

Market Basket Analysis

Este repositório é parte da disciplina de Projeto Integrador III, do curso de engenharia eletrônica do Instituto Federal de Santa Catarina

Conceituação

Uma das maiores necessidades de uma empresa é certamente a necessidade de estar sempre desenvolvendo novas maneiras de aumentar o volume de vendas de seus produtos. Uma das maneiras de atingir esse objetivo é vender mais para aqueles que já são clientes do negócio. Tendo isso em vista, surgiu a demanda de encontrar uma maneira de sugerir itens que podem vir a interessar e/ou complementar aqueles que os clientes já adquirem. Nesse objetivo surgiu a análise do carrinho de compras.

Utilizando de um conjunto de dados que representam as transações dos consumidores do negócio, é possível estabelecermos regras associativas do tipo $X \Rightarrow Y$, onde X representa um ou mais itens que o cliente adquiriu e Y um conjunto de itens sugeridos com base naqueles já adquiridos pelos clientes. Essas regras que embasarão as sugestões para os clientes, contudo antes de se estabelecer elas faz-se uma análise da base de dados com o fim de se estabelecer embasamento o suficiente para que a regra seja provada como efetiva.

Nessa análise da base de dados, as regras estabelecidas são mensuradas em sua maioria por 3 principais métricas.

- **Suporte:** Métrica que representa a frequência de ocorrência de compra do conjunto de itens em questão do total de compras. Sendo representada como:

$$Supp = \frac{freq(X, Y)}{N}$$

- **Confiança:** Representa a probabilidade de que a transação do do lado esquerdo da regra (antecedente) também contenha o conjunto do lado direito (consequente), sendo dada por:

$$Conf = \frac{freq(X, Y)}{freq(X)}$$

- **Lift:** A probabilidade de uma regra ocorrer de maneira independente da sugestão dos itens. Há 3 possibilidades:
 - Lift > 1: O lado esquerdo da regra eleva a possibilidade dos itens do lado direito serem comprados.
 - Lift = 1: A aquisição dos itens antecedentes não influenciam na aquisição dos consequentes
 - Lift < 1: O antecedente prejudica a aquisição dos itens consequentes.

$$Lift = \frac{freq(X, Y)}{freq(X) * freq(Y)}$$

Base de datos utilizada

Para simular a uma base de compras, foi utilizada uma base que está disponível na internet, no repositório de bases para machine learning do UCI, disponível neste [link](#). A base representa as compras feitas em uma loja virtual no período de 2 anos (01/01/2009 - 09/12/2011) e contém os campos InvoiceNo (Número da nota), StockCode (código do item), Description (Descrição do item), Quantity (Quantidade comprada do item), InvoiceDate (Data da compra), UnitPrice (Preço da compra), CustomerID (Código do cliente) e Country (País do cliente). Contudo, esta tabela não resumia bem a situação estudada na empresa, então foi necessário realizar o tratamento da mesma. Para isso, foi utilizada uma lista dos nomes e gêneros, também disponível publicamente no site da [UCI](#), para atribuir nomes e gêneros aos clientes.

Análise e tratamento da base

O primeiro procedimento realizado foi visualizar a base e contar o número de países nas mesmas. Após observar a base, tomou-se a decisão de separar a base em grupos de clientes, utilizando os países, porém como os dados foram colhidos de uma loja inglesa, aproximadamente 90% dos dados tinham como país de origem o reino unido, assim, fez-se uma separação dos consumidores do reino unido em vários grupos. As imagens abaixo mostram a base original e contagem de linhas separadas por país.

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	United Kingdom
417534 rows × 8 columns								

Dataset original

```
United Kingdom      379423
EIRE                 8710
Germany              8129
France               5710
Netherlands          2769
Spain                1278
Switzerland          1187
Belgium              1054
Portugal             1024
Channel Islands      906
Sweden               883
Italy                731
Australia            654
Cyprus                554
Austria              537
Greece               517
Denmark              428
Norway                369
Finland              354
United Arab Emirates 318
Unspecified          280
USA                  244
Japan                224
Poland                194
Malta                 172
Lithuania            154
Singapore            117
Canada                77
Thailand              76
Israel                74
Iceland               71
RSA                   65
Korea                 63
Brazil                62
West Indies           54
Bahrain               42
Nigeria               30
Name: Country, dtype: int64
```

Itens comprados por país

Para criar os grupos dos clientes ingleses, foi selecionado cada cliente distinto e separados em blocos de tamanhos distintos, dando números a esses grupos. Com isso, havia um dataframe com grupos de clientes originais e um datagrama que continha os clientes ingleses e seus novos grupos. Para unir esses dataframes e utilizar os novos grupos foi feita uma query utilizando o sqlite, conforme a imagem abaixo.

```

query = """
select
    df.Invoice as cd_compra,
    df.StockCode as cd_item,
    df.Description as tx_item,
    df.Quantity as nm_quantidade,
    strftime('%Y-%m-%d',df.InvoiceDate) as dt_compra,
    df.Price as nm_vl_item,
    df.Price*df.Quantity as nm_vl_total,
    cast(df."Customer ID" as integer) as cd_cliente,
    case
        when df2."Group ID" is not null then cast(df2."Group ID" as integer)
        else df."Group ID"
    end as cd_grupo_cliente
from
    df
left join
    df2
on
    df."Customer ID" = df2."Customer ID"
order by
    cd_grupo_cliente
"""

df3 = ps.sqldf(query)

```

Query para grupos de clientes

Além disso, para melhor adequação do dataset com o original, foi atribuído a cada grupo de clientes uma cidade da Inglaterra, assim, utilizou-se um dataset que continha as cidades mais populosas deste país e foi atribuído aos grupos as 47 mais populosas. Com isso foi criado um dataframe com as informações de ID do grupo, código do grupo, país e região que o grupo pertence.

