АССОЦИАТИВНЫЕ ПРАВИЛА

Обучение на ассоциативных правилах (Associations rules learning — ARL) представляет из себя, с одной стороны, простой, с другой — довольно часто применимый в реальной жизни метод поиска взаимосвязей (ассоциаций) в датасетах, или, если точнее, айтемсетах (itemsests).

В общем виде ARL можно описать как «Кто купил x, также купил y». В основе лежит анализ транзакций, внутри каждой из которых лежит свой уникальный itemset из набора items. При помощи ARL алгоритмов находятся те самые «правила» совпадения items внутри одной транзакции, которые потом сортируются по их силе. Все, в общем, просто.

В 1992 году группа по консалтингу в области ритейла компании Teradata под руководством Томаса Блишока провела исследование 1.2 миллиона транзакций в 25 магазинах для ритейлера Osco Drug (Drug Store — формат разнокалиберных магазинов у дома). После анализа всех этих транзакций самым сильным правилом получилось «Между 17:00 и 19:00 чаще всего пиво и подгузники покупают вместе». К сожалению, такое правило показалось руководству Osco Drug настолько контринтуитивным, что ставить подгузники на полках рядом с пивом они не стали. Хотя объяснение паре пиво-подгузники вполне себе нашлось: когда оба члена молодой семьи возвращались с работы домой (как раз часам к 5 вечера), жены обычно отправляли мужей за подгузниками в ближайший магазин. И мужья, недолго думая, совмещали приятное с полезным — покупали подгузники по заданию жены и пиво для собственного вечернего времяпрепровождения.

Пусть у нас имеется некий датасет (или коллекция) *D,* такой, что 𝐷 =

𝑑O … 𝑑j , где *d* — уникальная транзакция-itemset (например, кассовый чек). Внутри каждой *d* представлен набор *items (i — item)*, причем в идеальном случае он представлен в бинарном виде:

*d1 = [{Пиво: 1}, {Вода: 0}, {Кола: 1}, {...}],*

*d2 = [{Пиво: 0}, {Вода: 1}, {Кола: 1}, {...}]*.

Принято каждый itemset описывать через количество ненулевых значений (*k-itemset*), например, *[{Пиво: 1}, {Вода: 0}, {Кола: 1}]* является *2- itemset*.

Если изначально датасет в бинарном виде не представлен, можно при желании его преобразовать.

Таким образом, датасет представляет собой разреженную матрицу со значениями {1,0}. Это будет бинарный датасет. Существуют и другие виды записи — вертикальный датасет (показывает для каждого отдельного item вектор транзакций, где он присутствует) и транзакционный датасет (примерно, как в кассовом чеке).

Существует целый ряд базовых понятий в ARL:

## Support (поддержка)

𝑠𝑢𝑝𝑝(𝑋) =

{𝑡 ∈ 𝑇; 𝑋 ∈ 𝑡}

|𝑇|

, где X — itemset, содержащий в себе i-items, а T — количество транзакций. Т.е. в общем виде это показатель «частотности» данного itemset во всех анализируемых транзакциях. Но это касается только X. Нам же интересен скорее вариант, когда у нас в одном itemset встречаются *x1* и *x2* (например). Пусть *x1 = {Пиво}, а x2 = {Подгузники}*, значит нам необходимо посчитать, во скольких транзакциях встречается эта пара.

𝑠𝑢𝑝𝑝(𝑥1

∪ 𝑥2

) = 𝜎(𝑥1 ∪ 𝑥2)

|𝑇|

где 𝜎 — количество транзакций, содержащих 𝑥1 и 𝑥2

Разберемся с этим понятием на небольшом примере.

Создадим датафрейм, где указаны номера транзакций, а также в бинарном виде представлено, что покупалось на каждой транзакции



𝑠𝑢𝑝𝑝 =

Транзакции с пивом и подгузниками Все транзакции

= 𝑃(Пиво ∩ Подгузники)

3

𝑠𝑢𝑝𝑝 =

5

= 60%

## Confidence (достоверность)

Следующее ключевое понятие — confidence. Это показатель того, как часто наше правило срабатывает для всего датасета.

𝑐𝑜𝑛𝑓(𝑥1

∪ 𝑥2

) = 𝑠𝑢𝑝𝑝(𝑥1 ∪ 𝑥2)

𝑠𝑢𝑝𝑝(𝑥1)

Приведем пример: мы хотим посчитать confidence для правила «кто покупает пиво, тот покупает и подгузники».

Для этого сначала посчитаем, какой support у правила «покупает пиво», потом посчитаем support у правила «покупает пиво и подгузники», и просто поделим одно на другое. Т.е. мы посчитаем в скольких случаях (транзакциях) срабатывает правило «купил пиво» *supp*(X), «купил подгузники и пиво».

𝑠𝑢𝑝𝑝(𝑋 ∪ 𝑌)

Проверим на нашем примере:

𝑠𝑢𝑝𝑝(Пиво ∩ Подгузники)

𝑐𝑜𝑛𝑓(Пиво ∩ Подгузники) =

𝑠𝑢𝑝𝑝(Пиво)

3

𝑐𝑜𝑛𝑓 = 4 = 75%

= 𝑃(Подгузники | Пиво)

## Lift (поддержка)

Следующее понятие в нашем списке — lift. Грубо говоря, lift — это отношение «зависимости» items к их «независимости». Lift показывает, насколько items зависят друг от друга. Это очевидно из формулы:

𝑙𝑖𝑓𝑡(𝑥1

∪ 𝑥2

) = 𝑠𝑢𝑝𝑝(𝑥1 ∪ 𝑥2)

𝑠𝑢𝑝𝑝(𝑥1) × 𝑠𝑢𝑝𝑝(𝑥2)

Например, мы хотим понять зависимость покупки пива и покупки подгузников. Для этого считаем support правила «купил пиво и подгузники» и делим его на произведение правил «купил пиво» и «купил подгузники». В случае, если *lift = 1*, мы говорим, что items независимы и правил совместной покупки тут нет. Если же *lift > 1*, то величина, на которую lift, собственно, больше этой самой единицы, и покажет нам «силу» правила. Чем больше единицы, тем лучше. Если *lift* < 1, то это покажет, что правило основания x2 негативно влияет на правило x1. По-другому lift можно определить как отношение confidence к expected confidence, т.е. отношение достоверности правила, когда оба (или больше) элемента покупаются вместе к достоверности правила, когда один из элементов покупался (неважно, со вторым или без).

𝐶𝑜𝑛𝑓𝑖𝑑𝑒𝑛𝑐𝑒

𝑙𝑖𝑓𝑡 = 𝐸𝑥𝑝𝑒𝑐𝑡𝑒𝑑 𝑐𝑜𝑛𝑓𝑖𝑑𝑒𝑛𝑐𝑒 =

3/4

𝑃(Подгузники | Пиво)

𝑃(Подгузники)

𝑙𝑖𝑓𝑡 = 3 = 1.25

/

5

Т.е. наше правило, что пиво покупают с подгузниками, на 25% мощнее правила, что подгузники просто покупают.

