Mini Project: Point of Sale System

IMPLEMENTATION REPORT

Meghna Asthana

Software Technologies for Data Science

Table of Contents

TABLE OF FIGURES 2

INTRODUCTION 3

GETTING STARTED 4

SOFTWARE VERSIONS AND IDE 4
SETUP AND INITIAL RUN 4

DESCRIPTION OF THE CODE 5

TASK 1.1: UPGRADATION FROM PYTHON 2 TO PYTHON 3 5
CHANGES MADE: 5
TASK 1.2: IMPLEMENTATION OF ACTION PROGRAMS AND SETUP OF DATABASE 5
ACTION PROGRAMS: 5
SETUP OF DATABASE IN SQLITE: 7

TASK 1.3: MEAL DEAL IMPLEMENTATION 8

TASK 2.1: REPORTING DAILY AND PRODUCT SALES 9

TASK 2.2: APRIORI ALGORITHM IMPLEMENTATION 11

RESULTS ON GATHERED DATA 11
RESULTS ON SUPPLIED DATASET 11

DIFFICULTY IN CODE PRODUCTION AND TASK IMPLEMENTATION 12

OUTSTANDING ISSUES AND CONSTRAINTS 12

CONCLUSION AND RECOMMENDATIONS 13

APPENDICES 14

SCREENSHOT OF TRANSACTIONS 14

Table of figures

FIGURE 1: DATABASE SCHEMA 7
FIGURE 2: RESULT FOR DAILY PRODUCT FREQUENCY 9
FIGURE 3: RESULT FOR DAILY TOTAL SALES 10
FIGURE 4: RESULT FOR DAILY TRANSACTION TYPE COUNT 10
FIGURE 5: RESULT FOR DAILY DISCOUNT 10
FIGURE 6: RESULT FOR TOTAL SALES OF A PRODUCT 11
FIGURE 7: APRIORI RESULTS FOR GATHERED DATA 11
FIGURE 8: APRIORI RESULTS FOR SUPPLIED DATASET 11

Introduction

This project has been aimed at designing a backend python server for a till system, generally used for taking orders and keeping track of all transactions performed. We are provided with a fully functional frontend and skeleton of the backend code from which we have to build up to a system capable of taking up orders, identifying meal deals, calculating total and discount and updating all the information to the database.

In this report, we first provide an introduction to all the software and IDEs that have been utilized to accomplish every task. Following this, we have provided steps for setup and initial run which includes every command to be executed in the terminal. In the implementation section, we describe the code for each task. This includes detailed explanation of every step, information about important functions and sample results, if required. In the discussion section, we review the problems encountered during the development of this project which would be helpful for the reader to understand the thought process of the programmer. Furthermore, we highlight the persisting issues and bugs in the code which can be scrutinized for improving the code. In conclusion, we discuss about recommendations for improvement of code and its extension to incorporate new and complex features in the future.

Getting Started

Software Versions and IDE

The project encompasses various aspects which have been programmed using variety of languages and IDEs. The frontend of the tills has been written in HTML 4.01, styled using CSS and functionality has been provided by JavaScript. Though, no editing has been required on the frontend code, it has been viewed using Visual Studio code 2 v1.28.2. The frontend can be accessed as a website on the localhost server by opening the till.html or till2.html file on a compatible browser. The code has been made compatible to Firefox, Chrome, Opera, Safari, IE7+, IE6 and IE5.

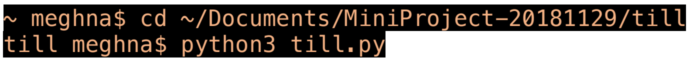
The backend of the tills project has been coded in Python 2 initial, which has been upgraded to Python 3 before implementing any new functionality. The backend has also been viewed and edited using Visual Studio Code 2 v1.28.2. The database management utilised for this project is SQLite 3 DB API 2.0 v2.5 and has been viewed using an online browser at sqliteonline.com. The port used for setting up connection between the backend python code and frontend tills interface is localhost:8080 which is separate from the default port of 80.

For the analysis portion of the project, Anaconda Navigator 3 and Jupyter Notebook v5.6.0 with Python 3 has been used.

Setup and initial run

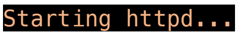
The project has been developed on Macbook Pro with macOS Sierra v10.12.6 installed. Thus, all the steps provided are for a Macintosh OS only. Following are the steps for setup and initial run:

a) The terminal of macOS is opened and command which changes the current directory to directory containing the python file is execute. Next, the python file is opened using the following:



```
~ meghna$ cd ~/Documents/MiniProject-20181129/till
till meghna$ python3 till.py
```

b) The python file is successfully running if the following message appears in the terminal:



```
Starting httpd...
```

c) Finally, the frontend of the till is accessed using the url http://localhost:8080

d) In order to terminate the current server Ctrl + C is used.

Description of the code

Task 1.1: Upgradation from Python 2 to Python 3

Changes made:

1. The `self.send_response(200)` was moved from bottom most position of the elif block to the topmost position. This allows the `'/till.css'` file to be loaded and read after the response has been sent to the server.

```
elif self.path == '/till.css':
fname = 'till.css'
file = open(fname, "r")
text = file.read()
self.send_header('Content-type', 'text/css')
self.send_response(200)

elif (self.path == '/till.css'):
self.send_response(200)
fname = 'till.css';
file = open(fname,"r")
text = file.read()
self.send_header('Content-type', 'text/css')
```

2. The port number for the HTTPServer has been changed from `port=80` to `port=8080`. This has been done in order to create a new port for the local server and not use the default server.

```
def run(server_class=HTTPServer, handler_class=S, port=80):

def run(server_class=HTTPServer, handler_class=S, port=8080):
```

3. The Python 3 version requires user to follow the print keyword with round brackets which encloses the target values to be printed.

```
print 'Starting httpd...'

print('Starting httpd...')
```

Task 1.2: Implementation of action programs and Setup of Database

Action programs:

In order to implement each of the action programs the incoming url requests from the front-end server has been parsed. A typical request looks like this:

```
"GET /action?action=cash&refund=0&type=3&cash=0&randn=57437 HTTP/1.1" 200 -
```

The parsing process includes recognizing the type of action requested by the url by splitting it into two halves by '?'. Based on the action, the second half string value has to be parsed. It is again separated into two halves by '=' out of which LHS is the parameter and RHS is its corresponding value.