	id_grupo	cd_grupo_cliente	cd_pais	tx_pais	cd_estado
0	0	0	GB	United Kingdom	London, City of
1	1	1	GB	United Kingdom	Birmingham
2	2	2	GB	United Kingdom	Manchester
3	3	3	GB	United Kingdom	Leeds
4	4	4	GB	United Kingdom	Newcastle upon Tyne
5	5	5	GB	United Kingdom	Leicester
6	6	6	GB	United Kingdom	Glasgow City
7	7	7	GB	United Kingdom	Liverpool
8	8	8	GB	United Kingdom	Portsmouth
9	9	9	GB	United Kingdom	Southampton
10	10	10	GB	United Kingdom	Nottingham
11	11	11	GB	United Kingdom	Bristol, City of
12	12	12	GB	United Kingdom	Sheffield
13	13	13	GB	United Kingdom	Kingston upon Hull, City of
14	14	14	GB	United Kingdom	Edinburgh, City of
15	15	15	GB	United Kingdom	Cardiff
16	16	16	GB	United Kingdom	Stoke-on-Trent
17	17	17	GB	United Kingdom	Coventry
18	18	18	GB	United Kingdom	Reading
19	19	19	GB	United Kingdom	Belfast
20	20	20	GB	United Kingdom	Derby
21	21	21	GB	United Kingdom	Plymouth
22	22	22	GB	United Kingdom	Wolverhampton
23	23	23	GB	United Kingdom	Swansea
24	24	24	GB	United Kingdom	Milton Keynes
25	25	25	GB	United Kingdom	Aberdeen City

Dataset grupos de clientes

Também se criou um dataframe contendo o código, ID e descrição dos itens e armazenados em um outro dataframe. O mesmo procedimento foi feito para o dataframe com nomes, gênero código e ID dos clientes

id_item cd_item tx_item				id_cliente cd_cliente cliente_nome cliente_sexo			
0	0	10002	INFLATABLE POLITICAL GLOBE	0	1136	12346	Mila F
1	1	10080	GROOVY CACTUS INFLATABLE	1	647	12347	Christy F
2	2	10109	BENDY COLOUR PENCILS	2	613	12348	Leona F
3	3	10120	DOGGY RUBBER	3	549	12349	Nina F
4	4	10123C	HEARTS WRAPPING TAPE	4	631	12351	Dora F
...
4026	4026	PADS	PADS TO MATCH ALL CUSHIONS	4378	1921	18283	Russel M
4027	4027	POST	POSTAGE	4379	3738	18284	Ora M
4028	4028	SP1002	KID'S CHALKBOARD/EASEL	4380	1880	18285	Odessa F
4029	4029	TEST001	This is a test product.	4381	1227	18286	Kaiden M
4030	4030	TEST002	This is a test product.	4382	79	18287	Cynthia F
4031 rows x 3 columns				4383 rows x 4 columns			
Dataframe itens				Dataframe clientes			

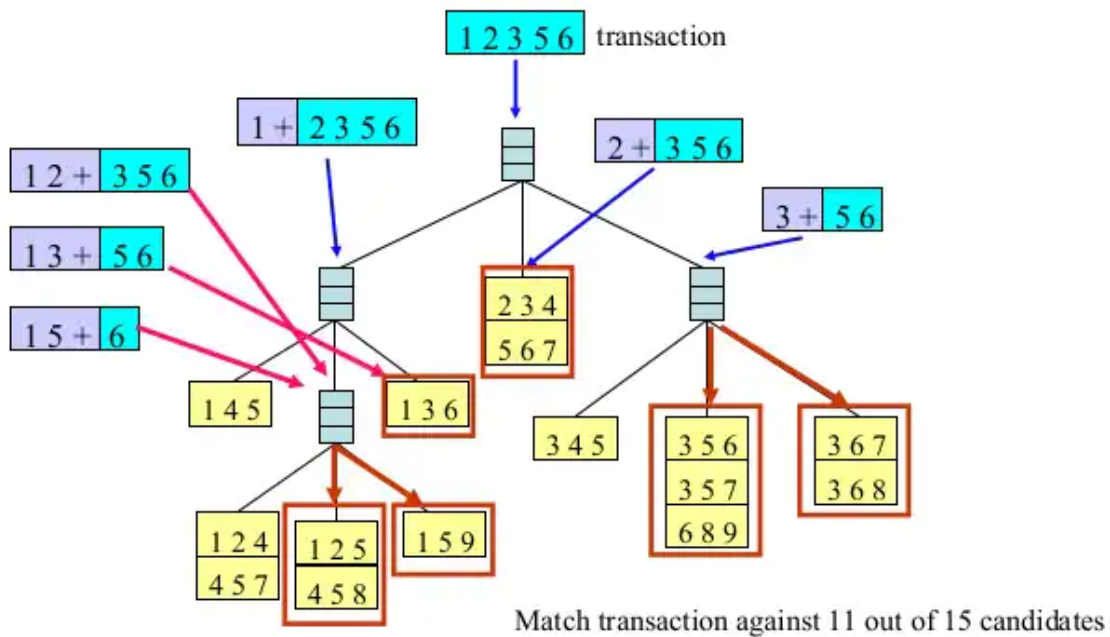
Além disso, para manter as tabelas normalizadas, os dataframes foram salvos em datasets separados e o dataframe original foi modificado para se adequar a essas características, tendo as informações que se encontram em outras tabelas substituídas pelo ID da informação, resultando no dataframe abaixo.

	id_compra	cd_compra	id_item	id_cliente	id_grupo	dt_compra	nm_quantidade	nm_vl_item	nm_vl_total
0	0	489434	3423	0	0	2009-12-01	12	6.95	83.40
1	1	489434	2740	0	0	2009-12-01	12	6.75	81.00
2	2	489434	2742	0	0	2009-12-01	12	6.75	81.00
3	3	489434	1258	0	0	2009-12-01	48	2.10	100.80
4	4	489434	617	0	0	2009-12-01	24	1.25	30.00
...
417529	525456	538145	2207	4382	46	2010-12-09	12	2.95	35.40
417530	525457	538145	1478	4382	46	2010-12-09	6	2.55	15.30
417531	525458	538145	2780	4382	46	2010-12-09	6	2.95	17.70
417532	525459	538145	1656	4382	46	2010-12-09	12	1.65	19.80
417533	525460	538145	542	4382	46	2010-12-09	8	1.69	13.52
417534 rows x 9 columns									

