



**Postgraduate Certificate in Cloud Native Computing
Postgraduate Certificate in Software Design with Artificial Intelligence**

Applied Scripting Languages

Assignment 1

Student ID: A00267813

Brief Description

Airbnb is worldwide online marketplace to lend individuals or small business's spare space in competitive manner. New York is one of the most popular and extremely expensive tourist destinations in the world. Hence New York was one of earliest cities where Airbnb started their service. The purpose of this assignment is to analyse factors which influence the price of Airbnb New York listings 2019.

Contents

Figures.....	3
Introduction.....	5
Data	6
Data Fields	7
Sample Data	8
Design	9
Design Principles.....	9
Modules.....	9
Database Table Diagram.....	12
Sequence Diagrams.....	13
Testing.....	20
Analysis and Visualisation.....	22
Conclusion	38
Appendix 1: Reflective Learning Log	42
Appendix 2: References	68

Figures

Figure 1 Module Diagram.....	9
Figure 2 Database Table Diagram	12
Figure 3 Sequence Diagram File Read to Pair Scatter Plot	13
Figure 4 Sequence Diagram Selecting Price Range 2	14
Figure 5 Sequence Diagram Analysing data Price > 500	15
Figure 6 Sequence Diagram Analysing data Price > 1000	17
Figure 7 Sequence Diagram Analysing data Price > 5000	18
Figure 8 Sequence Diagram Availability = 0, Availability >	19
Figure 9 Unit Test Coverage.....	20
Figure 10 Running all unittests	21
Figure 11 Summary of file read	22
Figure 12 Data Head	22
Figure 13 Data Summary	23
Figure 14 Pair Scatter Plots.....	24
Figure 15 Pair Scatter Plots with Correlations.....	25
Figure 16 Price Summary	25
Figure 17 Price Distribution - All Data.....	26
Figure 18 Price Summary. Price <=1000.....	26
Figure 19 Price Distribution. Price <= 1000.....	27
Figure 20 Price Summary. Price <=500.....	27
Figure 21 Price Distribution. Price <= 500	28
Figure 22 Price Distribution comparison	28
Figure 23 Room Listing by Room Type, Pie Chart (Price < 500)	29
Figure 24 Room Listing by Room Type, Count Bar Chart (Price < 500)	29
Figure 25 Room Listing by Neighbourhood Group, Pie Chart (Price < 500).....	30
Figure 26 Room Listing by Neighbourhood Group, Count Bar Chart (Price < 500)	30
Figure 27 Room listing by Room Type and Neighbourhood Group (Price <=\$ 500).....	31
Figure 28 Room listing by Neighbourhood Group and Room Type (Price <=\$ 500).....	31
Figure 29 Price KDE by Room Type (Price <= \$500)	32
Figure 30 Price KDE by Neighbourhood Group (Price <= \$500)	32
Figure 31 Price Summary. Price > 1000, Price > 5000	33
Figure 32. Price Distribution. Price > \$1000, Price > \$5000.....	33
Figure 33 Price Comparison. All Price, Price <\$500, Price > \$1000, Price > \$5000.....	33
Figure 34 Price Grouping.....	34
Figure 35 Price Grouping.....	35
Figure 36 Price Comparison for Availability.....	35
Figure 37 Setting up git repository	42
Figure 38 Facing Circular import issue	42
Figure 39 Fixing Module not found issue	43
Figure 40 Fixing Encoding issue while reading file	44
Figure 41 VC++ dependency issue while installation matplotlib.....	45
Figure 42 Installing VS Build tool as a fix	45
Figure 43 Python 3.8 is not supported but matplotlib.....	46
Figure 44 Searched and found solution for matplotlib version issue.....	46
Figure 45 Testing major math functions	47
Figure 46 Fixing overlapped x, y labels.....	48
Figure 47 Faced overlapped sub plots	49

