Market basket insight phase 4

Market Basket Analysis

Market basket analysis is a method or technique of data analysis for retail and marketing purpose. Market basket analysis is done to understand the purchasing behavior of customers. MBA (Market Business Analysis) is used to uncover what items are frequently brought together by the customer. Market basket analysis leads to effective sales and marketing.Market basket analysis measures the co-occurrence of products and services. Market basket analysis is only considered when there is a transaction between two or more items.

Eg:if a customer is buying bread then he is likely to buy butter, jam or milk to compliment bread.

Applications of Market Basket Analysis

Market basket analysis is applied to various fields of the retail sector in order to boost sales and generate revenue by identifying the needs of the customers and make purchase suggestions to them.

- Cross-selling is basically a sales technique in which seller suggests some related product to a customer after he buys a product.
- Product Placement: It refers to placing the complimentary (pen and paper)and substitute goods (tea and coffee) together so that the customer addresses the goods and will buy both the goods together.
- MBA has also been used in the field of healthcare for the detection of adverse drug reactions. It produces association rules that indicates what all combinations of medications and patient characteristics lead to ADRs.
- Fraud Detection: Market basket analysis is also applied to fraud detection. It may be possible to identify purchase behavior that can associate with fraud on the basis of market basket analysis data that contain credit card usage

How Market Based Analysis Works

In order to make it easier to understand, think of Market Basket Analysis in terms of shopping at a supermarket. Market Basket Analysis takes data at transaction level, which lists all items bought by a customer in a single purchase. The technique determines relationships of what

products were purchased with which other product(s). These relationships are then used to build profiles containing If-Then rules of the items purchased.

The rules are written as :

if {A} then {B} i.e. {A} => {B}

The If part of the rule (the {A} above) is known as the antecedent and the THEN part of the rule is known as the consequent (the {B} above). The antecedent is the condition and the consequent is the result.

Assosciation Rules

Association Rules are widely used to analyze retail basket or transaction data, and are intended to identify strong rules discovered in transaction data using measures of interestingness, based on the concept of strong rules.

Let I={i1,i2,i3,…,in} be a set of n attributes called items and D={t_1,t_2,…,t_n} be the set of transactions. It is called database. Every transaction, t_i in D has a unique transaction ID, and it consists of a subset of itemsets in I.

Assosciation rules are produced using algorithms like :

- Apriori Algorithm
- Eclat Algorithm
- FP-growth Algorithm

A rule can be defined as an implication, **X**$\rightarrow$**Y** where X and Y are subsets of I(X,Y$\subseteq$I), and they have no element in common. X and Y are the antecedent and the consequent of the rule, respectively.

Eg: {Bread,Egg}=> {Milk} ItemSet={Bread,Egg,Milk}

There are various metrics in place to help us understand the strength of assosciation between antecedent and consequent:

- Support

- Confidence
- Lift or Correlation or interest
- Leverage
- Conviction

Support

It gives an idea of how frequent an itemset is in all the transactions.To say in formal terms it's the fraction of total no. of transactions in which the itemset occurs.We refer to an itemset as a "frequent itemset" if you support is larger than a specified minimum-support threshold.

$$supp(X->Y)$$

$$=$$

$$(TransactionscontainingbothXandY)$$

$$(TotalNo.oftransactions)$$

Range:[0,1] Value of support helps us identifying the rules worth for future analysis.

Confidence

It defines the likelihood of occurence of consequent on the cart given that cart already has antecedent.It signifies the likelihood of item Y being purchased when item X is purchased.

$$confidence(X->Y)=$$

$$support(X->Y)$$

$$support(X)$$

Range:[0,1]

If confidence is 0.75 then that imples that 75%of transactions containing X also contain Y .It can also be interpreted as the conditional probability P(Y|X), i.e, the probability of finding the itemset Y in transactions given the transaction already contains X.

It has a major drawback i.e. It only takes into account the popularity of the itemset X and not the popularity of Y. If Y is equally popular as X then there will be a higher probability that a transaction containing X will also contain Y thus increasing the confidence. To overcome this drawback there is another measure called lift.

Lift

Lift gives the rise in the probability of having {Y} on the cart with the knowlede of {X} being present over the probability of having {Y} on the cart without knowlede about presence of {X}.

$$\text{Lift}(X->Y)=$$

$$\text{confidence}(X->Y)$$

$$\text{support}(Y)$$

Range:[0,Infinity]

It can simply be considered as correlation between the antecedent and consequent.If the value of lift is greater than 1, it means that the itemset Y is likely to be bought with itemset X, while a value less than 1 implies that itemset Y is unlikely to be bought if the itemset X is bought.

