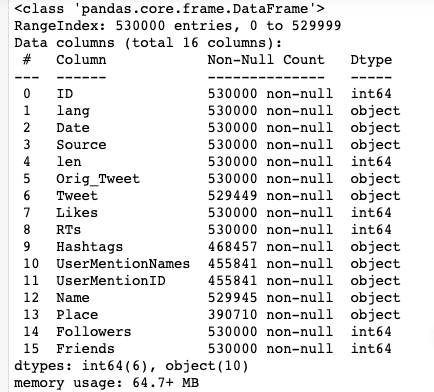
**Data acquisition**

I used three datasets in this chapter:

The **FIRST** dataset I used in this chapter is the Assoc\_Analysis\_Vidhya.dat.csv data provided by Dr. Santago. This dataset has a total of 315 transactions each of which consists of merchandise like “Bread”, “Wine”, “Eggs”, etc. Each row of this dataset represents a specific transaction by a customer.

The **SECOND** dataset is the GroceryStoreStacked.csv data. The transaction in this dataset is stacked compared to the first dataset--that is, each row only contains the transaction of one item while a customer may purchase multiple items. One transaction is therefore recorded using multiple rows.

The **LAST** dataset is called the FIFA.csv dataset that I found on Kaggle. This dataset contains a random collection of 530k tweets starting from the Round of 16 till the World Cup Final that took place on 15 July, 2018 & was won by France. Multiple features are available in this dataset as shown in the figure below:



Despite the fact that we have multiple features at hand, the specific feature of my interest is the “Tweet” feature which is essentially the content of each tweet. This feature is categorical and I will further process this feature as will be demonstrated later.

**Program development**

In this chapter, I implemented two algorithms related to the Association concept.

The **FIRST** thing I did in this chapter was to implement the Apriori Algorithm (algorithm 5.1 in textbook) for frequent itemset generation.

Code implementation can be found in Association\_Analysis.ipynb.

The Apriori Algorithm for frequent itemset generation is achieved through multiple iterations. Given a set of frequent itemset of length k-1, we generate a set of frequent item candidates, calculate the support for each candidate, and select the candidate with support above minsup as the frequent k-itemset. Some characteristics of my implementation are:

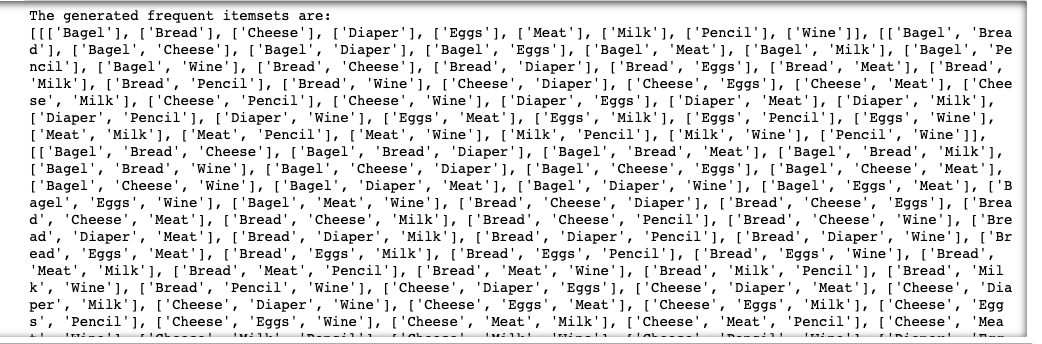
1. Two functions--”candidate\_gen” and “candidate\_prune”-- are defined in addition to the “Apriori\_frequent” function. “candidate\_gen” helps generate the candidates for the frequent k-itemset using F(k-1) \* F(k-1) method. “candidate\_prune” prunes any candidate which contains a k-1 subset that is not frequent.
2. The Frequent itemset F is constructed using a nested list. The first level contains k separate lists which are the k different frequent itemsets with length m, 1 <= m <= k. Within each of the k separate lists are the frequent itemsets of length m. Candidate is structured in the same way.
3. When calculating the support of each candidate, the data structure for storing the support for each candidate is a dictionary. The key of this dictionary is the name of the candidate itemset; and the value corresponding to a key is the support count of this key. Special transformations are required to set the key as the name of a candidate since a candidate is a list object. Therefore, I combine each element in a candidate as a single string object which is then used as the name of this candidate. In this way, whenever I want to refer to the support count of a candidate to see if it should be included in the frequent k-itemset, I can simply call support\_count[key].

The **SECOND** algorithm I implemented is the Apriori Rule Generation algorithm (algorithm 5.3). Code implementation can also be found in file Association\_Analysis.ipynb.

