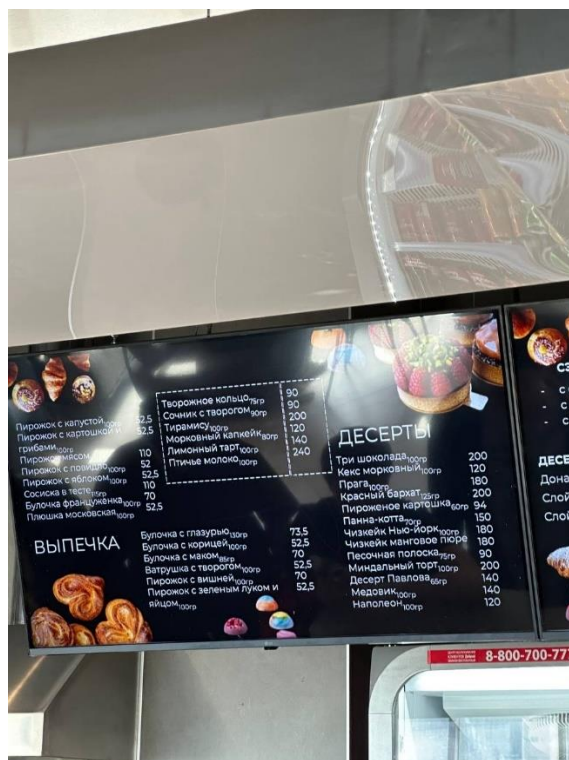


Рисунок 3.1.3 – Десерты и выпечка в продаже

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.