Levarage or Piatetsky-Snapiro

It computes the difference between the observed frequency of X & Y appearing together and the frequency that we would expect if A and C are independent.

$$\text{Leverage}(X->Y)=\text{support}(X->Y)$$

$$-\text{support}(X)*\text{support}(Y)$$

Range:[-1,1]

If X,Y are positively correlated then we get leverage>0 ,we need such type of rules.
If X,Y are negatively correlated then we get leverage<0.
If X,y are independent , then we get leverage = 0.

Conviction

It can be interpreted as the ratio of the expected frequency that X occurs without Y (that is to say, the frequency that the rule makes an incorrect prediction) if X and Y were independent divided by the observed frequency of incorrect predictions.

$$\text{Conviction}(X \rightarrow Y) = \frac{\text{support}(Y)}{\text{confidence}(X \rightarrow Y)}$$

Please mark in the above equation Y means it is Y bar i.e. a bar on Y

Range:[0,Infinity]

A high conviction value means that the consequent is highly depending on the antecedent. For instance, in the case of a perfect confidence score, the denominator becomes 0 (due to 1 - 1) for which the conviction score is defined as 'inf'. Similar to lift, if items are independent, the conviction is 1.

Apriori Algorithm

Apriori algorithm is a classical algorithm in data mining. It is used for mining frequent itemsets and relevant association rules. It is devised to operate on a database containing a lot of transactions, for instance, items brought by customers in a store.Association rule learning is a prominent and a well-explored method for determining relations among variables in large databases.

Rule - generation is a two step process. First is to generate frequent item set and second is to generate rules from the considered itemset.

1.Generating Frequent Itemset:

One approach to find the frequent itemsets is to check all possible subsets of the given item set and check the support value of each itemset and consider only those that have support values greater than the minimum threshold support value.

Here the Apriori uses the result of antimontone property of support and makes the generation of frequent Item set faster by reducing the search space.It has two principles:

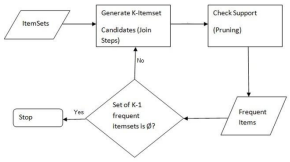
1. All subsets of a frequent itemset must be frequent
2. Similarly, for any infrequent itemset, all its supersets must be infrequent too

Apriori principle allows us to prune all supersets of an itemset which does nogt satisfy the minimum threshold condition for support.For example if {Milk,Bread} does not satify our threshold value , then the superset of {Milk,Bread} will also not cross the threshold value there by we can just prune them away i.e. do not consider the itemsets that wil be generated from the {Milk,Bread}.

Totally 3 major steps are involved here:

1. Generate all frequent itemsets each satisfying the minimum threshold and having only one item let it be L_1. Next use self join and generate all possible combinations of L_1 and now let the result be L_2.
2. At each step as we keep on generating candidate itemsets, for each candidate we scan entire database so as to know its support and remove the candidates that do not satisfy minimum threshold
3. **Here To reduce the no of comparisions, store the generated candidate items in a Hash Tree,Instead of matching each of candidate itemsets against each transaction,match each transaction with the candidates in hash tree(there by we can enhance the speed of apriori using this method)**
   In similar way create L_k from L_{k-1} until the point where we are unable to apply selfjoin.

This approach of extending a frequent itemset one at a time is called the “bottom up” approach.



2.Generating all possible rules from Frequent Itemsets

If n items are in set I , no of possible assosciation rules possible are 3<supn</sup>- 2^{n+1} + 1.It becomes computationally expensive to generate all the rules and there is no meaning in genearting all that many no. of rules.

So apriori simplifies this approach by following some methodology,

Rules are formed by binary partition of each itemset.From a list of all possible candidate rules, we aim to identify rules that fall above the minimum confidence level.Just like antimontone property of support, confidence of rules generated from same itemset also follow the anti montone property. It's antimontone w.r.t no. of elements in consequent.

=> CONF(A,B,C -> D) >= CONF(B,C -> A,D) >= CONF(C -> A,B,D)

On the basis of this rules are generated.

If you want to refer further on Advanced Apriori Algorithms, Please refer to Advanced Apriori Algorithms.

Apriori uses a breadth-first search strategy to count the support of itemsets and uses a candidate generation function which exploits the downward closure property of support.