Figure 48 Fixing overlapped subplots by introducing figsize	50
Figure 49 Facing empty subplot for variable to same variable scatter plot	50
Figure 50 Decided to fill empty subplot by a distribution graph	51
Figure 51 Printing summary in a table	52
Figure 52 Understanding Standard Deviation.....	53
Figure 53 Sub plotting without index causing issues.....	54
Figure 54 Testing grouped bar charts	56
Figure 55 Testing pie charts.....	56
Figure 56 Image saving issue.....	57
Figure 57 Fixing image save.....	57
Figure 58 Unittesting and test coverage.....	58
Figure 59 Unit testing for exceptions.....	59
Figure 60 Python doc Google Standard	60
Figure 61 Testing Python doc. See Also links are not working.....	60
Figure 62 Fixing Static code analysis warning and errors	61
Figure 63 Finding a way to hide internal methods	62
Figure 64 Unit tests are failing intermittently because of output order	63
Figure 65 File reading unit test are failing based on the location of test execution	64
Figure 66 Code refactoring	64
Figure 67: Large table print casing alignment issues.....	65
Figure 68: Fixed table size issue.....	65
Figure 69: Introducing user friendly logging	66

Introduction

According to ¹ Wikipedia, Airbnb, Inc. is an online marketplace for arranging or offering lodging, primarily homestays, or tourism experiences. The company does not own any of the real estate listings, nor does it host events; it acts as a broker, receiving commissions from each booking. The company is based in San Francisco, California, United States.

In 2008, the founders of AirBnb officially launched their business idea in a site called Airbedandbreakfast.com, then in 2009 they shorten the name Airbnb.com. Just after 2 years, Airbnb recorded 1 millionth booked, and after another year, in February 2012 they announced their 5 millionth and just after another 5 months, doubling numbers Airbnb announced 10 millionth booking.

Main features of the Airbnb are the host who list their lodging with their own price and tenet can rent the lodging through Airbnb, and both tenant and host can share their experience through reviews. So competitive price and tenant's reviews plays a major role for host business.

The purpose of this assignment is to analyse how tenants' reviews correlate to price and other factors which influence the price of Airbnb New York listings 2019

¹ <https://en.wikipedia.org/wiki/Airbnb>

Data

² Insideairbnb publishes various data related to their business for different countries and different cities. The data related to listing in New York 2019 can be downloaded from here <http://insideairbnb.com/get-the-data.html>

The original dataset has following files in CSV format or archive. As primary focus of the assignment is to analyse price variation, it was decided to analyse listings.csv.

12 September, 2019	New York City	listings.csv.gz	Detailed Listings data for New York City
12 September, 2019	New York City	calendar.csv.gz	Detailed Calendar Data for listings in New York City
12 September, 2019	New York City	reviews.csv.gz	Detailed Review Data for listings in New York City
12 September, 2019	New York City	listings.csv	Summary information and metrics for listings in New York City (good for visualisations).
12 September, 2019	New York City	reviews.csv	Summary Review data and Listing ID (to facilitate time based analytics and visualisations linked to a listing).
N/A	New York City	neighbourhoods.csv	Neighbourhood list for geo filter. Sourced from city or open source GIS files.
N/A	New York City	neighbourhoods.geojson	GeoJSON file of neighbourhoods of the city.

² <http://insideairbnb.com/>

Further, various data exploratory efforts related to this dataset can be found in kaggle

³<https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data>

Data Fields

listings.csv file consists 49080 records with 16 following fields

Field	Type	Description	Considered
id	Integer	AirBnB specific unique number. This field will be ignored for analysis	No
name	Text	Name of the listing	No
host_id	Integer	Host Id assigned by AirBnB	Yes
host_name	Text	Name of the host	Yes
neighbourhood_group	Text	Neighbourhood group where room located	Yes
neighbourhood	Text	Neighbourhood group where room located	Yes
latitude	Real	Latitude coordinate of the room	Yes
longitude	Real	longitude coordinate of the room	Yes
room_type	Text	Room Type	Yes
price	Real	Price of the room	Yes
minimum_nights	Integer	Minimum nights need to be booked	Yes
number_of_reviews	Integer	Total Number of Reviews for the room	Yes
last_review	Date	Last Review date for the room	Yes
reviews_per_month	Real	Average number of reviews per month	Yes
calculated_host_listings_count	Integer	Host listing count	Yes
availability_365	Integer	Availability of the room per year	Yes