1. ***Action=plu***

   The plu action has three parameters namely: refund, num and plu. Each of the parameter value are extracted and utilized in the following:

   a) The server is connected to the SQLite database where it searches for the product in the 'products' table with the plu value entered. If the product with a the plu value exists it is inserted into the current transaction row in SQLite database, otherwise, it is considered as a malicious request and no transaction is performed.

   b) A XML response is created which displays product info which include product name (extracted from SQLite database), number of items (num parameter) and whether transaction is refund (refund parameter) in the till display if the transaction is valid otherwise it sends an error message 'Plu ID not found'.

2. ***Action=cash***

   The cash action has three parameters refund, type, cash. Each of the parameter value are extracted and utilized as:

   a) Connection is made to the SQLIte database and all the items from the current transaction are saved in a list. The current transaction is identified by the status value in the 'transactions' table; the current transaction will always have status as 0 whereas older transactions will have status as 1.

   b) After collecting all product IDs in the transaction, the price of each is retrieved from the 'products' table and the total value of transaction is calculated.

   c) The total calculated is then compared to the cash parameter value and the change value is calculated by taking the difference of the two. These values are then fed into the XML response which displays the total and change for the transaction.

   d) The 'transactions' table is updated with the refund parameter as 'transactiontype', type as 'methodid' and 1 as 'status'.

3. ***Action=program***

   The program action has five parameters refund, num, pos, shift and value. Each of these are extracted and utilized as:

   a) Using the pos parameter value the 'productid', 'name', 'shift' and 'price' values for the item generating the url request is retrieved from 'products' table.

   b) As the shift values in the 'products' table and url request are not consistent, an additional if condition statement is introduced which checks the shift value and appropriate product size. The shift values are designated as:

Shift Value	Size
1	Small
2	Medium
3	Large
3 and only one product retrieved	No size
Any other value	No size

c) The current transaction ID is retrieved from the ‘transactions’ table with status=0 and the timestamp, transaction ID, product ID values are fed into the ‘items’ table. If no current transaction is found, it creates a new transaction and feeds all the values with the new transaction ID in the ‘items’ table.

d) The XML response is created with the size, product name and number of orders which is displayed on the till. It also displays the price of each item in the total div.

4. *Action=clr*

a) The clr action has no additional parameters. When requested, it connects to the database and updates the ‘transactions’ table with parameters; status=1 (signifying transaction ended), transactiontype=-1 and methodid=-1 (signifying a voided transaction).

b) The XML response resets the display and returns a message ‘Transaction voided.’

5. *Action=status*

The status action has no additional parameters. When requested, it connects to the database and fetches all items in the current transaction. It sends a XML response which displays all the items in the newly loaded window of till.html or till2.html.

Setup of Database in SQLite:

1. *Description of code:*

All the code regarding database creation is contained in database.py file. Following are the steps involved:

a) The sqlite3 library is imported in the database.py file and a database called ‘products.db’ is created. Following this, a cursor object is created to execute all SQLite queries.

b) As the file products.txt contains SQLite queries for three tables namely: products, payment method and shift, it is first read and each query is executed using the cursor object.

c) Two addition tables have been created for ease in recording transactions and items sold. After the commands, the changes are committed and the database is closed.

2. *SQLite Schema:*

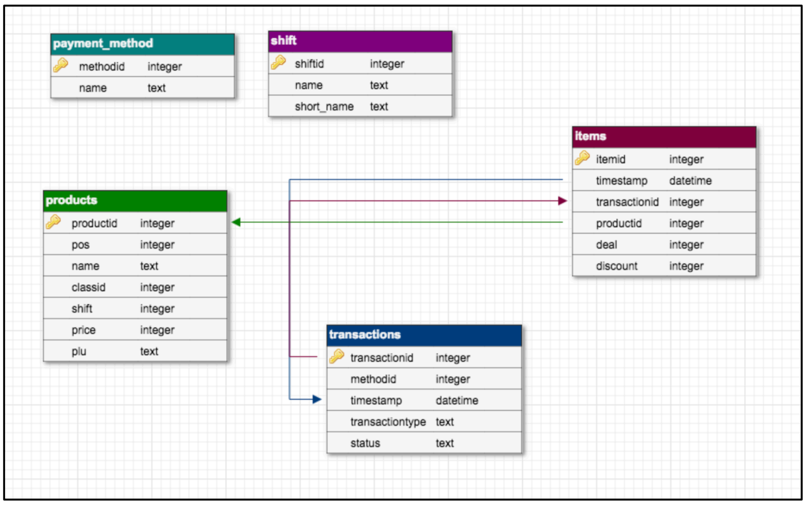


Figure 1: Database schema

3. ***Common SQLite commands used:***

SQLite COMMAND	Usage
sqlite3.connect()	Creates or connects to database
db.cursor()	Cursor object
cursor.execute("""CREATE TABLE…""")	Creates a new table
cursor.execute("""SELECT…""")	Selection query
cursor.execute("""INSERT into…""")	Insertion query
cursor.execute("""UPDATE…""")	Update query
db.commit()	Commits query to database
db.close()	Disconnects from database