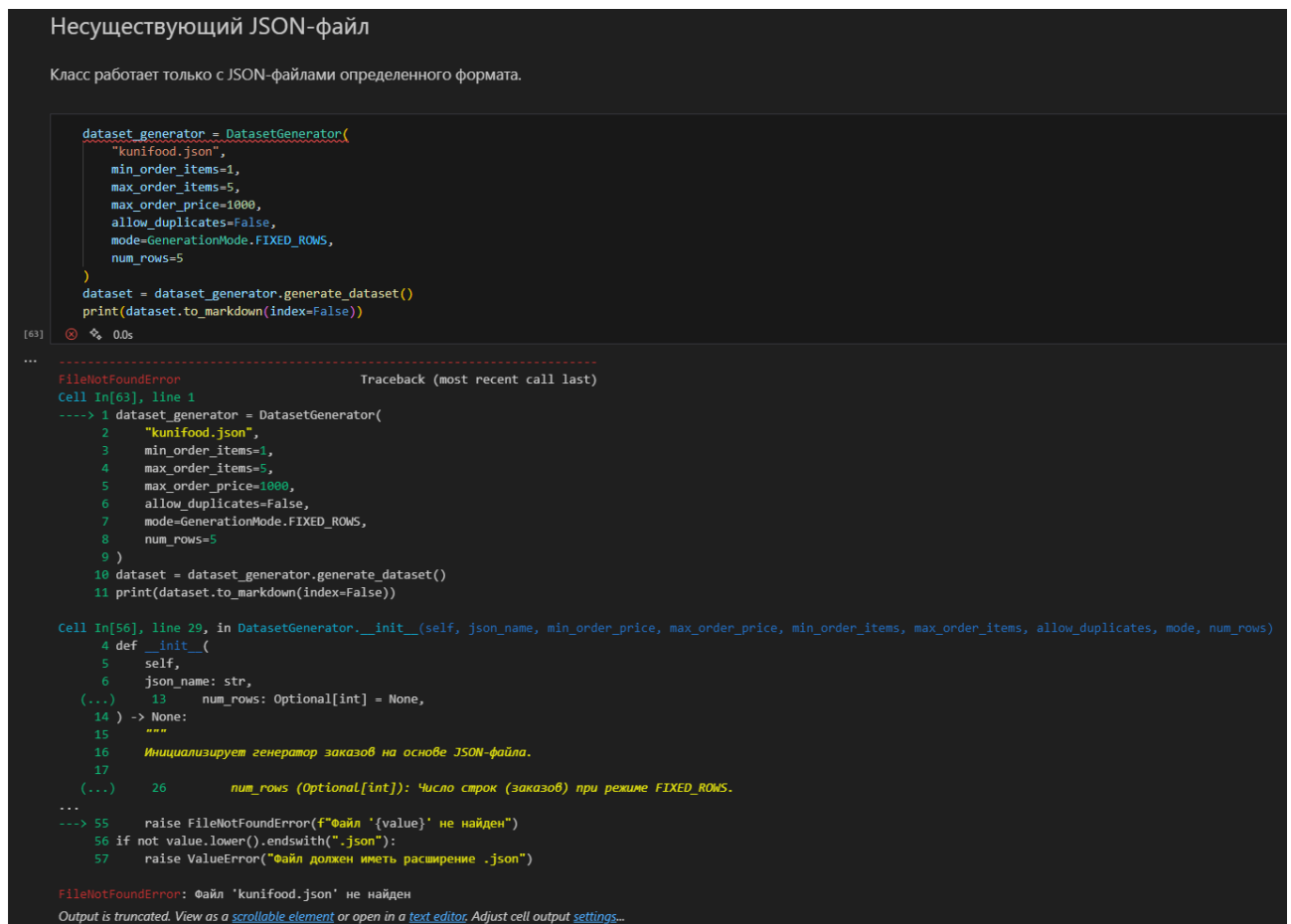
## 3.2 Генерация данных

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.



The screenshot shows a Jupyter Notebook interface. At the top, there is a title "Несуществующий JSON-файл" (Non-existent JSON file) and a note "Класс работает только с JSON-файлами определенного формата." (The class works only with JSON files of a specific format). Below this, a code cell contains the following Python code:

```
dataset_generator = DatasetGenerator(  
    "kunifood.json",  
    min_order_items=1,  
    max_order_items=5,  
    max_order_price=1000,  
    allow_duplicates=False,  
    mode=GenerationMode.FIXED_ROWS,  
    num_rows=5  
)  
dataset = dataset_generator.generate_dataset()  
print(dataset.to_markdown(index=False))
```

The output of the cell shows a `FileNotFoundError` traceback. The error message is "FileNotFoundError: Файл 'kunifood.json' не найден" (FileNotFoundError: File 'kunifood.json' not found). The traceback shows the error occurred in `Cell In[63], line 1` at the first line of the code cell. The notebook also shows the definition of the `DatasetGenerator` class in `Cell In[56]`, which includes a docstring in Russian: "Инициализирует генератор заказов на основе JSON-файла." (Initializes the order generator based on a JSON file).

**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.

[illegible]

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.

### 3.3 Анализ полученных данных

Общая информация о датасете отражена на Рисунке 3.3.1.

```

Число транзакций: 22745
Число признаков: 74
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22745 entries, 0 to 22744
Data columns (total 74 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   Бургер с говядиной                       22745 non-null  int64
1   Бургер куриный                           22745 non-null  int64
2   Хот-дог куриный                         22745 non-null  int64
3   Булочка с глазурью                      22745 non-null  int64
4   Булочка с корицей                       22745 non-null  int64
5   Булочка с маком                         22745 non-null  int64
6   Ватрушка с творогом                     22745 non-null  int64
7   Пирожок с вишней                        22745 non-null  int64
8   Пирожок с зеленым луком и яйцом        22745 non-null  int64
9   Пирожок с капустой                     22745 non-null  int64
10  Пирожок с картошкой и грибами          22745 non-null  int64
11  Пирожок с мясом                         22745 non-null  int64
12  Пирожок с повидлом                     22745 non-null  int64
13  Пирожок с яблоком                       22745 non-null  int64
14  Сосиска в тесте                         22745 non-null  int64
15  Булочка французенка                     22745 non-null  int64
16  Плюшка московская                       22745 non-null  int64
17  Десерт Павлова                           22745 non-null  int64
...
72  Яблочный с сельдереем фреш              22745 non-null  int64
73  Яблочный фреш                           22745 non-null  int64
dtypes: int64(74)
memory usage: 12.8 MB

```

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.