Pros of the Apriori algorithm:

- It is an easy-to-implement and easy-to-understand algorithm.
- It can be used on large itemsets.
- Cons of the Apriori Algorithm:
    - Sometimes, it may need to find a large number of candidate rules which can be computationally expensive.

- Calculating support is also expensive because it has to go through the entire database

Development

Importing Libraries

In [1]:

```
#Data manipulation libraries

import pandas as pd

import numpy as np



#Visualizations

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style("dark")

import squarify

import matplotlib



#for market basket analysis (using apriori)
```

```
from mlxtend.frequent_patterns import apriori

from mlxtend.frequent_patterns import association_rules



#for preprocessing

from mlxtend.preprocessing import TransactionEncoder




#to print all the interactive output without resorting to print, not only the last result.

from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast_node_interactivity = "all"
```

Importing data

```
data = pd.read_csv('/content/market basket .csv',
sep=';',parse_dates=['Date'])

df.head()
```

Output

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Country
0	536365	WHITE HANGING HEART T-LIGHT HOLDER	6.0	2010-01-12 08:26:00	2,55	17850.0	United Kingdom
1	536365	WHITE METAL LANTERN	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom
2	536365	CREAM CUPID HEARTS COAT HANGER	8.0	2010-01-12 08:26:00	2,75	17850.0	United Kingdom
3	536365	KNITTED UNION FLAG HOT	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom

		WATER BOTTLE					
4	536365	RED WOOLLY HOTTIE WHITE HEART.	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom

data.shape

Output

```
(133459, 7)
```

```
#head of data
```

```
data.head()
```

Output

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Country

0	536365	WHITE HANGING HEART T-LIGHT HOLDER	6.0	2010-01-12 08:26:00	2,55	17850.0	United Kingdom
1	536365	WHITE METAL LANTERN	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom
2	536365	CREAM CUPID HEARTS COAT HANGER	8.0	2010-01-12 08:26:00	2,75	17850.0	United Kingdom
3	536365	KNITTED UNION FLAG HOT WATER BOTTLE	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom
4	536365	RED WOOLLY HOTTIE WHITE HEART.	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom

```
data.tail()
```

Output

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Country
133454	548194	PIG KEYRING WITH LIGHT & SOUND	1.0	2011-03-29 15:30:00	2,46	NaN	United Kingdom
133455	548194	COFFEE MUG BLUE PAISLEY DESIGN	1.0	2011-03-29 15:30:00	4,96	NaN	United Kingdom
133456	548194	OFFICE MUG WARMER BLACK+ SILVER	1.0	2011-03-29 15:30:00	5,79	NaN	United Kingdom
133457	548194	BIRD DECORATION RED	4.0	2011-03-29 15:30:00	1,63	NaN	United Kingdom

		RETROS POT					
133458	548194	ROUND SNACK BOXES SET	NaN	NaT	NaN	NaN	NaN

Here we can find that data needs a lot of preprocessing.So let's preprocess the data.We use the TransactionEncoder() of mlxtend.preprocessing to do this work for us, if needed even we can implement the function i will provide the alternative as well.The TransactionEncoder() is an Encoder class for transaction data in Python list.It finds out what are all the different products in the transactions and will assign each transaction a list which contains a boolean array where each index represnts the corresponding product whether purchased in the transaction or not i.e. True or False.

It needs input as a python list of lists, where the outer list stores the n transactions and the inner list stores the items.

It returns the one-hot encoded boolean array of the input transactions, where the columns represent the unique items found in the input array in alphabetic order. For further details you can refer its documentation:TransactionEncoder

```
#converting into required format of TransactionEncoder()

trans=[]
```

```
for i in range(0,7501):
    trans.append([str(data.values[i,j]) for j in range(0,20)])

trans=np.array(trans)

print(trans.shape)
```

Output

```
(7501, 20)
```

```
t=TransactionEncoder()
data=t.fit_transform(trans)
data=pd.DataFrame(data,columns=t.columns_,dtype=int)

data.shape
```

Output

```
(7501, 121)
```

```
##here we also find nan as one of the columns so lets drop that column
```

```
data.drop('nan',axis=1,inplace=True)
```

now lets check shape

```
data.shape
```

```
#lets verify whether nan is present in columns
```

```
'nan' in data.columns
```

```
#so its proved that nan is not in columns
```

Output

```
(133459, 7)False
```

```
data.head()
```

Output

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.

and should_run_async(code)