Algoritmos testados

Algoritmo de apriori

Apriori: Implementation Using Hash Tree



Algoritmo proposto por Agrawal e Srikant em 1994, desenvolvido para a mineração de dados online que contenham transações. O algoritmo começa analisando os itens individualmente e expandindo um item por vez dos conjuntos que tiverem suporte maior que o dado como mínimo. Sendo assim, a ideia do algoritmo se baseia no fato de que se um conjunto não é frequente, todas as regras que derivam deste conjunto também não o serão. O pseudocódigo está apresentado abaixo:

```

Apriori(T,  $\epsilon$ )
    L1  $\leftarrow$  {large 1 - itemsets}
    k  $\leftarrow$  2
    while Lk-1 is not empty
        Ck  $\leftarrow$  Apriori_gen(Lk-1, k)
        for transactions t in T
            Dt  $\leftarrow$  {c in Ck : c  $\subseteq$  t}
            for candidates c in Dt
                count[c]  $\leftarrow$  count[c] + 1

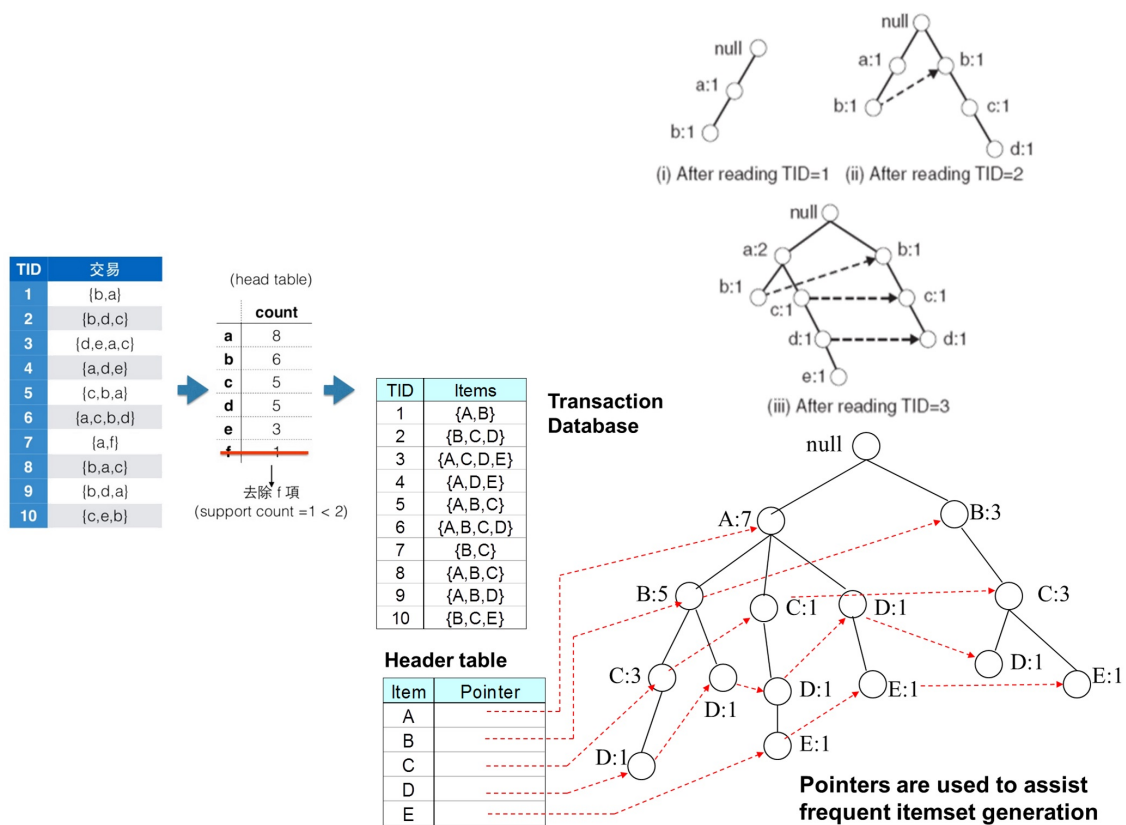
        Lk  $\leftarrow$  {c in Ck : count[c]  $\geq$   $\epsilon$ }
        k  $\leftarrow$  k + 1

    return Union(Lk)

Apriori_gen(L, k)
    result  $\leftarrow$  list()
    for all p  $\subseteq$  L, q  $\subseteq$  L where p1 = q1, p2 = q2, ..., pk-2 = qk-2 and pk-1 < qk-1
        c = p  $\cup$  {qk-1}
        if u  $\subseteq$  c for all u in L
            result.add(c)
    return result

```

FP-Growth



O algoritmo de FP-Growth é uma forma de aprimorar o algoritmo de apriori. O algoritmo lê a base de transações e faz a contagem da frequência em que os itens aparecem, aqueles itens que estão abaixo do suporte mínimo são retirados dos conjuntos e estessão reordenados por frequência. Com esse passo feito, cada linha do algoritmo é varrida e sendo representada na árvore, tendo o número de vezes que cada subconjunto aparece. Assim que a árvore está formada, o algortimo faz a montagem dos conjuntos de itens frequentes com base no número de aparições do item nos subconjuntos. Devido a isso, o algortimo se mostra mais rápido e menos oneroso ao sistema.

Apache Spark

Um framework de código aberto multi-plataforma voltado para processamento de dados em larga escala por meio de clusterização e paralelismo. Pode funcionar tanto localmente quanto em uma malha de computadores e suporta programação nas linguagens Scala, Java, Python e R.