## Conviction (убедительность)

В общем виде Conviction — это «частотность ошибок» нашего правила.

Т.е., например, как часто покупали пиво без подгузников и наоборот.

𝑐𝑜𝑛𝑣(𝑥1

∪ 𝑥2

) = 1 − 𝑠𝑢𝑝𝑝(𝑥2)

1 − 𝑐𝑜𝑛𝑓(𝑥1 ∪ 𝑥2)

1 − 𝑠𝑢𝑝𝑝(Подгузники) 1 − 0.6

𝑐𝑜𝑛𝑣(Пиво ∪ Подгузники) = 1 − 𝑐𝑜𝑛𝑓(Пиво ∪ Подгузники) = 1 − 0.75

= 1.6

Чем результат по формуле выше 1, тем лучше. Например, если conviction покупки пива и подгузников вместе был бы равен 1.2, это значит, что правило

«купил пиво и подгузники» было бы в 1.2 раза (на 20%) более верным, чем если бы совпадение этих items в одной транзакции было бы чисто случайным. Немного не интуитивное понятие, но оно и используется не так часто, как предыдущие три.

Существует ряд часто используемых алгоритмов, позволяющих находить правила в itemsets согласно перечисленным выше понятиям. Рассмотрим некоторые из них: Apriori и FP-Growth алгоритм

# Apriori Алгоритм

Используемые понятия:

* Множество объектов (itemset):

𝑋 ⊆ 𝐼 = {𝑥1, 𝑥2, . . . , 𝑥n}

* Множество идентификаторов транзакций (tidset):

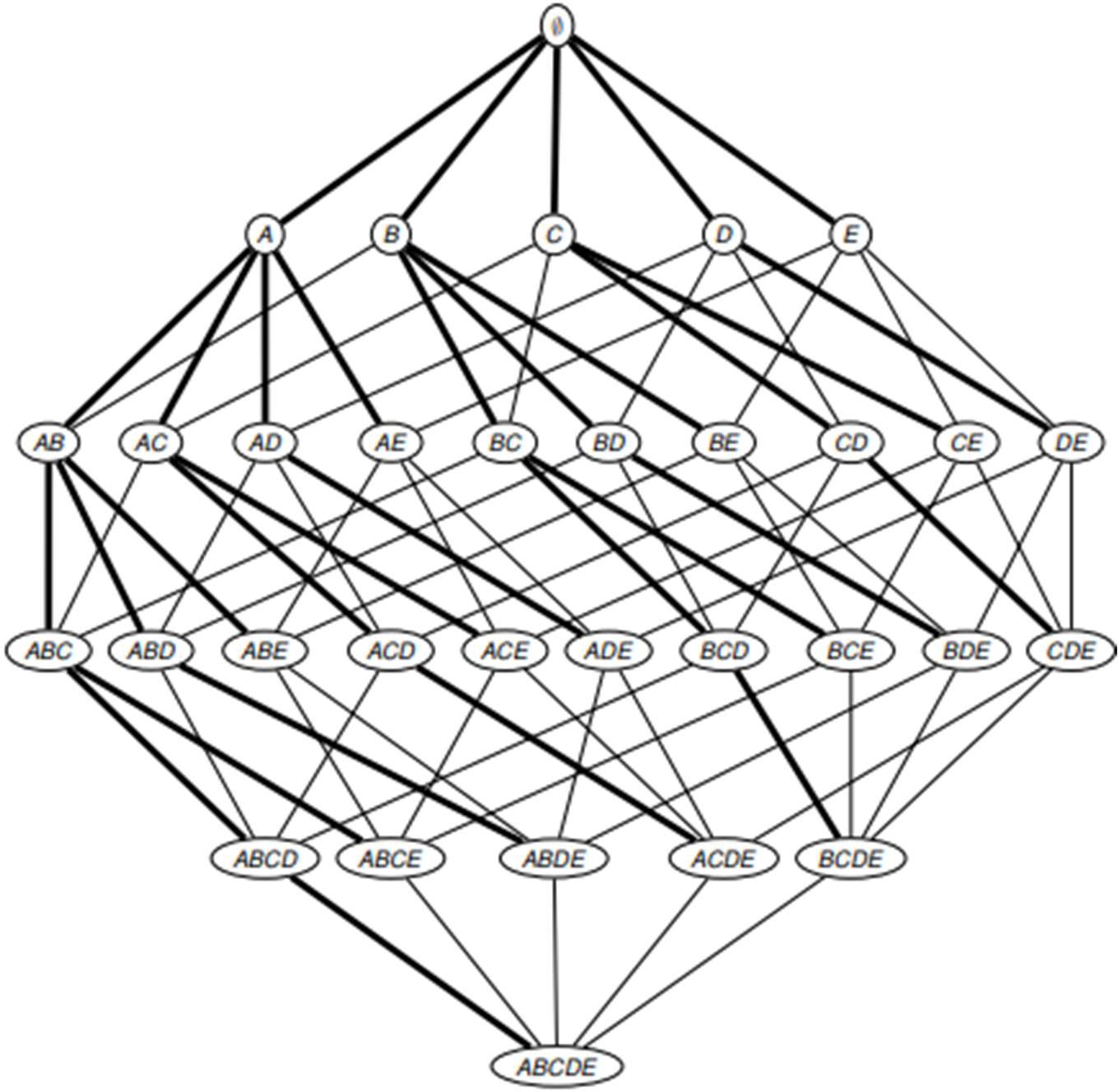
𝑇 = {𝑡1, 𝑡2, . . . , 𝑡m}

* Множество транзакций (transactions):

{(𝑡, 𝑋): 𝑡 ∈ 𝑇, 𝑋 ∈ 𝐼}

Введем дополнительно еще несколько понятий.

Будем рассматривать дерево префиксов (prefix tree), где 2 элемента X и Y соединены, если X является прямым подмножеством Y. Таким образом мы можем пронумеровать все подмножества множества I. Рисунок приведен ниже:



Рассмотрим алгоритм Apriori.

