

DSA8002 Assignment

Singapore Airbnb Data Analysis

Yogesh Kashiram Bore

Student No:40311162

Email: ybore01@qub.ac.uk

Contents

[1. Background: 2](#_Toc66631776)

[1.1 Domain information: 2](#_Toc66631777)

[1.2 Technical information: 2](#_Toc66631778)

[2. Class Design: 3](#_Toc66631779)

[2.1 Class information: 4](#_Toc66631780)

[3. Database Design: 5](#_Toc66631781)

[4. SQL Operation Design: 6](#_Toc66631782)

[5. Essential Key Code Snippets: 7](#_Toc66631783)

[6. Project Demo: 16](#_Toc66631784)

[7. Unit Testing Cases: 18](#_Toc66631785)

[8. Summary: 19](#_Toc66631786)

[9. References: 20](#_Toc66631787)

[10. Appendix: Source Code 21](#_Toc66631788)

[Appendix A: Host Code 21](#_Toc66631789)

[Appendix B: Neighbourhood Data Code 25](#_Toc66631790)

[Appendix C: Neighbourhood Group Data Code 28](#_Toc66631791)

[Appendix D: Review Data Code 32](#_Toc66631792)

[Appendix E: Room Data Code 37](#_Toc66631793)

[Appendix F: Room Type Data Code 41](#_Toc66631794)

[Appendix F: Inheritance Code 44](#_Toc66631795)

[Appendix G: Unit Testing Code 46](#_Toc66631796)

[Appendix H: Demo Function Code 48](#_Toc66631797)

# Background:

In this project Airbnb in Singapore listing data analysis, cleaning and converting into useful form is done along with data operations like insert, update, delete. I have tried to distribute every Airbnb listing based on their host, review, room type, Neighbourhood data, and few other things. Given 7907 sample data from listing csv file is used for Data analysis, cleaning and perform certain operations to convert data into normal form and store it into data frame by using python panda’s library. Then insert this sorted and cleaned data into database in Sqlite3. Then created a relational database to perform like display, insert, update, and delete operations. Python classes are used for development.

Data analysis on millions of listings provided through Airbnb is a difficult to make conclusions and take decisions for the organization. So we took this raw data transform it into a structured data using database normalization and make it ready for organization to take the decisions and to find out the answers of the questions by populating the data from the created database. In this project

We populated listing records that has 200 or above total reviews in the database. We can get answer of the multiple such questions by populating required data. It is really easy for organization to take decision from these answers and populated data.

## 1.1 Domain information:

I used an Airbnb in Singapore listing dataset from www.kaggle.com, Today Airbnb is one of the most commonly used service worldwide which is creating a huge amount of data day by day. Every day millions of listing generated. Airbnb is rapidly used in Singapore as it is great tourist place. So we used this data set to perform data operations and analysis.

The listing csv file which we get has the raw data present in it which is not in a structured format. These data has the duplicate and blanks values for some fields. Data redundancy is there similar type of data present for multiples records. We have to transform all these data into normal form, clean it and make it structured. So that we can make use of it.

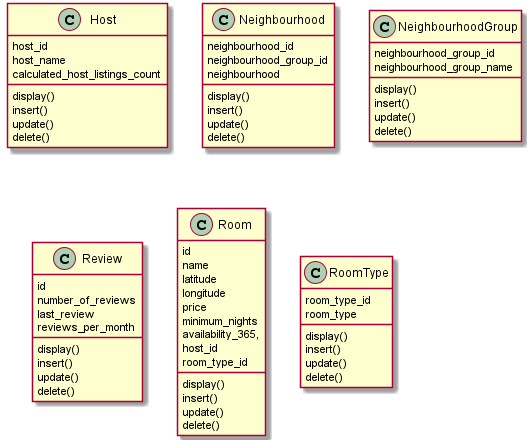
## 1.2 Technical information:

The development of this project Python 3.9.0, Any Python IDE (Pycharm Used), SQLite 3.34.1 for Data Base Python libraries pandas, numpy, and SQLite3 should be added in AN IDE, MS-Excel for input file all other details are provided in the readme file. Panda’s library is used to performance data manipulation in Python. Data frame is used to perform all the operations on the data. Data analysis requires cleaning, merging, and restructuring for which pandas is the best solution which can perform all these operations.

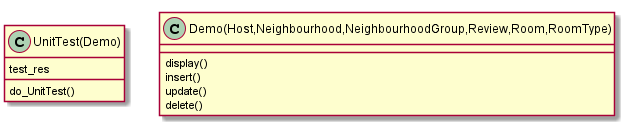
Sqlite3 is used for all the database operations like from create database to the inserting updating deleting record from table through python. SQLite library is imported into the python file to perform all this operations without any operations.

# Class Design:

I have created 6 python classes for Airbnb listing based on their host, review, room type, and Neighbourhood data for implementation.



There are more 2 classes created one is for Unit testing and one Demo class for project presentation and use:



## 2.1 Class information:

1. **Host :** Host class contains all the host attributes like host id , host name, calculated\_host\_listings\_count and functions to perform operations on host data like insert update, delete and display In this class I have extracted the all the host related data, performed data validation and data cleaning and stored it into Host table.
2. **Neighbourhood:** Neighbourhood class contains all the neighbourhood attributes like neighbourhood\_id, neighbourhood\_group\_id, neighbourhood and functions same like host calls we perform operations on neighbourhood data.
3. **NeighbourhoodGroup:** NeighbourhoodGroup class only have 2 class attributes which is used to perform all the operations on the Neighbourhood Group data.
4. **Review:** Review class is design to perform all the operations on the review data which, we got from the listing. It has the class attributes like id, number\_of\_reviews, last\_review, reviews\_per\_month
5. **Room:** In room class we have attributes designed for room related data: id, name, latitude, longitude, price, minimum\_nights, availability\_365, host\_id, room\_type\_id and all the operation functions which all other classes have display insert update delete.
6. **RoomType:** This class contains two attributes room id and room type like this class and all the functions which other classes have.
7. **UnitTest:** Unit testing class contains functions to test display, insert, update, delete operations and test\_res class attribute to check the test case passed or failed.
8. **Demo:** Demo class import all the other classes Host, Neighbourhood, NeighbourhoodGroup, Review, Room, RoomType and UnitTest. Objects of all classes created in this class and call the respective class functions as per the user’s choice. This class file is the entry point for all operations.Menu options are provided in this class.

# Database Design:

Airbnb in Singapore listing data csv is used to import the data, and design the database. Given dataset is having 7907 records. All the data given in a single file. Data is not in structured form. There are 16 columns given in the file some of columns have no values, also some redundant data present.

Normalization is used to convert this unstructured data into multiple 6 tables. Also removed the redundant data. And convert it into 1NF that first normal form where no redundant value is allowed for each table cell. And each record should be unique. Created a relationship between all 6 tables so that it can satisfies 1NF rules. And our data is clean and structured format.

Now to create a relationship between all tables we have to use the key. A KEY is used to identify a record in a table uniquely. Also Foreign Key is used to references the primary key of another Table. Given data is divided into below 6 tables and applied both primary and foreign key on it so that it can be in 1st normal form:

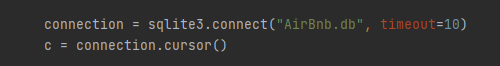
|  |  |  |
| --- | --- | --- |
| **Host Table : host** | | |
| **Column** | **DataType** | **Key** |
| host\_id | INTEGER | Primary Key |
| host\_name | TEXT |  |
| calculated\_host\_listings\_count | INTEGER |  |
|  |  |  |
| **Neighbourhood Table :nbh\_data** | | |
| **Column** | **DataType** | **Key** |
| neighbourhood\_id | INTEGER | Primary Key |
| neighbourhood\_group\_id | INTEGER | Foreign Key |
| neighbourhood | TEXT |  |
|  |  |  |
|  |  |  |
| **NeighbourhoodGroup Table :nbh\_grp** | | |
| **Column** | **DataType** | **Key** |
| neighbourhood\_group\_id | INTEGER | Primary Key |
| neighbourhood\_group\_name | TEXT |  |
|  |  |  |
|  |  |  |  |
| **Review Table :review** | | |
| **Column** | **DataType** | **Key** |
| id | INTEGER | Foreign Key |
| number\_of\_reviews | INTEGER |  |
| last\_review | TEXT |  |
| reviews\_per\_month | REAL |  |
|  |  |  |
|  |  |  |
| **Room Table: room** | | |
| **Column** | **DataType** | **Key** |
| id | INTEGER | Primary Key |
| name | TEXT |  |
| latitude | REAL |  |
| longitude | REAL |  |
| price | INTEGER |  |
| minimum\_nights | INTEGER |  |
| availability\_365, | INTEGER |  |
| host\_id | INTEGER | Foreign Key |
| room\_type\_id | INTEGER | Foreign Key |
|  |  |  |
| **RoomType Table :roomtype** | | |
| **Column** | **DataType** | **Key** |
| room\_type\_id | INTEGER | Primary Key |
| room\_type | TEXT |  |

# SQL Operation Design:

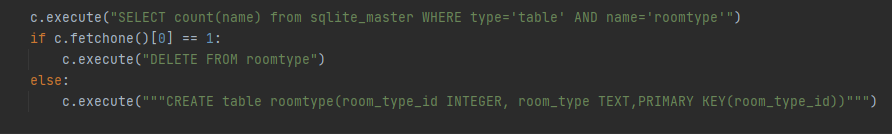
Sqlite3 is used to create a data base. First import the Sqlite3 as below:



Created a connection and cursor to perform operations on data:

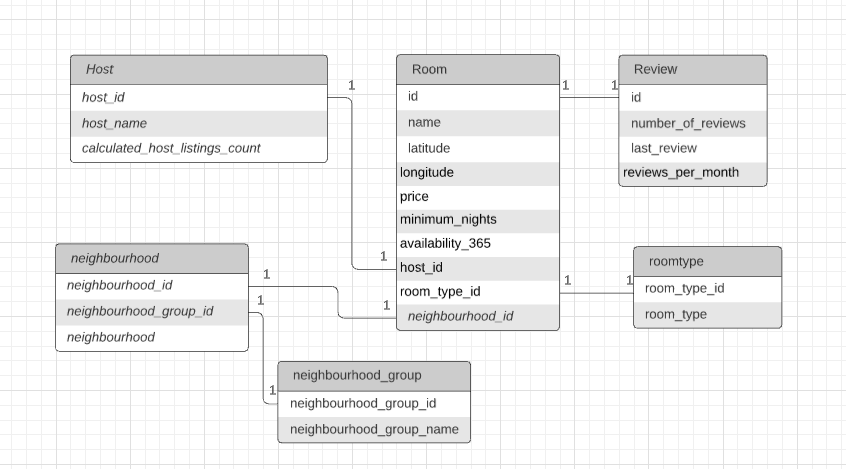


First check if the table with same name is created in work space if not created then create a new table using below logic:



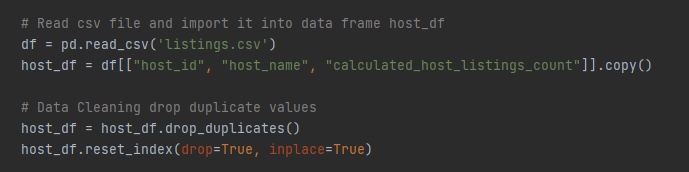
In above example created a table for roomtype, same logic is used for all other 6 table. All other 5 tables are created as mentioned in the above tables.