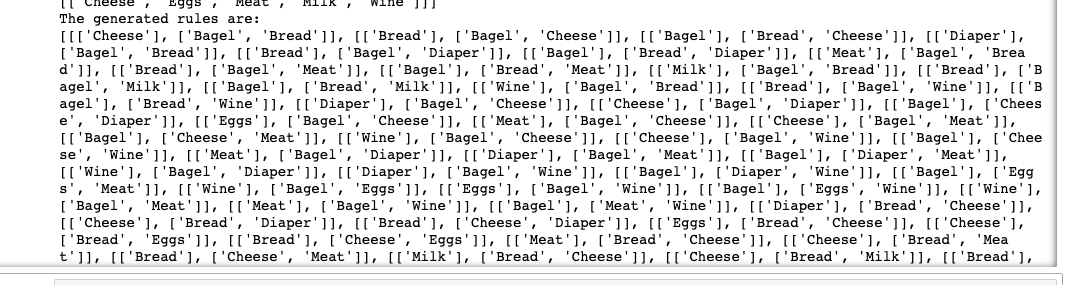
I implement this because I wanted to explore more into the Apriori algorithm and how it may be used to generate actual rules. The Apriori Algorithm for rule generation is achieved through recursion. The key is to find the k-item consequent of the rules. For each frequent k-itemset, I find the corresponding 1-item consequent in the first place. Rules with 1-item consequent can be generated through finding the confidence of each possible combination using these consequent. Now we have the base case, the algorithm then recursively finds rule consequences of length k+1 (assuming we have rule consequences of length k) and use that to generate rules for this itemset. Here below are some implementation details:

1. Confidence can be calculated as long as we have the support count stored in memory. Therefore, the confidence of each candidate need not be stored in any special data structure but float.
2. The candidate\_gen and candidate\_prune algorithms that are used to produce candidate k-item consequences are exactly the same as the ones I use for frequent itemset generation.
3. The k-item consequences are stored using nested lists just like the frequent itemset.
4. The generated rules are stored using nested lists. Every nested list in the list “rules” is a single rule that is generated. Within the nested list, the first element refers to the rule precedent in a rule, and the second means the rule consequent.

The two algorithms are then tested using the Assoc\_Analysis\_Vidhya.dat.csv data. Using a minsup level of 0.1 and minconf of 0.2, part of the generated frequent itemsets is shown below:



Also, part of the generated rules is also shown below:

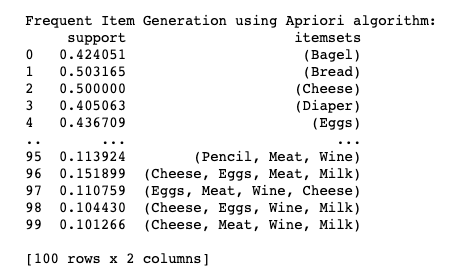
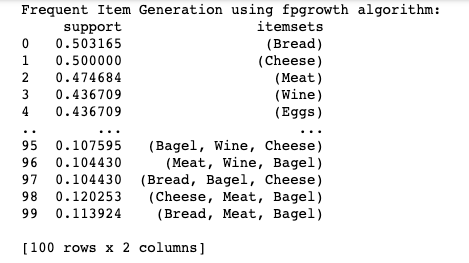


**Data analysis and package use**

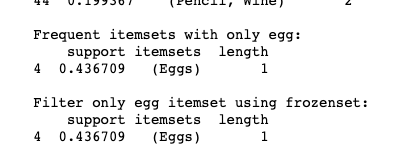
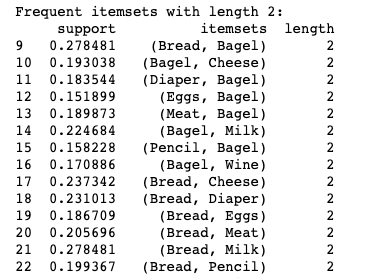
In this chapter, I use the package mlxtend to develop three programs related to association analysis. All detailed implementations of these programs can be found in the file Association\_Analysis.ipynb.

In the **FIRST** program, I use the Assoc\_Analysis\_Vidhya.dat.csv data once again to familiarize myself with the mlxtend package. Here below are a list of things I experimented with:

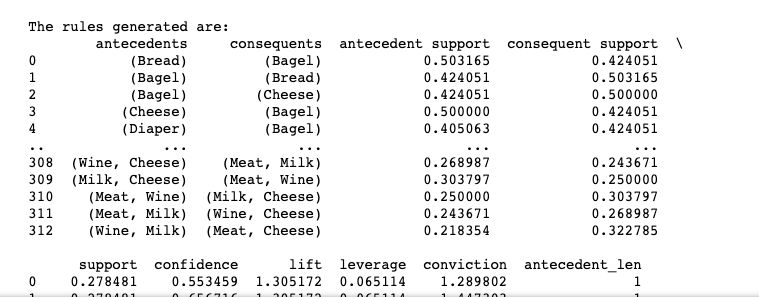
1. I start with transforming and encoding the raw data using the “TransactionEncoder” tool in the package “mlxtend.preprocessing”. The raw data is in the form of a nested list where each first level inner list represents a transaction. In order to process the data, “TransactionEncoder” transforms the raw data into a True/False data matrix where each entry indicates whether or not this transaction (row) contains the specific item that is indicated by this column.
2. Since numerous functionalities exist in the package, I was only able to experiment with a few of them. One thing I experimented with was to generate frequent itemsets using different algorithms. Using a minsup of 0.2, I was able to create two different frequent itemsets using the fpgrowth and the Apriori algorithms. The output are shown below:



1. I was also able to use the frequent itemset generated by this Apriori method to verify that my implementation does give the accurate results.
2. Several manipulations are available in mlxtend so that we may extract the information we need from the generated frequent itemset. For example, I tried to extract all frequent itemsets with length 2 and also the frequent itemset that has only “Eggs” in it.



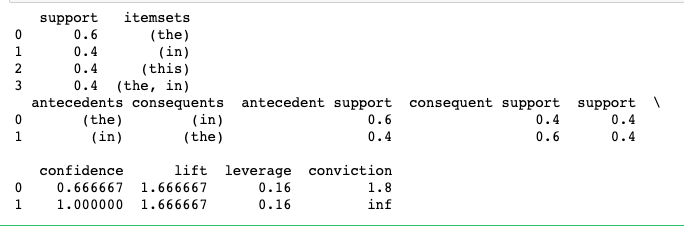
1. Mlxtend also allows me to generate a set of rules. For each rule, information like antecedent support and confidence are listed so I can better understand the relation between both sides of a rule.



My **SECOND** program makes use of the GroceryStoreStacked.csv data. In this dataset, each row only contains the transaction of one item while a customer may purchase multiple items. Therefore, some transformations are necessary in order to process the data. What I did is to combine all rows with the same customer ID, which will result in a data matrix just like the previous one, and then I can conduct analysis resembling my first program.

My **THIRD** program uses mlxtend to analyze the relationship between words in a particular situation. The data I used is the FIFA.csv dataset after data preprocessing--that is, with only the tweet and the ID variable in the dataframe. The intention of conducting association analysis on this dataset is that I want to explore if association analysis can be applied to the field of language processing. Intuitively, certain patterns of words should exist behind these tweets since they are all related to the FIFA 2018 competition.

Some other preprocessing processes are necessary still because it would be impossible to conduct associate analysis with the tweet being a long string of words. Therefore, I separated the tweet content in each tweet into a list of words. Since the processing time for conducting associate analysis on a dataset that has so many “items” (words actually) is enormous, I have to sample from my dataset. Here below are some steps I took and why eventually the method does not work:

1. I fit the model just like I did for the previous two programs. Even though the items are words in this dataset.
2. I sample from the complete list of more than 200k columns, since otherwise the time to process would be too much. Still, even if I just select around 10 data points, the processing time would still be too much.
3. The most severe problem with this method is the association is not giving me what I want but some trivials results. Here below is one of my trained results: 

Even though frequent itemsets are generated, I cannot extract any significant information from the simple fact that the word “the” usually accompanies the word “in” in the sentence. These words are collected as frequent itemsets simply because they have a higher frequency of occurrence in English than other words. However, it is those words that we are interested in and that can give us valuable information.

1. Through this program--even though I was unable to produce any significant results, I do realize that it would be very hard to conduct association studies on textual analysis because of the noises (frequent words that do not possess too much meaning). In addition, since the item (word) distribution is pretty sparse, building such a model on textual analysis is extremely time consuming.

**Student learning summary and self-assessment**

Major takeaway:

1. Implementation of both Apriori frequent itemset generation and rule generation algorithms allows me to have deep understanding on both of these concepts. I also gain solid understanding on the types of data structures that should be used for such implementations.
2. Different types of evaluation measures, besides support and confidence. Comprehensive explanation on the strengths and weaknesses of these evaluation measures (including support and confidence) enables me to choose the most appropriate evaluation measure when encountering different types of data/problems.
3. How the FPgrowth algorithm works!

Another aspect of this chapter that I think is really interesting is what are the areas that association analysis can be applied to. Throughout the study of this chapter, I was working with market transaction type of data all the time. It makes me wonder if association analysis should only be applied to these subjects. Even though I wasn’t able to find another type of data from other areas that association models can effectively untangle, I do believe there is potential that association analysis can be applied to textual type of data. In my case, if I were able to eliminate irrelevant “frequent words” like “the” and “in”, the rest of the analysis should be able to bring some merits. So this is certainly some other thing I would like to explore more in the future.

Besides that, something I want to explore more in this chapter are:

1. Compare the performance of FP-growth and Apriori algorithm when applied to actual data.
2. Try out different evaluation measures and compare their performance.

Self-evaluation:

I believe that my understanding of notions related to association is solid. One potential weakness is that I was unable to apply association methods to my own dataset successfully. Comprehensively, I would like to give myself an A/A- on this chapter.