Apriori использует следующее утверждение: если 𝑋 ⊆ 𝑌, то 𝑠𝑢𝑝𝑝(𝑋) ≥

𝑠𝑢𝑝𝑝(𝑌). Отсюда следуют следующие 2 свойства:

* если Y встречается часто, то любое подмножество

𝑋 ∶ 𝑋 ⊆ 𝑌

так же встречается часто

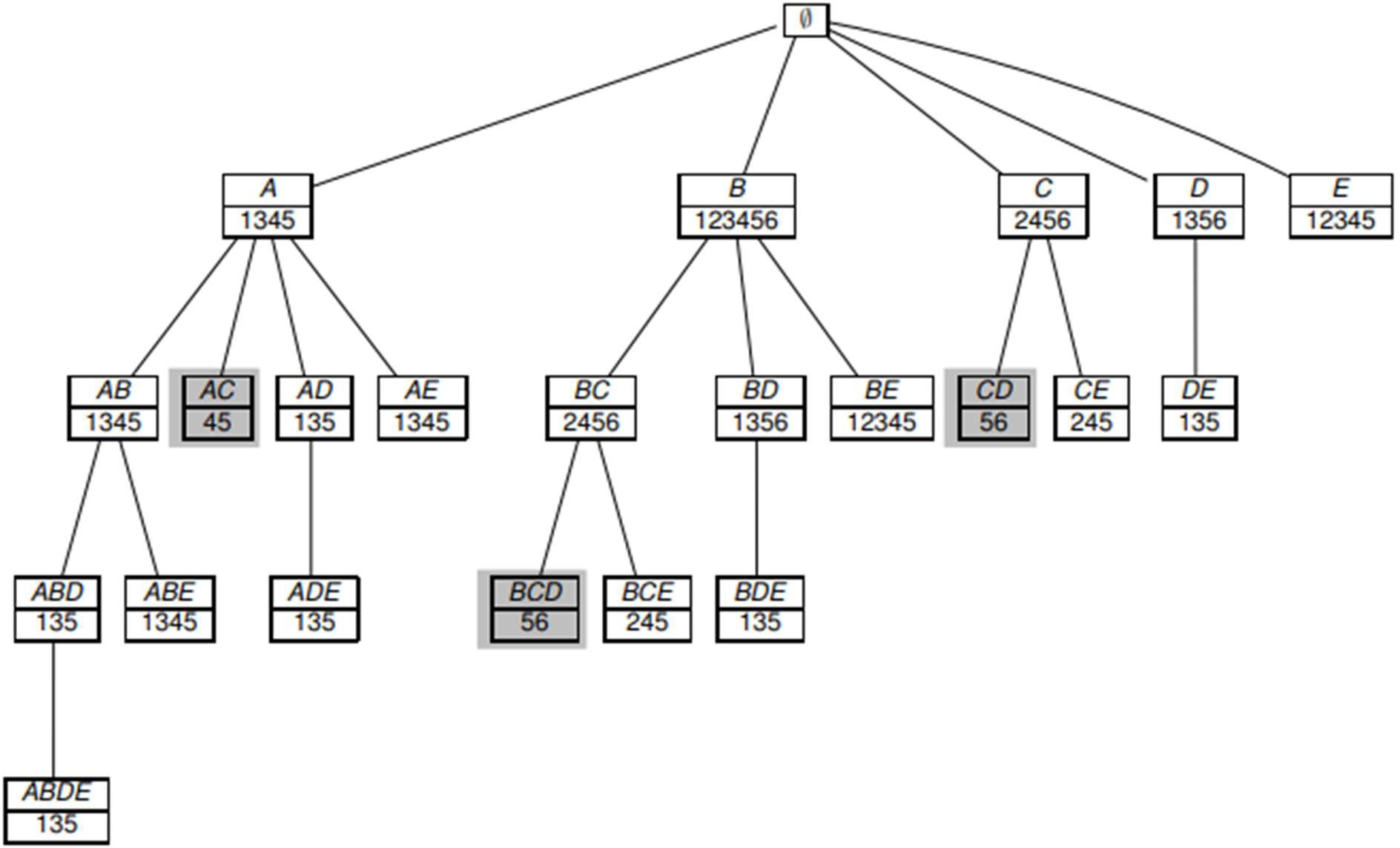
* если X встречается редко, то любое супермножество

𝑌 ∶ 𝑌 ⊇ 𝑋

так же встречается редко

Apriori алгоритм по-уровнево проходит по префиксному дереву и рассчитывает частоту встречаемости подмножеств X в D. Таким образом, в соответствии с алгоритмом:

* исключаются редкие подмножества и все их супермножества
* рассчитывается *supp(X)* для каждого подходящего кандидата X размера k на уровне k



Таким образом, Apriori сначала ищет все единичные (содержащие 1 элемент) itemsets, удовлетворяющие заданному пользователем *supp*, затем составляет из них пары по принципу иерархической монотонности, т.е. если

𝑥1 встречается часто и 𝑥2 встречается часто, то и [𝑥1, 𝑥2] встречается часто.

Явным минусом такого подхода является то, что необходимо

«просканировать» весь датасет, посчитать все *supp* на всех уровнях. Это также может сильно нагрузить оперативную память на больших датасетах, хотя алгоритм в плане скорости все равно намного эффективнее брутфорса.

## FP-Growth Algorithm

FP-Growth (Frequent Pattern Growth) более новый алгоритм, впервые он описан в 2000 году.

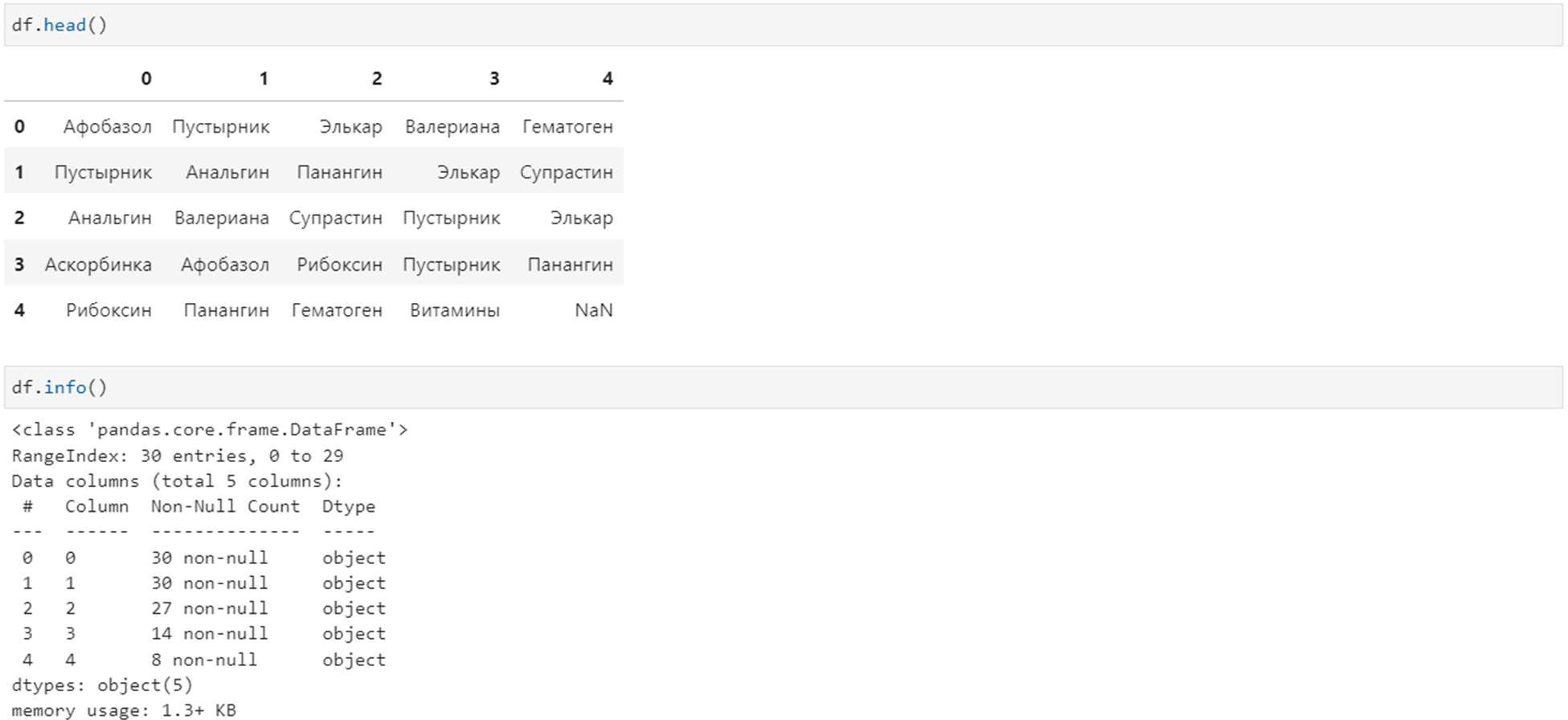
FP-Growth предлагает радикальную вещь — отказаться от генерации кандидатов (генерация кандидатов лежит в основе Apriori). Теоретически, такой подход позволит еще больше увеличить скорость алгоритма и использовать еще меньше памяти.