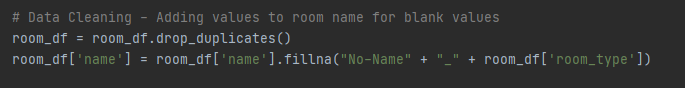
Data Base Design:



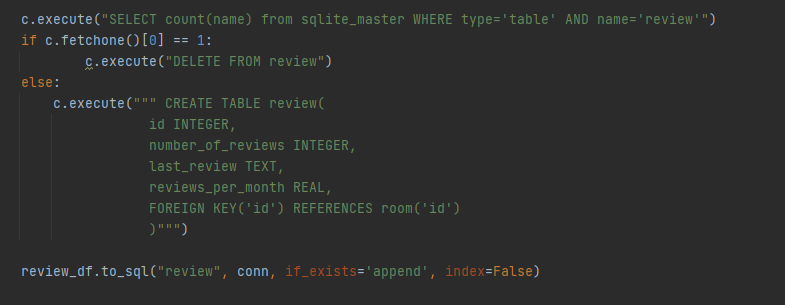
# Essential Key Code Snippets:

1. **Data cleaning:** By using below code snippets required fields from csv file are copied to the host\_df data frame and then drop the duplicate values from the data frame same code is used for all other tables imported the required data from the listing csv file according to the table columns and create a data frame then clean that data using below code:

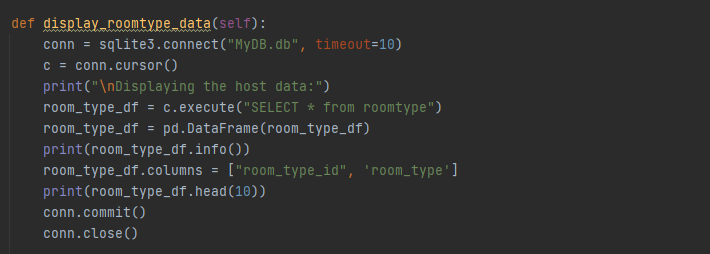


In room data observed some fields are blank for room name column. Fill that values with some data using below code snippet:

1. **Create DB Connection and Table:** Create a Sqlite3 connection and cursor c then create a table for respective class given above in SQL operation and insert that data from data frame to table :

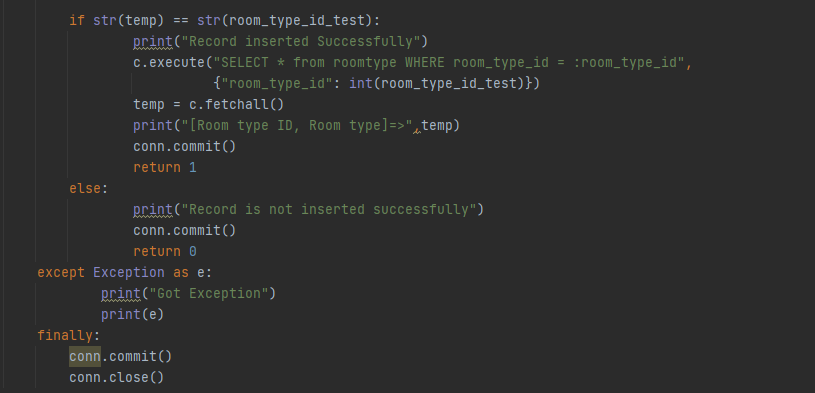


1. **Display the Records:** One option is given to check the data in demo so that we can get the data by calling to the display function. This type of function is written for all 6 classes. In below example display function is given for the roomtype class:



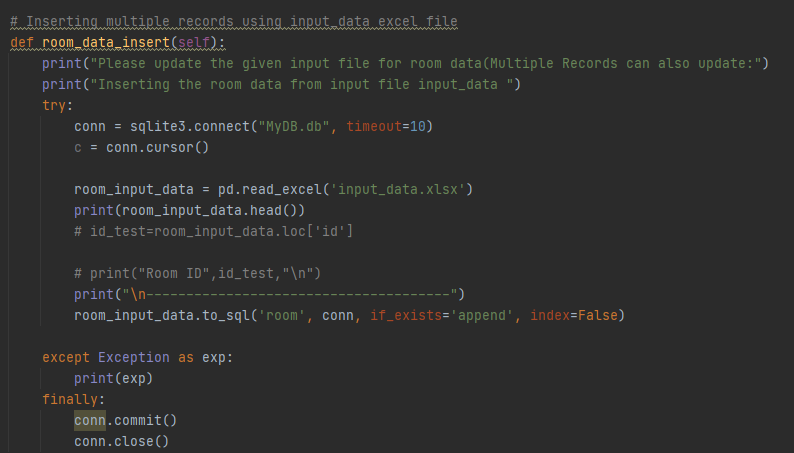
1. **Insert Data :** To insert the new data in the table for all 6 options host, room, review, room, neighbourhood data below function is used:

****

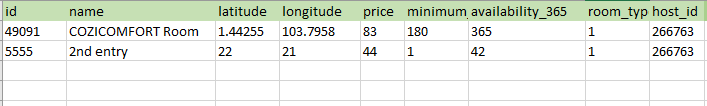
****

Same logic used for all other tables as well.

1. **Insert Room Data Using Excel file:** For room data insertion we can insert multiple records at a time using input\_data file given in project folder. Just we have to update the file for new data. We can apply the data validation in the excel file so that only valid data can be inserted into the table. Below code is used to update the room data. This feature is given only for room data:



**Input Excel File: input\_data.xlsx:**



1. **Update Existing Data:** To update the data we need some unique constraint for any table like room id, host id then for that record we can update the data as per the user input.

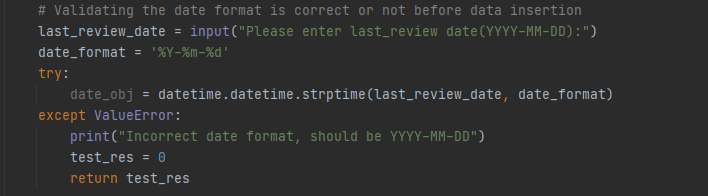
Here we are also checking the key constraints and data validation so we have to enter the valid data while updating any record. Below code is given for the neighbourhood data:



After this we will check by using select weather the data is updated correctly in the table or not by using below code:



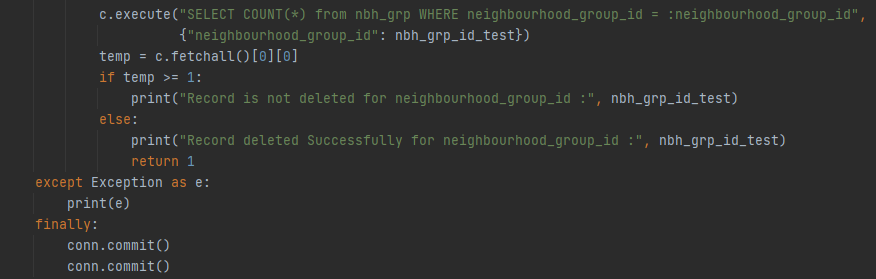
1. **Data Validation:**  While inserting new data into table we are checking whether data is already exist in the table by using primary key constraints and also validating input data using data type given while creating table. Also we are validating the date input entry whether it is in correct format like: YYYY-MM-DD or not. If not it will give an error message. We can see the unit cases for this. Below is code given to check the date in update and insert option here import datetime to validate date :

****

1. **Delete a Record:** To delete record we need some unique key for which we need to delete the record and It should be present in the table so first we will check the given input for which records need to delete is available in the table or not. If it is not available then throw an error message. Otherwise delete the record for valid id:

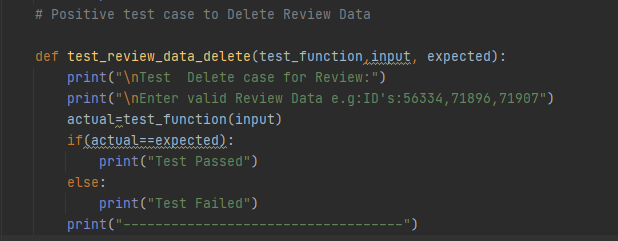
****

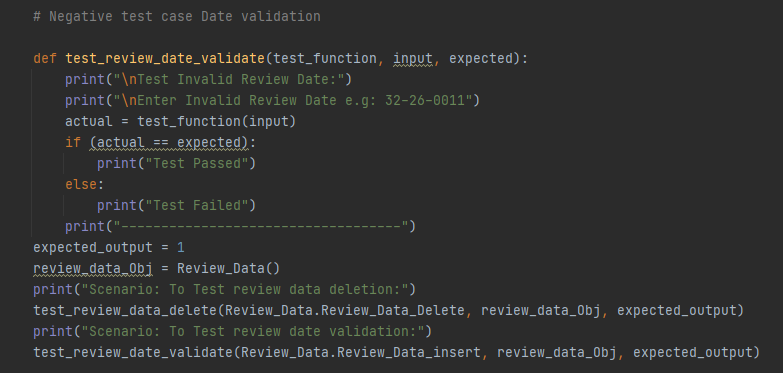
After deleting the record we have to cross check if the record is deleted successfully or not by using select query for id which we recently deleted if we didn’t get any output means record is deleted successfully. Below is the code continues to check if it deleted or not:



1. **Unit Test:** For unit testing of each functionality by passing the class object for which we have to test, the function which we have to test and if the function works successfully then we will get actual value as 1 then given functions works successfully. If we get other than 1 the function is not worked as expected and test case failed.

In below code we are checking if review data can be deleted successfully or not. We passed review delete function, review object and expected result as 1 to test\_review\_Data\_delete function and checked:



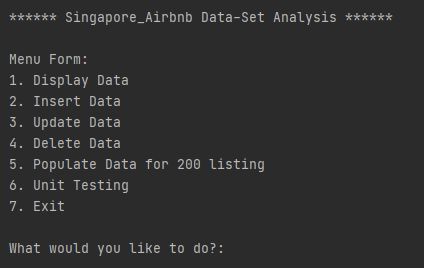


Also in negative test case we are checking date format review insert function to test function we can also do this using review update function.

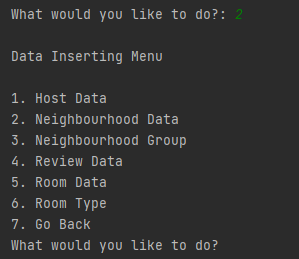
1. **Demo:** This class is the entry point for our project from where we can perform all the operations like data display, insert, update, delete. Also we can populate data on given condition in the assessment that populate all the data/listing that has 200 or above total reviews in option 5.

