|  |
| --- |
|  |
| ICT1002 Programming Fundamentals |
| Team Assignment 1 |

**Team 6**

**Done by:**

**Daniel Khoo**

**Dylan Lian**

**Jerahmeel Chua**

**Lee Jun Long**

**Lucas Yap**

**20th October 2016**

Table of Contents

**Team member contributions3**

**Utilized Technologies4 - 7**

CSV4

Miscellaneous5

Regular Expression Operations5 - 6

Itertools6

Tkinter, tkFileDialong, sys7

**Methodology & Algorithms8**

Main Methodogy8

Apriori’s Algorithm and Promotion Adviser 8

**Highlights9 - 10**

Adaptive GUI9

Multi-step function10

**Limitations and Recommendations11**

GUI discrepancies11

Pre-defined receipt formats11

Limited scenarios for recommending promotions11

Lack of real-time analysis11

# **Team member contributions**

Jerahmeel Chua - Function 1, 2, and 4

Taking the initiative, Jerahmeel got the ball rolling; establishing not only a function for the dataset(in .csv) to be read (Function 1) but exporting the individual receipts into ‘.txt’ files accordingly by reading and writing them as needed (Function 2). Following that and using self-defined functions to locate and sum the sale amounts from each receipt, function 4 was completed.

Jerahmeel also acted as the leader and main compiler in our group, ensuring that all functions were completed on time, adding and testing them to our existing code accordingly.

Dylan and Jun Loong – Function 5 and 6

Working as a team to complete the above functions, Dylan and Jun Loong first scanned and extracted all sold items from our self-generated dictionary containing the dataset into a list. This is done by using pre-defined search parameters analysed from the sample data given to limit the data extracted to only sold items. They then eliminated all repeated elements in the list by converting the list into a set and back to a list again. This results in a list with all unique items being sold by the respective merchants (Function 5).

They then created Function 6 by scanning and writing a new .csv file with the following information: Receipt ID, Company Name, Company Address, Date, Time, Items sold, Quantity of the items sold, their individual costs, and the Total transaction amount. The respective information listed above was extracted using a combination of regular expressions and search patterns analysed from the sample data. All the data is consolidated into one list, which is then written row by row to the new .csv file with respect to each receipt id.

Daniel and Lucas – Function 3, 7, and 8

For better efficiency, Function 3 was accomplished by printing out a previously compiled dictionary ‘merchantsales’ which is created when a ‘.csv’ file is read by our analyzer. This is an example of how the information obtained during our initial reading of the sample data is used in various parts of our program, removing the need to re-read the data.

Function 7 was then accomplished by scanning through the data compiled in a previously defined dictionary and comparing the frequency of items bought to each other. For example, to compare the number of times ‘beer’ and ‘carrot cake’ were bought with the total number of times ‘beer’ was bought using the Apriori Algorithm. Thus showing us the purchasing associations between unique items sold by each merchant.

Function 8 as an open question led to the creation of two new and unique functions in our transaction analyser, the first of which was the “Promotion Adviser”. This function, based on the correlation computed in function 7 would then calculate based on certain assumptions (refer to figures 12 and 14) recommends a promotion for a given period of time.

# **Utilized Technologies**

CSV

This module from the Python Standard Library allowed us to read from and write tabular data in ‘.csv’ format.

csv.reader (Used in Function 1)

Used in our self-defined ‘readCSV’ function, csv.reader allowed us to read line by line and extract the necessary data from the sample data into data structures such as ‘receiptids’ and ‘merchantDict’ as shown below.