Task 1.3: Meal Deal Implementation

Meal Deal implementation has been performed in the action=cash program, therefore, all meal deals are only identified when the final cash, credit or debit button is clicked on the tills frontend. The general idea behind optimized selection of products is identifying all deals possible after all ordered items are ready to be purchased. The algorithm first tries to find all products which fit Deal 2 and collects the leftover items and performs the same task for Deal 1 until no deals are applicable. Finally, the remaining items are registered at full price outside the meal deals. Following are the steps involved:

a) When the action=cash is requested in the url, the items in the current transaction are collected in a list. Other lists are created which contain the classid and price of the corresponding items, retrieved from the 'products' table.
b) First, we calculate the number of items which qualify for Meal Deal 2. This is achieved by enumerating valid sides and drinks in separate lists. The length of side and drinks list is compared and the minimum of the two is the number of Meal Deal 2 applicable.
c) A for loop runs for the number of Meal Deal 2 applicable and calculates the original and discounted price. These values are then updated into the deal and discount fields in the 'items' table for the current transaction. The discount is also calculated by taking the difference of original and discounted price.
d) Next, we append the remain sides and drinks in the main sides and drinks list. A new list of main items is created which will be used for assessing the number of Meal Deal 1 applicable. The process of evaluating this is exactly the same as for Meal Deal 2.
e) Finally, we find the leftover items in each category and calculate their cost outside of any meal deal at full price. These values are then updated in the deals and discount table as 0, signifying no deal and full price.
f) The XML response is generated which displays the output as the number of meal each deal applicable, price after discount, number of items outside meal deals and the total amount to be paid on the tills frontend.

Task 2.1: Reporting Daily and Product sales

The daily and product sales report have been generated based on the data saved on the ‘items’ and ‘transactions’ tables in the products database. The analysis has been performed using Python 3 in Jupyter Notebook. Following are the steps involved and major function defined:

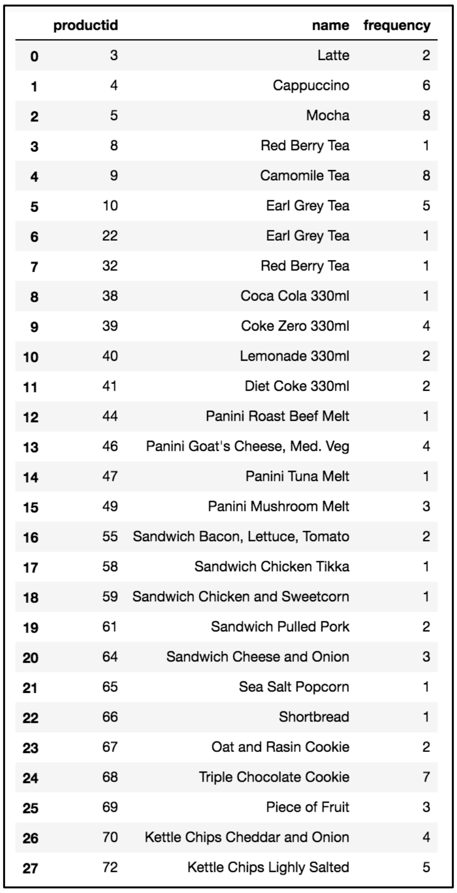
a) First, sqlite3, pandas, numpy and datetime libraries are imported and the Jupyter Notebook is connected to the products database.

b) Using pandas.read_sql_query(), items, transactions and products table are loaded into the notebook.

c) For ease, another column namely, dayofyear has been added to the itemsdf and transactiondf dataframes by converting the date in timestamp to nth day of the year. This has been achieved by creating a datetime object for every timestamp and using the timetuple().tm_yday function on the object. Similarly, price (converted from pence to pounds) column has added to itemsdf dataframe.

d) ***def daily_product_freq(day):***

The function takes nth day of the year as argument and returns the frequency of each product sold on that particular day as a dataframe. In order to achieve this, all the productids are retrieved from itemsdf where the dayofyear value is equal to argument ‘day’. The numpy.unique() function is used to obtain the frequency of each productid. These values are compiled in a dataframe productsolddf and returned.

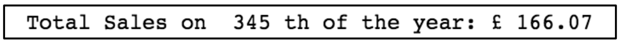


	productid	name	frequency
0	3	Latte	2
1	4	Cappuccino	6
2	5	Mocha	8
3	8	Red Berry Tea	1
4	9	Camomile Tea	8
5	10	Earl Grey Tea	5
6	22	Earl Grey Tea	1
7	32	Red Berry Tea	1
8	38	Coca Cola 330ml	1
9	39	Coke Zero 330ml	4
10	40	Lemonade 330ml	2
11	41	Diet Coke 330ml	2
12	44	Panini Roast Beef Melt	1
13	46	Panini Goat's Cheese, Med. Veg	4
14	47	Panini Tuna Melt	1
15	49	Panini Mushroom Melt	3
16	55	Sandwich Bacon, Lettuce, Tomato	2
17	58	Sandwich Chicken Tikka	1
18	59	Sandwich Chicken and Sweetcorn	1
19	61	Sandwich Pulled Pork	2
20	64	Sandwich Cheese and Onion	3
21	65	Sea Salt Popcorn	1
22	66	Shortbread	1
23	67	Oat and Rasin Cookie	2
24	68	Triple Chocolate Cookie	7
25	69	Piece of Fruit	3
26	70	Kettle Chips Cheddar and Onion	4
27	72	Kettle Chips Lighly Salted	5

Figure 2: Result for daily product frequency

e) ***def daily_total_sales(day):***

The function takes nth day of the year as argument and returns the total sale on that particular day. In order to achieve this, all the productids are retrieved from itemsdf where the dayofyear value is equal to argument ‘day’. The numpy.unique() function is used to obtain the frequency of each productid. The frequency is multiplied by the price for corresponding productid and returned as summed up.