While loop and if elif else is used for Demo main menu and submenus for all options:

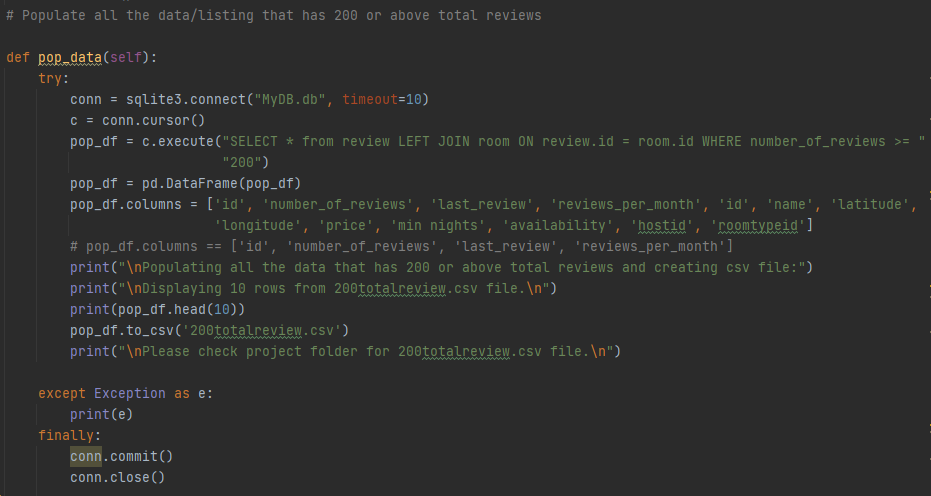
Main Menu as given below:



And Submenu for Insert, Update, and Delete is like:

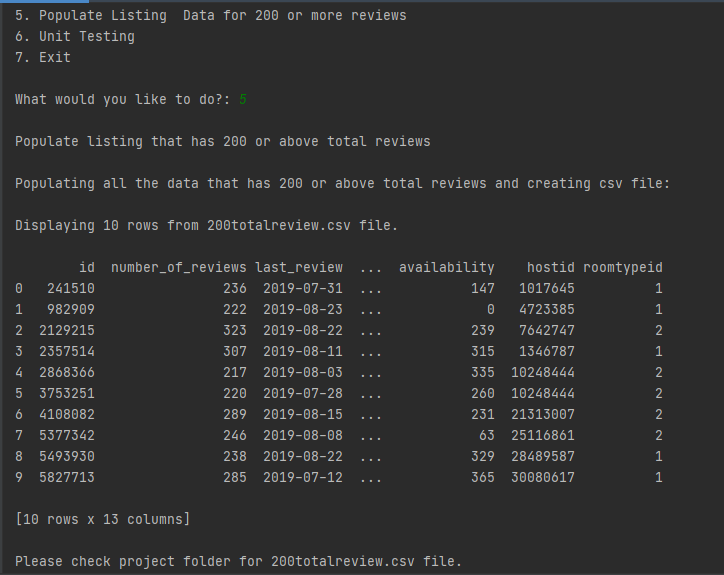


1. **Populate Data for 200 or more review in listing:**  We can use total number of review condition and filter the data from review table then by using inner join combine two join and populate the required data also we are creating a csv file to store a populated data. By using below code we can populate data for 200 or more review and store it into csv file:

****

**Result for Populate Data for 200 or more review in listing option 5 from main menu:**

Also check the project folder for created 200totalreview.csv file

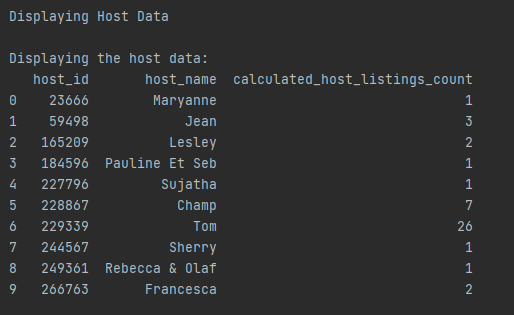
****

# Project Demo:

In this section we can see the execution of each operation like insert update delete and display for any random table. However same operations are possible for all other table. Only for room data we are using the excel file for data entry because there are multiple fields present in room data. And one advantage is we can insert multiple room records at time using file input which is shown earlier. Except of that all other operations are almost similar for all data just the input is changed according to the table.

The below snippets can show the execution for each operation we are performing for all data. For demo selected random tables below:

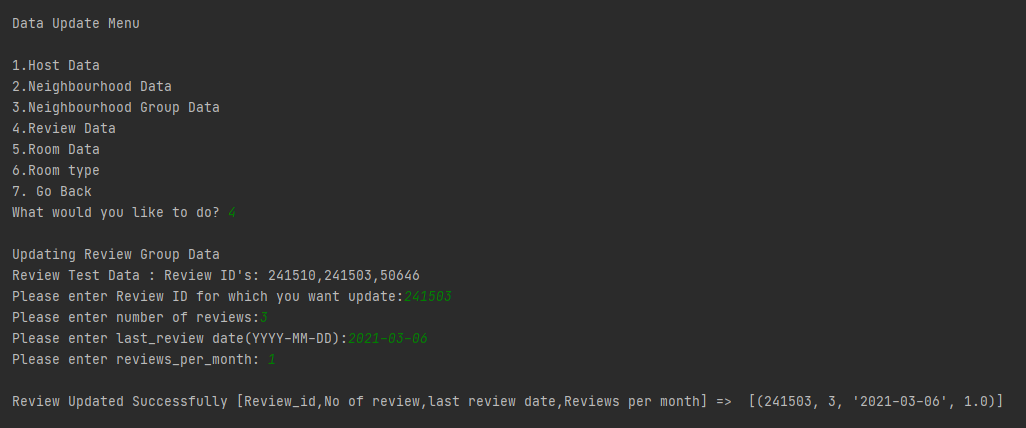
1. **Display Data : (e.g. Displaying Host Data)**



1. **Insert Data: (e.g. Inserting Neighbourhood Data)**

****

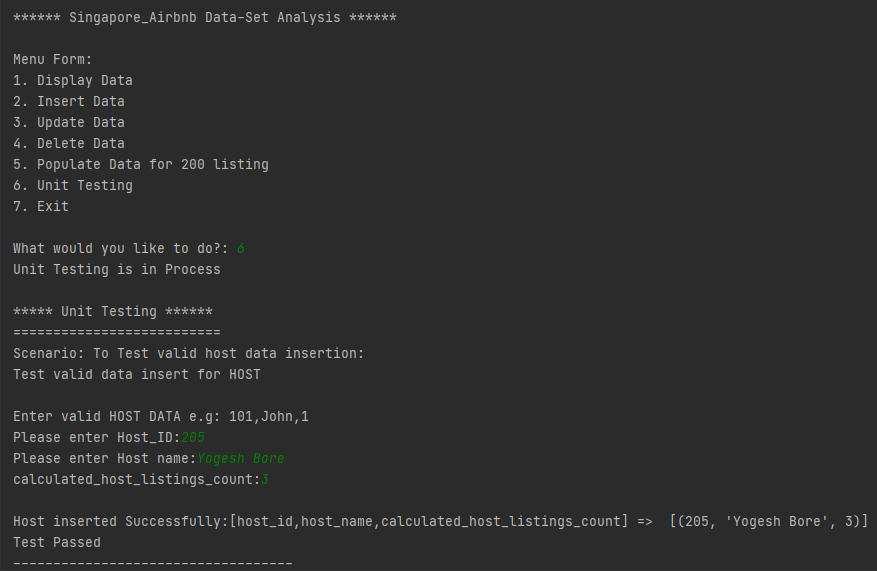
1. **Update Data: (e.g. Updating Review Data)**

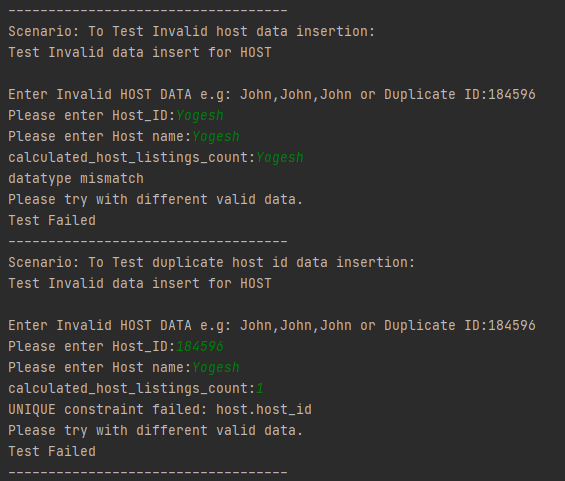
****

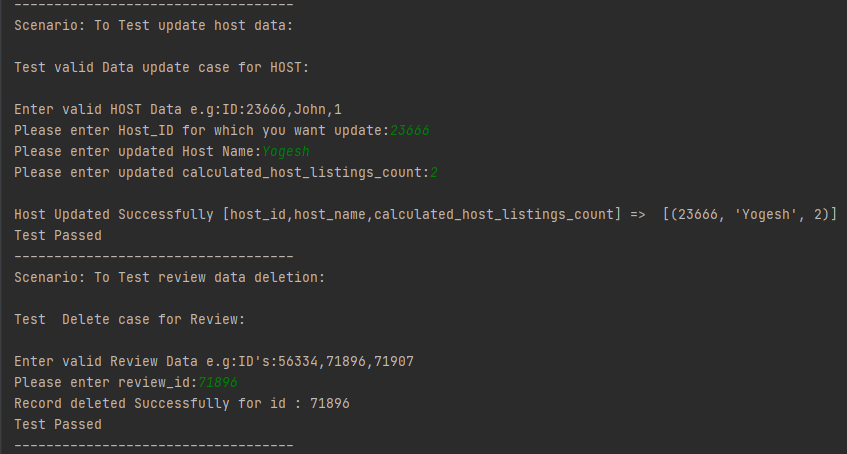
1. **Delete Data: (e.g. Deleting Room Data)**

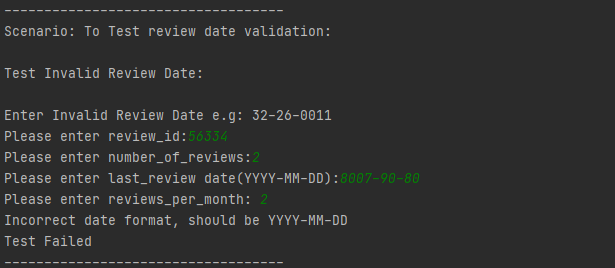
****

1. **Populate Data for 200 or more review in listing:**  Result is given in above section for this
2. **Unit Testing:** In next section unit testing demo is given separately.
3. **Exit:** To stop the execution of program and End.
4. Unit Testing Cases:

For unit testing created a separate class to perform some random scenario testing below results we will get after performing unit testing in option 6 from menu option







# Summary:

In this project Singapore Airbnb listing data Analysis we transformed unstructured data into a structured one and perform operations on the data like display, insert, update and delete. To perform all this operations we used data frames from the python panda’s library. And On the given condition we populated data and created a new csv file to insert populated data. Also for room data insertion we used excel file. So that we can insert multiple records at time. Data validation done while inserting and updating data using some panda’s inbuilt functions. We successfully converted raw data into structured format so that we can make use of it and take decision.