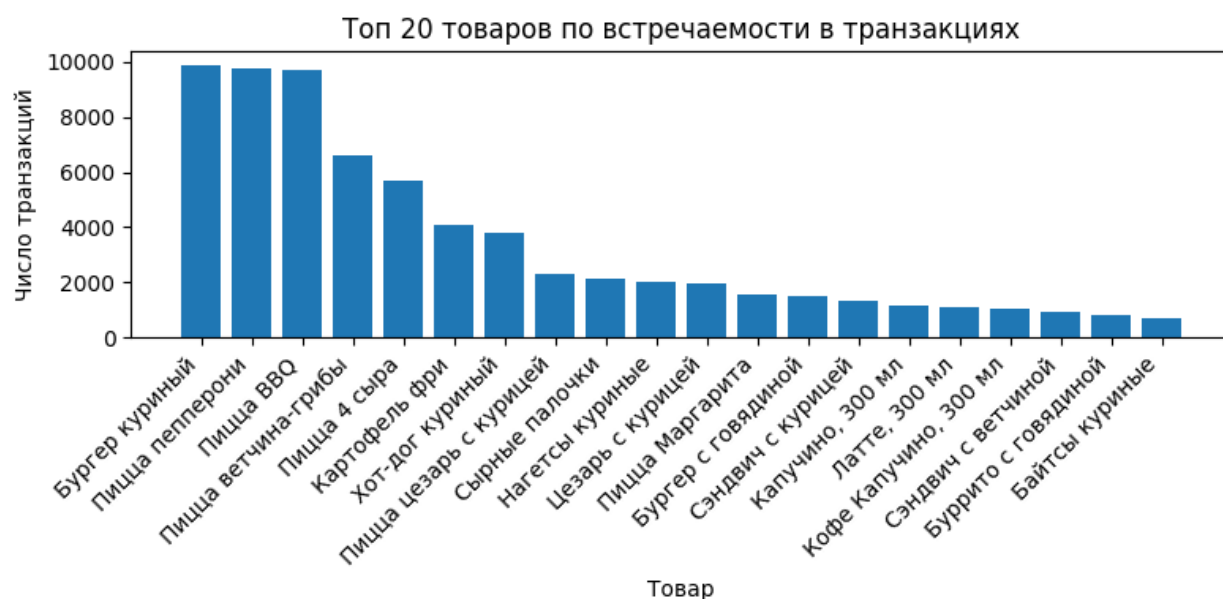


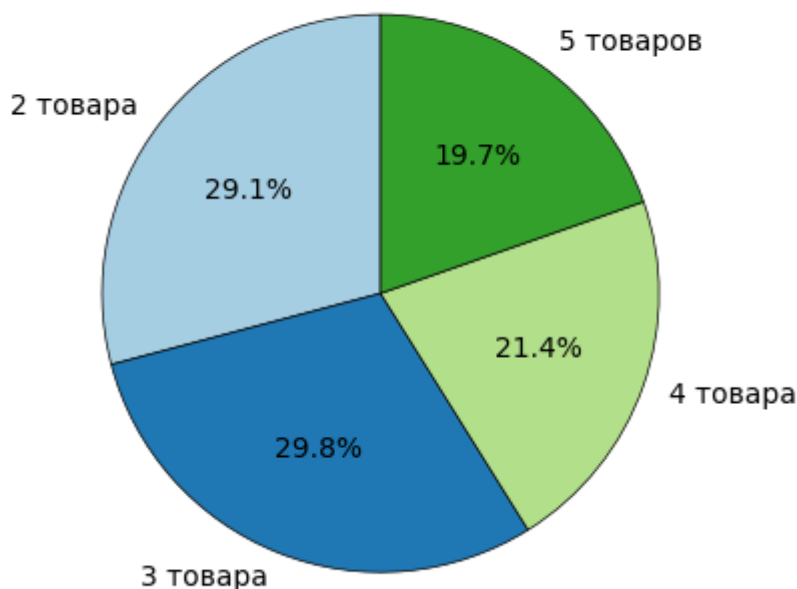
Рисунок 3.3.2 – Топ 20 самых популярных товаров по встречаемости в транзакциях



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.