³ <https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data>

Sample Data

<u>id</u>	<u>name</u>	<u>host_id</u>	<u>host_name</u>	<u>neighbourhood_group</u>	<u>neighbourhood</u>	<u>latitude</u>	<u>longitude</u>	<u>room_type</u>	<u>price</u>	<u>minimum_nights</u>	<u>number_of_reviews</u>	<u>last_review</u>	<u>reviews_per_month</u>	<u>calculated_availability_365</u>	
2539	Clean & q!	2787	John	Brooklyn	Kensington	40.64749	-73.9724	Private room	149	1	9	10/19/2018	0.21	6	365
2595	SkyLit Mid	2845	Jennifer	Manhattan	Midtown	40.75362	-73.9838	Entire home/apt	225	1	45	5/21/2019	0.38	2	355
3647	THE VILLA	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.9419	Private room	150	3	0			1	365
3831	Cozy Entr!	4869	LisaRoxan	Brooklyn	Clinton Hill	40.68514	-73.9598	Entire home/apt	89	1	270	7/5/2019	4.64	1	194
5022	Entire Apt	7192	Laura	Manhattan	East Harlem	40.79851	-73.944	Entire home/apt	80	10	9	11/19/2018	0.1	1	0
5099	Large Cozy	7322	Chris	Manhattan	Murray Hill	40.74767	-73.975	Entire home/apt	200	3	74	6/22/2019	0.59	1	129
5121	BlissArtsS	7356	Garon	Brooklyn	Bedford-Stuyvesant	40.68688	-73.956	Private room	60	45	49	10/5/2017	0.4	1	0
5178	Large Furr	8967	Shunichi	Manhattan	Hell's Kitchen	40.76489	-73.9849	Private room	79	2	430	6/24/2019	3.47	1	220
5203	Cozy Clea!	7490	MaryEllen	Manhattan	Upper West Side	40.80178	-73.9672	Private room	79	2	118	7/21/2017	0.99	1	0
5238	Cute & Co	7549	Ben	Manhattan	Chinatown	40.71344	-73.9904	Entire home/apt	150	1	160	6/9/2019	1.33	4	188

Figure 1 Sample Data. Spreadsheet view

Printing First 10 lines of data set														
	<u>name</u>	<u>host_id</u>	<u>neighbourhood_group</u>	<u>neighbourhood</u>	<u>latitude</u>	<u>longitude</u>	<u>room_type</u>	<u>price</u>	<u>minimum_nights</u>	<u>number_of_reviews</u>	<u>last_review</u>	<u>reviews_per_month</u>	<u>calculated_host_listings_count</u>	<u>availability_365</u>
1	Clean & quiet apt home by the park	2787	1	1	40.647	-73.972	Private room	149	1	9	2018-10-19	0.21	6	365
2	SkyLit Midtown Castle	2845	2	2	40.754	-73.984	Entire home/apt	225	1	45	2019-05-21	0.38	2	355
3	THE VILLAGE OF HARLEM..._NEW YORK !	4632	1	2	40.809	-73.942	Private room	150	3	0			1	365
4	Cozy Entire Floor of Brownstone	4869	2	1	40.685	-73.968	Entire home/apt	89	1	270	2019-07-05	4.64	1	194
5	Entire Apt; Spacious Studio/Loft by central park	7192	2	2	40.799	-73.944	Entire home/apt	80	10	9	2018-11-19	0.100	1	0
6	Large Cozy 1 BR Apartment In Midtown East	7322	2	2	40.748	-73.975	Entire home/apt	200	3	74	2019-06-22	0.598	1	129
7	BlissArtsSpace!	7356	1	1	40.687	-73.956	Entire home/apt	60	45	49	2017-10-05	0.408	1	0
8	Large Furnished Room Near B'way	8967	1	2	40.765	-73.985	Entire home/apt	79	2	430	2019-06-24	3.478	1	228
9	Cozy Clean Guest Room - Family Apt	7490	1	2	40.802	-73.967	Entire home/apt	150	1	118	2017-07-21	0.998	1	6
10	Cute & Cozy Lower East Side 1 bdrn	7549	2	2	40.713	-73.990	Entire home/apt	150	1	160	2019-06-09	1.338	4	188

Figure 2 Sample Data Application Output

Design

Design Principles.

