Market-Basket Analysis

UNIVERSITÀ
DEGLI STUDI
DI MILANO

Algorithms for Massive Data Course
Project September 2022

Mahdi Sotikhiabani

“I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.”

Introduction

This project presents a market basket analysis(MBA) implemented in Pyspark using the FP-growth algorithm on the tweets about Ukraine-Russian conflict provided by Kaggle. The idea behind MBA is to find the frequent itemsets which are the items that are frequently found in the same basket. The word frequent refers to a threshold number which is defined by the user. The most classical example of MBA is the purchases of customers at the marketplace. Each transaction is a basket and each product in that transaction is the item. Therefore, given that our dataset is composed of tweets which are textual information, the tweets are considered as baskets and unique words found in the tweets are considered as items.

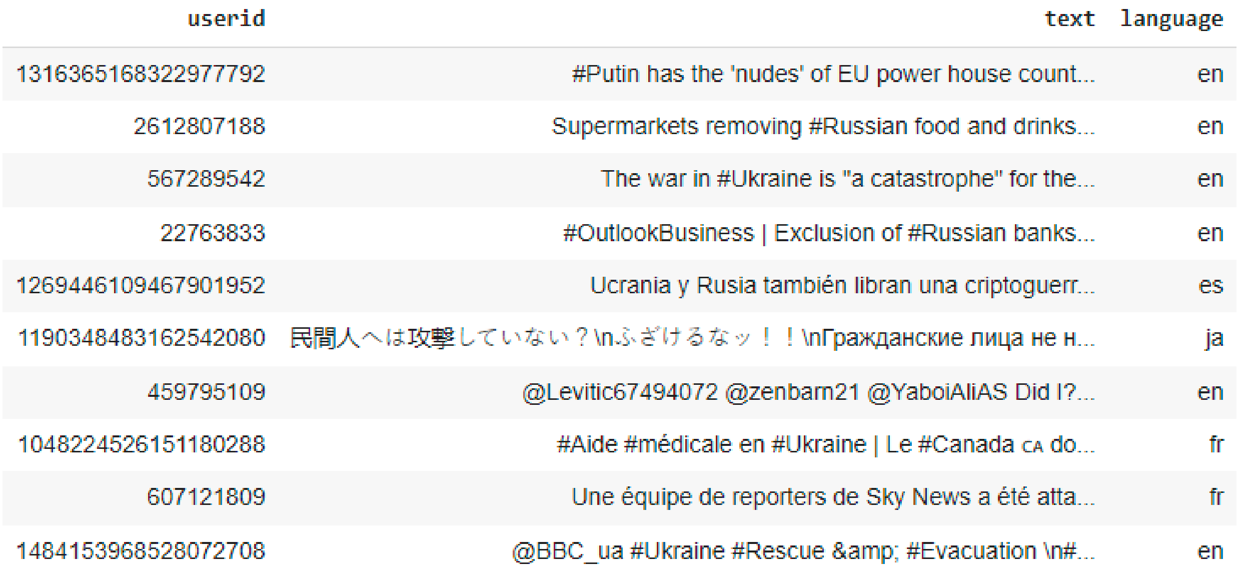
Dataset

Dataset is acquired from a Kaggle repository which provides tweets about Ukraine-Russian conflict from 27 February to 31 March. The total number of tweets found in the whole dataset is over 15 millions. However, given our computational resources, we are not able to use all the data provided. We thus use a smaller portion of the dataset but later try to show the scalability of the solution, there are 17 features provided like timestamp, location, language etc. However, given the scope of our analysis, we will need only two of them which are : language, text.

- Text feature presents the actual textual content of a given tweet.
- Language shows the language in which a tweet is tweeted.