Это достигается за счет хранения в памяти префиксного дерева (trie) не из комбинаций кандидатов, а из самих транзакций.

При этом FP-Growth генерирует таблицу заголовков для каждого item, чей *supp* выше заданного пользователем. Эта таблица заголовков хранит связанный список всех однотипных узлов префиксного дерева.

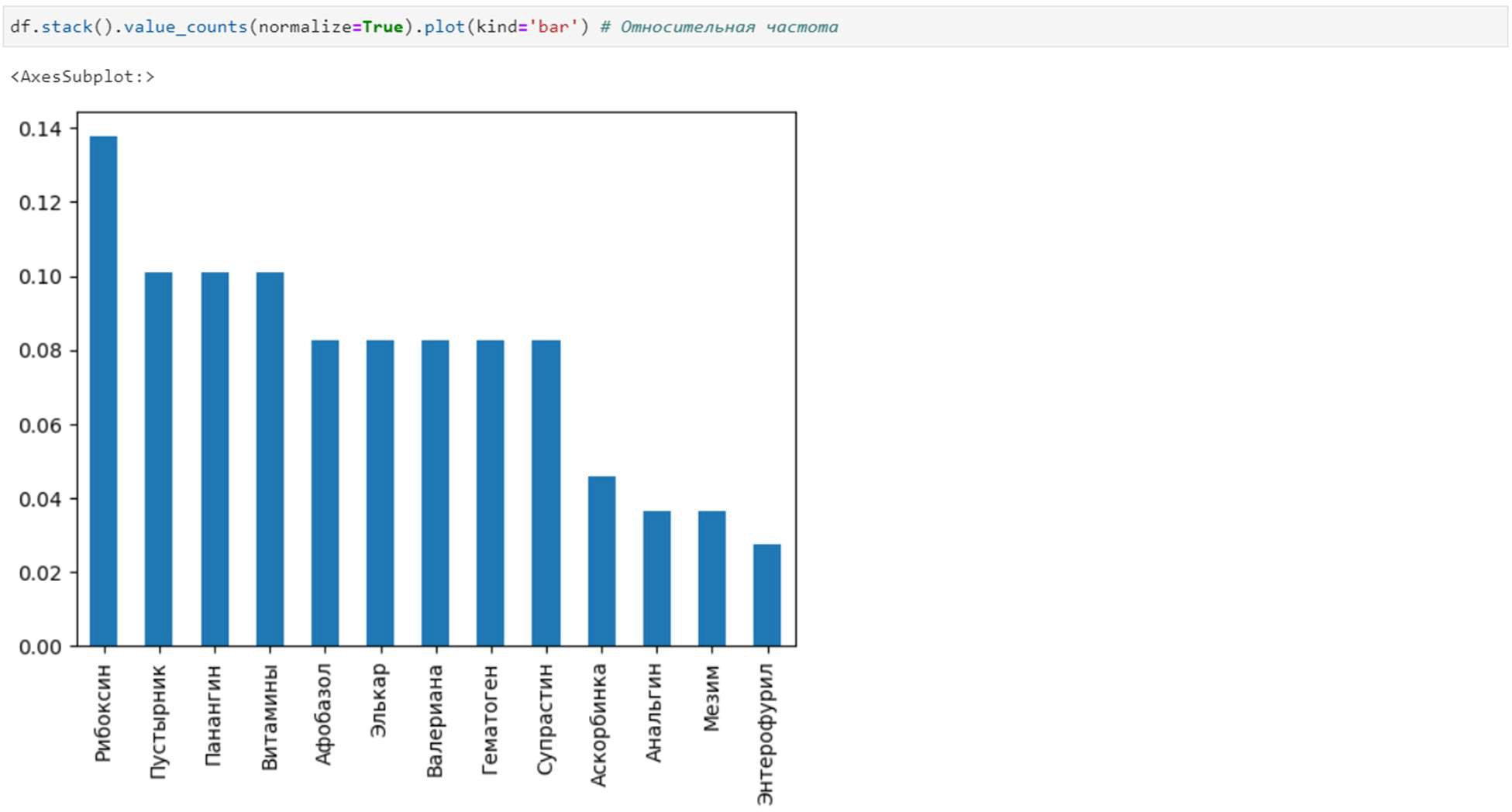
В качестве примера возьмем данные по покупкам лекарств в аптеке.

Датасет небольшой, содержит всего 30 транзакций (чеков).