# References:

1. Python: https://www.python.org/downloads/

2. SQLite3: https://www.sqlite.org/releaselog/3\_34\_1.html

3. Python Numpy Library: https://numpy.org/install/

4. Python Pandas Library: https://pandas.pydata.org/pandas-docs/

5. Airbnb listing dataset: https://www.kaggle.com/jojoker/singapore-irbnb

# Appendix: Source Code

## Appendix A: Host Code

import pandas as pd  
import sqlite3  
  
  
class Host\_Data:  
 # Read csv file and import it into data frame host\_df  
 df = pd.read\_csv('listings.csv')  
 host\_df = df[["host\_id", "host\_name", "calculated\_host\_listings\_count"]].copy()  
  
 # Data Cleaning drop duplicate values  
 host\_df = host\_df.drop\_duplicates()  
 host\_df.reset\_index(drop=True, inplace=True)  
  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 c.execute("SELECT count(name) from sqlite\_master WHERE type='table' AND name='host'")  
 if c.fetchone()[0] == 1:  
 c.execute("DELETE FROM host")  
 # c.execute("DROP TABLE host")  
  
 else:  
 c.execute('''CREATE TABLE host(  
 host\_id INTEGER PRIMARY KEY,  
 host\_name TEXT,  
 calculated\_host\_listings\_count INTEGER)''')  
  
 host\_df.to\_sql('host', conn, if\_exists='append', index=False)  
  
 conn.commit()  
 conn.close()  
  
 # c.execute("SELECT \* FROM host")  
 # print(c.fetchall())  
 # Insert Host data  
  
 @staticmethod  
 def host\_insert():  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 try:  
  
 test\_res = 1  
 host\_id\_test = input("Please enter Host\_ID:")  
 host\_name\_test = input("Please enter Host name:")  
 list\_count\_test = input("calculated\_host\_listings\_count:")  
  
 c.execute("INSERT INTO host VALUES(:host\_id,:host\_name,:calculated\_host\_listings\_count)",  
 {"host\_id": host\_id\_test, "host\_name": host\_name\_test,  
 "calculated\_host\_listings\_count": list\_count\_test})  
 # self.conn.commit()  
 c.execute("SELECT host\_id from host WHERE host\_id = :host\_id", {"host\_id": host\_id\_test})  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 # print(type(host\_id\_test))  
  
 if str(temp) == host\_id\_test:  
 c.execute("SELECT \* from host WHERE host\_id = :host\_id", {"host\_id": host\_id\_test})  
 inserted = c.fetchall()  
 print("\nHost inserted Successfully:[host\_id,host\_name,calculated\_host\_listings\_count] => ", inserted)  
 conn.commit()  
 return test\_res  
 else:  
 print("Host Record is not inserted successfully")  
 # self.conn.commit()  
 test\_res = 0  
 return test\_res  
  
 except Exception as e:  
 print(e)  
 print("Please try with different valid data.")  
 finally:  
 conn.commit()  
 conn.close()  
  
 # update host  
 @staticmethod  
 def host\_update():  
  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 try:  
 test\_res = 1  
 host\_id\_test = input("Please enter Host\_ID for which you want update:")  
 host\_id\_test = int(host\_id\_test)  
  
 host\_name\_test = input("Please enter updated Host Name:")  
 host\_name\_test = str(host\_name\_test)  
  
 list\_count\_test = input("Please enter updated calculated\_host\_listings\_count:")  
 list\_count\_test = float(list\_count\_test)  
  
 """self.c.execute(  
 'UPDATE host SET host\_name=:host\_name WHERE host\_id=:host\_id',  
 {'host\_name': host\_name\_test,'host\_id': host\_id\_test })"""  
  
 c.execute(  
 'UPDATE host SET host\_name=:host\_name,calculated\_host\_listings\_count=:calculated\_host\_listings\_count '  
 'WHERE host\_id=:host\_id',  
 {'host\_name': host\_name\_test, 'host\_id': host\_id\_test,  
 'calculated\_host\_listings\_count': list\_count\_test})  
  
 # self.conn.commit()  
 c.execute("SELECT host\_id from host WHERE host\_id = :host\_id", {"host\_id": host\_id\_test})  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 # print(type(host\_id\_test))  
  
 if temp == host\_id\_test:  
 c.execute("SELECT \* from host WHERE host\_id = :host\_id", {"host\_id": host\_id\_test})  
 updated = c.fetchall()  
 print("\nHost Updated Successfully [host\_id,host\_name,calculated\_host\_listings\_count] => ", updated)  
 # self.conn.commit()  
  
 return test\_res  
 else:  
 print("Record is not updated successfully")  
 test\_res = 0  
 return test\_res  
  
 except Exception as e:  
 print(e)  
 print("\nPlease try with different data")  
 finally:  
 conn.commit()  
 conn.close()  
  
 @staticmethod  
 def display\_host\_data():  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 print("\nDisplaying the host data:")  
 hostdata = c.execute("SELECT \* from host")  
 hostdata = pd.DataFrame(hostdata)  
 hostdata.columns = ["host\_id", 'host\_name', 'calculated\_host\_listings\_count']  
 print(hostdata.head(10))  
 conn.commit()  
 conn.close()  
  
 # Delete host data  
  
 @staticmethod  
 def host\_data\_delete():  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 try:  
 host\_id\_test = input("Please enter Host\_id:")  
 c.execute("SELECT COUNT(\*) from host WHERE host\_id = :host\_id", {"host\_id": host\_id\_test})  
 rowcount = c.fetchall()[0][0]  
 # print(rowcount)  
 if rowcount <= 0:  
 print("No records founds for host\_id:", host\_id\_test)  
 else:  
  
 c.execute("DELETE from host WHERE host\_id = :host\_id", {"host\_id": host\_id\_test})  
  
 c.execute("SELECT COUNT(\*) from host WHERE host\_id = :host\_id", {"host\_id": host\_id\_test})  
  
 conn.commit()  
  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 if temp >= 1:  
 print("Record is not deleted for Host id :", host\_id\_test)  
  
 else:  
 print("Record deleted Successfully for host id :", host\_id\_test)  
  
 return 1  
  
 except Exception as e:  
 print(e)  
 finally:  
 conn.commit()  
 conn.commit()  
  
  
# Host\_Obj = Host\_Data()  
# Host\_Obj.host\_data\_delete()  
  
# Host\_Obj = host\_Data()  
  
# Host\_Obj.Host\_Data\_Delete()  
"""  
Host\_Obj.host\_insert()  
Host\_Obj.display\_host\_data()  
Host\_Obj.host\_Update()  
Host\_Obj.display\_host\_data()  
#Host\_Obj.host\_insert()  
  
Host\_Obj.host\_Update() # Test data for valid scenario:23666  
  
  
Host\_Obj.display\_host\_data()"""

## Appendix B: Neighbourhood Data Code

import sqlite3  
import pandas as pd  
  
  
class Neighbourhood\_Data:  
 # Read csv file and save it into data frame neighbourhood\_df  
 df = pd.read\_csv('listings.csv')  
 neighbourhood\_df = df[["neighbourhood", "neighbourhood\_group"]].copy()  
  
 # Data cleaning  
 neighbourhood\_df = neighbourhood\_df.drop\_duplicates()  
 neighbourhood\_df.reset\_index(drop=True, inplace=True)  
  
 s1 = pd.Series(range(1, len(neighbourhood\_df.index) + 1))  
  
 neighbourhood\_df['neighbourhood\_id'] = s1  
  
 def makeid(data):  
 if data['neighbourhood\_group'] == 'North Region':  
 return 1  
 elif data['neighbourhood\_group'] == 'Central Region':  
 return 2  
  
 elif data['neighbourhood\_group'] == 'East Region':  
 return 3  
 elif data['neighbourhood\_group'] == 'West Region':  
  
 return 4  
 else:  
 return 5  
  
 neighbourhood\_df['neighbourhood\_group\_id'] = neighbourhood\_df.apply(makeid, axis=1)  
  
 # print(neighbourhood\_df)  
 del neighbourhood\_df['neighbourhood\_group']  
 # print(neighbourhood\_df)  
  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 c.execute("SELECT count(name) from sqlite\_master WHERE type='table' AND name='nbh\_data'")  
 if c.fetchone()[0] == 1:  
 # c.execute("DROP TABLE nbh\_data")  
 c.execute("DELETE FROM nbh\_data")  
 else:  
 c.execute("""CREATE TABLE nbh\_data(  
 neighbourhood\_id INTEGER PRIMARY KEY,  
 neighbourhood\_group\_id INTEGER,  
 neighbourhood TEXT,  
 FOREIGN KEY("neighbourhood\_group\_id") REFERENCES nbh\_grp("neighbourhood\_group\_id")   
 )""")  
  
 neighbourhood\_df.to\_sql("nbh\_data", conn, if\_exists='append', index=False)  
 c.execute("SELECT \* FROM nbh\_data")  
 # print(c.fetchall())  
  
 conn.commit()  
 conn.close()  
  
 # Insert New neighbourhood record  
  
 def nbh\_insert(self):  
  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 nbh\_id\_test = input("Please enter neighbourhood\_id:")  
 nbh\_grp\_id\_test = input("Please enter neighbourhood\_group\_id:")  
 nbh\_test = input("Please enter neighbourhood:")  
  
 c.execute("INSERT INTO nbh\_data VALUES(:neighbourhood\_id,:neighbourhood\_group\_id,:neighbourhood)",  
 {"neighbourhood\_id": int(nbh\_id\_test), "neighbourhood\_group\_id": int(nbh\_grp\_id\_test),  
 "neighbourhood": nbh\_test})  
  
 c.execute("SELECT neighbourhood\_id from nbh\_data WHERE neighbourhood\_id = :neighbourhood\_id",  
 {"neighbourhood\_id": int(nbh\_id\_test)})  
 temp = c.fetchall()[0][0]  
  
 if str(temp) == nbh\_id\_test:  
 print("Record inserted Inserted Successfully")  
  
 c.execute("SELECT \* from nbh\_data WHERE neighbourhood\_id = :neighbourhood\_id",  
 {"neighbourhood\_id": int(nbh\_id\_test)})  
 temp = c.fetchall()  
 print("\nNeighbourhood inserted Successfully:[neighbourhood\_id,neighbourhood\_grp\_id,neighbourhood] =>",  
 temp)  
 conn.commit()  
 return 1  
 else:  
 print("NeighbourhoodRecord is not inserted successfully")  
 return 0  
  
 except Exception as e:  
 print("Got Exception as ", e)  
 print(str(e))  
 finally:  
 conn.commit()  
 conn.close()  
  