- Arquitetura do Spark é subdividida em:
 - Driver:** Máquina na qual aplicação é executada. Responsável por:
 - Manter informações sobre a aplicação
 - Apresentar respostas ao usuário
 - Gerenciar a distribuição de tarefas pelos executores
 - Jobs:** Ações paralelizadas que são separadas em uma série de passos ordenados que são chamados de Stages
 - Executores:** Armazenam uma partição Spark para ser processada, sendo responsáveis por:
 - Executar o código designado pelo driver

- Reportar o status da computação para o driver
- **Slots:** Partes do executor que são responsáveis por realizar as Tasks
- **Task:** São responsáveis pelo processamento de dados nas partições
- **SparkSession:** Acesso ao SparkContext
- **SparkContext:** Conexão com o Cluster
- As modificações nos dados dos dataframes só ocorrem quando uma ação é executada. Esse comportamento recebe o nome de **Lazy evaluation**
- Estruturas de dados:
 - **RDD** (Resilient distributed datasets):
 - Estrutura de baixo nível
 - Imutável
 - Complexo
 - Otimização difícil
 - **DataFrame:**
 - Estrutura de dados de mais alto nível
 - Imutável
 - Schema conhecido
 - Mais otimizado pelo Spark

Configurando o ambiente Spark

- **Java JDK:** Para a instalação utilizou-se o JRE 1.8.0-301
 - Necessário criar a variável de ambiente JAVA_HOME com o diretório de instalação do JRE
 - Adicionar na variável de ambiente Path: %JAVA_HOME%\bin
- **Hadoop:** Foi feita a utilização da versão 2.7.4
 - Para a utilização é necessário realizar o download dos arquivos winutils.exe e hadoop.dll, disponíveis [aqui](#) e inseri-los na pasta bin do local de instalação do hadoop
 - A extração dos arquivos do .tar.gz devem ser feitas utilizando o 7-zip como administrador, para que sejam feitos os links simbólicos das bibliotecas nativas.
 - No windows, recomenda-se a criação do diretório do Hadoop em uma pasta raiz (Ex: C:\hdp) devido à estrutura de arquivos do hadoop ter muitas pastas e o caminho delas exceder o número máximo de caracteres do SO.
 - Necessário criar a variável de ambiente HADOOP_HOME com o diretório de instalação do HADOOP
 - Adicionar na variável de ambiente Path: %HADOOP_HOME%\bin
- **Spark:** Versão 3.12 com hadoop 2.7
 - Necessário criar a variável de ambiente SPARK_HOME com o diretório de instalação do SPARK
 - Adicionar na variável de ambiente Path: %SPARK_HOME%\bin
- **Pyspark:** Para a instalação foi utilizado a versão 3.1.2 do anaconda
 - Para a correta instalação e utilização com o jupyter notebook, é necessário criar as variáveis de ambiente:
 - PYSPARK_DRIVER_PYTHON = "jupyter"
 - PYSPARK_DRIVER_PYTHON_OPTS = "notebook"
 - PYSPARK_HADOOP_VERSION: Esta variável pode ter valor 2.7, 3.2 ou "without" dependendo da versão do hadoop que se deseja utilizar. Para a utilização foi utilizada a versão 2.7

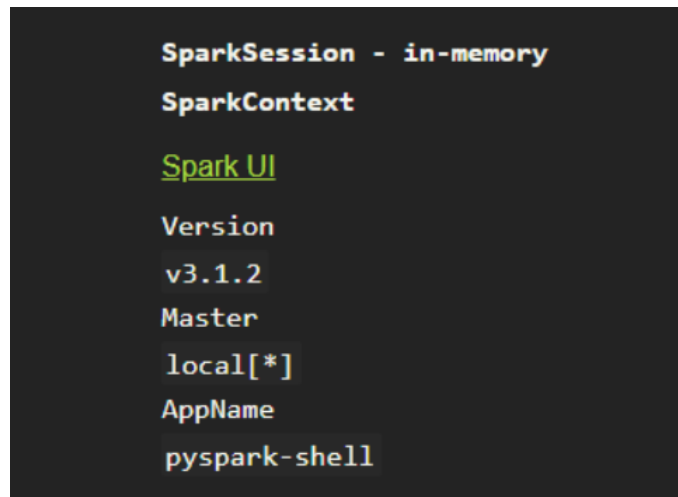
- PYTHONPATH: %SPARK_HOME%/python;\$SPARK_HOME/python/lib/py4j-0.10.9-src.zip
- Instalar o módulo findspark com o pip

Utilizando o pyspark

- Inicializando uma SparkSession utilizando python

```
import findspark
from pyspark.sql import SparkSession
## Spark Conf
findspark.init()
spark = SparkSession.builder.master("local[*]").getOrCreate()
spark = SparkSession.builder.appName("marketBasket").getOrCreate()
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
spark
```

- Será exibida a mensagem abaixo, ao clicar e, "Spark UI" será aberta uma guia com informações sobre a seção do spark sendo utilizada



```
SparkSession - in-memory
SparkContext
Spark UI
Version
v3.1.2
Master
local[*]
AppName
pyspark-shell
```

Testes realizados

Apriori

Para o algoritmo de apriori, foi necessário criar uma função de hot encode, onde retorna 1 caso a quantidade de itens for maior ou igual a 1 e 0 caso contrário.

Inicialmente se utilizou todos os grupos para realização dos testes, assim de juntou as tabelas que tínhamos à disposição e se formaram *baskets* para cada grupo de cliente, como apresentado abaixo

```

0 - basket_london: (300, 3419)
1 - basket_birmingham: (47, 1278)
2 - basket_manchester: (6, 159)
3 - basket_leeds: (17, 479)
4 - basket_newcastle_upon_tyne: (15, 419)
5 - basket_leicester: (4, 1999)
6 - basket_glasgow_city: (68, 1457)
7 - basket_liverpool: (18, 499)
8 - basket_portsmouth: (6, 137)
9 - basket_southampton: (9, 285)
10 - basket_nottingham: (23, 880)
11 - basket_bristol: (2, 133)
12 - basket_sheffield: (25, 683)
13 - basket_kingston_upon_hull: (12, 544)
14 - basket_edinburgh: (11, 511)
15 - basket_cardiff: (7, 416)
16 - basket_stoke_on_trent: (4, 419)
17 - basket_coventry: (5, 219)
18 - basket_reading: (10, 363)
19 - basket_belfast: (16, 424)
20 - basket_derby: (4, 288)
21 - basket_plymouth: (8, 225)
22 - basket_wolverhampton: (14, 673)
23 - basket_swansea: (5, 219)
24 - basket_milton_keynes: (1, 30)
25 - basket_aberdeen_city: (1, 162)
26 - basket_norfolk: (1, 65)
27 - basket_luton: (1, 110)
28 - basket_islington: (2, 39)
29 - basket_swindon: (1, 76)
30 - basket_croydon: (2, 73)
31 - basket_essex: (1, 113)
32 - basket_bournemouth: (1, 49)
33 - basket_west_sussex: (2, 52)
34 - basket_suffolk: (1, 60)
35 - basket_redcar_and_cleveland: (1, 77)
36 - basket_sunderland: (1, 70)
40 - basket_redbridge: (300, 3386)
41 - basket_warrington: (600, 3551)
42 - basket_slough: (500, 3200)
43 - basket_kirklees: (635, 3147)
44 - basket_oxfordshire: (200, 2194)
45 - basket_york: (253, 2483)
46 - basket_poole: (300, 2390)
47 - basket_harrow: (405, 2328)
48 - basket_dundee_city: (359, 2200)
49 - basket_hertfordshire: (183, 1751)