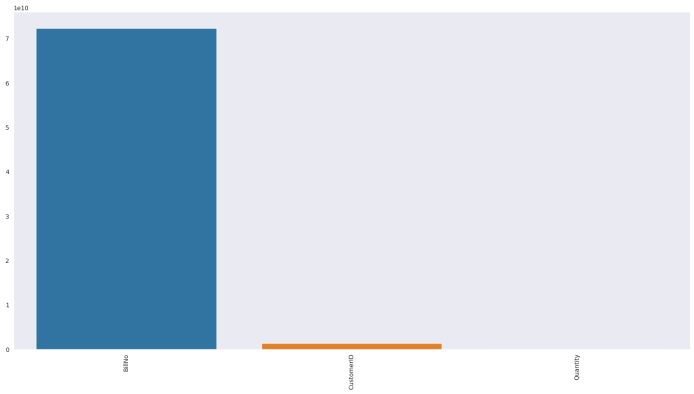
	BillNo	Itemname	Quantity	Date	Price	CustomerID	Country
0	536365	WHITE HANGING HEART T-LIGHT HOLDER	6.0	2010-01-12 08:26:00	2,55	17850.0	United Kingdom
1	536365	WHITE METAL LANTERN	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom
2	536365	CREAM CUPID HEARTS COAT HANGER	8.0	2010-01-12 08:26:00	2,75	17850.0	United Kingdom
3	536365	KNITTED UNION FLAG HOT WATER BOTTLE	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom

4	536365	RED WOOLLY HOTTIE WHITE HEART.	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom

Data Visualizations

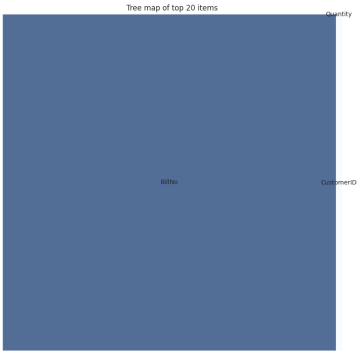
```
##Lets consider the top 20 items purchased freequently

r=data.sum(axis=0).sort_values(ascending=False)[:20]

#altering the figsize

plt.figure(figsize=(20,10))

s=sns.barplot(x=r.index,y=r.values)

s.set_xticklabels(s.get_xticklabels(), rotation=90)
```

Output



```
# create a color palette, mapped to these values

my_values=r.values

cmap = matplotlib.cm.Blues

mini=min(my_values)

maxi=max(my_values)

norm = matplotlib.colors.Normalize(vmin=mini, vmax=maxi)

colors = [cmap(norm(value)) for value in my_values]




#treemap of top 20 frequent items

plt.figure(figsize=(10,10))

squarify.plot(sizes=r.values, label=r.index, alpha=.7,color=colors)

plt.title("Tree map of top 20 items")

plt.axis('off')
```

Output



Work on apriori

#let us return items and ietmsets with atleast 5% support:

freq_items=apriori(data,min_support=0.05,use_colnames=True

freq_items

Output

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.

and should_run_async(code)

	support	itemsets
0	1.0	(BillNo)
1	1.0	(Itemname)
2	1.0	(Quantity)

```
3     1.0   (Date)

4     1.0   (Price)

...   ...   ...

122   1.0   (Itemname, Quantity, Country, Price, CustomerI...

123   1.0   (Itemname, Country, Price, Date, CustomerID, B...

124   1.0   (Quantity, Country, Price, Date, CustomerID, B...

125   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

126   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns
```

```
#Now let's generate association rules
```

```
res=association_rules(freq_items,metric="lift",min_threshold=1.3)
```

```
res
```

Output

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.

and should_run_async(code)

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric

Selecting and Filtering the Results

```
frequent_itemsets = apriori(data, min_support = 0.05,
use_colnames=True)
```

```
frequent_itemsets['length'] =
frequent_itemsets['itemsets'].apply(lambda x: len(x))
```

```
frequent_itemsets
```

Output

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.

and should_run_async(code)

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type

warnings.warn(

	support	itemsets	length
0	1.0	(BillNo)	1
1	1.0	(Itemname)	1
2	1.0	(Quantity)	1
3	1.0	(Date)	1
4	1.0	(Price)	1
...	...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...	6

123	1.0	(Itemname, Country, Price, Date, CustomerID, B...	6
124	1.0	(Quantity, Country, Price, Date, CustomerID, B...	6
125	1.0	(Itemname, Quantity, Country, Price, Date, Cus...	6
126	1.0	(Itemname, Quantity, Country, Price, Date, Cus...	7