 # Updating the neighbourhood data:  
 def nbh\_data\_Update(self):  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 test\_res = 1  
 nbh\_id\_test = input("Please enter neighbourhood\_id for which you want update:")  
 nbh\_id\_test = int(nbh\_id\_test)  
  
 neighbourhood = input("Please enter neighbourhood Name:")  
 neighbourhood = str(neighbourhood)  
  
 c.execute(  
 'UPDATE nbh\_data SET neighbourhood=:neighbourhood WHERE neighbourhood\_id=:neighbourhood\_id',  
 {'neighbourhood': neighbourhood, 'neighbourhood\_id': nbh\_id\_test})  
  
 c.execute("SELECT neighbourhood\_id from nbh\_data WHERE neighbourhood\_id = :neighbourhood\_id",  
 {"neighbourhood\_id": nbh\_id\_test})  
 temp = c.fetchall()[0][0]  
 if temp == nbh\_id\_test:  
 c.execute("SELECT \* from nbh\_data WHERE neighbourhood\_id = :neighbourhood\_id", {"neighbourhood\_id":  
 nbh\_id\_test})  
 updated = c.fetchall()  
 print("\nNeighbourhood Updated Successfully [neighbourhood\_id, neighbourhood\_group\_id,"  
 "neighbourhood] "  
 " => ", updated)  
 return test\_res  
 else:  
 print("Record is not updated successfully")  
 test\_res = 0  
 return test\_res  
 except Exception as e:  
 print(e)  
 print("\nPlease try with different data")  
 finally:  
 conn.commit()  
 conn.close()  
  
 def display\_nbh\_data(self):  
 conn = sqlite3.connect("MyDB.db")  
 c = conn.cursor()  
 neighbourhood\_df = c.execute("SELECT \* from nbh\_data")  
 neighbourhood\_df = pd.DataFrame(neighbourhood\_df)  
 neighbourhood\_df.columns = ["neighbourhood\_id", 'neighbourhood\_group\_id', 'neighbourhood']  
 print(neighbourhood\_df.head(10))  
 conn.commit()  
 conn.close()  
  
 def nbh\_Data\_Delete(self):  
  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 nbh\_id\_test = input("Please enter neighbourhood\_id:")  
 c.execute("SELECT COUNT(\*) from nbh\_data WHERE neighbourhood\_id = :neighbourhood\_id",  
 {"neighbourhood\_id": nbh\_id\_test})  
 rowcount = c.fetchall()[0][0]  
 # print(rowcount)  
 if rowcount <= 0:  
 print("No records founds for neighbourhood\_id:", nbh\_id\_test)  
 else:  
  
 c.execute("DELETE from nbh\_data WHERE neighbourhood\_id = :neighbourhood\_id",  
 {"neighbourhood\_id": nbh\_id\_test})  
  
 c.execute("SELECT COUNT(\*) from nbh\_data WHERE neighbourhood\_id = :neighbourhood\_id",  
 {"neighbourhood\_id": nbh\_id\_test})  
  
 conn.commit()  
  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 if temp >= 1:  
 print("Record is not deleted for neighbourhood\_id :", nbh\_id\_test)  
  
 else:  
 print("Record deleted Successfully for neighbourhood\_id :", nbh\_id\_test)  
  
 return 1  
  
 except Exception as e:  
 print(e)  
 finally:  
 conn.commit()  
 conn.commit()  
  
# neighbourhood\_data\_Obj = neighbourhood\_data()  
  
# neighbourhood\_data\_Obj.display\_nbh\_data()  
# neighbourhood\_data\_Obj.nbh\_data\_Update()  
# print(neighbourhood\_data\_Obj.nbh\_insert())  
# neighbourhood\_data\_Obj.nbh\_Data\_Delete()

## Appendix C: Neighbourhood Group Data Code

import sqlite3  
import pandas as pd  
  
  
class Neighbourhood\_Group:  
 df = pd.read\_csv('listings.csv')  
 nbh\_grp\_df\_old = df[["neighbourhood\_group"]].copy()  
 nbh\_grp\_df = nbh\_grp\_df\_old["neighbourhood\_group"].unique()  
  
 s1 = pd.Series(nbh\_grp\_df)  
 len = len(s1.index)  
 # print(len)  
  
 s2 = pd.Series(range(1,len+5))  
 # print(s2)  
  
 nbh\_grp\_df = pd.DataFrame(nbh\_grp\_df)  
 nbh\_grp\_df.columns = ['neighbourhood\_group\_name']  
 nbh\_grp\_df['neighbourhood\_group\_id'] = s2  
  
 # print(nbh\_grp\_df)  
 # Data Cleaning  
  
 conn = sqlite3.connect('MyDB.db',timeout=10)  
 c = conn.cursor()  
  
 c.execute("SELECT count(name) from sqlite\_master WHERE type='table' AND name='nbh\_grp'")  
 if c.fetchone()[0] == 1:  
 # c.execute("DROP TABLE nbh\_grp")  
 c.execute("DELETE FROM nbh\_grp")  
 else:  
 c.execute(""" CREATE TABLE nbh\_grp(  
 neighbourhood\_group\_id INTEGER PRIMARY KEY,  
 neighbourhood\_group\_name TEXT  
 )""")  
  
 nbh\_grp\_df.to\_sql("nbh\_grp", conn, if\_exists='append', index=False)  
 c.execute("SELECT \* FROM nbh\_grp")  
 # print(c.fetchall())  
# Insert the data for neighbourhood group  
  
 conn.commit()  
 conn.close()  
  
 def nbh\_grp\_inser(self):  
  
 try:  
 conn = sqlite3.connect('MyDB.db', timeout=10)  
 c = conn.cursor()  
  
 nbh\_grp\_id\_test = input("Please enter Neighbourhood\_group\_ID:")  
 nbh\_grp\_name\_test=input("Please enter Neighbourhood\_group\_Name:")  
  
  
 c.execute("INSERT INTO nbh\_grp VALUES(:neighbourhood\_group\_id,:neighbourhood\_group\_name)",  
 {"neighbourhood\_group\_id":nbh\_grp\_id\_test,"neighbourhood\_group\_name":nbh\_grp\_name\_test})  
  
  
 c.execute("SELECT neighbourhood\_group\_id from nbh\_grp WHERE neighbourhood\_group\_id = "  
 ":neighbourhood\_group\_id",{"neighbourhood\_group\_id":nbh\_grp\_id\_test})  
 temp = c.fetchall()[0][0]  
  
 if str(temp) == nbh\_grp\_id\_test:  
 print("Neighbourhood\_group record inserted Successfully")  
 c.execute(  
 "SELECT \* from nbh\_grp WHERE neighbourhood\_group\_id = :neighbourhood\_group\_id",  
 {"neighbourhood\_group\_id": nbh\_grp\_id\_test})  
 temp = c.fetchall()  
 print("[Neighbourhood\_group\_id , neighbourhood\_group\_name]==>", temp)  
 conn.commit()  
 return 1  
 else:  
 print("Neighbourhood\_group record is not inserted successfully")  
 return 0  
  
  
 except Exception as e:  
 print(e)  
 finally:  
 #print("Closing DB Connection")  
 conn.commit()  
 c.close()  
 conn.close()  
  
 # update nbh group  
  
 def nbh\_group\_update(self):  
  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 test\_res = 1  
 nbh\_id\_test = input("Please enter neighbourhood\_group\_id for which you want update:")  
 nbh\_id\_test = int(nbh\_id\_test)  
  
 nbh\_grp\_name\_test = input("Please enter updated neighbourhood\_group\_name:")  
 nbh\_grp\_name\_test = str(nbh\_grp\_name\_test)  
  
 c.execute(  
 'UPDATE nbh\_grp SET neighbourhood\_group\_name=:neighbourhood\_group\_name WHERE '  
 'neighbourhood\_group\_id=:neighbourhood\_group\_id',  
 {'neighbourhood\_group\_name': nbh\_grp\_name\_test, 'neighbourhood\_group\_id': nbh\_id\_test})  
  
 # self.conn.commit()  
 c.execute("SELECT neighbourhood\_group\_id from nbh\_grp WHERE neighbourhood\_group\_id = "  
 ":neighbourhood\_group\_id", {"neighbourhood\_group\_id": nbh\_id\_test})  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 # print(type(host\_id\_test))  
  
 if temp == nbh\_id\_test:  
 c.execute("SELECT \* from nbh\_grp WHERE neighbourhood\_group\_id = :neighbourhood\_group\_id",  
 {"neighbourhood\_group\_id": nbh\_id\_test})  
 updated = c.fetchall()  
 print("\nNeighbourhood Updated Successfully [Neighbourhood\_group\_id,Neighbourhood\_group\_name] => ", updated)  
 # self.conn.commit()  
  
 return test\_res  
 else:  
 print("Record is not updated successfully")  
 test\_res = 0  
 return test\_res  
  
 except Exception as e:  
 print(e)  
 print("\nPlease try with different data")  
 finally:  
 conn.commit()  
 conn.close()  
  
  
 # Deleting the neighbourhood group data  
 def nbh\_grp\_Delete(self):  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 nbh\_grp\_id\_test = input("Please enter neighbourhood\_group\_id:")  
 c.execute("SELECT COUNT(\*) from nbh\_grp WHERE neighbourhood\_group\_id = :neighbourhood\_group\_id",  
 {"neighbourhood\_group\_id": nbh\_grp\_id\_test})  
 rowcount = c.fetchall()[0][0]  
 # print(rowcount)  
 if rowcount <= 0:  
 print("No records founds for neighbourhood\_group\_id:", nbh\_grp\_id\_test)  
 else:  
 c.execute("DELETE from nbh\_grp WHERE neighbourhood\_group\_id = :neighbourhood\_group\_id",  
 {"neighbourhood\_group\_id": nbh\_grp\_id\_test})  
  
 c.execute("SELECT COUNT(\*) from nbh\_grp WHERE neighbourhood\_group\_id = :neighbourhood\_group\_id",  
 {"neighbourhood\_group\_id": nbh\_grp\_id\_test})  
 temp = c.fetchall()[0][0]  
 if temp >= 1:  
 print("Record is not deleted for neighbourhood\_group\_id :", nbh\_grp\_id\_test)  
 else:  
 print("Record deleted Successfully for neighbourhood\_group\_id :", nbh\_grp\_id\_test)  
 return 1  
 except Exception as e:  
 print(e)  
 finally:  
 conn.commit()  
 conn.commit()  
  
 def display\_nbh\_Grp\_data(self):  
 conn = sqlite3.connect('MyDB.db', timeout=10)  
 c = conn.cursor()  
 nbh\_grp\_df = c.execute("SELECT \* from nbh\_grp")  
 nbh\_grp\_df = pd.DataFrame(nbh\_grp\_df)  
 nbh\_grp\_df.columns = ['neighbourhood\_group\_id', 'neighbourhood\_group\_name']  
 # print(nbh\_grp\_df.info())  
 print(nbh\_grp\_df.head(10))  
 # print(hostdata.columns)  
 conn.commit()  
 c.close()  
 conn.close()  
  
# nbh\_grp\_Obj = neighbourhood\_group()  
# nbh\_grp\_Obj.nbh\_grp\_Delete()  
# nbh\_grp\_Obj.nbh\_grp\_inser()  
# nbh\_grp\_Obj.display\_nbh\_Grp\_data()  
# nbh\_grp\_Obj.nbh\_group\_update()

## Appendix D: Review Data Code

import sqlite3  
import pandas as pd  
import datetime  
  
  
class Review\_Data:  
 df = pd.read\_csv('listings.csv')  
 review\_df = df[["id", "number\_of\_reviews", "last\_review", "reviews\_per\_month"]].copy()  
  
 review\_df = review\_df.dropna()  
 review\_df = review\_df.drop\_duplicates()  
  
