Problem - 1

Note : All the dataset files must be placed in the same folder as the code.

Step 1 : Pre-processing step (**preprocess.py)** - Extract information from **monthwisePriceList.csv** and populate 4 dictionaries - **items** stores the mapping from item id to item name while **cp** stores the mapping from item name to item id. **oldp** stores the mapping of item-id to its present price whereas **newp** stores the mapping from item-id to its new price.
Then we use the 4 csv files - **augSales.csv. sepSales.csv, octSales.csv** and **novSales.csv** and generate one CSV file for each segment that has a segment weight associated with it. Here, there are 8 segments that are needed for calculation of the penalty - F1, F2, F3, F4, F5, H1, H2 and Others (containing the P group and F0 (cash) group). So, by combining the transactions of the 4 months (Aug to Nov), one CSV file per segment is made. Each such CSV file has 2 attributes - Slot and Item name. If a food item has multiple occurrences in a transaction, that number of entries are recorded in the segment CSV file for that given time slot. This step is needed as the quantity of the item in a transaction has to be captured while calculating the support of an itemset later and so noting down multiple entries for a given item in a transaction is important.
For example, if a transaction in slot 18 contains Butter Naan with quantity value 2, in the corresponding segment csv file, the following transactions are recorded:

Slot	Item
18	Butter Naan
18	Butter Naan

At the end of this step, a new file **newPrices_read.csv** is generated which is an exact copy of **newPrices.csv**. This file is used later in the final step of modifying the prices of items.

Step 2 : Rule-generation step (**rules.py)** - For the rule-generation step, Orange library is used in Python. One by one, an Orange data object is initialised with the segment table. The **AssociationRulesInducer** is then invoked on this data object with minsup=0.0001 and minconf=0.01. The reason for such a small support is because there is a large number of transactions for each segment and so the probability of a particular rule of the form **Slot=x->Item=y** having a large presence is very less. Moreover, the confidence of a rule does not play a major role here as each slot is of 1 hour and a large number of transactions of a great variety take place every hour in ANC resulting in low confidence value for an item sold in a particular slot to a particular segment. An additional parameter called **classificationRules** is set to 1 to ensure that each association rule has the food item in the consequent and the time slot appears in the antecedent as we require association rules of only that form.
The association rules thus obtained are added to a rule dictionary for each segment after some processing. The association rules are ordered in the decreasing order of their contribution to the maximization of the revenue by sorting them according to the support of the itemset in the rule. Larger the support of the item, more likely it is to have a higher frequency of occurrence compared to other items in December transactions, thereby bringing about a higher increase in

the revenue. Thus, the support is used to list the association rules obtained for each segment in descending order of their usefulness in maximizing the revenue increase.
Once the association rules have been obtained from each segment csv file, a dictionary (storing the mapping from segment to the top rules) is constructed. For choosing the top association rules, the segments F2,F3,F4 are in one category while the rest of the segments are placed in another category. Let **i** denote the number of top association rules chosen for F2,F3,F4 category while **j** denotes the number of top association rules chosen for the rest of the segments. By varying i and j suitably, it is observed that at **i=48** and **j=85**, the penalty is the least with a percentage increase in revenue exceeding 5%.
For the segments F2,F3,F4, the dictionary **d** contains a list of tuples corresponding to the top 39 association rules. For the rest of the segments, **d** contains a list of tuples corresponding to the top 81 rules found for them after sorting according to the criteria explained before. Each tuple is of the form (**Slot,ItemID**), which implies that for the given segment, the price of the item having id equal to **ItemID** must be increased by **newp[ItemID]-oldp[ItemID]** for the given **Slot**.
The reason that the same value **i** is used for F2,F3,F4 while a different value **j** is used for the rest is because F2,F3 and F4 have a much higher segment weight compared to the others and also, the number of transactions for F2,F3 and F4 is much higher compared to the other segments (an average of 1.4MB CSV file size for the first category compared to an average of 296.1kB CSV file size for the second category).

Step 3 - Calculation of parameters (**revenue.py)**
The old revenue is calculated by parsing through the file **decSales.csv** and and making use of the **oldp** dict which stores the original prices for the month of December. The dictionary of relevant rules for each segment **d** is imported from the **rules** file and used to calculate the new revenue of the month of December by making use of the new prices for items corresponding to the given rules in the given slot. After, the new revenue is calculated, the percentage increase in the revenue for the month of December is evaluated using the formula :

Percent increase = (new_rev/old_rev-1)*100

The percentage increase observed is 5.006%.
The next step involves the calculation of the penalty incurred from this price increase. The list of tuples corresponding to each segment is sorted by itemid and the time slot so that calculation of the penalty becomes easier and more efficient in terms of time complexity. A custom comparison function **cmpFn** is defined for sorting the list of tuples. After the slot values of 0,1,2 have been converted to 24,25,26 and the sorting has been performed, the penalty is calculated. This sorting and penalty calculation step is performed for each segment (as different segments have different weights) and the total penalty incurred is stored in **penalty**. For each step, the penalty is calculated as :

pen = (c*c*(newp[iid]-oldp[iid])*wt)

where **wt = segment weight, newp[iid]-oldp[iid] = price increase** and **c*c = hour weight**
The penalty incurred comes out to be 51198.9.
For the modification of the **newPrices.csv**, a new data structure **newprice** is constructed which is a dictionary of dictionaries. Each key in **newprice** corresponds to an item and its value is a dictionary which has (key,value) pairs of the form (Slot,List of segments for which this item's

price is changed in the given Slot). Construction of **newprice** from **d** makes updating **newPrices.csv** a whole lot easier.

The final task is to modify the **newPrices.csv** file. A copy of this file **newPrices_read.csv** is made and stored in the same folder where all these files are stored. This is because we require to read the file and make modification at the same time and the **csv** module of python does not give us the flexibility of reading and writing on the same csv file at the same time. So, **new_Prices.csv** is opened for writing while **newPrices_read.csv** is opened for reading. For each segment, a linear scan is made through the list **newprice[itemID][slot]** to see whether the segment is present there or not. If the segment is present, then the updated price of that item is written else the original price of the item is used.