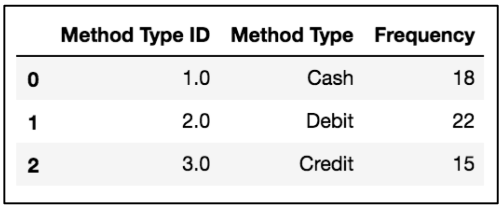


```
Total Sales on  345 th of the year: £ 166.07
```

Figure 3: Result for daily total sales

f) ***def daily_transaction_type_count(day):***

The function takes nth day of the year as argument and returns the frequency of each type of payment on that particular day as a dataframe. In order to achieve this, all the methodids are retrieved from transactionsdf where the dayofyear value is equal to argument ‘day’. The numpy.unique() function is used to obtain the frequency of each methodid. These values are compiled in a dataframe methodsdf and returned.

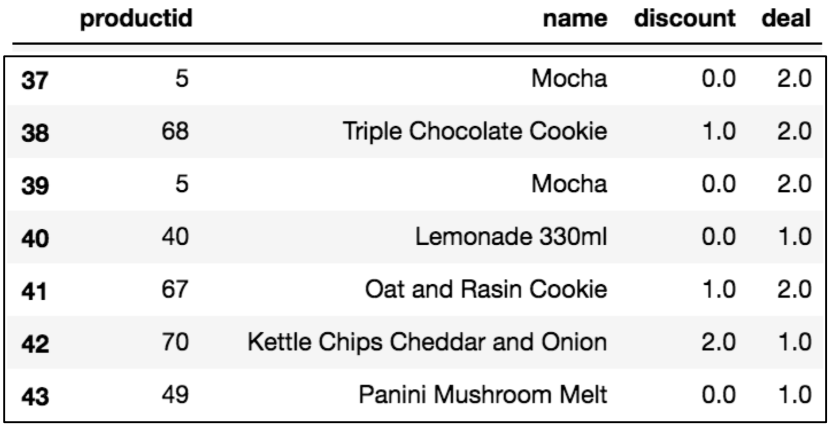


	Method Type ID	Method Type	Frequency
0	1.0	Cash	18
1	2.0	Debit	22
2	3.0	Credit	15

Figure 4: Result for daily transaction type count

g) ***def daily_discount(day):***

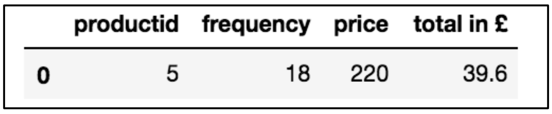
The function takes nth day of the year as argument and returns the frequency of each type of deal and discount on that particular day as a dataframe. Here, the discount value of 0 signifies paid full price, 1 for paid half price and 2 for free item. The deal column values signify 1 for Meal Deal 1 and 2 for Meal Deal 2.



	productid	name	discount	deal
37	5	Mocha	0.0	2.0
38	68	Triple Chocolate Cookie	1.0	2.0
39	5	Mocha	0.0	2.0
40	40	Lemonade 330ml	0.0	1.0
41	67	Oat and Rasin Cookie	1.0	2.0
42	70	Kettle Chips Cheddar and Onion	2.0	1.0
43	49	Panini Mushroom Melt	0.0	1.0

Figure 5: Result for daily discount

h) ***def total_sales(productid):***
The function takes product ID as argument and returns the frequency of product sold on every day of the recording period as a dataframe.