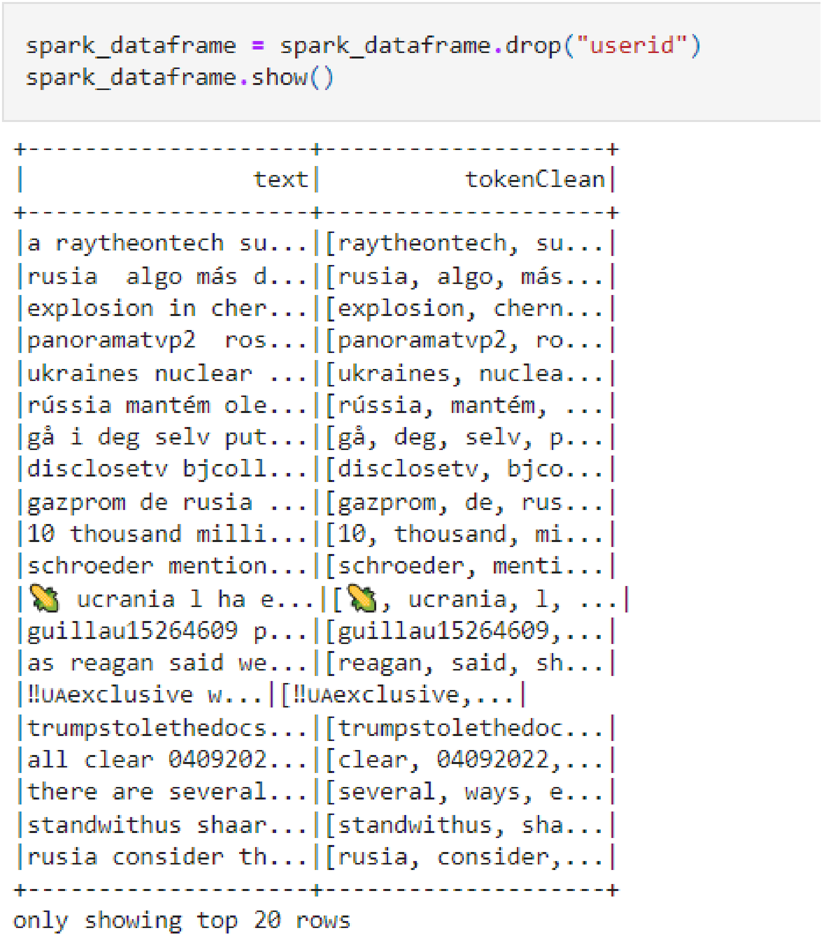
Pre-processing

Given the problem at hand(MBA), we need to create a basket of words where a basket represents an actual basket for MBA and words in it represent the items. Another thing to consider here is to have unique words for each basket. For example, the word “Ukraine” should not occur two times in a basket. We thus first tokenize the text, clean the text from stopwords, get only the unique items and eventually remove white spaces caused by this cleaning process. However, before diving into the pre-processing, the first thing we should do is to extract only the english tweets since tokenization and stopwords removal might create problems for other languages not recognized by Pyspark. English language filtering is followed by removing duplicates(re-tweets).



userid	text	language
1316365168322977792	#Putin has the 'nudes' of EU power house count...	en
2612807188	Supermarkets removing #Russian food and drinks...	en
567289542	The war in #Ukraine is "a catastrophe" for the...	en
22763833	#OutlookBusiness | Exclusion of #Russian banks...	en
1269446109467901952	Ucrania y Rusia también libran una criptoguerr...	es
1190348483162542080	民間人へは攻撃していない？\nふざけるなッ！！\nГражданские лица не н...	ja
459795109	@Levitic67494072 @zenbarn21 @YaboiAliAS Did I?...	en
1048224526151180288	#Aide #médicale en #Ukraine | Le #Canada CA do...	fr
607121809	Une équipe de reporters de Sky News a été atta...	fr
1484153968528072708	@BBC_ua #Ukraine #Rescue &amp; #Evacuation \n#...	en

