***Airline Seat Assignment***

***MIS40750 - Analytics Research & Implementation***

2017

Team Members:

|  |  |  |
| --- | --- | --- |
| Balaji Sridhar Subramanian | Andrew Meade | Shalini Shivam |
| 16203531 | 13214665 | 16200815 |

Introduction

The aim of this project was to create a software algorithm that facilitated the booking of seat on an airplane subject to a set of conditions.

1. The conditions we had to adhere to included:
2. Bookings of multiple passengers should be seated together where possible.
3. Bookings containing more passengers than available seats should be rejected.
4. A database should be updated to reflect the number of passengers refused and the number of passengers seated separately.

User Guide:

We need to do the following to run the python script:

1. Open the command prompt and input the seat\_assign\_16203531\_13214665\_16200815.py. This will open the code.
2. The code will automatically read a CSV file called booking.csv, and update the airline\_seating.db database file. These are the default files taken unless new parameters are provided.

Assumptions made:

The plane for which bookings are being taken has 15 rows, each containing 4 seats. The seats are labelled A,C,D,F.

If at least one passenger is seated in a separate row to the rest of their booking party, then all passengers in that booking are deemed to be seated separately.

Passengers are seated from left to right and from the front of the plane to the back. One exception exists, when seats are left vacant to accommodate a booking in a separate row, the skipped seats with be filled with the next appropriate booking size or when separating passengers becomes a necessity.

Overview of the code:

In the first section of our code we have defined a function which will allocate the flight tickets by reading the database file. After reading the file we need to check if the number of seats available is greater than the request and perform the seat allocation function. If both criteria are not me for these functions, the booking request will be rejected.

As previously stated, our code begins by reading in a CSV file. Next, we call a function called *load\_seating\_layout* which is used for storing the seat plans in a variable. After this, *load\_seat\_avail* is used to load the current availability status from the database.

*Check\_seating\_avail* is used to check the number of available seats and determine whether a group of passengers can be seated together. The output here tells us how many separate seats and consecutive sears are available.

The function *confirm\_booking*, as expected, confirms the booking if the number of seats are available. If there are more passengers than available seats, we reject the booking. If there are less passengers than available seats, we assign seats in the same row where possible, consecutively where there is no row with sufficient seats, and separately in assorted seating if necessary. The seating table is updated with the passenger names and assigned seats.

Once booking is completed, records are inserted into the database by using the *update\_reports* function. For each reservation request we update the reporting table for the number of rejections and for the number of passengers seated in a split party. For successful reservations, we update the seating table.

Finally *load\_metrics* is used to sum the data from *update\_records* to provide the required output for the number of passengers that have been seated separately and the number of passengers that have been rejected outright, in addition to the number of booking requests handled.

Testing

In order to test the code we have the following test cases. We have verified the code for non-consecutive booking, number of seats available, number of rejections and the consecutive bookings.

**Verify non-consecutive booking:**Given - Only 4 seats(non consecutive) are available in database  
 And Metrics table is empty  
When - Reservation request is for 4 passenger is issued  
Then - Seats are allocated to all 4 passenger  
 And Metrics database is updated  
 And Number of separation count is 4**Verify number of seats available:**

Given - When flight is fully booked  
 And Metrics table is empty  
When - Reservation request is for 1 passenger  
Then - No seats are allocated  
 And Metrics database is updated  
 And Number of refusal count is 1**Verify refusal count:**

Given - When the flight booking is empty  
 And maximum seat capacity is 60  
 And Metrics table is empty

When - Reservation request is issued for 100 seats  
Then - No seats are allocated  
 And Metrics table is updated  
 And Number of passenger refused count is 100**Verify consecutive bookings:**Given - When there are 4 consecutive seats and 2 non- consecutive seats available  
 And Metrics table is empty

When - Reservation request is issued for 4  
Then - 4 consecutive seats are allocated  
 And Metrics table is not updated  
 And Number of passenger separation is zero

**Test CSV Loader**  
 Given - A csv file with 100 rows are available  
 When - Reservation requests in the files are processed  
 Then - Total number of requests handled in the session should be 100  
  
**Test Column pattern**  
 Given - Seating layout with ACDF is defined in table rows\_cols  
 When - Seating layout is loaded  
 Then - self.seating\_layout[0][1] should be ACDF  
  
**Test Row Numbers**  
 Given - Noof rows is 15 in rows\_cols table  
 When - Seating layout is loaded  
 Then - self.seating\_layout[0][0] should be 15

Conclusion

We have effectively fulfilled the requirements of the assignment. We have successfully designed a booking system that aims to seat all passengers together where possible. The system endeavours to seat passengers as close as possible where accommodating the full party together is not possible. By working together and developing the necessary functions and iterations we have developed a working model that we can all be confident in.