```

tx_item	10 COLOUR SPACEBOY PEN	12 PENCIL SMALL TUBE WOODLAND	12 PENCILS SMALL TUBE RED RETROSPOT	12 PENCILS SMALL TUBE SKULL	3 HEARTS HANGING DECORATION RUSTIC	36 FOIL STAR CAKE CASES	36 PENCILS TUBE RED SPOTTY	36 PENCILS TUBE WOODLAND	36 PENCILS TUBE WOODLAND	3D TRADITIONAL CHRISTMAS STICKERS	6 CROCHET STRAWBERRIES	6 RIBBONS RUSTIC CHARM	60 CAKE CASES DOLLY GIRL DESIGN	60 CAKE CASES VINTAGE CHRISTMAS	72 SWEETHEART FAIRY CAKE CASES	ABSTRACT CIRCLES NOTEBOOK	f CAL GI
cd_cliente																	
12358	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
12360	1	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0
12368	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0
12370	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12373	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12374	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12414	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12440	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12817	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
12865	0	0	0	0	1	1	0	0	0	0	0	1	0	1	1	0	0

Em seguida, para cada grupo se aplicou o algoritmo de apriori, com suporte mínimo de 10%, 20% e 30% porém no caso deste dataset, era necessário que a máquina tivesse mais de 45 GB de memória RAM disponível, o que tornou inviável a utilização com 10%. Já no caso com 20%, o número de arquivos supera o tamanho máximo possível do array no python. Por fim, com 30%

passamos a ter baskets com nenhum item comum, o que resulta em erro no algoritmo. O arquivo Jupyter Notebook com o código utilizado se encontra em [Market Basket v1.ipynb](#)

```
In [18]: frequencies = []
rules = []
for i in baskets:
    frq = 'frequent_' + i.replace('basket_', '')
    rls = 'rules_' + i.replace('basket_', '')
    print(f"Begin - {i.replace('basket_', '')}")
    globals()[frq] = apriori(globals()[i], min_support=0.1, use_colnames=True)
    globals()[rls] = association_rules(globals()[frq], metric='lift', min_threshold=1).sort_values(['confidence', 'lift'], ascending=[False, False])
    print(f"End - {i.replace('basket_', '')}")
    frequencies.append(frq)
    rules.append(rls)
```

```
Begin - london
End - london
Begin - birmingham
```

```
-----
MemoryError                                Traceback (most recent call last)
<ipython-input-18-bec6b104f0> in <module>
      5 rls = 'rules_' + i.replace('basket_', '')
      6 print(f"Begin - {i.replace('basket_', '')}")
----> 7 globals()[frq] = apriori(globals()[i], min_support=0.1, use_colnames=True)
      8 globals()[rls] = association_rules(globals()[frq], metric='lift', min_threshold=1).sort_values(['confidence', 'lift'], ascending=[False, False])
      9 print(f"End - {i.replace('basket_', '')}")
```

```
C:\Anaconda\lib\site-packages\mlxtend\frequent_patterns\apriori.py in apriori(df, min_support, use_colnames, max_len, verbose, low_memory)
    300     _bools = _bools & (X[:, combin[i, 0]] == all_ones)
    301     else:
--> 302         _bools = np.all(X[:, combin], axis=2)
    303
    304     support = _support(np.array(_bools), rows_count, is_sparse)
```

```
MemoryError: Unable to allocate 43.8 GiB for an array with shape (17883664, 7, 47) and data type int64
```

```
In [19]: frequencies = []
rules = []
for i in baskets:
    frq = 'frequent_' + i.replace('basket_', '')
    rls = 'rules_' + i.replace('basket_', '')
    print(f"Begin - {i.replace('basket_', '')}")
    globals()[frq] = apriori(globals()[i], min_support=0.2, use_colnames=True)
    globals()[rls] = association_rules(globals()[frq], metric='lift', min_threshold=1).sort_values(['confidence', 'lift'], ascending=[False, False])
    print(f"End - {i.replace('basket_', '')}")
    frequencies.append(frq)
    rules.append(rls)
```

```
Begin - london
End - london
Begin - birmingham
End - birmingham
Begin - manchester
End - manchester
Begin - leeds
End - leeds
Begin - newcastle_upon_tyne
End - newcastle_upon_tyne
Begin - leicester
```

```
-----
MemoryError                                Traceback (most recent call last)
<ipython-input-19-7b2675da8bb7> in <module>
      5 rls = 'rules_' + i.replace('basket_', '')
      6 print(f"Begin - {i.replace('basket_', '')}")
----> 7 globals()[frq] = apriori(globals()[i], min_support=0.2, use_colnames=True)
      8 globals()[rls] = association_rules(globals()[frq], metric='lift', min_threshold=1).sort_values(['confidence', 'lift'], ascending=[False, False])
      9 print(f"End - {i.replace('basket_', '')}")
```

```
C:\Anaconda\lib\site-packages\mlxtend\frequent_patterns\apriori.py in apriori(df, min_support, use_colnames, max_len, verbose, low_memory)
    285     else:
    286         combin = generate_new_combinations(itemset_dict[max_itemset])
--> 287         combin = np.fromiter(combin, dtype=int)
    288         combin = combin.reshape(-1, next_max_itemset)
    289
```

```
MemoryError: cannot allocate array memory
```

```
In [11]: frequencies = []
rules = []
for i in baskets:
    frq = 'frequent_' + i.replace('basket_', '')
    rls = 'rules_' + i.replace('basket_', '')
    print(f"Begin - {i.replace('basket_', '')}")
    globals()[frq] = apriori(globals()[i], min_support=0.3, use_colnames=True)
    globals()[rls] = association_rules(globals()[frq], metric='lift', min_threshold=1).sort_values(['confidence', 'lift'], ascending=[False, False])
    print(f"End - {i.replace('basket_', '')}")
    frequencies.append(frq)
    rules.append(rls)
```

```
Begin - london
End - london
Begin - birmingham
End - birmingham
Begin - manchester
End - manchester
Begin - leeds
End - leeds
Begin - newcastle_upon_tyne
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-11-e367465656ac> in <module>
      6 print(f"Begin - {i.replace('basket_', '')}")
      7 globals()[frq] = apriori(globals()[i], min_support=0.3, use_colnames=True)
----> 8 globals()[rls] = association_rules(globals()[frq], metric='lift', min_threshold=1).sort_values(['confidence', 'lift'], ascending=[False, False])
      9 print(f"End - {i.replace('basket_', '')}")
     10 frequencies.append(frq)
```

```
C:\Anaconda\lib\site-packages\mlxtend\frequent_patterns\association_rules.py in association_rules(df, metric, min_threshold, support_only)
    78     """
    79     if not df.shape[0]:
--> 80         raise ValueError('The input DataFrame "df" containing '
    81                           'the frequent itemsets is empty.')
    82
```

```
ValueError: The input DataFrame "df" containing the frequent itemsets is empty.
```