This is how our dataset looks like before cleaning and before getting it into the distributed framework of Pyspark. The initial shape of the dataset is (507452). After removing the duplicates, the shape of the dataset is (185564,3). Afterwards we filter only tweets in english.After the pre-processing, this is how our data looks like:



```
spark_dataframe = spark_dataframe.drop("userid")
spark_dataframe.show()
```

```
+--------------------+--------------------+
|                text|          tokenClean|
+--------------------+--------------------+
|a raytheontech su...|[raytheontech, su...|
|rusia  algo más d...|[rusia, algo, más...|
|explosion in cher...|[explosion, chern...|
|panoramatvp2  ros...|[panoramatvp2, ro...|
|ukraines nuclear ...|[ukraines, nuclea...|
|rússia mantém ole...|[rússia, mantém, ...|
|gå i deg selv put...|[gå, deg, selv, p...|
|disclosetv bjcoll...|[disclosetv, bjco...|
|gazprom de rusia ...|[gazprom, de, rus...|
|10 thousand milli...|[10, thousand, mi...|
|schroeder mention...|[schroeder, menti...|
|🌽 ucrania l ha e...|[🌽, ucrania, l, ...|
|guillau15264609 p...|[guillau15264609,...|
|as reagan said we...|[reagan, said, sh...|
|‼UAexclusive w...|[‼UAexclusive,...|
|trumpstolethedocs...|[trumpstolethedoc...|
|all clear 0409202...|[clear, 04092022,...|
|there are several...|[several, ways, e...|
|standwithus shaar...|[standwithus, sha...|
|rusia consider th...|[rusia, consider,...|
+--------------------+--------------------+
only showing top 20 rows
```

As seen, the text is transformed into a tokenClean column which is a basket of unique words coming from the text column.

Shortcomings Of Apriori Algorithm

1. Using Apriori requires a generation of candidate itemsets. These itemsets may be large in number if the itemset in the database is huge.
2. 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.**

Frequent Pattern 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 a “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:

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).

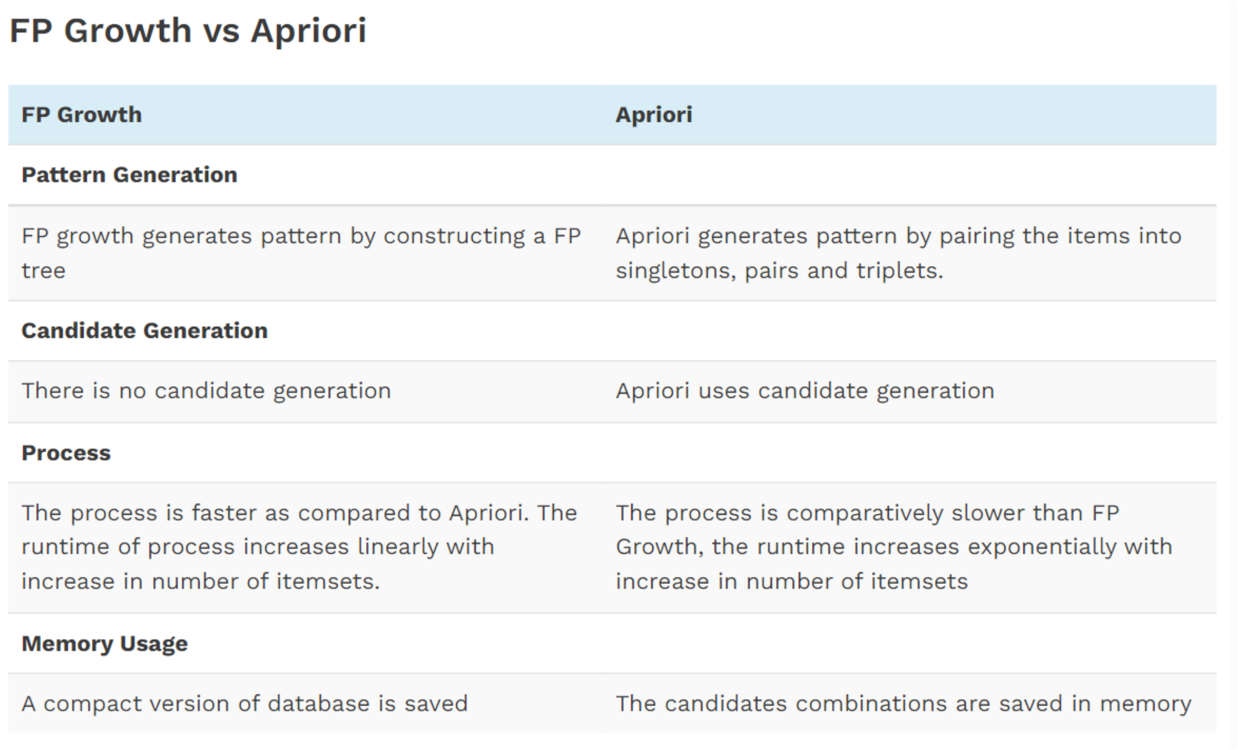
7) Construct a Conditional FP Tree, which is formed by a count of itemsets in the path. The itemsets meeting the threshold support are considered in the Conditional FP Tree.
8) Frequent Patterns are generated from the Conditional FP Tree.