**Распределение размеров корзин**



**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).

	Бургер с говядиной	Бургер куриный	Хот-дог куриный	Булочка с глазурью	Булочка с корицей	Булочка с маком	Ватрушка с творогом	Пирожок с вишней	Пирожок с зеленым луком и яйцом	Пирожок с капустой	—	Сырные палочки	Сэндвич с семгой	Сэндвич с курицей	Сэндвич с ветчиной	Апельсиновый фреш	Морковно-яблочный фреш	Морковный фреш	Яблочно-апельсиновый фреш	Яблочный с сельдереем фреш	Яблочный фреш
0	True	False	False	False	False	False	False	False	False	False	—	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	—	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	—	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	True	False	False	False	False	False	—	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	—	False	False	False	False	False	False	False	False	False	False

5 rows x 24 columns

Рисунок 4.1.1 – Обработанный датасет

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

	support	itemsets
1	0.433942	(Бургер куриный)
17	0.428094	(Пицца пепперони)
14	0.426643	(Пицца BBQ)
18	0.289250	(Пицца ветчина-грибы)
40	0.254781	(Бургер куриный, Пицца BBQ)
...	...	...
33	0.010332	(Бургер с говядиной, Пицца ветчина-грибы)
42	0.010244	(Бургер куриный, Пицца Маргарита)
37	0.010156	(Нагетсы куриные, Бургер с говядиной)
38	0.010156	(Сырные палочки, Бургер с говядиной)
34	0.010024	(Бургер с говядиной, Пицца цезарь с курицей)

146 rows x 2 columns

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).

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_metric	jaccard	certainty	kulczynski
292	(Бургер куриный, Пицца ветчина-грибы)	(Пицца пепперони, Пицца 4 сыра, Пицца BBQ)	0.140207	0.061596	0.029721	0.211979	3.441438	1.0	0.021085	1.190836	0.825110	0.172713	0.160254	0.347246
285	(Пицца пепперони, Пицца 4 сыра, Пицца BBQ)	(Бургер куриный, Пицца ветчина-грибы)	0.061596	0.140207	0.029721	0.482512	3.441438	1.0	0.021085	1.661477	0.755990	0.172713	0.398126	0.347246
295	(Пицца ветчина-грибы, Пицца BBQ)	(Пицца пепперони, Пицца 4 сыра, Бургер куриный)	0.136865	0.064234	0.029721	0.217154	3.380674	1.0	0.020929	1.195338	0.815864	0.173422	0.163417	0.339925
282	(Пицца пепперони, Пицца 4 сыра, Бургер куриный)	(Пицца ветчина-грибы, Пицца BBQ)	0.064234	0.136865	0.029721	0.462697	3.380674	1.0	0.020929	1.606420	0.752540	0.173422	0.377498	0.339925
297	(Пицца пепперони, Пицца ветчина-грибы)	(Пицца 4 сыра, Бургер куриный, Пицца BBQ)	0.141965	0.064630	0.029721	0.209353	3.239271	1.0	0.020546	1.183044	0.805665	0.168034	0.154723	0.334608
21	(Хот-дог куриный)	(Пицца Маргарита)	0.166586	0.069070	0.011563	0.069411	1.004942	1.0	0.000057	1.000367	0.005900	0.051599	0.000367	0.118410
131	(Хот-дог куриный)	(Пицца 4 сыра, Пицца BBQ)	0.166586	0.112596	0.018817	0.112959	1.003218	1.0	0.000060	1.000409	0.003849	0.072273	0.000408	0.140040
128	(Пицца 4 сыра, Пицца BBQ)	(Хот-дог куриный)	0.112596	0.166586	0.018817	0.167122	1.003218	1.0	0.000060	1.000644	0.003615	0.072273	0.000643	0.140040
143	(Пицца 4 сыра, Пицца ветчина-грибы)	(Хот-дог куриный)	0.112904	0.166586	0.018817	0.166667	1.000484	1.0	0.000009	1.000097	0.000545	0.072188	0.000097	0.139813
146	(Хот-дог куриный)	(Пицца 4 сыра, Пицца ветчина-грибы)	0.166586	0.112904	0.018817	0.112959	1.000484	1.0	0.000009	1.000062	0.000580	0.072188	0.000062	0.139813

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).