 # print(review\_df)  
  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 c.execute("SELECT count(name) from sqlite\_master WHERE type='table' AND name='review'")  
 if c.fetchone()[0] == 1:  
 c.execute("DELETE FROM review")  
 else:  
 c.execute(""" CREATE TABLE review(  
 id INTEGER,  
 number\_of\_reviews INTEGER,  
 last\_review TEXT,  
 reviews\_per\_month REAL,  
 FOREIGN KEY('id') REFERENCES room('id')  
 )""")  
  
 review\_df.to\_sql("review", conn, if\_exists='append', index=False)  
  
 c.execute("SELECT \* FROM review")  
 # print(c.fetchall())  
  
 conn.commit()  
 conn.close()  
  
 # Insert New review record  
  
 def Review\_Data\_insert(self):  
  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 test\_res = 1  
  
 review\_id\_test = input("Please enter review\_id:")  
 review\_id\_test = int(review\_id\_test)  
 number\_of\_reviews\_test = input("Please enter number\_of\_reviews:")  
 number\_of\_reviews\_test = int(number\_of\_reviews\_test)  
 last\_review\_test = input("Please enter last\_review date(YYYY-MM-DD):")  
 reviews\_per\_month\_test = input("Please enter reviews\_per\_month: ")  
 reviews\_per\_month\_test = float(reviews\_per\_month\_test)  
  
 date\_format = '%Y-%m-%d'  
 try:  
 date\_obj = datetime.datetime.strptime(last\_review\_test, date\_format)  
 # print(date\_obj)  
 except ValueError:  
 print("Incorrect date format, should be YYYY-MM-DD")  
 test\_res = 0  
 return test\_res  
  
 if test\_res != 0:  
 c.execute("INSERT INTO review VALUES(:id,:number\_of\_reviews,:last\_review,:reviews\_per\_month)",  
 {"id": review\_id\_test, "number\_of\_reviews": number\_of\_reviews\_test,  
 "last\_review": last\_review\_test, "reviews\_per\_month": reviews\_per\_month\_test})  
  
 c.execute("SELECT id from review WHERE id = :id", {"id": review\_id\_test})  
  
 temp = c.fetchall()[0][0]  
  
 if temp == review\_id\_test:  
 print("Record inserted Successfully")  
 c.execute("SELECT \* from review WHERE id = :id", {"id": review\_id\_test})  
  
 temp = c.fetchall()  
 conn.commit()  
 print("[Id,no of review,last review,review per month]==>", temp)  
 return 1  
 else:  
 print("Reviews record is not inserted successfully")  
 conn.commit()  
 return 0  
 except Exception as e:  
 print("Please enter valid data")  
 print(e)  
 finally:  
 conn.commit()  
 conn.close()  
  
 def Review\_Data\_Delete(self):  
  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 test\_res = 1  
  
 review\_id\_test = input("Please enter review\_id:")  
 c.execute("SELECT COUNT(\*) from review WHERE id = :id", {"id": review\_id\_test})  
 rowcount = c.fetchall()[0][0]  
 # print(rowcount)  
 if rowcount <= 0:  
 print("No records founds for :", review\_id\_test, "review id")  
 else:  
  
 c.execute("DELETE from review WHERE id = :id", {"id": review\_id\_test})  
  
 c.execute("SELECT COUNT(\*) from review WHERE id = :id", {"id": review\_id\_test})  
  
 conn.commit()  
  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 if temp >= 1:  
 # c.execute("DELETE from review WHERE id = :id", {"id": review\_id\_test})  
 # print(c.fetchall())  
 print("Record is not deleted Successfully for Review id :", review\_id\_test)  
 test\_res = 0  
 return test\_res  
 else:  
 print("Record deleted Successfully for id :", review\_id\_test)  
 conn.commit()  
 return test\_res  
  
 except Exception as e:  
 print(e)  
 finally:  
 conn.commit()  
 conn.close()  
  
 # update host  
  
 def review\_data\_update(self):  
  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 test\_res = 1  
 print("Review Test Data : Review ID's: 241510,241503,50646")  
 review\_id\_test = input("Please enter Review ID for which you want update:")  
 review\_id\_test = int(review\_id\_test)  
  
 no\_reviews\_test = input("Please enter number of reviews:")  
 no\_reviews\_test = str(no\_reviews\_test)  
 # Validating the date format is correct or not before data insertion  
 last\_review\_date = input("Please enter last\_review date(YYYY-MM-DD):")  
 date\_format = '%Y-%m-%d'  
 try:  
 date\_obj = datetime.datetime.strptime(last\_review\_date, date\_format)  
 except ValueError:  
 print("Incorrect date format, should be YYYY-MM-DD")  
 test\_res = 0  
 return test\_res  
  
 if test\_res != 0:  
 reviews\_per\_month\_test = input("Please enter reviews\_per\_month: ")  
 reviews\_per\_month\_test = float(reviews\_per\_month\_test)  
  
 c.execute(  
 'UPDATE review SET number\_of\_reviews=:number\_of\_reviews,last\_review=:last\_review,'  
 'reviews\_per\_month=:reviews\_per\_month WHERE id=:id',  
 {'number\_of\_reviews': no\_reviews\_test, 'last\_review': last\_review\_date,  
 'reviews\_per\_month': reviews\_per\_month\_test, 'id': review\_id\_test})  
  
 # self.conn.commit()  
 c.execute("SELECT id from review WHERE id = :id", {"id": review\_id\_test})  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 # print(type(host\_id\_test))  
  
 if temp == review\_id\_test:  
 c.execute("SELECT \* from review WHERE id = :id", {"id": review\_id\_test})  
 updated = c.fetchall()  
 print("\nReview Updated Successfully [Review\_id,No of review,last review date,Reviews per month] "  
 "=> "  
 "", updated)  
 # self.conn.commit()  
  
 return test\_res  
 else:  
 print("Record is not updated successfully")  
 test\_res = 0  
 return test\_res  
  
 except Exception as e:  
  
 print(e)  
 print("\nPlease try with different data")  
 finally:  
 conn.commit()  
 conn.close()  
  
 def display\_review\_data(self):  
  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 review\_df = c.execute("SELECT \* from review")  
 review\_df = pd.DataFrame(review\_df)  
 review\_df.columns = ['id', 'number\_of\_reviews', 'last\_review', 'reviews\_per\_month']  
 print(review\_df.head(10))  
 conn.commit()  
 conn.close()  
 # Populate all the data/listing that has 200 or above total reviews  
  
 def pop\_data(self):  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 pop\_df = c.execute("SELECT \* from review LEFT JOIN room ON review.id = room.id WHERE number\_of\_reviews >= "  
 "200")  
 pop\_df = pd.DataFrame(pop\_df)  
 pop\_df.columns = ['id', 'number\_of\_reviews', 'last\_review', 'reviews\_per\_month', 'id', 'name', 'latitude',  
 'longitude', 'price', 'min nights', 'availability', 'hostid', 'roomtypeid']  
 # pop\_df.columns == ['id', 'number\_of\_reviews', 'last\_review', 'reviews\_per\_month']  
 print("\nPopulating all the data that has 200 or above total reviews and creating csv file:")  
 print("\nDisplaying 10 rows from 200totalreview.csv file.\n")  
 print(pop\_df.head(10))  
 pop\_df.to\_csv('200totalreview.csv')  
 print("\nPlease check project folder for 200totalreview.csv file.\n")  
  
 except Exception as e:  
 print(e)  
 finally:  
 conn.commit()  
 conn.close()  
  
# Review\_Data\_Obj = Review\_Data()  
# print(Review\_Data\_Obj.Review\_Data\_insert())  
# Review\_Data\_Obj.Review\_Data\_Delete()  
# Review\_Data\_Obj.Review\_Data\_insert()  
# Review\_Data\_Obj.review\_data\_update()  
# Review\_Data\_Obj.display\_review\_data()  
# Review\_Data\_Obj.pop\_data()

## Appendix E: Room Data Code

import pandas as pd  
import sqlite3  
  
  
class Room\_Data:  
  
 df = pd.read\_csv('listings.csv')  
  
 room\_df = df[["id", "name", "latitude", "longitude", "price", "minimum\_nights", "availability\_365", "room\_type",  
 "host\_id"]].copy()  
  
 # Data Cleaning - Adding values to room name for blank values  
 room\_df = room\_df.drop\_duplicates()  
 room\_df['name'] = room\_df['name'].fillna("No-Name" + "\_" + room\_df['room\_type'])  
  
 room\_df.reset\_index(drop=True, inplace=True)  
  
 # Create a roomtype id  
 def makermid(data):  
 if data['room\_type'] == 'Private room':  
 return 1  
 elif data['room\_type'] == 'Entire home/apt':  
 return 2  
 elif data['room\_type'] == 'Shared room':  
 return 3  
 else:  
 return 0  
  
 room\_df['room\_type\_id'] = room\_df.apply(makermid, axis=1)  
 del room\_df['room\_type']  
  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 c.execute("SELECT count(name) from sqlite\_master WHERE type='table' AND name='room'")  
 if c.fetchone()[0] == 1:  
 # c.execute("DELETE FROM room")  
 c.execute("DROP Table room")  
 else:  
 c.execute(  
 """CREATE TABLE room(id INTEGER,name TEXT,latitude REAL,longitude REAL,price REAL, minimum\_nights INTEGER,  
 availability\_365 INTEGER,room\_type\_id INTEGER, host\_id INTEGER,neighbourhood\_id INTEGER,  
 PRIMARY KEY(id),  
 FOREIGN KEY (room\_type\_id) REFERENCES roomtype(room\_type\_id),  
 FOREIGN KEY (host\_id) REFERENCES host(host\_id) ,   
 FOREIGN KEY (neighbourhood\_id) REFERENCES nbh\_data(neighbourhood\_id)   
 )""")  
  
 room\_df.to\_sql('room', conn, if\_exists='append', index=False)  
  
 c.execute("SELECT \* FROM room")  
  
 conn.commit()  
 conn.close()  
 # Inserting multiple records using input\_data excel file  
  
  
 def room\_data\_insert(self):  
 print("Please update the given input file for room data(Multiple Records can also update:")  
 print("Inserting the room data from input file input\_data ")  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 #c = conn.cursor()  
  
 room\_input\_data = pd.read\_excel('input\_data.xlsx')  
 print(room\_input\_data.head())  
 # id\_test=room\_input\_data.loc['id']  
  
 # print("Room ID",id\_test,"\n")  
 print("\n--------------------------------------")  
 room\_input\_data.to\_sql('room', conn, if\_exists='append', index=False)  
  
 except Exception as exp:  
 print(exp)  
 finally:  
 conn.commit()  
 conn.close()  
  