Advantages Of FP Growth Algorithm

1. This algorithm needs to scan the database only twice when compared to Apriori which scans the transactions for each iteration.
2. The pairing of items is not done in this algorithm and this makes it faster.
3. The database is stored in a compact version in memory.
4. It is efficient and scalable for mining both long and short frequent patterns.

Disadvantages Of FP-Growth Algorithm

1. FP Tree is more difficult to build than Apriori.
2. It may be expensive.
3. When the database is large, the algorithm may not fit in the shared memory.

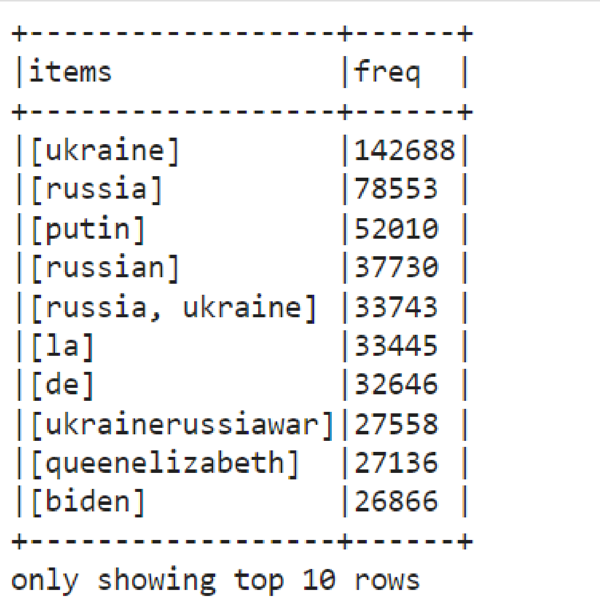


FP Growth vs Apriori

FP Growth	**Apriori**
Pattern Generation	
FP growth generates pattern by constructing a FP tree	Apriori generates pattern by pairing the items into singletons, pairs and triplets.
Candidate Generation	
There is no candidate generation	Apriori uses candidate generation
Process	
The process is faster as compared to Apriori. The runtime of process increases linearly with increase in number of itemsets.	The process is comparatively slower than FP Growth, the runtime increases exponentially with increase in number of itemsets
Memory Usage	
A compact version of database is saved	The candidates combinations are saved in memory