Para contornar estes problemas de limitação de hardware, foram selecionados alguns grupos para realizar a predição. No ambiente empresarial, se escolheu os grupos que tinham maior representatividade na empresa. No ambiente de testes, escolhemos os grupos com formatos mais condizentes com os dados originais e também que possibilitariam apresentar um resultado. Assim, estabeleceu-se a seguinte query, onde são selecionados apenas os grupos contidos na lista. Ao aplicarmos o algoritmo com suporte mínimo de 15%, obtivemos o mesmo problema apresentado no teste anterior, com isso subimos o valor para 20%, o que possibilitou termos o estabelecimento de regras de associação e, caso continuássemos com este algoritmo, nos permitiria construir um modelo preditivo. Abaixo são apresentadas algumas regras associativas geradas e os valores das métricas calculadas. O arquivo Jupyter Notebook com o código utilizado se encontra em [Market Basket v1.2.ipynb](#)

```
## Query das tabelas
query_tabelas = """
SELECT
    ds.id_compra,
    ds.cd_compra,
    it.cd_item,
    it.tx_item,
    cl.cd_cliente,
    cl.cliente_nome,
    gc.cd_grupo_cliente,
    gc.cd_pais,
    gc.tx_pais,
    gc.cd_estado,
    ds.dt_compra,
    ds.nm_quantidade,
    ds.nm_vl_item,
    ds.nm_vl_total
FROM
    dataset ds
LEFT JOIN
    clientes cl
ON
    cl.id_cliente = ds.id_cliente
LEFT JOIN
    grupo_cliente gc
ON
    gc.id_grupo = ds.id_grupo
LEFT JOIN
    itens it
ON
    it.id_item = ds.id_item
WHERE
    gc.cd_grupo_cliente in (0,1,2,3,4,6,7,10,19,22)
"""
df = ps.sqldf(query_tabelas)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
5	(RED HANGING HEART T-LIGHT HOLDER)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.223333	0.413333	0.200000	0.895522	2.166586	0.107689	5.615238
3	(JUMBO BAG PINK WITH WHITE SPOTS)	(JUMBO BAG RED RETROSPOT)	0.233333	0.323333	0.206667	0.885714	2.739323	0.131222	5.920833
1	(60 TEATIME FAIRY CAKE CASES)	(PACK OF 72 RETRO SPOT CAKE CASES)	0.226667	0.290000	0.200000	0.882353	3.042596	0.134267	6.035000
0	(PACK OF 72 RETRO SPOT CAKE CASES)	(60 TEATIME FAIRY CAKE CASES)	0.290000	0.226667	0.200000	0.689655	3.042596	0.134267	2.491852
2	(JUMBO BAG RED RETROSPOT)	(JUMBO BAG PINK WITH WHITE SPOTS)	0.323333	0.233333	0.206667	0.639175	2.739323	0.131222	2.124762
4	(WHITE HANGING HEART T-LIGHT HOLDER)	(RED HANGING HEART T-LIGHT HOLDER)	0.413333	0.223333	0.200000	0.483871	2.166586	0.107689	1.504792

Apesar de conseguirmos ter resultados para os grupos selecionados, podemos perceber que são poucas sugestões, o que acarretaria em uma baixa possibilidade de eficácia do modelo. Além disso, ao utilizarmos a base de dados da empresa, enfrentou-se outros problemas, como o uso exagerado de armazenamento em disco (O algoritmo ocupou os 130GB disponíveis no disco rígido do computador utilizado), além de necessitar de mais de 12 horas de treinamento para os grupos selecionados).

FP-Growth

A utilização do algoritmo de FP-Growth foi feita utilizando o mesmo notebook que o algoritmo de apriori, mudando apenas a função utilizada, conforme apresentado abaixo.