127 rows × 3 columns

```
# getting th item sets with length = 2 and support more han 10%
```

```
frequent_itemsets[ (frequent_itemsets['length'] == 2) &

                   (frequent_itemsets['support'] >= 0.01) ]
```

```
Output
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
```

```
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.

  and should_run_async(code)
```

	support	itemsets	length
7	1.0	(Itemname, BillNo)	2
8	1.0	(Quantity, BillNo)	2
9	1.0	(Date, BillNo)	2
10	1.0	(Price, BillNo)	2
11	1.0	(BillNo, CustomerID)	2
12	1.0	(Country, BillNo)	2
13	1.0	(Itemname, Quantity)	2
14	1.0	(Itemname, Date)	2
15	1.0	(Itemname, Price)	2
16	1.0	(Itemname, CustomerID)	2
17	1.0	(Itemname, Country)	2
18	1.0	(Quantity, Date)	2
19	1.0	(Price, Quantity)	2
20	1.0	(Quantity, CustomerID)	2

```
21    1.0   (Country, Quantity)    2

22    1.0   (Price, Date)    2

23    1.0   (Date, CustomerID)     2

24    1.0   (Country, Date)  2

25    1.0   (Price, CustomerID)    2

26    1.0   (Country, Price) 2

27    1.0   (Country, CustomerID)  2
```

```
# getting th item sets with length = 2 and support more han 10%
```

```
frequent_itemsets[ (frequent_itemsets['length'] == 1) &

                   (frequent_itemsets['support'] >= 0.01) ]
```

```
Output
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.

  and should_run_async(code)
```

```
support    itemsets    length

0     1.0   (BillNo)    1
```

1 1.0 (Itemname) 1

2 1.0 (Quantity) 1

3 1.0 (Date) 1

4 1.0 (Price) 1

5 1.0 (CustomerID) 1

6 1.0 (Country) 1

ECLAT ALGORITHM

BottleNecks of Apriori:

Candidate generation can result in huge candidate sets

Multiple Scans of Database--- needs (n+1) scans,n is the longest pattern

To solve some of the above problems,Eclat has been introduced.

The ECLAT algorithm stands for Equivalence Class Clustering and bottom-up Lattice Traversal. It is one of the popular methods of Association Rule mining. It is a more efficient and scalable version of the Apriori algorithm. While the Apriori algorithm works in a horizontal sense imitating the Breadth-First Search of a graph, the ECLAT algorithm works in a vertical manner just like the Depth-First Search of a graph. This vertical approach of the ECLAT algorithm makes it a faster algorithm than the Apriori algorithm.

How it works:

The basic idea is to use Transaction Id Sets(tidsets) intersections to compute the support value of a candidate and avoiding the generation of subsets which do not exist in the prefix tree. In the first call of the function, all single items are used along with their tidsets. Then the function is called recursively and in each recursive call, each item-tidset pair is verified and combined with other item-tidset pairs. This process is continued until no candidate item-tidset pairs can be combined

Eg:

t1={a,b,c} t2={a,b} t3={a}

now above is horizontal layout where t1,t2,t3 are transactions a,b,c are products.now let's make it into vertical layout....

k=1,min_support=0.5 a={t1,t2,t3},sup=1 b={t1,t2},sup=0.66
c={t1},sup=0.33

now we eliminate c as is supp<min_support and them generate itemsets of length k=2

{a,b}={t1,t2} supp=0.5

and we can't generate anymore sets so we end up with only {a,b}.

This method has an advantage over Apriori as it does not require scanning the database to find the support of k+1 itemsets. This is because the Transaction set will carry the count of occurrence of each item in the transaction (support). The bottleneck comes when there are many transactions taking huge memory and computational time for intersecting the sets.

If you want further reference you can visit : Eclat Algo

Advantages over Apriori algorithm:-

Memory Requirements: Since the ECLAT algorithm uses a Depth-First Search approach, it uses less memory than Apriori algorithm.

Speed: The ECLAT algorithm is typically faster than the Apriori algorithm.

Number of Computations: The ECLAT algorithm does not involve the repeated scanning of the data to compute the individual support values.

FP GROWTH(Frequent Pattern Growth)

Shortcomings Of Apriori Algorithm

-Using Apriori needs a generation of candidate itemsets. These itemsets may be large in number if the itemset in the database is huge.

-Apriori needs multiple scans of the database to check the support of each itemset generated and this leads to high costs. These shortcomings can be overcome using the FP growth algorithm.