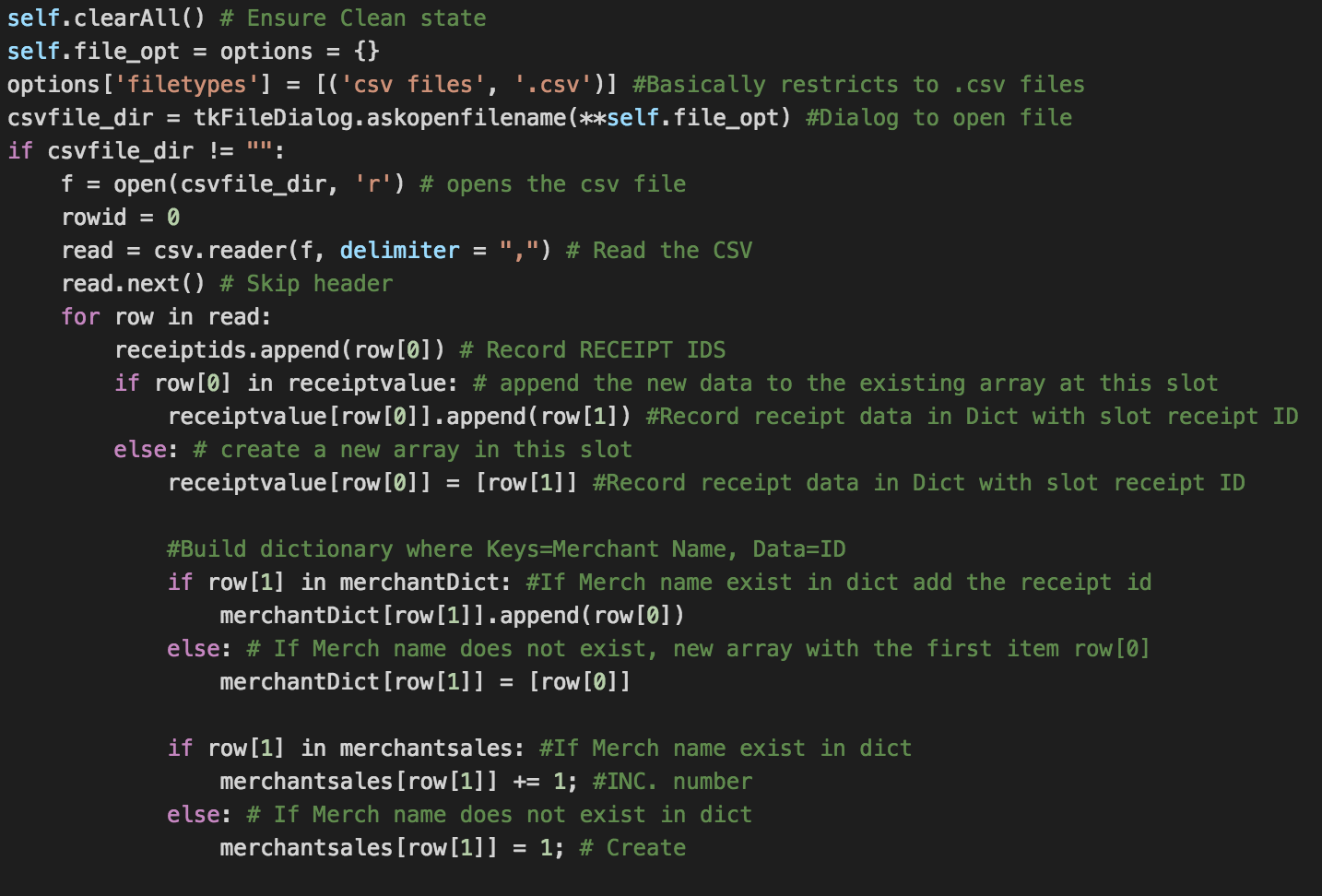




Figure 1 An Extract from our readCSV function and 'merchantDict' as stored in our analyzer

csv.writer (Used in Function 6)

Used in our self-defined ‘exportCSV’ function, csv.writer allowed us to transfer relevant data from our data structures into a new ‘.csv’ file.