```
frequencies = []
rules = []
for i in baskets:
    frq = 'frequent_' + i.replace('basket_', '')
    rls = 'rules_' + i.replace('basket_', '')
    print(f"Begin - {i.replace('basket_', '')}")
    #globals()[frq] = apriori(globals()[i], min_support=0.2, use_colnames=True)
    globals()[frq] = fpgrowth(globals()[i], min_support=0.2, use_colnames=True)
    globals()[rls] = association_rules(globals()[frq], metric='lift', min_threshold=1).sort_values(['confidence', 'lift'], ascending=[False, False])
    print(f"End - {i.replace('basket_', '')}")
    frequencies.append(frq)
    rules.append(rls)
```

Inicialmente se testou com 10% de suporte mínimo, resultando em problemas fechamentos inesperados do browser. Foram feitas diversas tentativas para validar se o problema havia ocorrido por falha no sistema operacional ou no script, contudo, apesar de haver reiniciado a máquina e ter deixado o mínimo de aplicações executando juntamente com o script, o resultado foi o mesmo, e como log de erro surgiu a mensagem abaixo.

```
Begin - london
End - london
Begin - birmingham

ERROR:root:Internal Python error in the inspect module.
Below is the traceback from this internal error.

ERROR:root:Internal Python error in the inspect module.
Below is the traceback from this internal error.

ERROR:root:Internal Python error in the inspect module.
Below is the traceback from this internal error.
```

Com esse resultado, realizou-se a alteração do suporte mínimo para 15%, o que resultou em problemas de memória semelhantes aos do algoritmo de apriori. Contudo o script não apresentou mais detalhes do erro, o que não pode dar uma conclusão clara se o algoritmo de fp-growth é ou não menos custoso para a máquina. Sendo assim, seguiu-se com ele e aumentamos o suporte mínimo para 20%, resultando em um sucesso na execução do bloco com maior velocidade que o algoritmo anterior. Entretanto, quando utilizando a tabela da empresa, não foi possível obter um resultado para os grupos com mais itens, assim como não foi obtido resultados satisfatórios quando excluídos estes grupos. Visto que necessitou utilizar uma máquina com 65GB de memória RAM para executar o script com sucesso. Assim, realizando estudos de como poderíamos contornar a situação e como particionar essa análise, chegou-se ao framework do Apache Spark, o qual realiza o processamento de maneira paralela, gerando uma menor demanda de recursos de memória RAM.

```
MemoryError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_17692\2799935493.py in <module>
      5 rls = 'rules_' + i.replace("basket_", "")
      6 print(f"Begin - {i.replace("basket_", "")}")
----> 7 globals()[frq] = fpgrowth(globals()[i], min_support=0.15, use_colnames=True)
      8
      9 print(f"End - {i.replace("basket_", "")}")

~\Anaconda3\envs\Market\lib\site-packages\mlxtend\frequent_patterns\fpgrowth.py in fpgrowth(df, min_support, use_colnames, max_len, verbose)
    86 generator = fpg_step(tree, minsup, colname_map, max_len, verbose)
    87
--> 88     return fpc.generate_itemsets(generator, len(df.index), colname_map)
    89
    90

~\Anaconda3\envs\Market\lib\site-packages\mlxtend\frequent_patterns\fpccommon.py in generate_itemsets(generator, num_itemsets, colname_map)
    63 supports = []
    64 for sup, iset in generator:
--> 65     itemsets.append(frozenset(iset))
    66     supports.append(sup / num_itemsets)
    67

MemoryError:
```

Uma das vantagens da utilização do Apache Spark é a possibilidade de monitorar as tarefas que estão ocorrendo no cluster devido à existência de uma interface do usuário que é criada com a inicialização de uma sessão no Spark. Nesta tela é possível ver os Jobs em andamento, os Stages, o armazenamento (quando implementado a utilização de outras soluções Apache), As configurações do ambiente, monitoramento dos executores e informações das queries SQL dos Spark.

3.1.2
Jobs Stages Storage Environment Executors SQL
marketBasket application UI

Spark Jobs ⁽⁷⁾

User: rds5827
 Total Uptime: 45 s
 Scheduling Mode: FIFO
 Active Jobs: 1
 Completed Jobs: 2

▶ Event Timeline

- Active Jobs (1)

Page:

Job Id +	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	count at FFGrowth.scale17B <small>count at FFGrowth.scale17B</small>	2022/02/15 21:39:14 <small>(UI)</small>	11 s	2/4	217/1408 (9 running)

Page:

- Completed Jobs (2)

Page:

Job Id +	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	rdd at FFGrowth.scale169 <small>rdd at FFGrowth.scale169</small>	2022/02/15 21:39:08	5 s	1/1	8/8
0	rdd at Instrumentation.scale62 <small>rdd at Instrumentation.scale62</small>	2022/02/15 21:39:02	6 s	1/1	8/8

Page:

Para a realização dos testes, foi necessário realizar várias mudanças na estrutura dos códigos. Para a separação em Baskets, o Spark utiliza de uma estrutura diferente da utilizada nos modelos anteriores. É criado uma linha para cada cliente, onde uma coluna contém o código do cliente e uma segunda coluna contém a lista de itens comprados por este cliente. O código que gera os Baskets é exibido abaixo. Foi utilizada a função do python `globals()`, que permite a criação de variáveis passando strings como parâmetro, permitindo a criação automática dos baskets divididos por grupo de clientes.

```

## Criação de baskets diferentes para cada grupo
baskets = []
sparkdf = spark.createDataFrame(df)
sparkdf.registerTempTable("spdf")
for i in list_grupos:
    bask = 'basket_' + grupos.loc[grupos['cd_grupo_cliente'] == i, 'tx_grupo_cliente'].values[0]
    raw = spark.sql(f"SELECT
                        cd_cliente,
                        cd_item
                    FROM
                        spdf
                    WHERE
                        cd_grupo_cliente = {i} AND
                        nm_quantidade >= 1
                    ORDER BY
                        cd_cliente,
                        cd_item
                ")
    # Utilização de globals() [string] para gerar variáveis utilizando o nome do basket atual
    globals()[bask] = raw.groupBy(['cd_cliente']).agg(collect_set('cd_item').alias('items')).select(['cd_cliente', 'items']).repartition(500)
    baskets.append(bask)

```

Para realizar o treinamento dos modelos, foi utilizado o pacote `pyspark.ml.fpm` que contém a estrutura para criação de modelos de mineração de padrões frequentes. A diferença do modelo de FPGrowth utilizado no Apache Spark para os demais modelos deste algoritmo é a possibilidade de clusterização da execução do algoritmo. Ou seja, o Apache Spark permite que as tarefas sejam divididas em blocos e processadas separadamente e em paralelo. Para uma efetiva utilização do potencial do paralelismo, é necessário montar um ambiente multi-cluster, onde há uma máquina conhecida como master/driver que distribui as tarefas e interage com os clientes e máquinas Workers/executors que realizam as tarefas atribuídas, permitindo um processamento rápido e confiável. Contudo, mesmo em ambiente local, o Spark apresenta resultados acima da média, visto que, por processar em memória em pequenos blocos, apresenta uma velocidade muito superior aos demais. O código utilizado para realizar o treinamento e salvamento dos arquivos está apresentado abaixo.