This algorithm is an improvement to the Apriori method. A frequent pattern is generated without the need for candidate generation. FP growth algorithm represents the database in the form of a tree called a frequent pattern tree or FP tree.

This tree structure will maintain the association between the itemsets. The database is fragmented using one frequent item. This fragmented part is called “pattern fragment”. The itemsets of these fragmented patterns are analyzed. Thus with this method, the search for frequent itemsets is reduced comparatively.

FP TREE

Frequent Pattern Tree is a tree-like structure that is made with the initial itemsets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the itemset.

The root node represents null while the lower nodes represent the itemsets. The association of the nodes with the lower nodes that is the itemsets with the other itemsets are maintained while forming the tree.

Frequent Pattern Algorithm Steps

The frequent pattern growth method lets us find the frequent pattern without candidate generation.

Let us see the steps followed to mine the frequent pattern using frequent pattern growth algorithm:

1) The first step is to scan the database to find the occurrences of the itemsets in the database. This step is the same as the first step of Apriori. The count of 1-itemsets in the database is called support count or frequency of 1-itemset.

2) The second step is to construct the FP tree. For this, create the root of the tree. The root is represented by null.

3) The next step is to scan the database again and examine the transactions. Examine the first transaction and find out the itemset in it. The itemset with the max count is taken at the top, the next itemset with lower count and so on. It means that the branch of the tree is constructed with transaction itemsets in descending order of count.

4) The next transaction in the database is examined. The itemsets are ordered in descending order of count. If any itemset of this transaction is already present in another branch (for example in the 1st transaction), then this transaction branch would share a common prefix to the root.

This means that the common itemset is linked to the new node of another itemset in this transaction.

5) Also, the count of the itemset is incremented as it occurs in the transactions. Both the common node and new node count is increased by 1 as they are created and linked according to transactions.

6) The next step is to mine the created FP Tree. For this, the lowest node is examined first along with the links of the lowest nodes. The lowest node represents the frequency pattern length 1. From this, traverse the path in the FP Tree. This path or paths are called a conditional pattern base.

Conditional pattern base is a sub-database consisting of prefix paths in the FP tree occurring with the lowest node (suffix).

7) Construct a Conditional FP Tree, which is formed by a count of itemsets in the path. The itemsets meeting the threshold support are considered in the Conditional FP Tree.

8) Frequent Patterns are generated from the Conditional FP Tree.

```
#Importing Libraries



from mlxtend.frequent_patterns import fpgrowth

#running the fpgrowth algorithm

res=fpgrowth(data,min_support=0.05,use_colnames=True)

res
```

Output

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.

  and should_run_async(code)
```

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)
...	...	...

```
122   1.0   (Itemname, Quantity, Country, Price, CustomerI...

123   1.0   (Itemname, Quantity, Country, Date, CustomerID...

124   1.0   (Itemname, Quantity, Country, Price, Date, Bil...

125   1.0   (Itemname, Quantity, Price, Date, CustomerID, ...

126   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns
```

```
res=association_rules(res,metric="lift",min_threshold=1)
```

```
res
```

Output

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.

and should_run_async(code)

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric

0	(Country)	(CustomerID)	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0
1	(CustomerID)	(Country)	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0
2	(Country)	(Price)	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0
3	(Price)	(Country)	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0
4	(Country)	(Date)	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0
...	...	...	...	...	...	...	...	...	...	...
1927	(Country)	(Itemname, Quantity, Price, Date, Custo	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0

		merID, ...								
1928	(Price)	(Itemname, Quantity, Country, Date, CustomerID ...	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0
1929	(Date)	(Itemname, Quantity, Country, Price, CustomerI...	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0
1930	(CustomerID)	(Itemname, Quantity, Country, Price, Date, Bil...	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0

1931	(BillNo)	(Itemname, Quantity, Country, Price, Date, Cus...	1.0	1.0	1.0	1.0	1.0	0.0	inf	0.0

1932 rows × 10 columns

Apriori Vs FP Growth

Since FP-Growth doesn't require creating candidate sets explicitly, it can be magnitudes faster than the alternative Apriori algorithm. FP-Growth is about 5 times faster.Let's look at it.