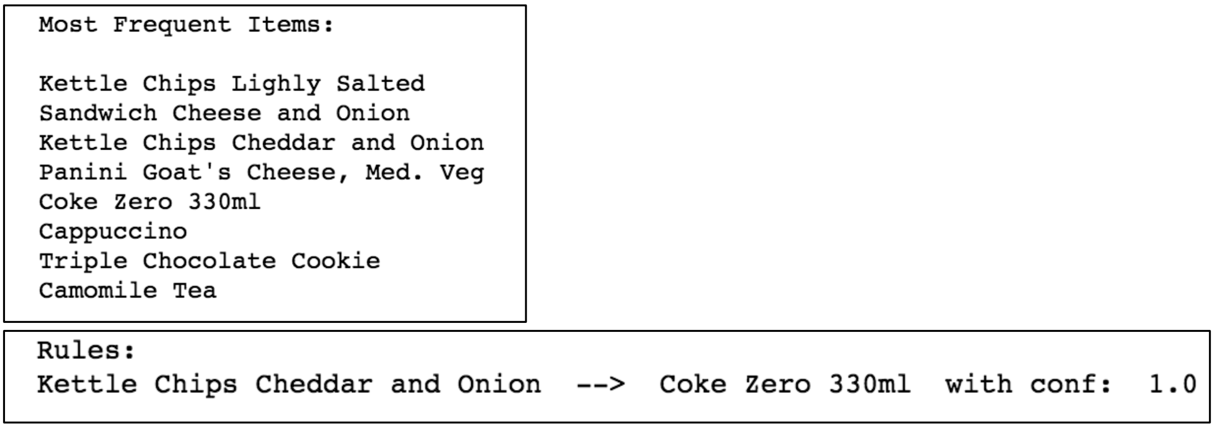


	productid	frequency	price	total in £
0	5	18	220	39.6

Figure 6: Result for total sales of a product

Task 2.2: Apriori Algorithm Implementation

Results on gathered data



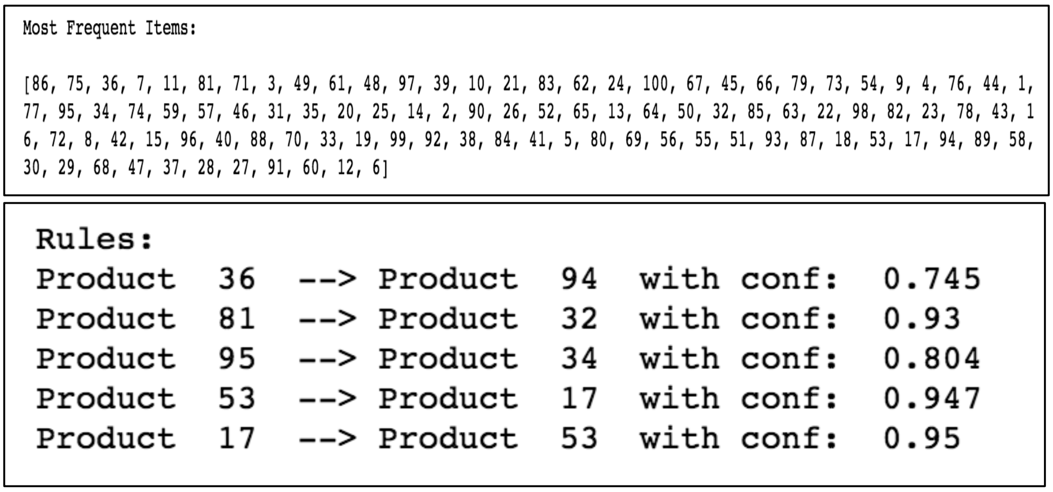
```
Most Frequent Items:

Kettle Chips Lighly Salted
Sandwich Cheese and Onion
Kettle Chips Cheddar and Onion
Panini Goat's Cheese, Med. Veg
Coke Zero 330ml
Cappuccino
Triple Chocolate Cookie
Camomile Tea
```

```
Rules:
Kettle Chips Cheddar and Onion  -->  Coke Zero 330ml  with conf:  1.0
```

Figure 7: Apriori results for gathered data

Results on supplied dataset



```
Most Frequent Items:

[86, 75, 36, 7, 11, 81, 71, 3, 49, 61, 48, 97, 39, 10, 21, 83, 62, 24, 100, 67, 45, 66, 79, 73, 54, 9, 4, 76, 44, 1,
77, 95, 34, 74, 59, 57, 46, 31, 35, 20, 25, 14, 2, 90, 26, 52, 65, 13, 64, 50, 32, 85, 63, 22, 98, 82, 23, 78, 43, 1
6, 72, 8, 42, 15, 96, 40, 88, 70, 33, 19, 99, 92, 38, 84, 41, 5, 80, 69, 56, 55, 51, 93, 87, 18, 53, 17, 94, 89, 58,
30, 29, 68, 47, 37, 28, 27, 91, 60, 12, 6]
```

```
Rules:
Product  36  --> Product  94  with conf:  0.745
Product  81  --> Product  32  with conf:  0.93
Product  95  --> Product  34  with conf:  0.804
Product  53  --> Product  17  with conf:  0.947
Product  17  --> Product  53  with conf:  0.95
```

Figure 8: Apriori results for supplied dataset

Difficulty in code production and task implementation

1. During upgradation of backend code from Python 2 to Python 3, scouting for changes other than obvious ones were quite time consuming due to insufficient experience in Python 2.
2. Initial running of python file through terminal window was challenging as there were many hassle while changing directories, to overcome this the mini project folder was moved to Documents and accessed directly from there.
3. There were issues regarding the system being locked in macOS. This was overcome by using 'sudo' before every command which gives the admin access to any user provided he or she knows the system password.
4. For macOS the command py till.py did not seem to work. After research, it was resolved by using python3 till.py.
5. It was difficult to parse the url request generated for each action and create an appropriate XML response to simultaneously display corresponding values on the frontend and update the database.
6. Whenever need for a new column arose in any of the tables, the complete database had to be scraped and created from scratch. This led to loss of transaction data multiple times.
7. Due to inconsistency in values in shift table and url request, it was tedious to handle sizes of drinks.
8. It is a tedious task to generate a transaction dataset large enough to implement Apriori algorithm appropriately.

Outstanding Issues and Constraints

1. The transactions voided are not deleted from the database, they are marked transactiontype equal to -1 in the 'transactions' table. Thus, careful attention is needed to identify and separate out these transactions and their items when calculating daily sales and product sales.
2. In some action cases, the num function has not be utilized and multiple entries are required for entering more than one item with the same product ID.
3. For action=status, when a transaction is in progress, the display only shows selected items from the previous window when a new item is selected otherwise the display remains clear.
4. While switching from till window 1 to till window 2 and vice versa, the total sets to a default value of £2.00 and displays the item price when a new item is selected.
5. The price of small and medium sizes has been interchanged.
6. In the Meal Deal implementation, when MD#2 is not available, the algorithm is unable to recognize whether MD#1 is implementable or not.
7. The value of payable amount and discount displayed on the target div in till shows the amount in pence.
8. The update query for deal and discount values does not work for non-deal items.