```

# Treinamento de modelos FPGrowth distribuídos
models = []
fp_growth = FPGrowth(itemsCol='items', minSupport=0.2, minConfidence=0.6)
for i in baskets:
    mdl = 'model_' + i.replace('basket_', '')
    models.append(mdl)
    print(f"-----\nModel {mdl} fit - Begin")
    globals()[i].createOrReplaceTempView('baskets')
    bask_tmp = spark.sql("select items from baskets")
    globals()[mdl] = fp_growth.fit(bask_tmp)
    print(f"Model {mdl} fit - End\n-----")

for i in models:
    print('saving ' + i + ' begin')
    globals()[i].write().overwrite().save(r'C:\Github\MarketBasketAnalysis'+f'\Modelos\\{i}')
    print('saving ' + i + ' end\n')

```

Para este framework, utilizou-se também de 20% de suporte mínimo e 60% de confiança mínima. O treinamento do modelo é realizado de maneira rápida, o único ponto negativo deste framework é que a velocidade de salvamento dos modelos é dispendiosa, visto que deve realizar a manipulação de diversos arquivos fragmentados em diversas pastas. Utilizando os dados criados para este estudo, os resultados obtidos ficaram próximos daqueles dos modelos anteriores, visto que a separação dos clientes em grupos ocorreu de maneira aleatória e não por um padrão de segmento de atuação. No dataset utilizado na empresa estudada, foi possível realizar a redução do suporte mínimo de cerca de 30% para 5%, o que permite um volume de sugestões muito mais assertivo que os algoritmos anteriores.

Sugestões

A criação das sugestões também foi feita em ambiente spark. Utilizou-se os dados dos últimos 2 meses do dataset utilizado para o treinamento, criando cestas de compras menores que aquelas utilizadas no treinamento. A lógica das tabelas é similar aquela utilizada no treinamento. A leitura dos modelos e a geração das sugestões são apresentadas nas figuras abaixo.


```

## Carregando modelos salvos
models = []
for i in grupos_cliente:
    mdl_name = "model_" + grupos.loc[grupos['cd_grupo_cliente'] == i, 'tx_grupo_cliente'].values[0]
    print(f"Reading: {mdl_name}")
    models.append(mdl_name)
    globals()[mdl_name] = FPGrowthModel.read().load(path+f"\\Modelos\\{mdl_name}")
    print(f"Readed: {mdl_name}\n")
print("Done!")

```

```

## Gerando Sugestões de compras
predicoes = []
for i in pedidos:
    res = i.replace('pedidos', 'predicao')
    predicoes.append(res)
    print(f'Working on: {res}')
    globals()[res] = globals()[i.replace('pedidos', 'model')].transform(globals()[i])
    print(f'Done: {res}\n')
print('Done!')

```

Para o dataset deste trabalho é gerado uma planilha no excel que compila as sugestões de cada cliente, contendo o código e nome do mesmo, itens comprados e itens sugeridos. Para a empresa, foi criado uma tabela no banco de dados que é lida pelo software de Business Intelligence (BI) e é criada uma Dashboard que apresenta os itens para os executivos de vendas, os quais fazem as negociações com os clientes finais.

Trabalhos futuros

Para continuidade deste trabalho, em estudos futuros é interessante a realização da criação de um ambiente Spark com várias máquinas, possibilitando a utilização de todo o potencial do Framework. Também é possível realizar a criação de uma aplicação web, onde seja permitido inserir o grupo do cliente e os itens comprados, que retornaria a sugestão para os itens em questão e que também permita a inserção de um arquivo que será processado pelo algoritmo e gerará uma base de sugestões utilizando o mesmo.

Referências

- [1] <https://select-statistics.co.uk/blog/market-basket-analysis-understanding-customer-behaviour/>
- [2] <https://medium.com/@fabio.italiano/the-apriori-algorithm-in-python-expanding-thors-fan-base-501950d55be9>
- [3] <https://www.geeksforgeeks.org/item-to-item-based-collaborative-filtering/>
- [4] <https://www.geeksforgeeks.org/trie-insert-and-search/>
- [5] <https://www.linkedin.com/pulse/market-basket-analysis-why-lift-odd-metric-nadamuni-ramesh/>
- [6] [Dahdouh, Karim et al. Large-scale e-learning recommender system based on Spark and Hadoop](#)
- [7] [Han, Jiawei et al. Mining frequent patterns without candidate generation](#)
- [8] [Xin, Dong et al. Mining Compressed Frequent-Pattern Sets](#)
- [9] [Li, Haouyuan et al. Pfp: parallel fp-growth for query recommendation](#)
- [10] [Bueher, Gregory et al. Toward terabyte pattern mining: An Architecture-conscious Solution](#)
- [11] [Zhou, Yunhong et al. Large-scale Parallel Collaborative Filtering for the Netflix Prize](#)
- [12] [Maradin, Octavian. Using purchase histories for predicting the next market basket](#)
- [13] https://en.wikipedia.org/wiki/Apriori_algorithm

[14] <https://towardsdatascience.com/fp-growth-frequent-pattern-generation-in-data-mining-with-python-implementation-244e561ab1c3>

[15] https://pt.wikipedia.org/wiki/Apache_Spark