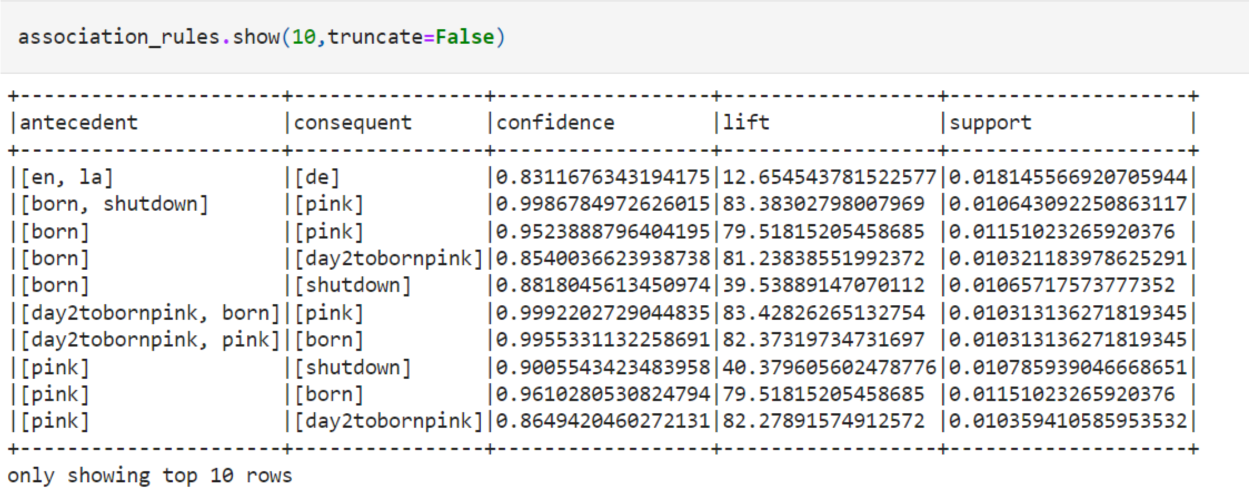
FP-Growth Implementation

For FP-growth, we tried the Pyspark implementation. We tried a range of values for the minimum threshold and decided to go on with 0.01 as threshold. Below this number we had a few frequent items and above that, we had a lot of frequent items. With threshold 0.01, we had 213 frequent items.



```
+------------------+------+
|items             |freq  |
+------------------+------+
|[ukraine]         |142688|
|[russia]          |78553 |
|[putin]           |52010 |
|[russian]         |37730 |
|[russia, ukraine] |33743 |
|[la]              |33445 |
|[de]              |32646 |
|[ukrainerussiawar]|27558 |
|[queenelizabeth]  |27136 |
|[biden]           |26866 |
+------------------+------+
only showing top 10 rows
```

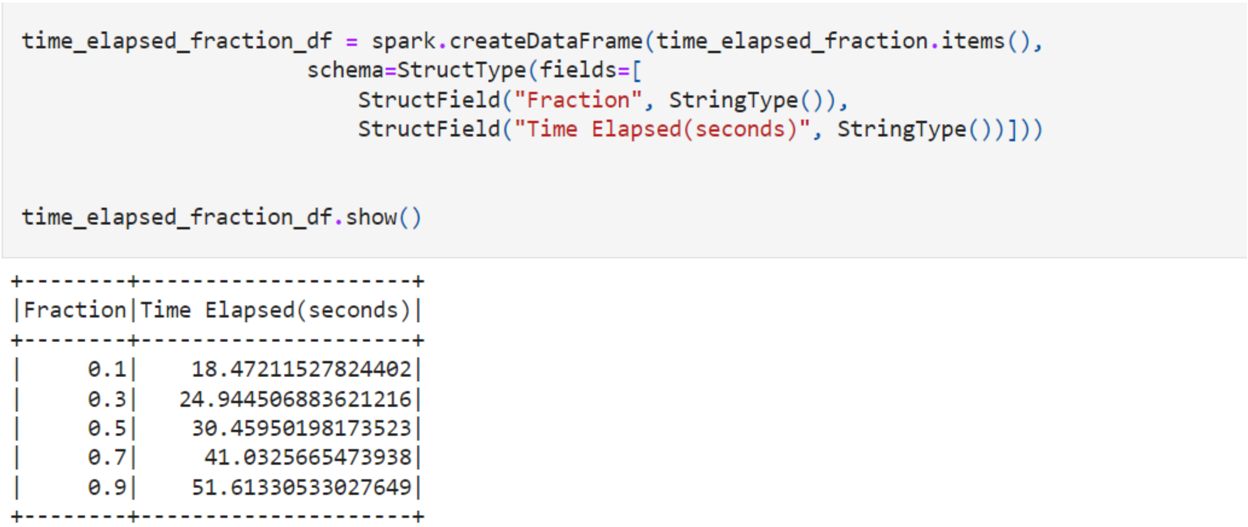
The most frequent items are Ukraine, Russia and Putin. This is expected and gives a clear mirror of the tweets about the conflict. We have 18 association rules provided by the FP-growth algorithm for the threshold used.



```
association_rules.show(10,truncate=False)
```

```
+----------------------+----------------+------------------+------------------+--------------------+
|antecedent            |consequent      |confidence        |lift              |support             |
+----------------------+----------------+------------------+------------------+--------------------+
|[en, la]              |[de]            |0.8311676343194175|12.654543781522577|0.018145566920705944|
|[born, shutdown]      |[pink]          |0.9986784972626015|83.38302798007969 |0.010643092250863117|
|[born]                |[pink]          |0.9523888796404195|79.51815205458685 |0.01151023265920376 |
|[born]                |[day2tobornpink]|0.8540036623938738|81.23838551992372 |0.010321183978625291|
|[born]                |[shutdown]      |0.8818045613450974|39.53889147070112 |0.01065717573777352 |
|[day2tobornpink, born]|[pink]          |0.9992202729044835|83.42826265132754 |0.010313136271819345|
|[day2tobornpink, pink]|[born]          |0.9955331132258691|82.37319734731697 |0.010313136271819345|
|[pink]                |[shutdown]      |0.9005543423483958|40.379605602478776|0.010785939046668651|
|[pink]                |[born]          |0.9610280530824794|79.51815205458685 |0.01151023265920376 |
|[pink]                |[day2tobornpink]|0.8649420460272131|82.27891574912572 |0.010359410585953532|
+----------------------+----------------+------------------+------------------+--------------------+
only showing top 10 rows
```