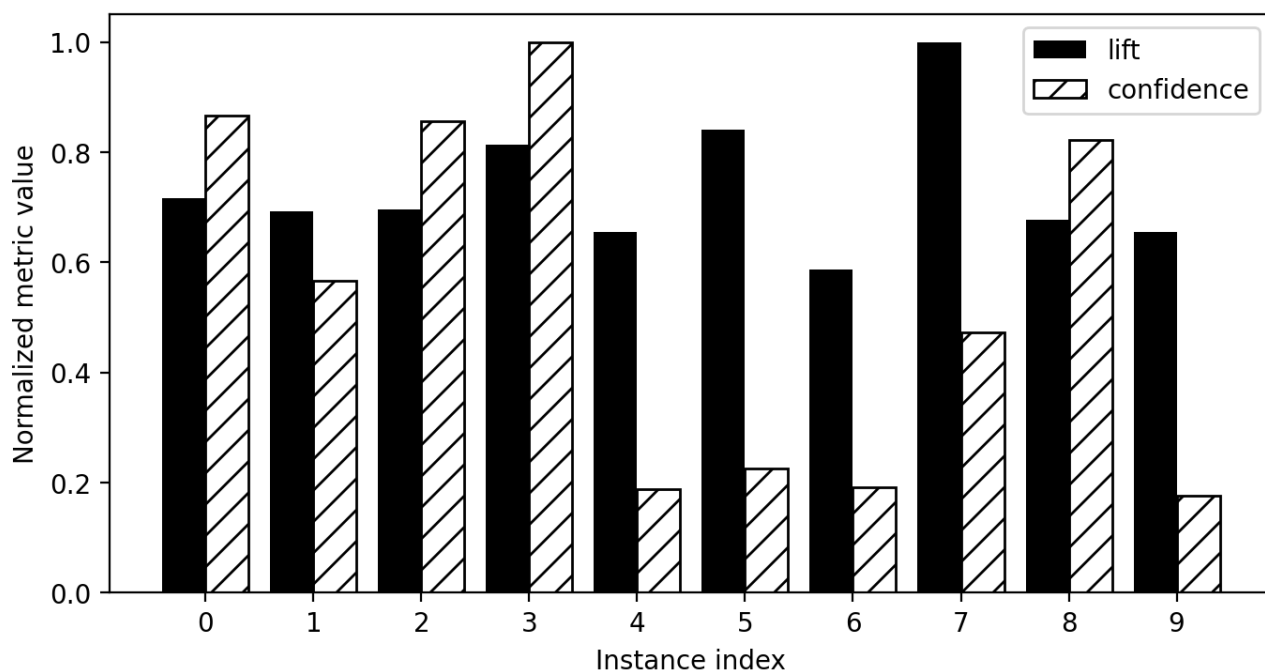


Рисунок 4.1.4 – Визуализация метрик lift, confidence для случайных десяти правил

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  $X \rightarrow Y$  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.

	itemsets	support
1	(Бургер куриный)	0.433942
17	(Пицца пепперони)	0.428094
14	(Пицца BBQ)	0.426643
18	(Пицца ветчина-грибы)	0.289250
95	(Бургер куриный, Пицца BBQ)	0.254781
...	...	...
59	(Бургер с говядиной, Пицца ветчина-грибы)	0.010332
41	(Бургер куриный, Пицца Маргарита)	0.010244
48	(Сырные палочки, Бургер с говядиной)	0.010156
56	(Нагетсы куриные, Бургер с говядиной)	0.010156
35	(Бургер с говядиной, Пицца цезарь с курицей)	0.010024
146 rows × 2 columns		

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.