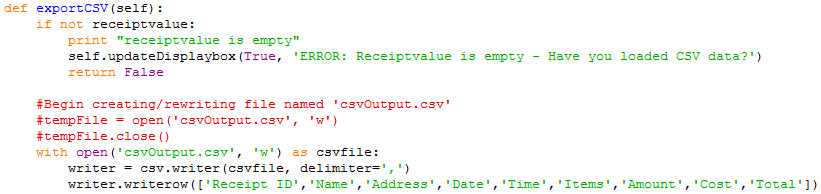




Figure 2 An extract from our 'exportCSV' function, writing the headings for our new '.csv' file and the output of the '.writerow' command in the output file

Miscellaneous

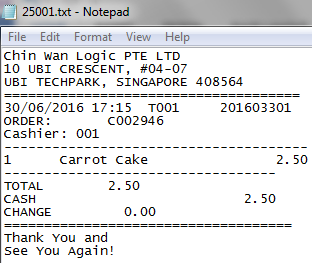
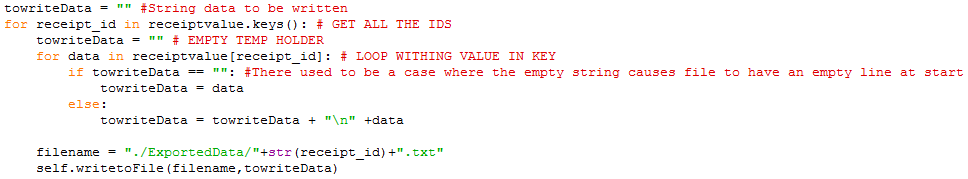
output\_file.write (Used in Function 2)

Figure 3 An extract from our ‘exportDataToFile’ function and an example of an output '.txt' file

Using the output\_file.write API allowed us to write data

one by one from our receiptvalue dictionary to individual ‘.txt’ files

each named after their respective receiptIDs.

Regular expression operations

re.findall (Used in Function 4)

.findall returns all unique matches of a pattern in a string in the form of a list.

Utilizing this allowed us to obtain the total sale amount from each receipt and store it in a dictionary (receiptvalue).



Figure 4 An extract from our 'extractFloat' function

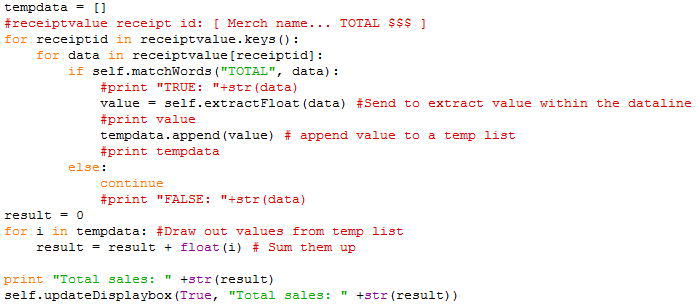
Following that, we sum the values in order to obtain the total sales amount for all receipts.



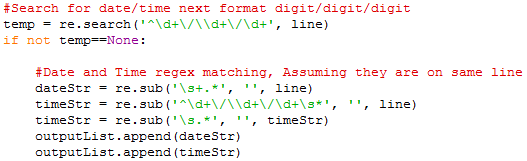
Figure 5 An extract from our 'totalSales' function and the list of receipt totals as stored in our analyzer

re.search and re.sub (Used in Function 6)

Allowing us to scan through strings (and obtain either a ‘True’ or ‘None’) .search was used in function 6 to obtain the data that we needed to write to a new ‘.csv’ file.

Used in conjuction with .sub, (which allows us to substitute patterns within strings with other characters) we were able to validate if date was present in the list and reformat it accordingly.

For example in the figures below, our code scans to see if the appropriate date format is found. After which it extracts it and stores it in a list in order to be written to the output file later on.



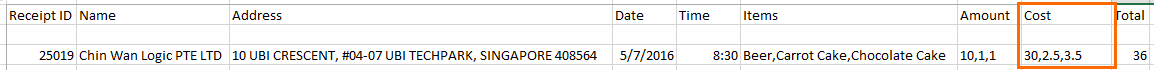


Figure 6 An extract from our 'exportCSV' function and an extract from our output ‘.csv’ file

Itertools

itertools.permutations (Used in Function 7)

This API returns a number of tuples with fixed length, accounting for all possible orderings with no repeated elements.