Following design principles considered when designing modules

Modularity

Modules should be independent from each other as much as possible

⁴High cohesion and low coupling

Single Responsibility

Reusability

Modules should be reusable and should be used with other data sets as well.

Modules

It was identified following modules will be implemented for analysing the data set.

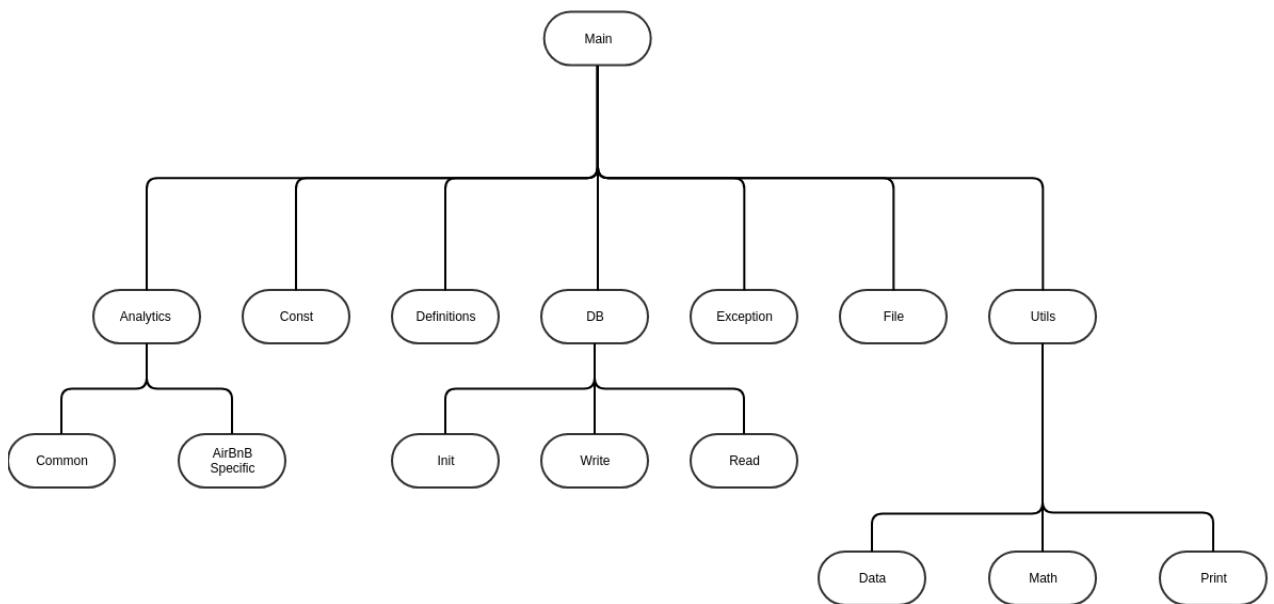


Figure 3 Module Diagram

⁴ <https://enterprisecraftsmanship.com/posts/cohesion-coupling-difference/>

Main

Main module is the starting point of the program

Analytics

Responsibility of this module is all graphs related activities. It consists 2 sub modules for Common and AirBnB specific graphs.

Constant

Shared module which can be used with other modules to specify constants.

Definition

All the application specific definitions / configurations would be defined here

DB

DB module is responsible for all db related operation. This module has 3 sub modules responsible for db Init operations such as creating tables, indexes, Write for all db insert operations. And Read for db sql selects.