To show how our solution scales up with the dataset, we measured the time.

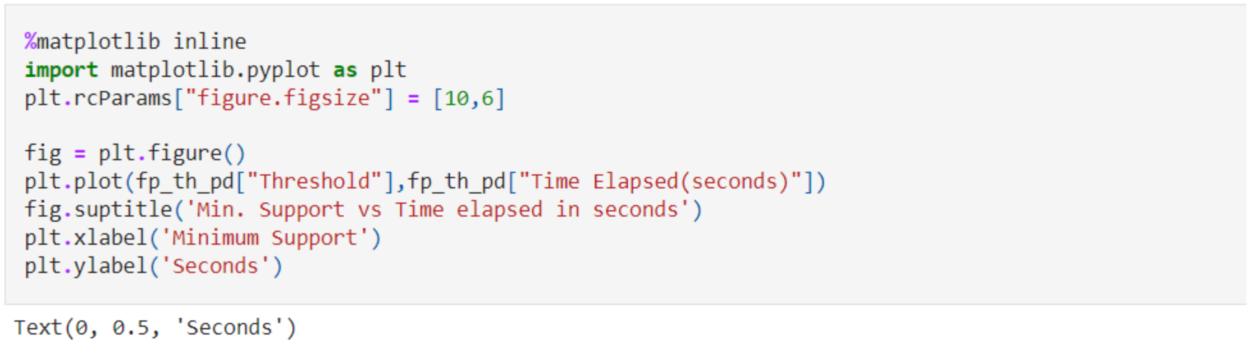


```
time_elapsed_fraction_df = spark.createDataFrame(time_elapsed_fraction.items(),
                      schema=StructType(fields=[
                          StructField("Fraction", StringType()),
                          StructField("Time Elapsed(seconds)", StringType())]))


time_elapsed_fraction_df.show()
```

```
+--------+---------------------+
|Fraction|Time Elapsed(seconds)|
+--------+---------------------+
|     0.1|    18.47211527824402|
|     0.3|   24.944506883621216|
|     0.5|    30.45950198173523|
|     0.7|     41.0325665473938|
|     0.9|    51.61330533027649|
+--------+---------------------+
```