Used in our self-defined ‘findAsssociations’ function, it allowed us to create and format a dictionary of all possible combinations between the products being sold at our merchants.

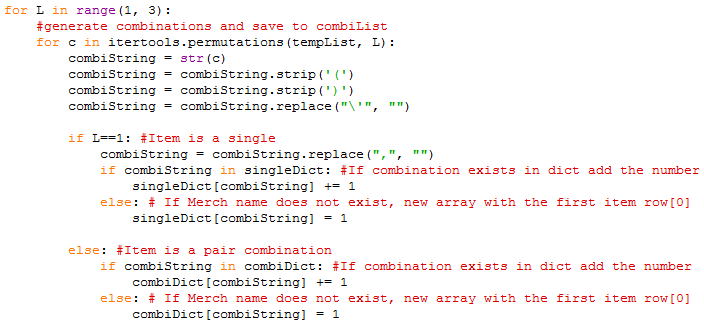




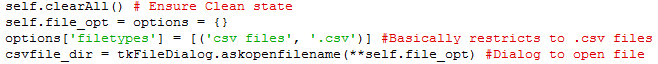
Figure 7 An extract from our 'findAssociations' function and the dictionary as stored in our analyzer

Tkinter, tkFileDialog, sys (GUI)

Tkinter is a standard Python interface that allows us to utilize the Tk GUI toolkit.

tkFileDialog

Allows us to enable the user to open a specific file which in the case of our analyser was restricted to ‘.csv’ files.



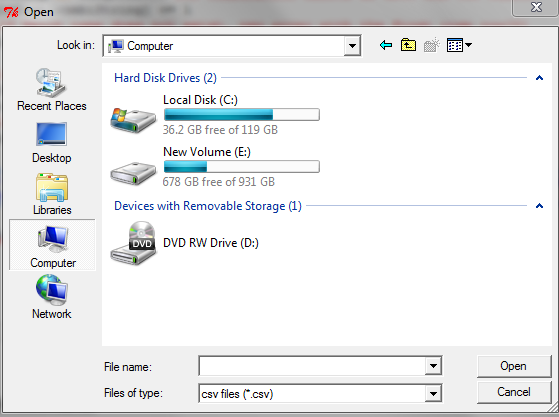


Figure 8 An extract from our ‘readCSV’ function and the way it is presented in the GUI

sys.platform

Also a part of the Python Standard Library, the sys module and more specifically the sys.platform command allows us to detect what OS our anaylzer is being run on.

Using this information we managed to make our analyzer recognize if it was being run on Windows or MAC OS X and run the appropriate UI configuration accordingly.

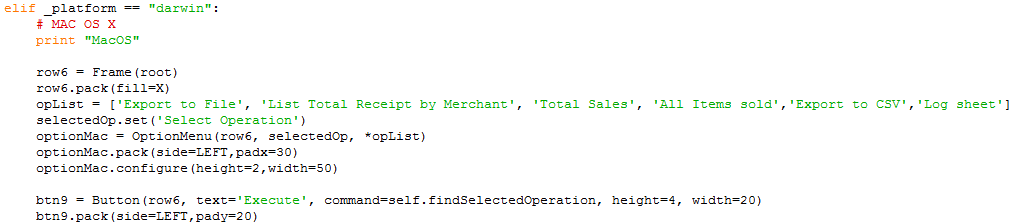
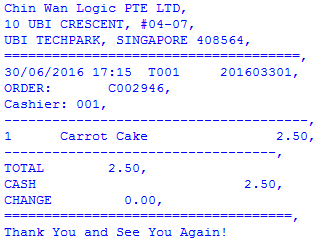


Figure 9 An extract from our ‘createWidgets’ function which initializes our GUI

# **Methodology/Algorithms**

Main Methodology

Our main methodology for extracting and processing data in our analyzer was to sieve through the data structures obtained from reading the sample data and find the information we required.