 # Del func  
  
 def Room\_Data\_Delete(self):  
  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 print("Room ID Test Data:71907,56334")  
 room\_id\_test = input("Please enter room\_id:")  
 c.execute("SELECT COUNT(\*) from room WHERE id = :id", {"id": room\_id\_test})  
 rowcount = c.fetchall()[0][0]  
 # print(rowcount)  
 if rowcount <= 0:  
 print("No records founds for room id:", room\_id\_test)  
 else:  
  
 c.execute("DELETE from room WHERE id = :id", {"id": room\_id\_test})  
  
 c.execute("SELECT COUNT(\*) from room WHERE id = :id", {"id": room\_id\_test})  
  
 conn.commit()  
  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 if temp >= 1:  
 print("Record is not deleted for Review id :", room\_id\_test)  
  
 else:  
 print("Record deleted Successfully for room id :", room\_id\_test)  
  
 return 1  
  
 except Exception as e:  
 print(e)  
 finally:  
 conn.commit()  
 conn.commit()  
  
 # update room Data  
  
 def room\_data\_update(self):  
  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 test\_res = 1  
  
 room\_id\_test = int(input("Please enter Room\_ID for which you want update:"))  
 room\_name\_test = input("Please enter updated Room Name:")  
 min\_night\_test = int(input("Please enter updated minimum nights count:"))  
 availability\_test = int(input("Please enter the availability\_test out of 365 days:"))  
 room\_price\_test = int(input("Please enter room price:"))  
 c.execute("Update room set name = :name, minimum\_nights =:minimum\_nights, availability\_365 "  
 "=:availability\_365, price =:price "" where id = :id",  
 {"name": room\_name\_test,  
 "minimum\_nights": min\_night\_test,  
 "availability\_365": availability\_test,  
 "price": room\_price\_test,  
 "id": room\_id\_test})  
  
 c.execute("SELECT id from room WHERE id = :id", {"id": room\_id\_test})  
 temp = c.fetchall()[0][0]  
 print(temp)  
 if temp == room\_id\_test:  
 c.execute("SELECT \* from room WHERE id = :id", {"id": room\_id\_test})  
 updated = c.fetchall()  
 print("\nRoom Data Updated Successfully \n['id','name','latitude, 'longitude',Price, "  
 "'minimum\_nights','availability\_365',host id,room id] => \n", updated)  
 return test\_res  
 else:  
 print("Record is not updated successfully")  
 test\_res = 0  
 return test\_res  
 except Exception as e:  
 print(e)  
 print("\nPlease try with different data")  
 finally:  
 conn.commit()  
 conn.close()  
  
 def display\_room\_data(self):  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 room\_df = c.execute("SELECT \* from room")  
 room\_df = pd.DataFrame(room\_df)  
 room\_df.columns = ["id", "name", "latitude", "longitude", "price", "minimum\_nights", "availability\_365",  
 "host\_id", "room\_type\_ID"]  
 # print(room\_df.info())  
 print((room\_df).head(10))  
 conn.commit()  
 conn.close()  
  
  
#Room\_Data\_Obj = Room\_Data()  
# Room\_Data\_Obj.Room\_Data\_Delete()  
# Room\_Data\_Obj.room\_data\_insert()  
  
# Room\_Data\_Obj.Room\_Data\_Delete()  
#Room\_Data\_Obj.display\_room\_data()  
# Room\_Data\_Obj.room\_data\_update()

## Appendix F: Room Type Data Code

import pandas as pd  
import sqlite3  
  
  
class Room\_type:  
 df = pd.read\_csv('listings.csv')  
 room\_type\_df = df[["room\_type"]].copy()  
  
 room\_type\_df = room\_type\_df.drop\_duplicates()  
 room\_type\_df.reset\_index(drop=True, inplace=True)  
  
 # print(room\_type\_df)  
  
 # Make a roomtypeid  
 def makermid(data):  
 if data['room\_type'] == 'Private room':  
 return 1  
 elif data['room\_type'] == 'Entire home/apt':  
 return 2  
 elif data['room\_type'] == 'Shared room':  
 return 3  
 else:  
 return 4  
  
 room\_type\_df['room\_type\_id'] = room\_type\_df.apply(makermid, axis=1)  
 # del room\_df['room\_type']  
 # print(room\_df['room\_type\_id'])  
  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
  
 c.execute("SELECT count(name) from sqlite\_master WHERE type='table' AND name='roomtype'")  
 if c.fetchone()[0] == 1:  
 c.execute("DELETE FROM roomtype")  
 else:  
 c.execute("""CREATE table roomtype(room\_type\_id INTEGER, room\_type TEXT,PRIMARY KEY(room\_type\_id))""")  
  
 room\_type\_df.to\_sql("roomtype", conn, if\_exists='append', index=False)  
  
 # c.execute("SELECT \* FROM roomtype")  
 # print(c.fetchall())  
  
 conn.commit()  
 conn.close()  
  
 # Insert New roomtype  
 def room\_type\_insert(self):  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 room\_type\_id\_test = int(input("Please enter room\_type\_id:"))  
 room\_type\_name\_test = input("Please enter room\_type\_name:")  
 print(room\_type\_id\_test)  
 c.execute("INSERT INTO roomtype VALUES(:room\_type\_id,:room\_type)",  
 {"room\_type\_id": room\_type\_id\_test, "room\_type": room\_type\_name\_test})  
 c.execute("SELECT room\_type\_id from roomtype WHERE room\_type\_id = :room\_type\_id",  
 {"room\_type\_id": int(room\_type\_id\_test)})  
 temp = c.fetchall()[0][0]  
  
 if str(temp) == str(room\_type\_id\_test):  
 print("Record inserted Successfully")  
 c.execute("SELECT \* from roomtype WHERE room\_type\_id = :room\_type\_id",  
 {"room\_type\_id": int(room\_type\_id\_test)})  
 temp = c.fetchall()  
 print("[Room type ID, Room type]=>", temp)  
 # conn.commit()  
 return 1  
 else:  
 print("Record is not inserted successfully")  
 # conn.commit()  
 return 0  
 except Exception as e:  
 print("Got Exception")  
 print(e)  
 finally:  
 conn.commit()  
 conn.close()  
  
 def Roomtype\_Update(self):  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 roomtype\_id\_test = input("Please enter Roomtype\_ID for which you want update:")  
 roomtype\_id\_test = int(roomtype\_id\_test)  
  
 ls = [1, 2, 3, 4]  
  
 if (roomtype\_id\_test in ls):  
 roomtype\_test = input("Please enter updated room type name given room id:")  
 roomtype\_test = str(roomtype\_test)  
  
 c.execute(  
 'UPDATE roomtype SET room\_type=:room\_type WHERE room\_type\_id=:room\_type\_id',  
 {'room\_type\_id': roomtype\_id\_test, 'room\_type': roomtype\_test})  
  
 conn.commit()  
 c.execute("SELECT room\_type\_id from roomtype WHERE room\_type\_id = :room\_type\_id",  
 {"room\_type\_id": roomtype\_id\_test})  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 # (type(roomtype\_id\_test))  
  
 if temp == roomtype\_id\_test:  
 print("Record updated Successfully")  
 c.execute("SELECT \* from roomtype WHERE room\_type\_id = :room\_type\_id",  
 {"room\_type\_id": roomtype\_id\_test})  
 temp = c.fetchall()  
 print(temp)  
 conn.commit()  
 return 1  
 else:  
 print("Record is not updated successfully")  
 conn.commit()  
 return 0  
 else:  
 print("Sorry for given room id no records available ")  
  
 except Exception:  
 # print(e)  
 print("Please try with different data entered Host-ID is not exist")  
 finally:  
 conn.commit()  
 conn.close()  
  
 # Delete host data  
  
 # Del func  
  
 def room\_Data\_Delete(self):  
  
 try:  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 room\_id = input("Please enter room\_id:")  
 c.execute("SELECT COUNT(\*) from roomtype WHERE room\_type\_id = :room\_type\_id", {"room\_type\_id": room\_id})  
 rowcount = c.fetchall()[0][0]  
 # print(rowcount)  
 if rowcount <= 0:  
 print("No records founds for room\_type\_id:", room\_id)  
 else:  
  
 c.execute("DELETE from roomtype WHERE room\_type\_id = :room\_type\_id", {"room\_type\_id": room\_id})  
  
 c.execute("SELECT COUNT(\*) from roomtype WHERE room\_type\_id = :room\_type\_id", {"room\_type\_id": room\_id})  
  
 conn.commit()  
  
 temp = c.fetchall()[0][0]  
 # print(temp)  
 if temp >= 1:  
 print("Record is not deleted for room\_type\_id :", room\_id)  
  
 else:  
 print("Record deleted Successfully for room\_type\_id :", room\_id)  
  
 return 1  
  
 except Exception as e:  
 print(e)  
 finally:  
 conn.commit()  
 conn.commit()  
  
 def display\_roomtype\_data(self):  
 conn = sqlite3.connect("MyDB.db", timeout=10)  
 c = conn.cursor()  
 print("\nDisplaying the host data:")  
 room\_type\_df = c.execute("SELECT \* from roomtype")  
 room\_type\_df = pd.DataFrame(room\_type\_df)  
 print(room\_type\_df.info())  
 room\_type\_df.columns = ["room\_type\_id", 'room\_type']  
 print(room\_type\_df.head(10))  
 conn.commit()  
 conn.close()  
  
# update fun  
  
  
# Room\_type\_Obj = Room\_type()  
# Room\_type\_Obj.display\_roomtype\_data()  
  
# Room\_type\_Obj.room\_Data\_Delete()  
  
# print(Room\_type\_Obj.room\_type\_insert())  
  
# print(Room\_type\_Obj.Roomtype\_Update())

## Appendix F: Inheritance Code

# Parent class  
class Property\_Types(object):  
 print("In super class")  
 # Constructor  
  
 def \_\_init\_\_(self, name, price, availability):  
 self.name = name  
 self.price = price  
 self.availability = availability  
  
 def display(self):  
 print("\nIn parent class display method")  
 print("Name:", self.name)  
 print("Availability:", self.price)  
 print("Price:", self.availability)  
  
# Child Class 1  
class Private\_room(Property\_Types):  
 print("In subclass private room")  
 str1 = 'Private room'  
 def \_\_int\_\_(self, name, price, availability, room\_id):  
 self.room\_id = room\_id  
  
 # calling parent class constructor  
 Property\_Types.\_\_init\_\_(self,name, price, availability)  
  
 print("\n In private room child class")  
  
room = input("Enter room Name:")  
availability = int(input("Enter Availability:"))  
price = int(input("Enter Price:"))  
  
# Creating a Object of child class  
private\_room\_obj = Private\_room(room, availability, price)  
  
private\_room\_obj.display()  
  
# Child Class 2  
class Entire\_Home(Property\_Types):  
 print("In subclass Entire\_Home")  
 str = "Sweet Home"  
 def \_\_int\_\_(self, name, price, availability, home\_id):  
 self.home\_id = home\_id  
  
 # calling parent class constructor  
 Property\_Types.\_\_init\_\_(self,name, price, availability)  
  
 print("\n In Entire\_Home child class")  
  
room = input("Enter Home Name:")  
availability = int(input("Enter Availability:"))  
price = int(input("Enter Price:"))  
  
class Shared(Private\_room, Entire\_Home):  
 def \_\_int\_\_(self):  
 print("Calling constructor of private room and Entire Home subclasses")  
 Private\_room.\_\_init\_\_(self, price, availability)  
 self.price=price  
 self.availability=availability  
  