	antecedents	consequents	support	confidence	lift	leverage	conviction
448	(Бургер куриный, Пицца ветчина-грибы)	(Пицца пепперони, Пицца 4 сыра, Пицца BBQ)	0.029721	0.211979	3.441438	0.021085	1.190836
449	(Пицца пепперони, Пицца 4 сыра, Пицца BBQ)	(Бургер куриный, Пицца ветчина-грибы)	0.029721	0.482512	3.441438	0.021085	1.661477
455	(Пицца пепперони, Пицца 4 сыра, Бургер куриный)	(Пицца ветчина-грибы, Пицца BBQ)	0.029721	0.462697	3.380674	0.020929	1.606420
442	(Пицца ветчина-грибы, Пицца BBQ)	(Пицца пепперони, Пицца 4 сыра, Бургер куриный)	0.029721	0.217154	3.380674	0.020929	1.195338
445	(Пицца пепперони, Пицца ветчина-грибы)	(Пицца 4 сыра, Бургер куриный, Пицца BBQ)	0.029721	0.209353	3.239271	0.020546	1.183044
...	...	...	...	...	...	...	...
87	(Хот-дог куриный)	(Пицца Маргарита)	0.011563	0.069411	1.004942	0.000057	1.000367
161	(Хот-дог куриный)	(Пицца 4 сыра, Пицца BBQ)	0.018817	0.112959	1.003218	0.000060	1.000409
164	(Пицца 4 сыра, Пицца BBQ)	(Хот-дог куриный)	0.018817	0.167122	1.003218	0.000060	1.000644
305	(Хот-дог куриный)	(Пицца 4 сыра, Пицца ветчина-грибы)	0.018817	0.112959	1.000484	0.000009	1.000062
308	(Пицца 4 сыра, Пицца ветчина-грибы)	(Хот-дог куриный)	0.018817	0.166667	1.000484	0.000009	1.000097
304 rows × 7 columns							

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).

```
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
```

Рисунок 4.1.7 – Класс для фильтрации правил

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

	antecedents	consequents	support	confidence	lift	leverage	conviction
271	(Хот-дого куриный, Пицца ветчина-грибы)	(Картофель фри)	0.019609	0.364379	2.026358	0.009932	1.290361
157	(Пицца 4 сыра, Хот-дого куриный)	(Картофель фри)	0.018378	0.357877	1.990197	0.009144	1.277294
229	(Бургер куриный, Хот-дого куриный)	(Картофель фри)	0.018158	0.346477	1.926799	0.008734	1.255013
349	(Пицца пепперони, Хот-дого куриный)	(Картофель фри)	0.018246	0.346411	1.926433	0.008774	1.254886
205	(Хот-дого куриный, Пицца BBQ)	(Картофель фри)	0.018685	0.346373	1.926225	0.008985	1.254814
273	(Пицца ветчина-грибы, Картофель фри)	(Хот-дого куриный)	0.019609	0.319943	1.920585	0.009399	1.225505
207	(Пицца BBQ, Картофель фри)	(Хот-дого куриный)	0.018685	0.318591	1.912469	0.008915	1.223074
350	(Пицца пепперони, Картофель фри)	(Хот-дого куриный)	0.018246	0.313918	1.884421	0.008563	1.214744
158	(Пицца 4 сыра, Картофель фри)	(Хот-дого куриный)	0.018378	0.311243	1.868364	0.008541	1.210027
230	(Бургер куриный, Картофель фри)	(Хот-дого куриный)	0.018158	0.296482	1.779755	0.007955	1.184638
254	(Пицца пепперони, Хот-дого куриный)	(Пицца 4 сыра)	0.018993	0.360601	1.442721	0.005828	1.173062
321	(Бургер куриный, Хот-дого куриный)	(Пицца 4 сыра)	0.018554	0.354027	1.416419	0.005455	1.161124
159	(Хот-дого куриный, Картофель фри)	(Пицца 4 сыра)	0.018378	0.352148	1.408903	0.005334	1.157757
309	(Хот-дого куриный, Пицца ветчина-грибы)	(Пицца 4 сыра)	0.018817	0.349673	1.399000	0.005367	1.153351
165	(Хот-дого куриный, Пицца BBQ)	(Пицца 4 сыра)	0.018817	0.348818	1.395580	0.005334	1.151837
325	(Бургер куриный, Хот-дого куриный)	(Пицца ветчина-грибы)	0.019829	0.378356	1.308056	0.004670	1.143338
169	(Пицца пепперони, Хот-дого куриный)	(Пицца ветчина-грибы)	0.019872	0.377295	1.304391	0.004637	1.141391
272	(Хот-дого куриный, Картофель фри)	(Пицца ветчина-грибы)	0.019609	0.375737	1.299003	0.004514	1.138542
307	(Пицца 4 сыра, Хот-дого куриный)	(Пицца ветчина-грибы)	0.018817	0.366438	1.266855	0.003964	1.121832
266	(Хот-дого куриный, Пицца BBQ)	(Пицца ветчина-грибы)	0.018114	0.335778	1.160857	0.002510	1.070049