Figure 10 A value in 'sortedmerchantDict' as stored in our analyser

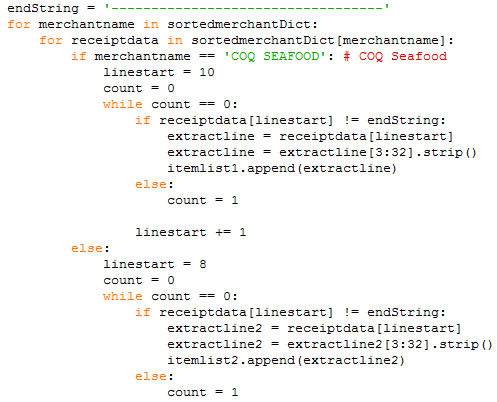
As seen in figure 11, when wanting to display all items sold by the respective merchants, we used nested loops to locate and extract the items, and stored them in a list.

Figure 11 An extract from our 'listAllSoldItems' depicting how we strip parts of the data

In extracted code on the left we can see that the analyser first discerns which merchant the receipt is from. Specific data is then extracted based on pre-defined search parameters analysed from the sample data. For e.g. (data extracted between receiptdata[linestart] to endString). It then strips the data of any unwanted space or characters to obtain only the sold items and store them into the respective ‘itemlists’.

Apriori’s Algorithm and Promotion Adviser

The association algorithm that we decided to use was the Apriori Algorithm proposed by Agrawal, et al.

It basically identifies item set that appear frequently in the database starting from single item sets to multi-items sets depending on the data. In our case as the dataset was small, we restricted it to item sets of 1 and 2 items only.

In addition we used other general association rule learning concepts such as Support and Confidence in order to determine association and make recommendations. (Refer to figure 13)

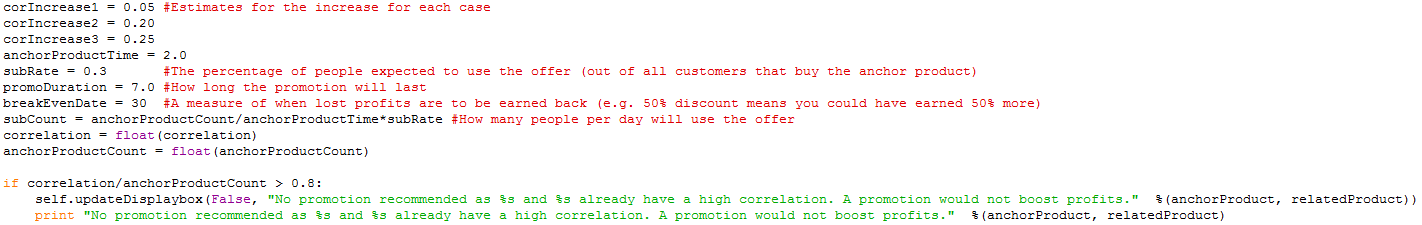
Using this algorithm to calculate correlation, our analyzer is then able to make recommendations based on four pre-determined scenarios.

Figure 12 An extract from our ‘promoAdviser’ function, showing the scenario where the correlation between the products is more than 0.8

# **Highlights**

Adaptive GUI

Needless to say, the most enriching aspect of this project has been the creation and interaction with some of the various GUI commands that are more easily accessible using Python.

By using the modules previously mentioned (Tkinter, sys, etc.) our group was able to make a dynamic GUI that detects and reacts according to the OS our analyzer is being run on.