Exception.

All application specific exception will be defined in this module.

File

This module will be implemented for csv file read and filtering invalid records.

Utils.

A reusable utility module consists of 3 sub modules Data, Math, and Print

Data

Reusable module can be used with any data set to filter, manipulate, or transform data.

Math

Math module is a reusable module which can be used to calculate main statistical elements such as min, median, mode, standard deviation etc.

Print

Print module is responsible for printing output standard output (console) in user friendly manner.

Database Table Diagram

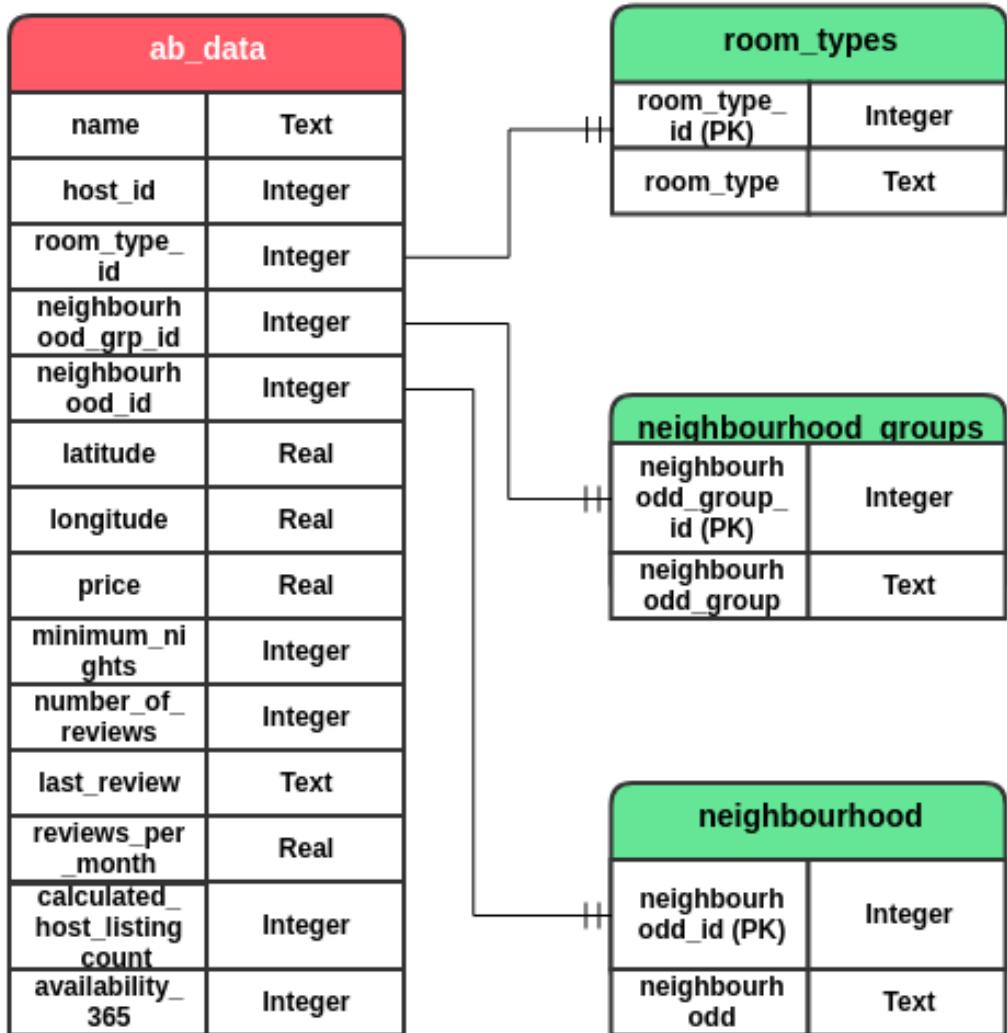


Figure 4 Database Table Diagram

Data repetition was identified with Room Type, Neighbourhood Group and Neighbourhood fields. So, it was decided to normalise these data into separate tables.