  
 Entire\_Home.\_\_init\_\_(self, price, availability)  
 self.price = price  
 self.availability = availability  
  
 print("Shared room class formed from 2 classes Private room & EntireHome by Multiple Inheritance")  
  
 def shared\_display(self):  
 print("Got attribute from one parent class:", private\_room\_obj.str1)  
 print("Got attribute from one parent class:", shared\_obj.str)  
  
# Creating a Object of child class  
shared\_obj = Shared(room, availability, price)  
shared\_obj.shared\_display()

## Appendix G: Unit Testing Code

from host\_data import Host\_Data  
from review\_data import Review\_Data  
  
class UnitTesing:  
 def do\_unit\_test(Self ):  
 print("\n\*\*\*\*\* Unit Testing \*\*\*\*\*\*")  
 print("==========================")  
  
 def test\_host\_insertion(test\_function,input, expected):  
 print("Test valid data insert for HOST")  
 print("\nEnter valid HOST DATA e.g: 101,John,1")  
 actual=test\_function()  
  
 if(actual==expected):  
 print("Test Passed")  
 else:  
 print("Test Failed")  
 print("-----------------------------------")  
  
  
 def test\_host\_update(test\_function,input, expected):  
 print("\nTest valid Data update case for HOST:")  
 print("\nEnter valid HOST Data e.g:ID:23666,John,1")  
 actual=test\_function()  
  
 if(actual==expected):  
 print("Test Passed")  
 else:  
 print("Test Failed")  
 print("-----------------------------------")  
  
 def test\_review\_data\_delete(test\_function,input, expected):  
 print("\nTest Delete case for Review:")  
 print("\nEnter valid Review Data e.g:ID's:56334,71896,71907")  
 actual=test\_function(input)  
  
 if(actual==expected):  
 print("Test Passed")  
 else:  
 print("Test Failed")  
 print("-----------------------------------")  
  
  
 # Invalid input  
  
 def test\_In\_host\_insertion(test\_function, input, expected):  
 print("Test Invalid data insert for HOST")  
 print("\nEnter Invalid HOST DATA e.g: John,John,John or Duplicate ID:184596")  
 actual = test\_function()  
  
 if (actual == expected):  
 print("Test Passed")  
 else:  
 print("Test Failed")  
 print("-----------------------------------")  
  
  
 def test\_review\_date\_validate(test\_function, input, expected):  
 print("\nTest Invalid Review Date:")  
 print("\nReview ID Test Data e.g:ID's:56334,71896,71907")  
 print("\nEnter Invalid Review Date e.g: 32-26-0011")  
 actual = test\_function(input)  
  
 if (actual == expected):  
 print("Test Passed")  
 else:  
 print("Test Failed")  
 print("-----------------------------------")  
  
  
 host\_Data\_Obj = Host\_Data()  
 user\_input\_host = host\_Data\_Obj  
 expected\_output = 1  
 print("Scenario 1: To Test valid host data insertion:")  
 test\_host\_insertion(Host\_Data.host\_insert,user\_input\_host, expected\_output)  
  
  
 print("Scenario 2: To Test Invalid host data insertion for duplicate Host ID:")  
 test\_In\_host\_insertion(Host\_Data.host\_insert,user\_input\_host, expected\_output)  
  
  
 print("Scenario 3: To Test Invalid host data insertion:")  
 test\_In\_host\_insertion(Host\_Data.host\_insert, user\_input\_host, expected\_output)  
  
  
 print("Scenario 4: To Test Valid host data update:")  
 test\_host\_update(Host\_Data.host\_update,user\_input\_host, expected\_output)  
  
  
 review\_data\_Obj = Review\_Data()  
 #user\_input\_review = review\_data\_Obj  
 print("Scenario 5: To Test Review Data Delete:")  
 test\_review\_data\_delete(Review\_Data.Review\_Data\_Delete,review\_data\_Obj, expected\_output)  
  
  
 print("Scenario 6: To Test Review Date validation:")  
 test\_review\_date\_validate(Review\_Data.Review\_Data\_insert,review\_data\_Obj, expected\_output)  
  
  
  
#unit\_Obj = UnitTesing()  
#unit\_Obj.do\_unit\_test()

## Appendix H: Demo Function Code

from host\_data import Host\_Data  
from room\_data import Room\_Data  
from review\_data import Review\_Data  
from room\_type\_data import Room\_type  
from neighbourhood\_data import Neighbourhood\_Data  
from neighbourhood\_group import Neighbourhood\_Group  
from unit\_test import UnitTesing  
  
# Creating objects for all the class file  
  
Host\_obj = Host\_Data()  
neighbourhood\_data\_Obj = Neighbourhood\_Data()  
nbh\_grp\_Obj = Neighbourhood\_Group()  
Review\_Data\_Obj = Review\_Data()  
Room\_Data\_Obj = Room\_Data()  
Room\_type\_Obj = Room\_type()  
unit\_testing\_Obj = UnitTesing()  
  
# Main Menu option for all the operations  
  
  
def menu():  
 ans = True  
 while ans:  
 print("Menu Form:")  
 print("1. Display Data\n2. Insert Data\n3. Update Data\n4. Delete Data\n5. Populate Listing Data "  
 "for 200 or more reviews\n6. Unit Testing \n7. Exit\n")  
 ans = input("What would you like to do?: ")  
 if ans == "1":  
  
 sub\_wh = True  
 while sub\_wh:  
 print("\nDisplay Menu:")  
 print(  
 "\n1. Host Data\n2. Neighbourhood Data\n3. Neighbourhood Group\n4. Review\n5. Room"  
 "\n6. Room Type\n7. Go Back")  
  
 subm = input("What would you like to do? ")  
 if subm == "1":  
 print("\nDisplaying Host Data")  
 Host\_obj.display\_host\_data()  
 elif subm == "2":  
 print("\nDisplaying Neighbourhood Data")  
 neighbourhood\_data\_Obj.display\_nbh\_data()  
 elif subm == "3":  
 print("\nDisplaying Neighbourhood Group Data")  
 nbh\_grp\_Obj.display\_nbh\_Grp\_data()  
 elif subm == "4":  
 print("\nDisplaying Review Data")  
 Review\_Data\_Obj.display\_review\_data()  
 elif subm == "5":  
 print("\nDisplaying Room Data")  
 Room\_Data\_Obj.display\_room\_data()  
  
 elif subm == "6":  
 print("\nDisplaying Room Type Data")  
 Room\_type\_Obj.display\_roomtype\_data()  
 elif subm == "7":  
 sub\_wh = False  
 print("\nReturning Back to previous menu")  
 else:  
 print("\nYou have selected invalid option")  
 elif ans == "2":  
 sub\_wh = True  
 while sub\_wh:  
 print("\nData Inserting Menu")  
 print(  
 "\n1. Host Data\n2. Neighbourhood Data\n3. Neighbourhood Group\n4. Review Data"  
 "\n5. Room Data\n6. Room Type\n7. Go Back")  
 subm = input("What would you like to do? ")  
 if subm == "1":  
 print("\nInserting Host Data")  
 Host\_obj.host\_insert()  
 elif subm == "2":  
 print("\nInserting Neighbourhood Data")  
 neighbourhood\_data\_Obj.nbh\_insert()  
 elif subm == "3":  
 print("\nInserting Neighbourhood Group Data")  
 nbh\_grp\_Obj.nbh\_grp\_inser()  
 elif subm == "4":  
 print("\nInserting Review Data")  
 Review\_Data\_Obj.Review\_Data\_insert()  
 elif subm == "5":  
 print("\nInserting Room Data")  
 Room\_Data\_Obj.room\_data\_insert()  
 elif subm == "6":  
 print("\nInserting Room Type Data")  
 Room\_type\_Obj.room\_type\_insert()  
 elif subm == "7":  
 sub\_wh = False  
 print("\nReturning Back to previous menu")  
 else:  
 print("\nYou have selected invalid option")  
 elif ans == "3":  
 sub\_wh = True  
 while sub\_wh:  
 print("\nData Update Menu")  
 print(  
 "\n1.Host Data\n2.Neighbourhood Data\n3.Neighbourhood Group Data\n4.Review Data"  
 "\n5.Room Data \n6.Room type \n7. Go Back")  
 subm = input("What would you like to do? ")  
 if subm == "1":  
 print("\nUpdating Host Data")  
 Host\_obj.host\_update()  
 elif subm == "2":  
 print("\nUpdating Neighbourhood Data")  
 neighbourhood\_data\_Obj.nbh\_data\_Update()  
 elif subm == "3":  
 print("\nUpdating Neighbourhood Group Data")  
 nbh\_grp\_Obj.nbh\_group\_update()  
 elif subm == "4":  
 print("\nUpdating Review Group Data")  
 Review\_Data\_Obj.review\_data\_update()  
 elif subm == "5":  
 print("\nUpdating Room Data")  
 Room\_Data\_Obj.room\_data\_update()  
 elif subm == "6":  
 print("\nUpdating Room Type")  
 Room\_type\_Obj.Roomtype\_Update()  
 elif subm == "7":  
 sub\_wh = False  
 print("\nReturning Back to previous menu")  
 else:  
 print("\nYou have selected invalid option")  
 elif ans == "4":  
 sub\_wh = True  
 while sub\_wh:  
 print("\nData Delete Menu")  
 print(  
 "\n1. Host Data\n2. Neighbourhood Data\n3. Neighbourhood Group\n4. Review Data"  
 "\n5. Room Data"  
 "\n6. Room Type\n7. Go Back")  
 subm = input("What would you like to do? ")  
 if subm == "1":  
 print("\nDeleting Host Data")  
 Host\_obj.host\_data\_delete()  
 elif subm == "2":  
 print("\nDeleting Neighbourhood Data")  
 neighbourhood\_data\_Obj.nbh\_Data\_Delete()  
 elif subm == "3":  
 print("\nDeleting Neighbourhood Group Data")  
 nbh\_grp\_Obj.nbh\_grp\_Delete()  
 elif subm == "4":  
 print("\nDeleting Review Data")  
 Review\_Data\_Obj.Review\_Data\_Delete()  
 elif subm == "5":  
 print("\nDeleting Room Data")  
 Room\_Data\_Obj.Room\_Data\_Delete()  
 elif subm == "6":  
 print("\nDeleting Room Type Data")  
 Room\_type\_Obj.room\_Data\_Delete()  
 elif subm == "7":  
 sub\_wh = False  
 print("\nReturning Back to previous menu")  
 else:  
 print("\nYou have selected invalid option")  
 elif ans == "5":  
 print("\nPopulate listing that has 200 or above total reviews")  
 Review\_Data\_Obj.pop\_data()  
 elif ans == "6":  
 print("Unit Testing is in Process")  
 unit\_testing\_Obj.do\_unit\_test()  
 elif ans == "7":  
 ans = False  
 print("\nGoodbye.....!")  
 elif ans != "":  
 print("\nYou have selected invalid option")  
  
  
class Demo:  
 try:  
 print('\n\*\*\*\*\*\* Singapore\_Airbnb Data-Set Analysis \*\*\*\*\*\*\n')  
  
 except Exception as e:  
 print(e)  
  
  
demo\_obj = Demo()  
menu()