```
import time

l=[0.01,0.02,0.03,0.04,0.05]

t=[]

for i in l:

    t1=time.time()

    apriori(data,min_support=i,use_colnames=True)

    t2=time.time()

    t.append((t2-t1)*1000)
```

Output

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.


  and should_run_async(code)


/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcom
mon.py:110: DeprecationWarning: DataFrames with non-bool types result
in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type


  warnings.warn(


support    itemsets


0     1.0   (BillNo)


1     1.0   (Itemname)


2     1.0   (Quantity)


3     1.0   (Date)


4     1.0   (Price)


...   ...   ...


122   1.0   (Itemname, Quantity, Country, Price, CustomerI...


123   1.0   (Itemname, Country, Price, Date, CustomerID, B...


124   1.0   (Quantity, Country, Price, Date, CustomerID, B...


125   1.0   (Itemname, Quantity, Country, Price, Date, Cus...
```

```
126   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns
```

```
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcom
mon.py:110: DeprecationWarning: DataFrames with non-bool types result
in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type

  warnings.warn(
```

	support	itemsets
0	1.0	(BillNo)
1	1.0	(Itemname)
2	1.0	(Quantity)
3	1.0	(Date)
4	1.0	(Price)
...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...
123	1.0	(Itemname, Country, Price, Date, CustomerID, B...

```
124   1.0   (Quantity, Country, Price, Date, CustomerID, B...

125   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

126   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns
```

```
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcom
mon.py:110: DeprecationWarning: DataFrames with non-bool types result
in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type

  warnings.warn(
```

	support	itemsets
0	1.0	(BillNo)
1	1.0	(Itemname)
2	1.0	(Quantity)
3	1.0	(Date)
4	1.0	(Price)
...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...

```
123   1.0   (Itemname, Country, Price, Date, CustomerID, B...

124   1.0   (Quantity, Country, Price, Date, CustomerID, B...

125   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

126   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns
```

```
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcom
mon.py:110: DeprecationWarning: DataFrames with non-bool types result
in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type

  warnings.warn(
```

	support	itemsets
0	1.0	(BillNo)
1	1.0	(Itemname)
2	1.0	(Quantity)
3	1.0	(Date)
4	1.0	(Price)
...	...	...

```
122   1.0   (Itemname, Quantity, Country, Price, CustomerI...

123   1.0   (Itemname, Country, Price, Date, CustomerID, B...

124   1.0   (Quantity, Country, Price, Date, CustomerID, B...

125   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

126   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns
```

```
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcom
mon.py:110: DeprecationWarning: DataFrames with non-bool types result
in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type

  warnings.warn(
```

	support	itemsets
0	1.0	(BillNo)
1	1.0	(Itemname)
2	1.0	(Quantity)
3	1.0	(Date)
4	1.0	(Price)

```
...   ...   ...

122   1.0   (Itemname, Quantity, Country, Price, CustomerI...

123   1.0   (Itemname, Country, Price, Date, CustomerID, B...

124   1.0   (Quantity, Country, Price, Date, CustomerID, B...

125   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

126   1.0   (Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns
```

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.

and should_run_async(code)

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type

warnings.warn(

	support	itemsets

0	1.0	(BillNo)
1	1.0	(Itemname)
2	1.0	(Quantity)
3	1.0	(Date)
4	1.0	(Price)
...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...
123	1.0	(Itemname, Country, Price, Date, CustomerID, B...

124	1.0	(Quantity, Country, Price, Date, CustomerID, B...
125	1.0	(Itemname, Quantity, Country, Price, Date, Cus...
126	1.0	(Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type

warnings.warn(

	support	itemsets
0	1.0	(BillNo)
1	1.0	(Itemname)

2	1.0	(Quantity)
3	1.0	(Date)
4	1.0	(Price)
...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...
123	1.0	(Itemname, Country, Price, Date, CustomerID, B...
124	1.0	(Quantity, Country, Price, Date, CustomerID, B...
125	1.0	(Itemname, Quantity, Country, Price, Date, Cus...

126	1.0	(Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type

warnings.warn(

	support	itemsets
0	1.0	(BillNo)
1	1.0	(Itemname)
2	1.0	(Quantity)
3	1.0	(Date)

4	1.0	(Price)
...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...
123	1.0	(Itemname, Country, Price, Date, CustomerID, B...
124	1.0	(Quantity, Country, Price, Date, CustomerID, B...
125	1.0	(Itemname, Quantity, Country, Price, Date, Cus...
126	1.0	(Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type

warnings.warn(

	support	itemsets
0	1.0	(BillNo)
1	1.0	(Itemname)
2	1.0	(Quantity)
3	1.0	(Date)
4	1.0	(Price)
...	...	...