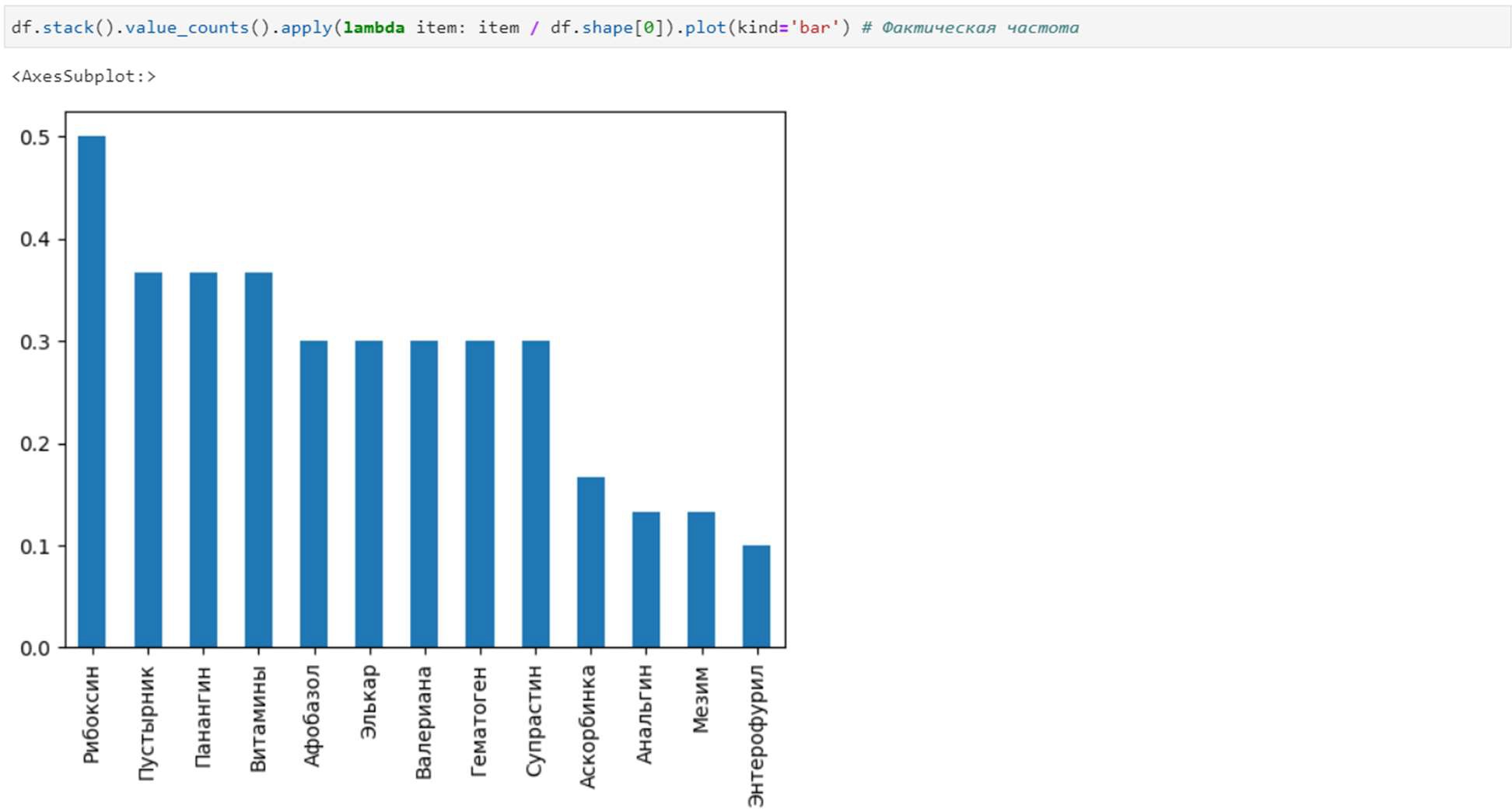


Визуализируем данные для лучшего представления. Посчитаем количество покупок конкретных лекарств во всех чеках. Также выполним нормализацию, чтобы вывести относительную частоту встречаемости.



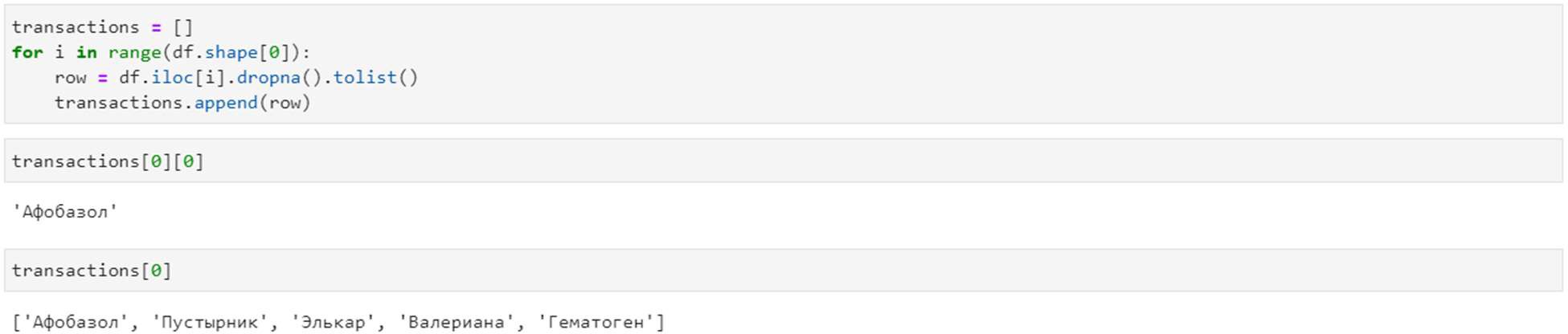


Выведем также и фактическую частоту. То есть, глядя на полученную гистограмму можно сказать, что в 50% корзин есть Рибоксин, в 37% Пустырник и т.д.



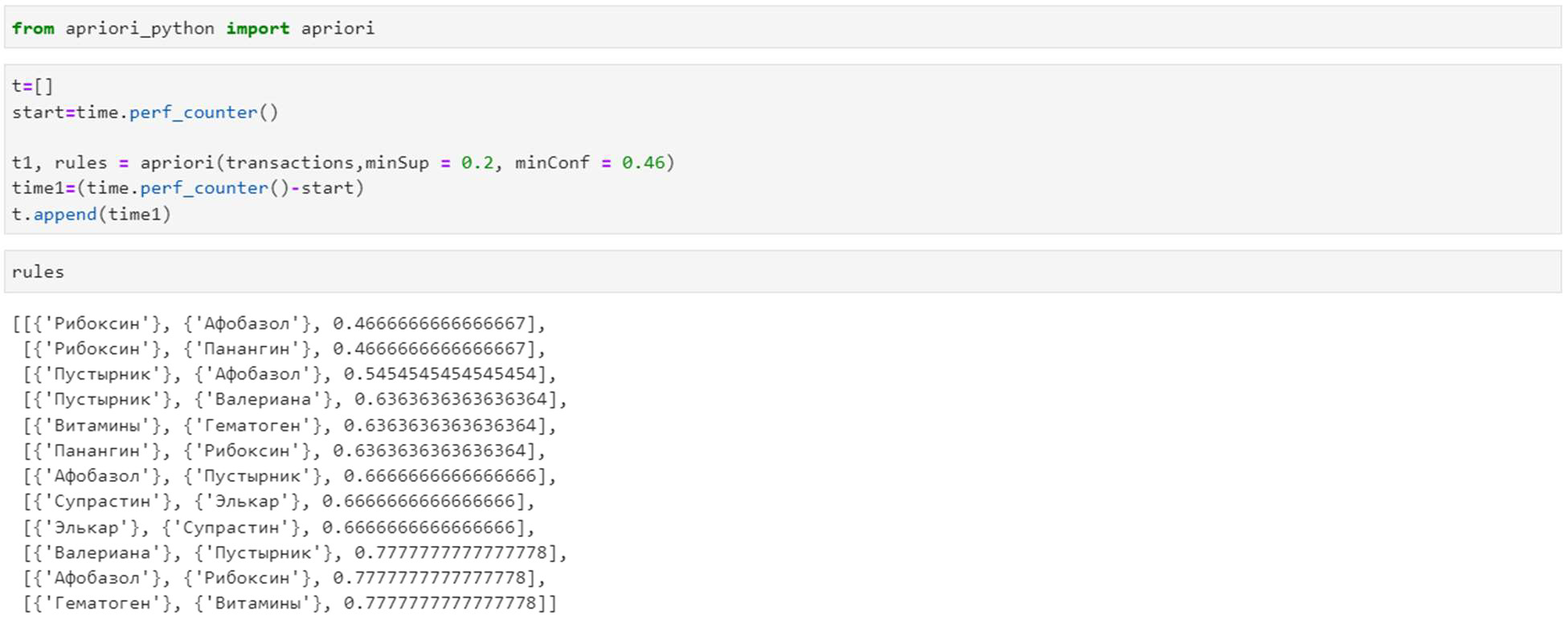
В большинстве библиотек, которые реализуют алгоритмы поиска ассоциативных правил в качестве входных значений необходимо подавать список транзакций, то есть список списков, поэтому нам необходимо преобразовать наш датасет в такой формат. При этом необходимо