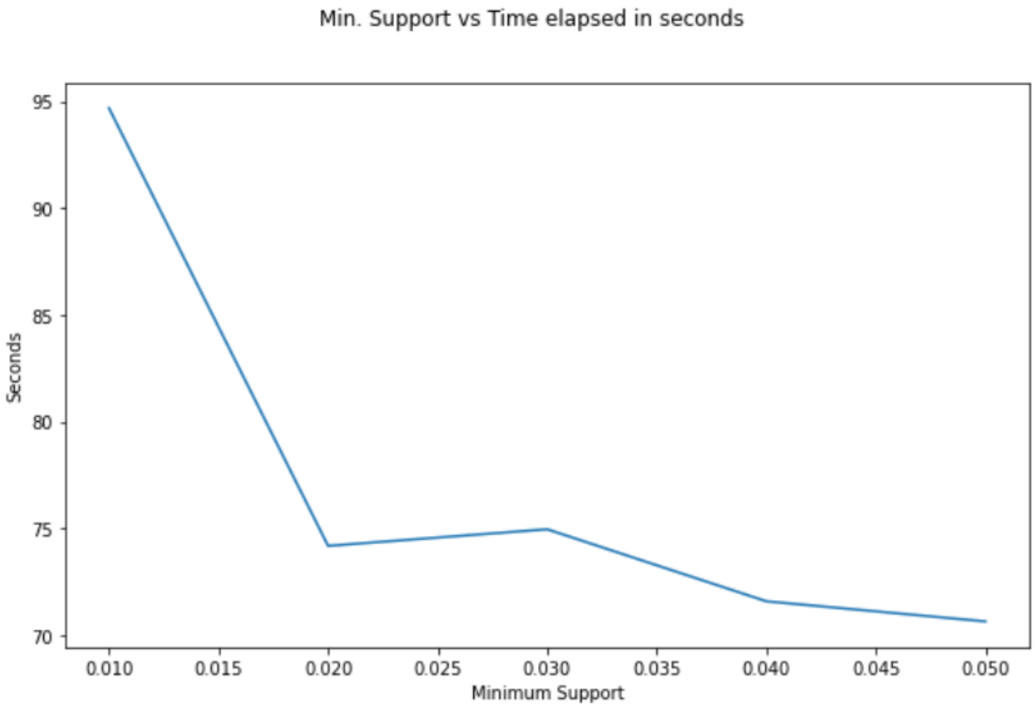
As seen, our solution scales up with the threshold use without consuming all of the RAM. Recall that lower thresholds require more time, given the randomness of the process, even if there is sometimes a weird decrease as threshold increases, the trend is clearly downward.



```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = [10,6]

fig = plt.figure()
plt.plot(fp_th_pd["Threshold"],fp_th_pd["Time Elapsed(seconds)"])
fig.suptitle('Min. Support vs Time elapsed in seconds')
plt.xlabel('Minimum Support')
plt.ylabel('Seconds')
```