122	1.0	(Itemname, Quantity, Country, Price, CustomerI...
123	1.0	(Itemname, Country, Price, Date, CustomerID, B...
124	1.0	(Quantity, Country, Price, Date, CustomerID, B...
125	1.0	(Itemname, Quantity, Country, Price, Date, Cus...
126	1.0	(Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns

```
l=[0.01,0.02,0.03,0.04,0.05]

f=[]

for i in l:

    t1=time.time()

    fpgrowth(data,min_support=i,use_colnames=True)
```

```
    t2=time.time()

    f.append((t2-t1)*1000)
```

Output

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please pass the
result to `transformed_cell` argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.

  and should_run_async(code)

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110:
DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance
and their support might be discontinued in the future.Please use a DataFrame with bool type

  warnings.warn(

support itemsets

0       1.0     (Country)

1       1.0     (CustomerID)

2       1.0     (Price)

3       1.0     (Date)

4       1.0     (Quantity)

...     ...     ...

122     1.0     (Itemname, Quantity, Country, Price, CustomerI...

123     1.0     (Itemname, Quantity, Country, Date, CustomerID...
```

124	1.0	(Itemname, Quantity, Country, Price, Date, Bil...
125	1.0	(Itemname, Quantity, Price, Date, CustomerID, ...
126	1.0	(Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type

warnings.warn(

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)
...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...

123	1.0	(Itemname, Quantity, Country, Date, CustomerID...
124	1.0	(Itemname, Quantity, Country, Price, Date, Bil...
125	1.0	(Itemname, Quantity, Price, Date, CustomerID, ...
126	1.0	(Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type

warnings.warn(

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)
...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...

123	1.0	(Itemname, Quantity, Country, Date, CustomerID...
124	1.0	(Itemname, Quantity, Country, Price, Date, Bil...
125	1.0	(Itemname, Quantity, Price, Date, CustomerID, ...
126	1.0	(Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type

warnings.warn(

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)
...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...

123	1.0	(Itemname, Quantity, Country, Date, CustomerID...
124	1.0	(Itemname, Quantity, Country, Price, Date, Bil...
125	1.0	(Itemname, Quantity, Price, Date, CustomerID, ...
126	1.0	(Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type

warnings.warn(

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)
...	...	...
122	1.0	(Itemname, Quantity, Country, Price, CustomerI...

123 1.0 (Itemname, Quantity, Country, Date, CustomerID...

124 1.0 (Itemname, Quantity, Country, Price, Date, Bil...

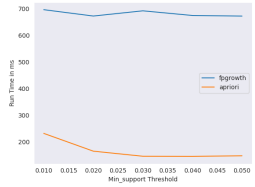
125 1.0 (Itemname, Quantity, Price, Date, CustomerID, ...

126 1.0 (Itemname, Quantity, Country, Price, Date, Cus...

127 rows × 2 columns

```
sns.lineplot(x=l,y=f,label="fpgrowth")

sns.lineplot(x=l,y=t,label="apriori")

plt.xlabel("Min_support Threshold")

plt.ylabel("Run Time in ms")
```

Output



Conclusion:

Association analysis, often referred to as market basket analysis in the context of retail, is a technique used to discover interesting relationships between items in a dataset. This analysis is commonly used to identify which items are frequently purchased together, helping businesses make informed decisions about product placement, promotions, and more.

Here are the basic steps for performing association analysis and generating insights in market basket analysis:

1. **Data Collection**: Gather transaction data, where each transaction lists the items purchased by a customer.

2. **Data Preprocessing**: Organize the data into a suitable format, such as a matrix or list of transactions, with items as columns and transactions as rows.

3. **Support and Confidence**: Calculate support and confidence for item pairs. Support measures how often items appear together, while confidence measures how likely one item is purchased when another is.

4. **Setting Thresholds**: Decide on minimum support and confidence thresholds. This helps filter out associations that are too weak or infrequent to be meaningful.

5. **Finding Association Rules**: Use algorithms like Apriori or FP-growth to find association rules. An association rule typically has the form: {Item A} => {Item B} with a support and confidence value.

6. **Interpreting Results**: Explore the discovered association rules to gain insights. Focus on rules with high support and confidence, as these are the most reliable associations.

7. **Visualizing and Presenting Insights**: Visualize the results with tools like scatter plots, bar charts, or word clouds. These visuals can make it easier to understand and communicate the insights.

8. **Implementing Business Decisions**: Use the insights to inform business decisions, such as optimizing product placement, creating cross-selling strategies, or designing promotions.

9. **Continual Monitoring**: Market basket analysis is an ongoing process. Continually gather data and re-run the analysis to adapt to changing customer behavior and market trends.