Conclusion and Recommendations

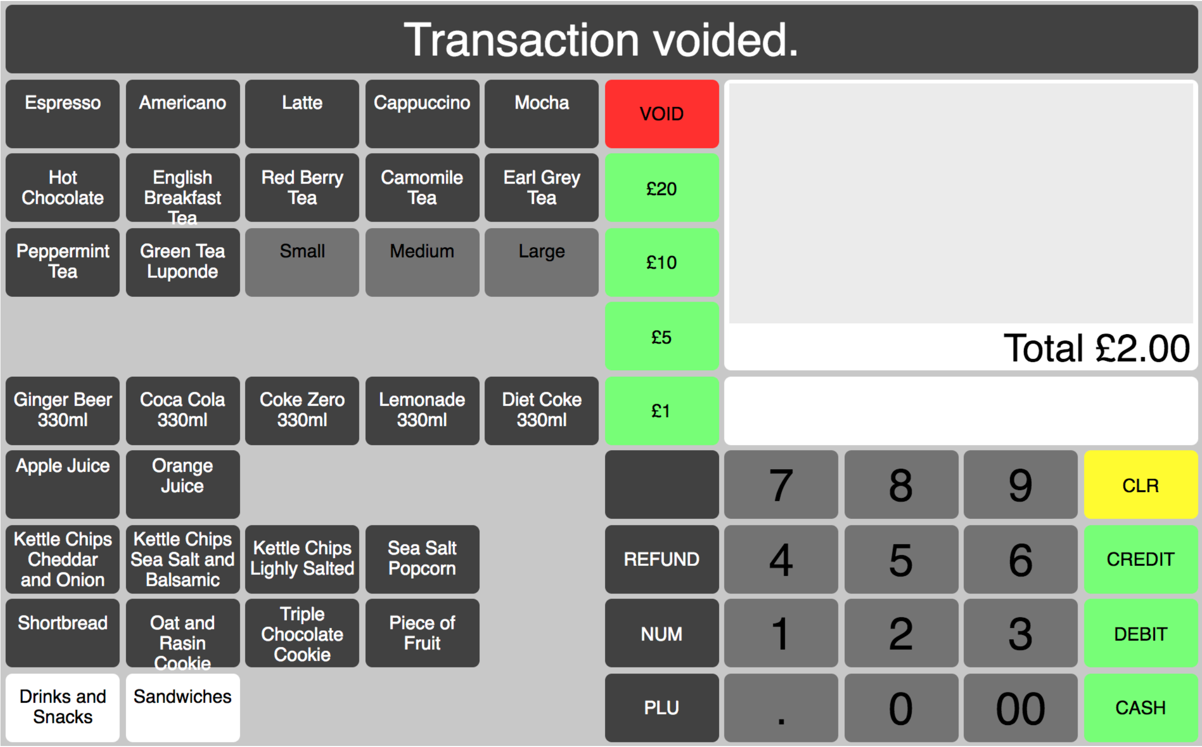
In this report, we gave an introduction to setup and run the system on macOS which includes every command to be executed in the terminal. In the implementation section, we provided detailed explanation of every step, information about important functions and sample outputs. We discussed about the problems faced during the development of this project. Moreover, we emphasized on the persisting issues and bugs in the code for further scrutiny of the code and providing explanation of some incorrect results. Finally, we will be discussing about recommendations for improvement of code and its extension to incorporate new and complex features in the future.

Following the analysis in task 2.1 and 2.2, a recommendation system can be built which provide optimized daily or hourly deals depending upon customer preference at that point of time. It can be further extended to predict the optimal inventory stock required to cope up with the demands of different items. This will not only reduce wastage of unwanted items but also increase the overall sale of popular items which get sold out due to poor inventory management.

Appendices

Screenshot of Transactions

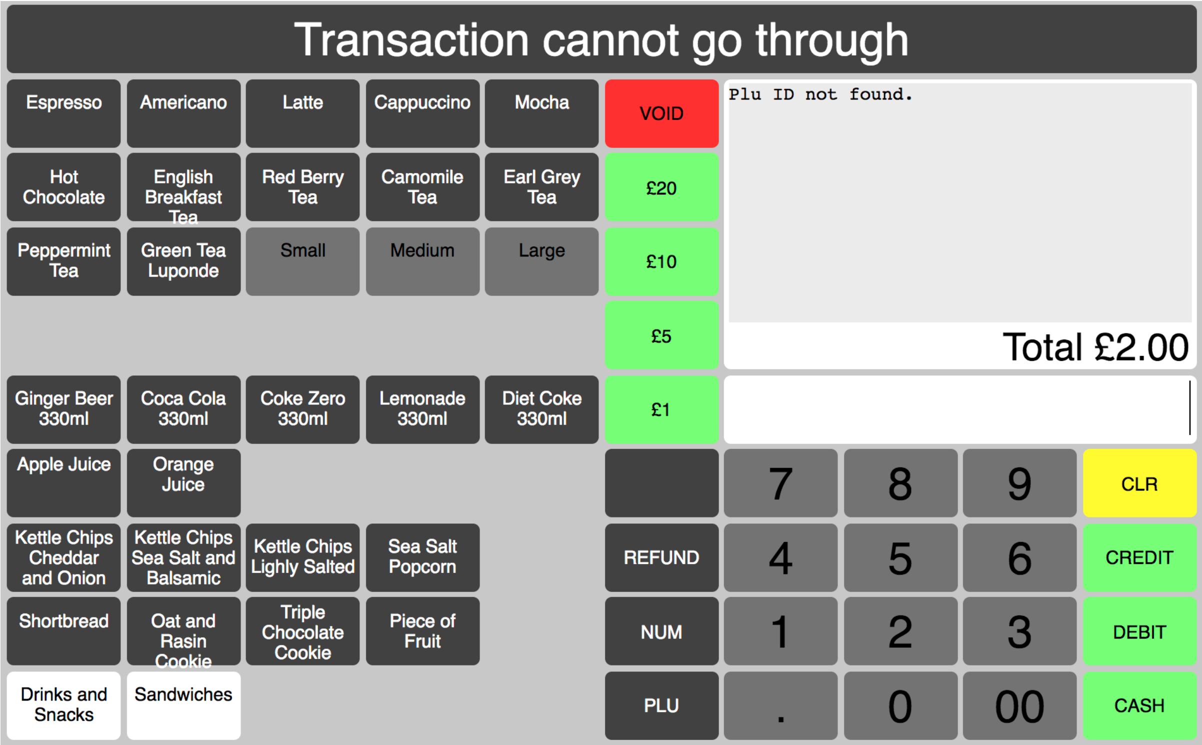
Transaction Voided:



Transaction voided.

Total £2.00

Plu ID not found:

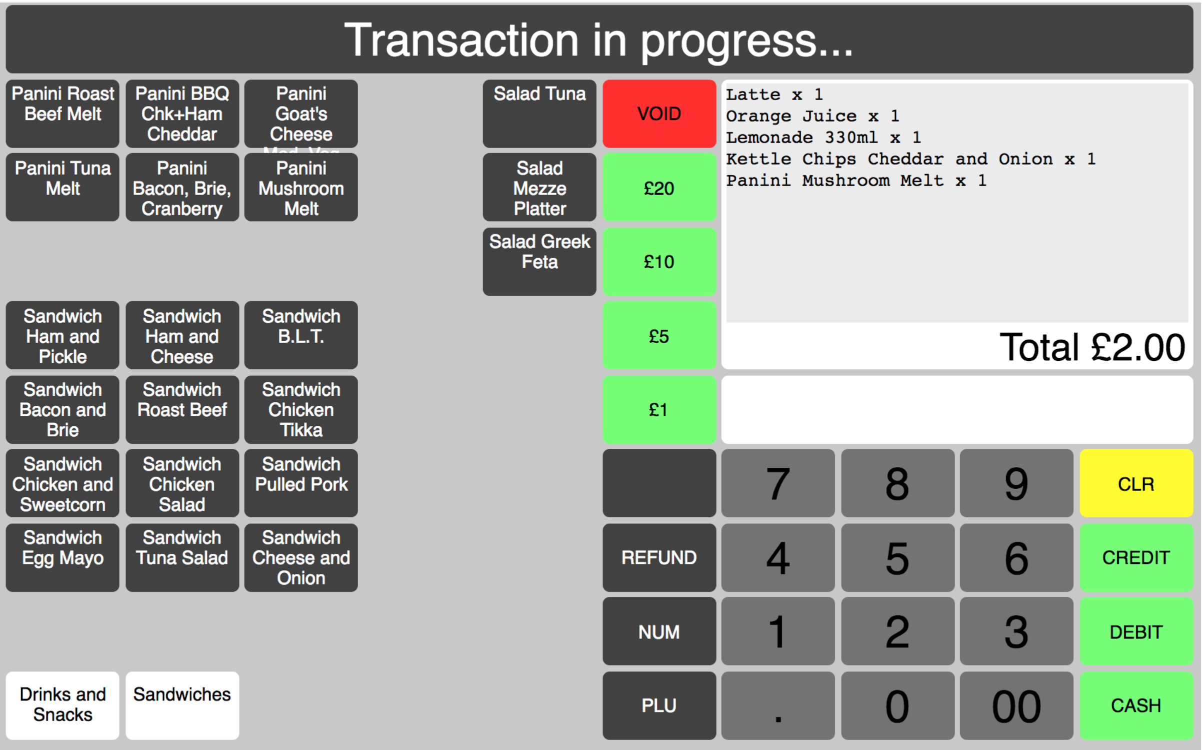


Transaction cannot go through

```
Plu ID not found.
```

Total £2.00

Transaction showing change of till windows:

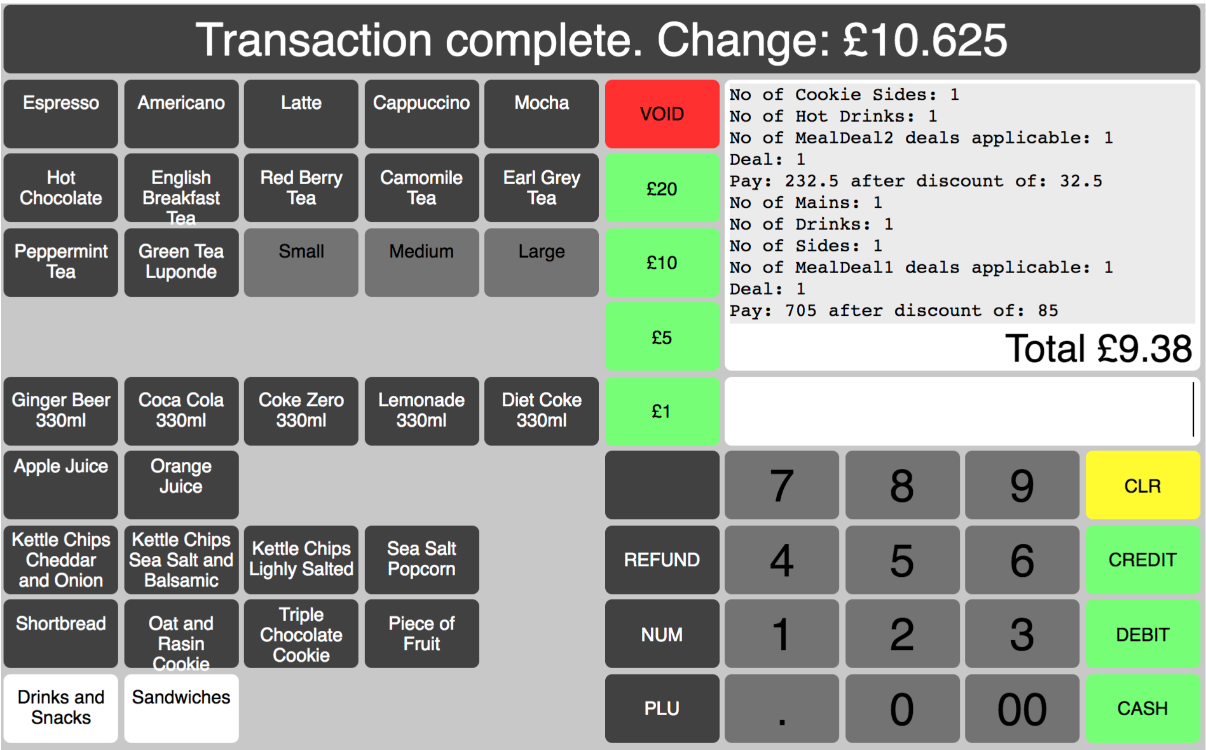


Transaction in progress...

```
Latte x 1
Orange Juice x 1
Lemonade 330ml x 1
Kettle Chips Cheddar and Onion x 1
Panini Mushroom Melt x 1
```

Total £2.00

One Meal Deal 1 and One Meal Deal 2:

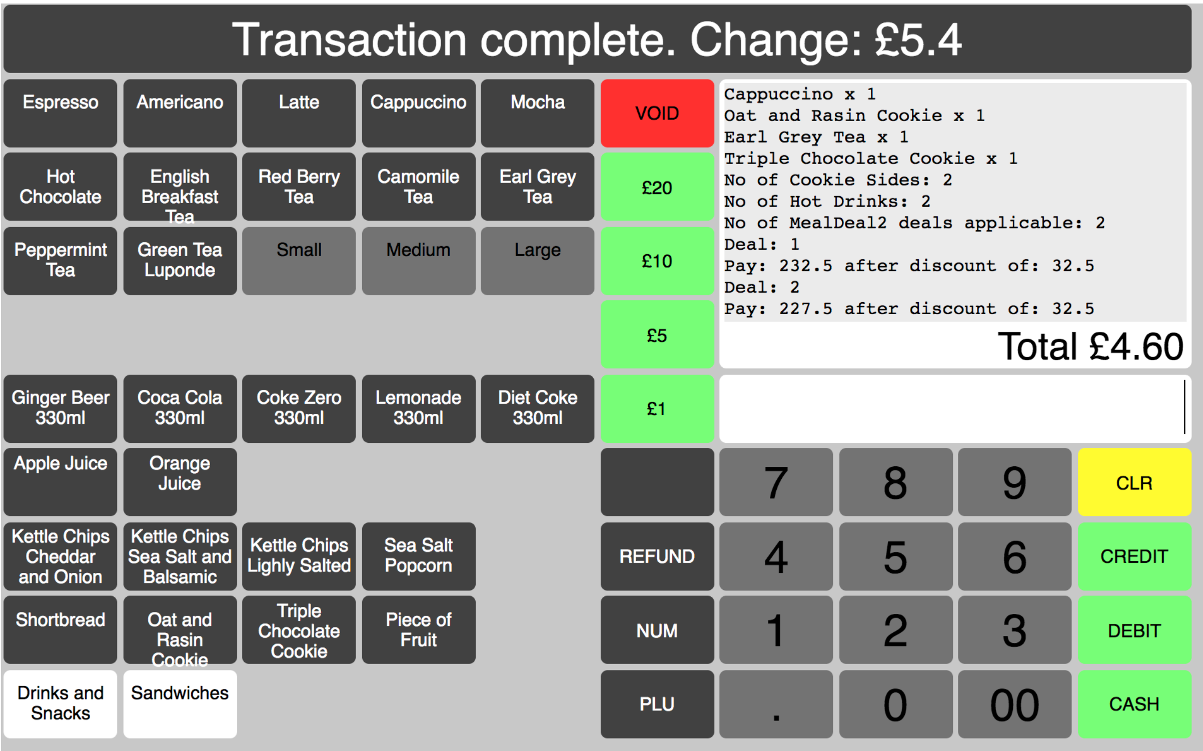


Transaction complete. Change: £10.625

```
No of Cookie Sides: 1
No of Hot Drinks: 1
No of MealDeal2 deals applicable: 1
Deal: 1
Pay: 232.5 after discount of: 32.5
No of Mains: 1
No of Drinks: 1
No of Sides: 1
No of MealDeal1 deals applicable: 1
Deal: 1
Pay: 705 after discount of: 85
```

Total £9.38

Two Meal Deal 2:



Transaction complete. Change: £5.4

```
Cappuccino x 1
Oat and Rasin Cookie x 1
Earl Grey Tea x 1
Triple Chocolate Cookie x 1
No of Cookie Sides: 2
No of Hot Drinks: 2
No of MealDeal2 deals applicable: 2
Deal: 1
Pay: 232.5 after discount of: 32.5
Deal: 2
Pay: 227.5 after discount of: 32.5
```

Total £4.60