контролировать пустые значения в данных (NaN), их добавлять в список не надо.



Также выведем первый элемент и первый список, и сравним с нашим датасетом, чтобы удостоверится, что преобразование прошло удачно.

Далее воспользуемся первой реализацией алгоритма Aprori из библиотеки apriori\_python.



В качестве аргументов принимается наш список списков с покупками, и настраиваются два параметра minSup и minConf. Для каждых данных они будут **свои**! Необходимо их определить так, чтобы количество правил было не слишком малым и не слишком большим. Если эти параметры сделать слишком малыми, то может существенно увеличится время выполнения алгоритма и найдется слишком много ненадежных правил. А если параметры будут слишком большие, то правил может вообще не найтись. Поэтому при первой прогонке рекомендуется сделать параметры поменьше, чтобы вывести как можно больше правил, а затем постепенно их уменьшать, пока количество правил не станет разумным.

minSup – это минимальная поддержка. Значение поддержки меняется от 0 (когда условие и следствие не встречаются вместе ни в одной транзакции) до 1 (когда условие и следствие во всех транзакциях появляются совместно).

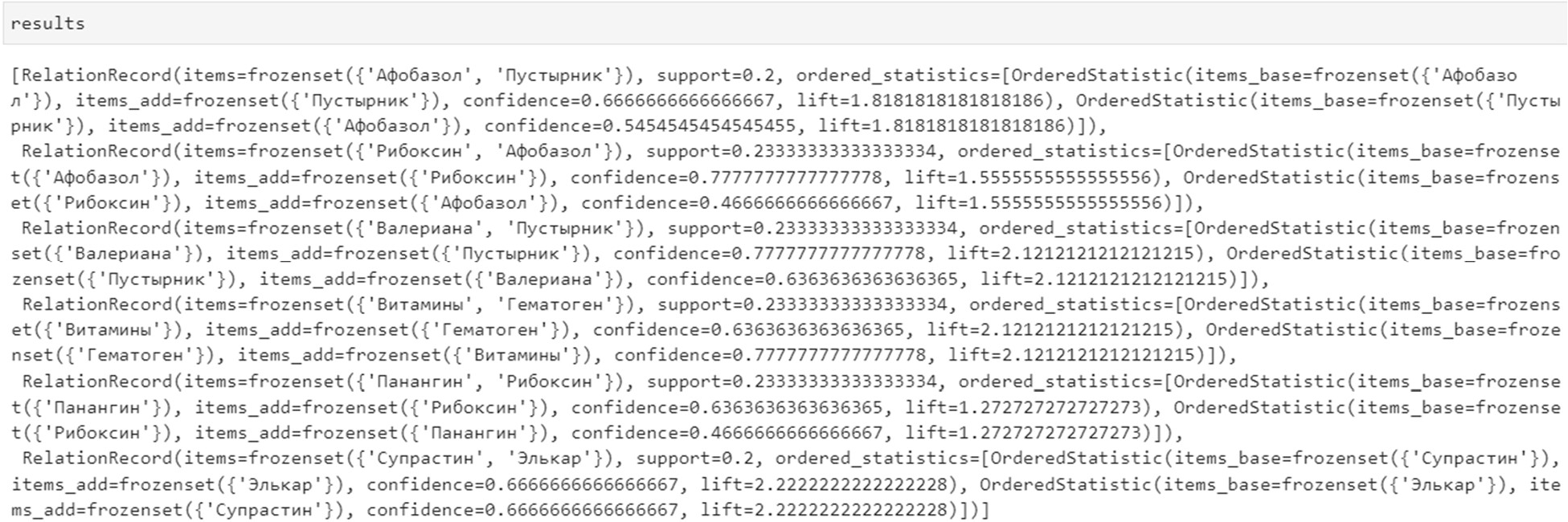
В общем случае поддержка является мерой надежности, с которой ассоциативное правило выражает ассоциативную связь между условием и следствием. Если поддержка S > 0,8, то связь сильная, а само правило заслуживает доверия. В случае, когда 0,5 < S < 0,8, ассоциативная связь средняя, а правило следует использовать с осторожностью. При S < 0,5 связь слабая, а ассоциативное правило является сомнительным.

minConf – минимальная достоверность. Это показатель, характеризующий уверенность в том, что ассоциация A → B является ассоциативным правилом. То есть предположение о том, что появление события A влечёт за собой появление события B, является достаточно достоверным.

Воспользуемся другой библиотекой для применения алгоритма Apriori.