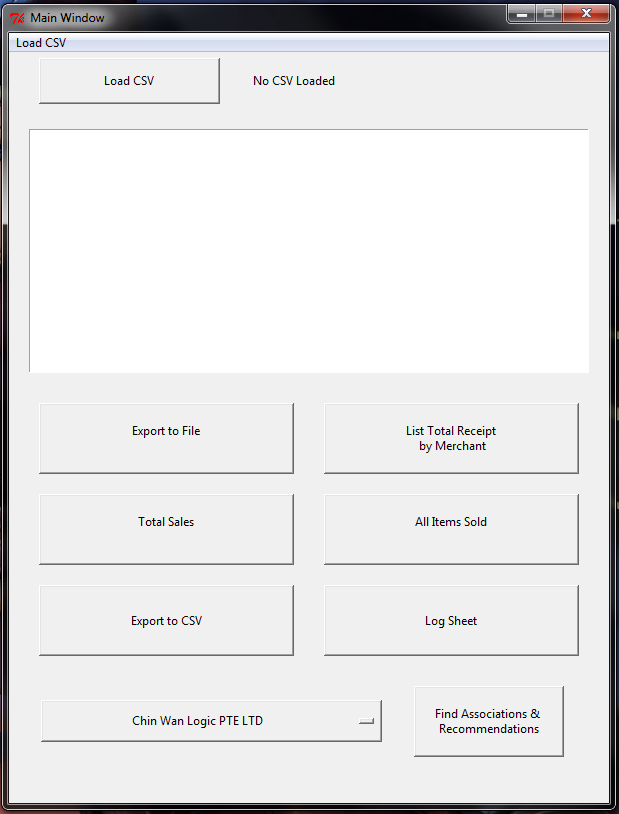
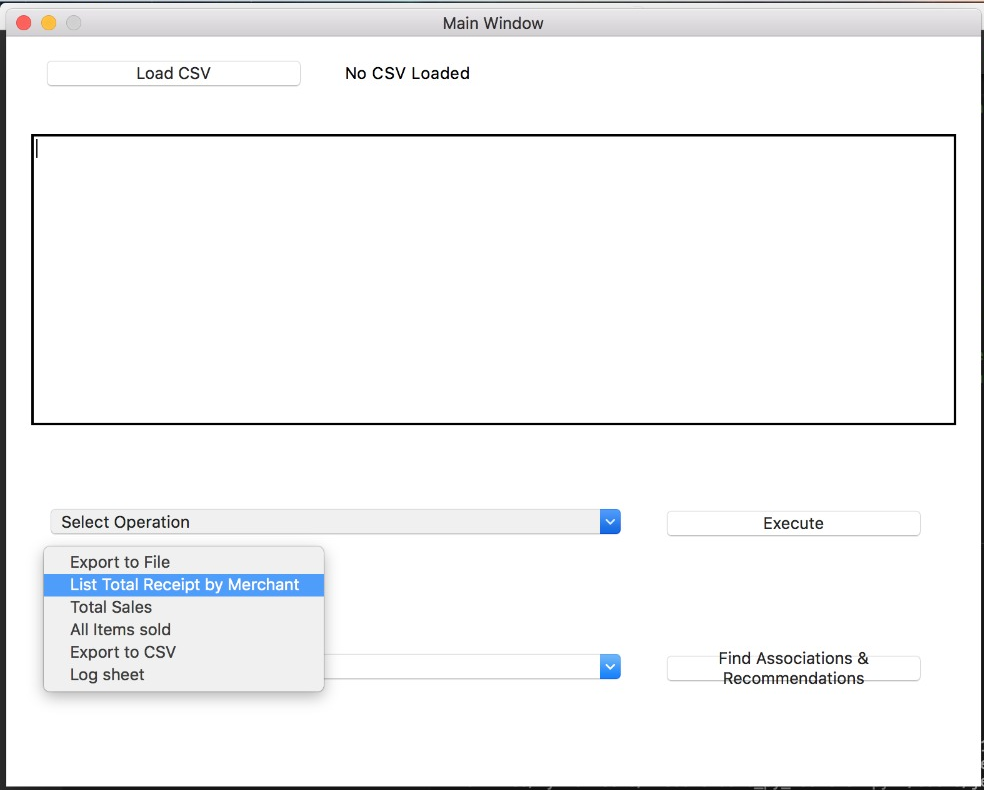
 

Figure 13 Side by side comparison of our GUI when run with Windows and MAC OS X

Multi-step function

Another notable feature of our GUI would be our multi-step function (7 & 8) – “Find Associations & Recommendations”

Scripting the function so that clicking on the button will create another pop-up window, our function then allows the user to select two products from separate dropdown lists.

