**Shehzan Maredia**

**I have uploaded my answers in this document as well as emailed the .csv files and my jupyter notebook. Please let me know if you need anything else.**

**Shipt – Interview Challenge – Data Science Intern**

**Introduction**

For this position, you will be performing data science research and development to support Shipt’s personalization efforts in the web and mobile customer apps. To that end, this challenge attempts to provide tasks that are more or less representative of this kind of work.

There are three data sets provided with these questions: *orders.csv*, *order\_lines.csv*, and *products.csv*. The column names should be self explanatory. You will need to join these datasets to perform your analysis.

For the following, please use either R or Python, and you are free to use any packages you want to help answer these questions.

**Required Questions**

**Data Understanding**

Questions 1, 2, and 3 deal with *orders.csv* and *order\_lines.csv.*

1. Join these two datasets by *order\_id* returning all rows from both regardless of whether there is a match between the two data sets.
   1. Answer in .csv file
   2. Code in jupyter notebook
2. Join these two datasets by *order\_id* returning only the rows from *orders.csv* that have no corresponding record in *order\_lines.csv*.
   1. Answer in .csv file
   2. Code in jupyter notebook
3. Join your result from #1 with *products.csv* so that you also have product names:
   1. What are the top 10 most frequently occurring product names?

* Below I include the 10 most frequent names and their counts. I just took the value counts of “name” in the dataframe which was created after joining the products.csv file.
  + - * Banana                               5929
      * Strawberries                         3040
      * Hass Avocado                         2993
      * 2% Reduced Fat Milk                  2068
      * Red Seedless Grapes                  1970
      * Large Grade A Eggs                   1798
      * Whole Milk                           1629
      * Publix 93% Lean Ground Beef          1529
      * Boneless Skinless Chicken Breasts    1494
      * Honey Wheat Bread                    1321
  1. What are the summary statistics (mean, median, and IQR) for the number of distinct product ids per order? How would you visualize these?
* Below, I include the summary statistics for the number of distinct product ids per order. I solved this question by using the order\_lines dataset, and I figured out how many distinct product ids corresponded to each order id using “groupby”. That data was then analyzed and the results are below:
  + - * mean        18.285471
      * 25%         11.000000
      * 50% (also the median)         16.000000
      * 75%         23.000000
      * IQR 12.0000
      * count    16464.000000
      * std         10.562312
      * min          1.000000
      * max        127.000000
* Although there are several ways to visualize this data, using a boxplot would make most sense to visualize the mean, median, and IQR.

**Market Basket Analysis**

In the following questions, you will mine frequent item sets found in *orders.csv*, *order\_lines.csv*, and *products.csv*, using the apriori algorithm. Apriori takes a collection of "transactions" and mines them for frequent item sets and association rules. Before proceeding, you should transform your data such that each *order\_id* corresponds to a transaction and each product name in a given order to an item.

Once you have your data in the correct format, you can run apriori using any library you prefer. Play around with different parameter settings until you get around 15,000 item sets (or thereabouts). Keep the length of item sets to no more than 3.

Here are a couple of examples that will get you in the ball park:

In R:

library(arules)

fsets <- apriori(data=transactions,

parameter=list(supp=0.001,

conf=0.1,

maxlen=3,

target='frequent itemsets'))

In Python:

from apyori import apriori

rules = apriori(transactions,

min\_support=0.001,

min\_confidence=0.1,

min\_lift=1.0,

max\_length=3)

1. What items are in the 10 item sets with the highest support? How do these compare with the top items found in (3)?

I was unable to get my computer to reach 15,000+ item sets due to computation time, but I got close. Here are the results:

A screenshot of a cell phone

Description automatically generated

The 10 items sets with highest support mainly involve the top 10 items we found earlier. This makes total sense as support refers to the popularity of an item and can be calculated by number of transactions containing an items divided by total transactions. With that definition in mind, the items that occur most frequently should of course have highest support.

1. How many item sets of each length (1,2, or 3) are there in the total result set? Are these counts surprising in any way? Why or why not?

Item sets of length 1: 1189

Item sets of length 2: 6276

Item sets of length 3: 5383

No this makes sense. Most people if they buy one item might buy another item, so these counts are not surprising in any way.

1. In general, does the order of association within an item set matter? In other words, assuming that products X and Y occur as an item set, does P(X|Y) = P(Y|X)? For example, is the probability of canned tomatoes being in the basket given that we have frozen pizza the same as the probability of pizza being in the basket given that we have canned tomatoes? Why or why not?

No, the order of the antecedent and consequent matter. In conditional probability, it is not necessarily true that does P(X|Y) = P(Y|X). That only occurs if P(X)=P(Y), which is not necessarily true for our case.

**Additional Questions – Pick One**

Frequent Item Sets can be thought of as an example of frequently bought together recommendations. Assume that we want to use this approach as the basis for such a recommender, then **pick only one** of the following questions and answer it in 400 words or less**.**

Question B:

For our web and mobile apps, we prefer to keep response time for user interactions low. What could we do with the model to ensure that it delivers results with as little latency as possible? Also, keeping in mind that the dataset in this exercise comes from a single store and single city (or metro), how could we scale this modeling approach to many different stores and metros?

To deliver results with as little latency as possible, we can take several measures to decrease computation time and provide a fast user experience. We will need to provide users with accurate, intelligent recommendations, and since we will be working with a lot of data, we want to make sure we can make this process as fast as possible. One strategy we can implement is implementing a clustering algorithm to group users with similar buying history together. This can enable us to pre-solve the problem for each cluster ahead of time, so that when we provide recommendations in real time, we can use these insights in addition to what they bought in that order to provide accurate metrics. We can also provide suggestions based on their buying history and incorporate that with suggestions in real time.

To add onto that, all computations should be sent to a server, since computing on the user’s phone would be very slow and ineffective. This was a problem I encountered when working on my contact tracing app, NOVID, and I found computing on a scalable server can reduce latency very significantly.

Using a static dataset for the items available to be recommended can also help with latency issues, and this dataset could be updated every day or 2-3 days. In terms of databases, a graph database could be very effective for this type of relational data. Instead of resolving certain relations in real time per query, the graph DB could visualize the relations as edges in a graph and significantly speed up queries based on relations between nodes.

Further, it seems that using an alternating least squares implementation has proven to be successful in recommendation engines and accounting for latency as it alternates between improving the item and the user factors respectively. This could be a very effective approach to apply in the model, and it can help make this model more scalable.

Ultimately, there are various techniques we can implement to reduce latency and improve user experiences.