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.

	itemsets	support
1	(Бургер куриный)	0.433942
17	(Пицца пепперони)	0.428094
14	(Пицца BBQ)	0.426643
18	(Пицца ветчина-грибы)	0.289250
45	(Бургер куриный, Пицца BBQ)	0.254781
...	...	...
65	(Бургер с говядиной, Пицца ветчина-грибы)	0.010332
50	(Бургер куриный, Пицца Маргарита)	0.010244
61	(Нагетсы куриные, Бургер с говядиной)	0.010156
68	(Сырные палочки, Бургер с говядиной)	0.010156
67	(Бургер с говядиной, Пицца цезарь с курицей)	0.010024
146 rows × 2 columns		

Рисунок 4.2.1 – Найденные частые наборы

Извлеченные ассоциативные правила представлены на Рисунке 4.2.2.

	antecedents	consequents	support	confidence	lift	leverage	conviction
66	(Бургер куриный, Пицца ветчина-грибы)	(Пицца пепперони, Пицца 4 сыра, Пицца BBQ)	0.029721	0.211979	3.441438	0.021085	1.190836
79	(Пицца пепперони, Пицца 4 сыра, Пицца BBQ)	(Бургер куриный, Пицца ветчина-грибы)	0.029721	0.482512	3.441438	0.021085	1.661477
76	(Пицца пепперони, Пицца 4 сыра, Бургер куриный)	(Пицца ветчина-грибы, Пицца BBQ)	0.029721	0.462697	3.380674	0.020929	1.606420
69	(Пицца ветчина-грибы, Пицца BBQ)	(Пицца пепперони, Пицца 4 сыра, Бургер куриный)	0.029721	0.217154	3.380674	0.020929	1.195338
74	(Пицца 4 сыра, Бургер куриный, Пицца BBQ)	(Пицца пепперони, Пицца ветчина-грибы)	0.029721	0.459864	3.239271	0.020546	1.588553
...	...	...	...	...	...	...	...
425	(Хот-дог куриный)	(Пицца Маргарита)	0.011563	0.069411	1.004942	0.000057	1.000367
347	(Хот-дог куриный)	(Пицца 4 сыра, Пицца BBQ)	0.018817	0.112959	1.003218	0.000060	1.000409
350	(Пицца 4 сыра, Пицца BBQ)	(Хот-дог куриный)	0.018817	0.167122	1.003218	0.000060	1.000644
363	(Хот-дог куриный)	(Пицца 4 сыра, Пицца ветчина-грибы)	0.018817	0.112959	1.000484	0.000009	1.000062
366	(Пицца 4 сыра, Пицца ветчина-грибы)	(Хот-дог куриный)	0.018817	0.166667	1.000484	0.000009	1.000097
304 rows × 7 columns							

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  $\text{lift} > 1$  и  $\text{confidence} > 0.25$ , 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  $2\times$  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  $\times$  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 французенка",
    "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 ВВQ",
    "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
},
},
```



```
{
    "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.
```

```
"""

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
```

```
@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}")
```

```
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)
```

```

        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_idx = random.choice(available_indices, k=num_items)
            else:
                if num_items > len(available_indices):
                    continue
                chosen_idx = random.sample(available_indices, k=num_items)

            cnt = {}
            for i in chosen_idx:
                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).
        """
```

```

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{'a' 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(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).
        """
```

```

    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(
        [

```

```

        {"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,
            }
        )

```



```
        "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:
```

```

        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

```

```
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,
                }
            )
```

```
        )

    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)
```