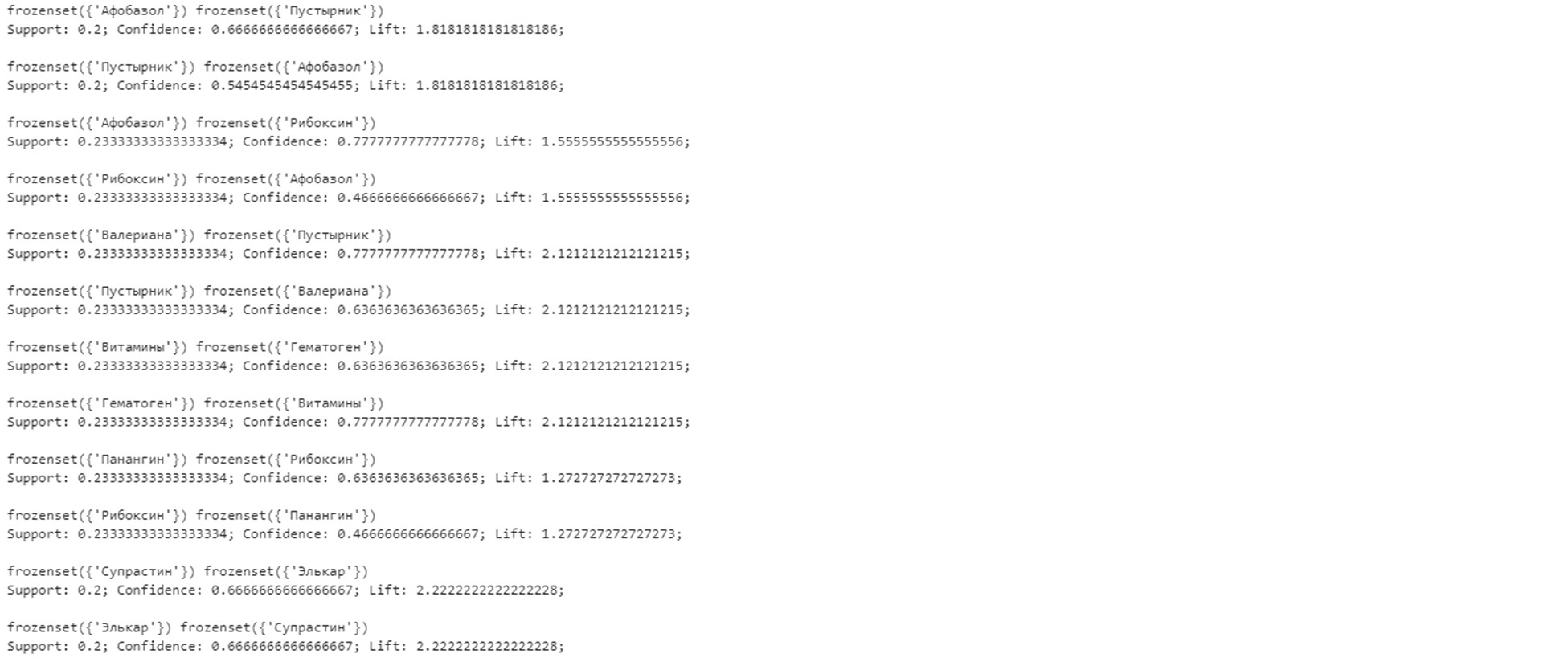


Здесь принимаются те же аргументы, что и в прошлой библиотеке, но необходимо указать минимальный лифт чуть больше 1, чтобы исключить вывод независимых правил.

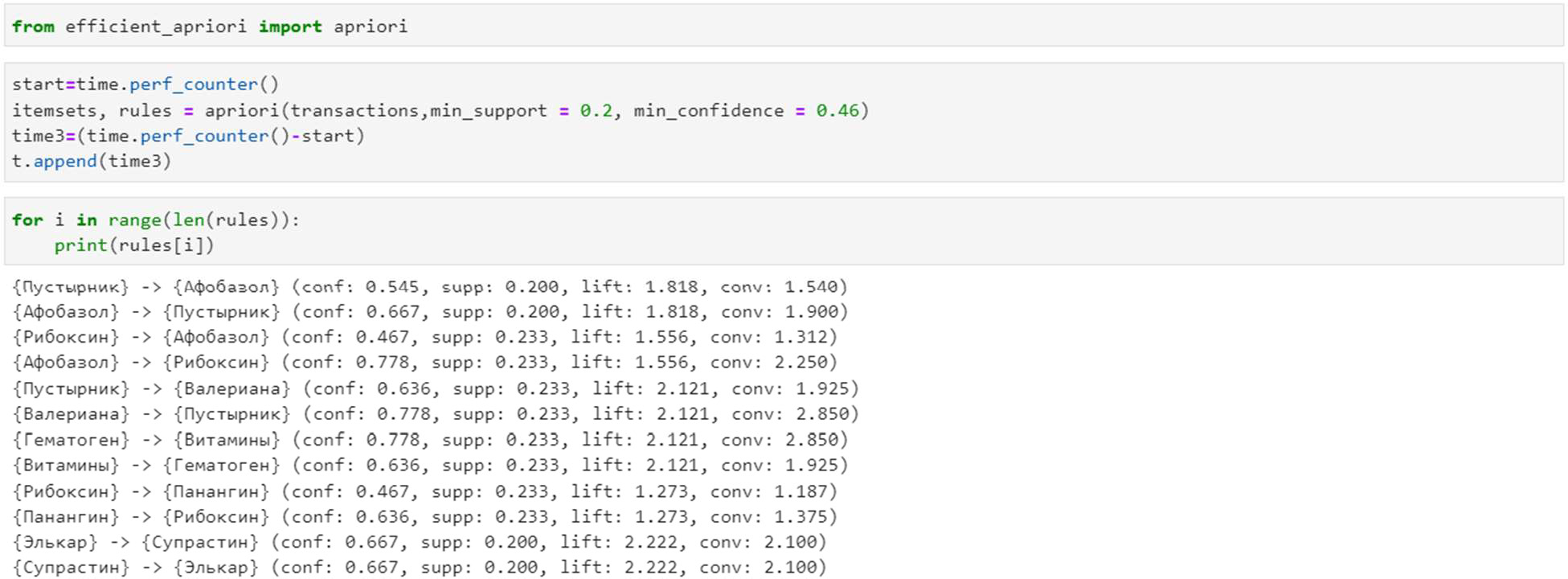


Для более удобной интерпретации результатов отформатируем вывод:



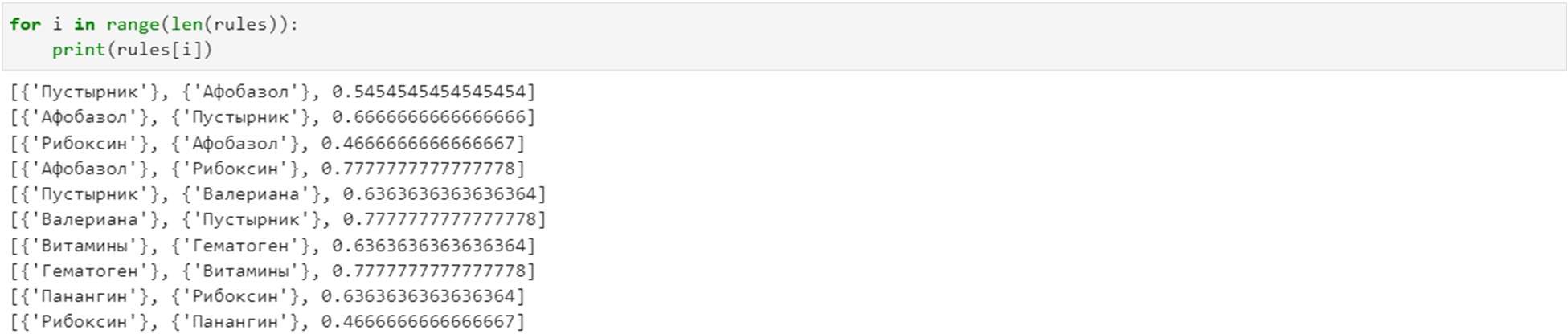


Воспользуемся третьей реализацией с помощью библиотеки efficient\_apriori. Аналогично зададим параметры минимальной поддержки и минимальной достоверности.