Our algorithm as discussed above then takes the input of the products and the frequency of their purchases in order to give a recommended promotion.

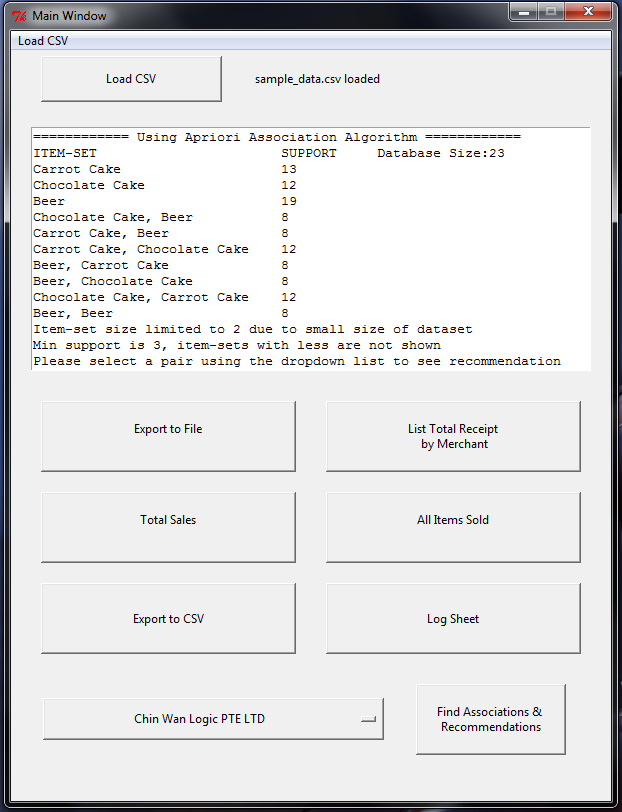
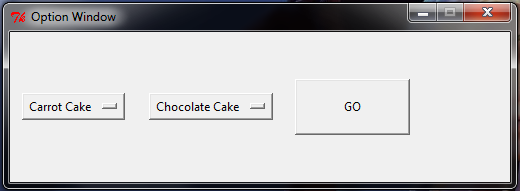
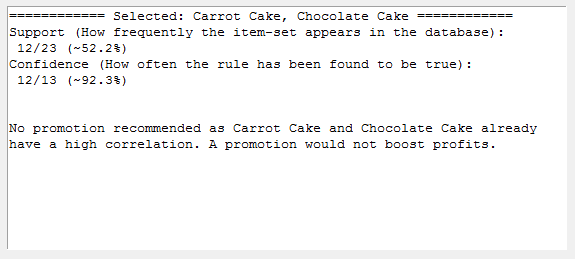


Figure 14 Our multi-step function

# **Limitations**

GUI discrepancies

Due to operating system-specific style differences. We were unable to initialize a standardized GUI between separate platforms. In order to overcome that limitation our group created a separate GUI for OS X systems so that our analyzer would still be functional.

However, should this have been an industry-level project; it would have been more efficient to create an

OS-independent user interface to seamlessly work regardless of platform.

Pre-defined receipt formats

With the resources at hand, we were only able to react to two fixed pre-determined receipts in our analyzer. As a result our group would have to add separate sections in our code to accommodate for a new format of receipt should it be necessary.

Once again, should this have been an industry-level project; it would have made more sense to create a general algorithm in order to detect the company, address, etc.

Limited scenarios for recommending promotions

Given the nature of function 7 & 8, where we calculated and interpreted the associations between two unique items sold by the merchants, we only take into account four different scenarios of correlations. (Refer to figure 12)

In an industry-level project it would be more desirable to have a more comprehensive and flexible algorithm which for example could also take into account how long the product has been on the menu and/or the time of day that the product sees the most sales.

Lack of real-time analysis

As our analyzer is currently only capable of post-data analysis, another limitation would be the inability to give

real-time updates.

Should there be a need however, it would be feasible to connect our analyzer and adapt it to a database such that it would be able to process live updates; possibly in the form of updating instances of the sample data ‘.csv’ file.