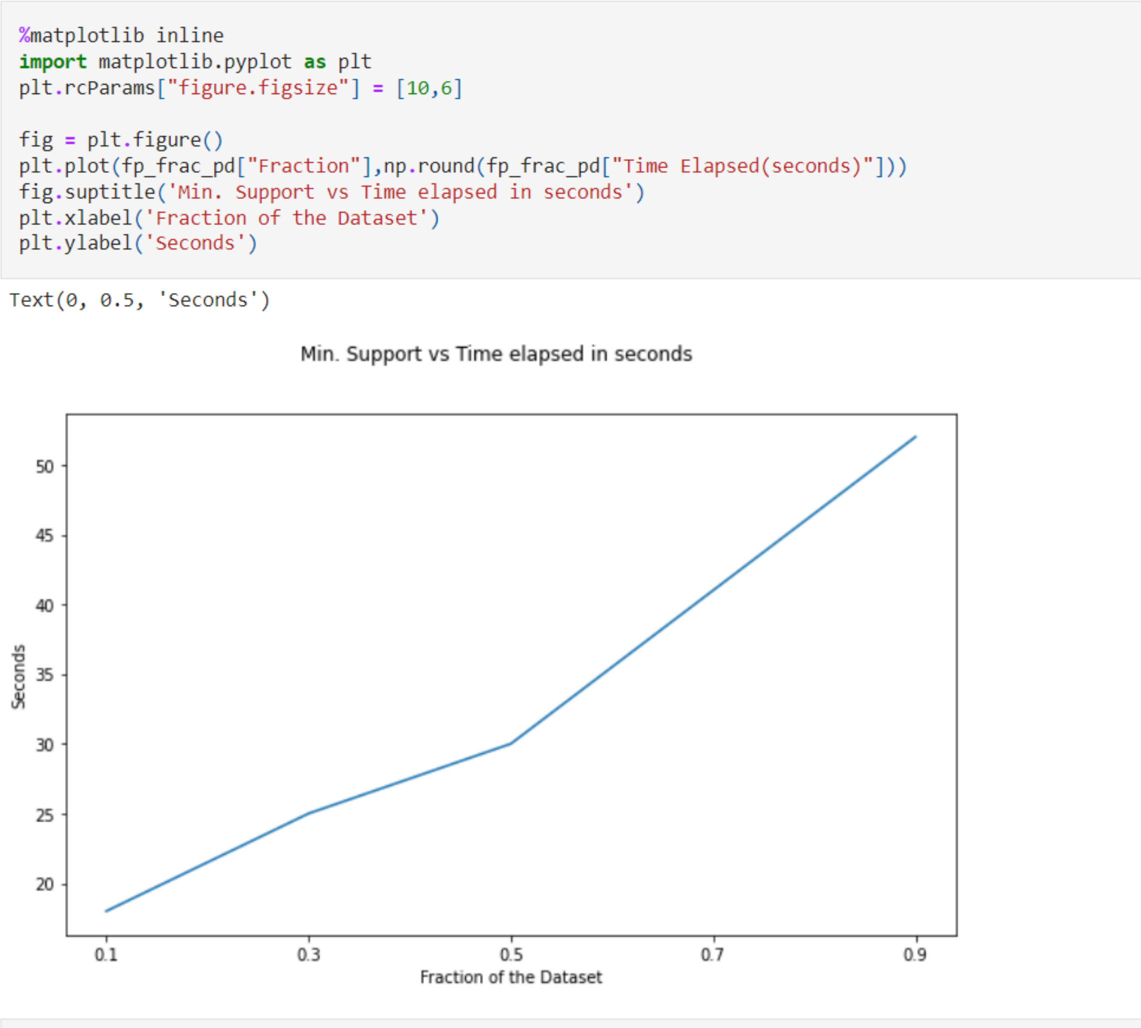
```
Text(0, 0.5, 'Seconds')
```



As the minimum threshold increases, the time required for a pass of FP-growth decreases.

We did the same measurement with respect to fraction of dataset as well to have a better picture of scalability

As we use more and more data, the time required increases.



```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = [10,6]

fig = plt.figure()
plt.plot(fp_frac_pd["Fraction"],np.round(fp_frac_pd["Time Elapsed(seconds)"]))
fig.suptitle('Min. Support vs Time elapsed in seconds')
plt.xlabel('Fraction of the Dataset')
plt.ylabel('Seconds')
```

```
Text(0, 0.5, 'Seconds')
```

So we can conclude that given any threshold and size of the dataset, the algorithm is able to process the data scaling up nicely with the problem.

Conclusion

In this project, we tried to define frequent itemsets contained in the tweets using the FP-growth algorithm of Pyspark implementation. We tested the scalability of the FP-growth algorithm by measuring the time elapsed for each threshold and fraction of the dataset. We saw that the result provided was consistent with respect to the choice of threshold and fraction used.

Acknowledgment

- I used the“https://www.softwaretestinghelp.com/fp-growth-algorithm-data-mining/” link to give a theoretical background on the FP-Growth algorithm as well as comparing FP-growth algorithm with Apriori algorithm.