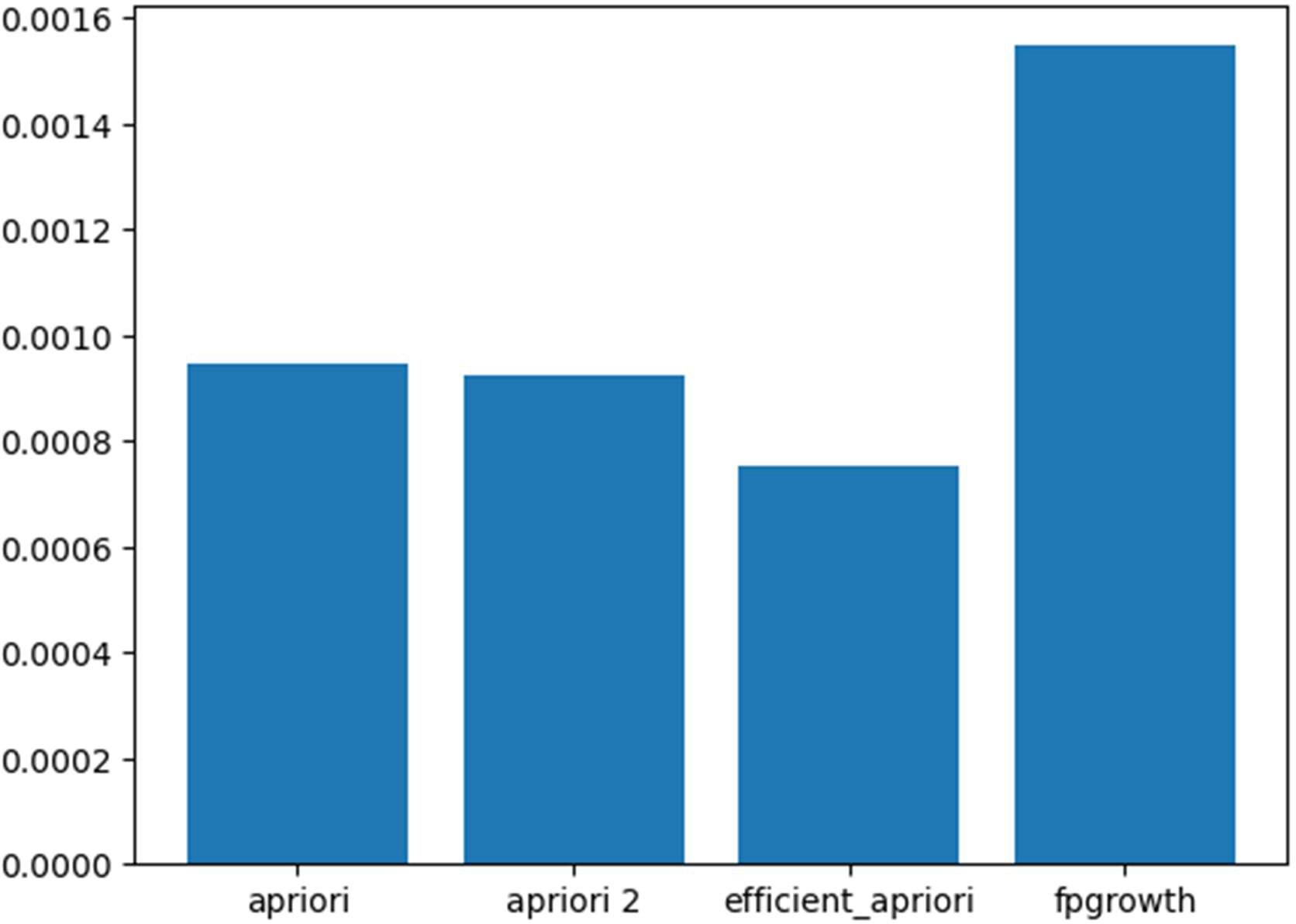
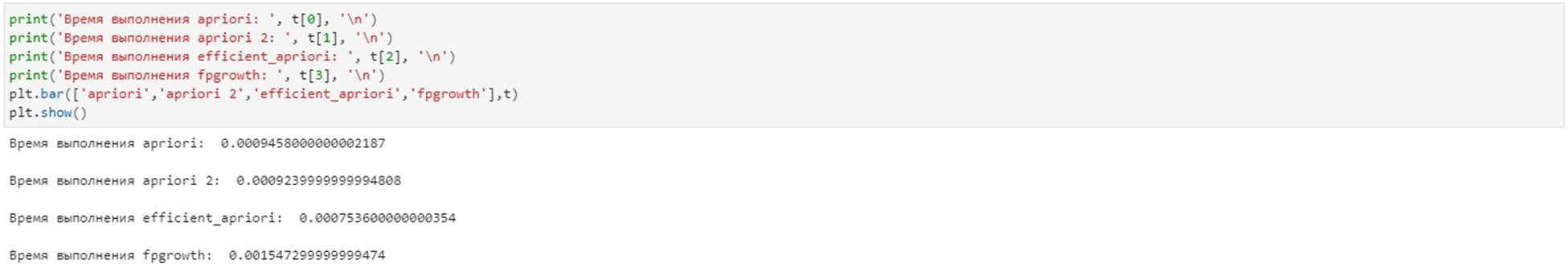


Применим алгоритмом FP-Growth из библиотеки fpgrowth\_py. Аналогично прошлым алгоритмам, необходимо задать минимальные поддержку и достоверность.





Сравним время выполнения наших алгоритмов:



Как видим на небольших данных время выполнения почти не различается, а также преимущества алгоритма FPGrowth нивелируется достаточно хорошей оптимизации алгоритма Apriori в используемых библиотеках. В

# Практическая работа

1. Загрузить данные Market\_Basket\_Optimisation.csv.
2. Визуализировать данные (отразить на гистограммах относительную и фактическую частоту встречаемости для **20** наиболее популярных товаров).
3. Применить алгоритм Apriori, используя 3 разные библиотеки (apriori\_python, apyori, efficient\_apriori). Подобрать гиперпараметры для алгоритмов так, чтобы выводилось порядка **10** наилучших правил.
4. Применить алгоритм FP-Growth из библиотеки fpgrowth\_py. Подобрать гиперпараметры для алгоритма так, чтобы выводилось порядка **10** наилучших правил.
5. Сравнить время выполнения всех алгоритмов и построить гистограмму.
6. Загрузить данные data.csv.
7. Визуализировать данные (отразить на гистограммах относительную и фактическую частоту встречаемости для **20** наиболее популярных товаров).
8. Применить алгоритм Apriori, используя 3 разные библиотеки (apriori\_python, apyori, efficient\_apriori). Подобрать гиперпараметры для алгоритмов так, чтобы выводилось порядка **10** наилучших правил.
9. Применить алгоритм FP-Growth из библиотеки fpgrowth\_py. Подобрать гиперпараметры для алгоритма так, чтобы выводилось порядка **10** наилучших правил.
10. Сравнить время выполнения всех алгоритмов и построить гистограмму. 11.Сформулировать выводы и сделать отчет.