Sequence Diagrams

Init

Main

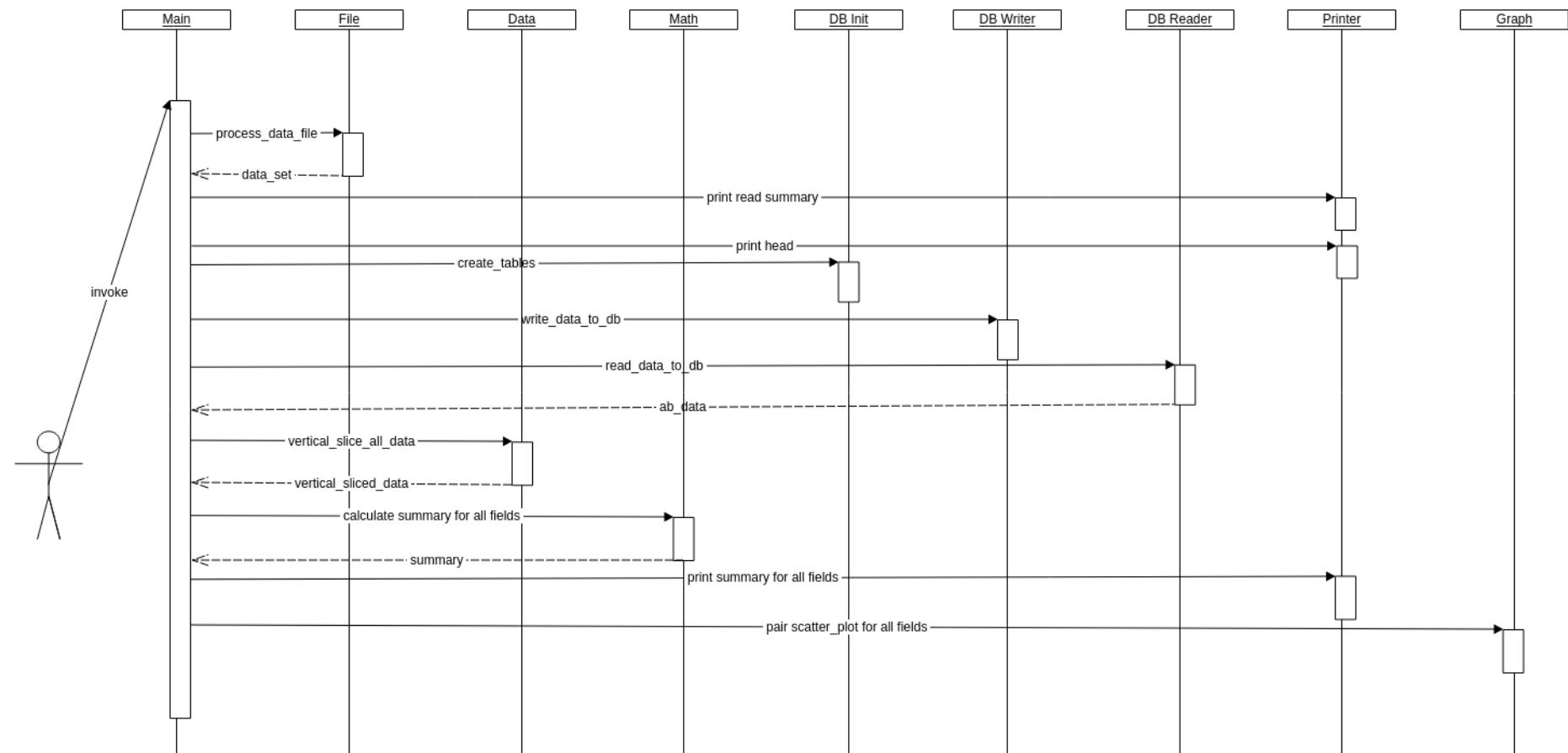


Figure 5 Sequence Diagram File Read to Pair Scatter Plot

Comparing Price

Comparing Price

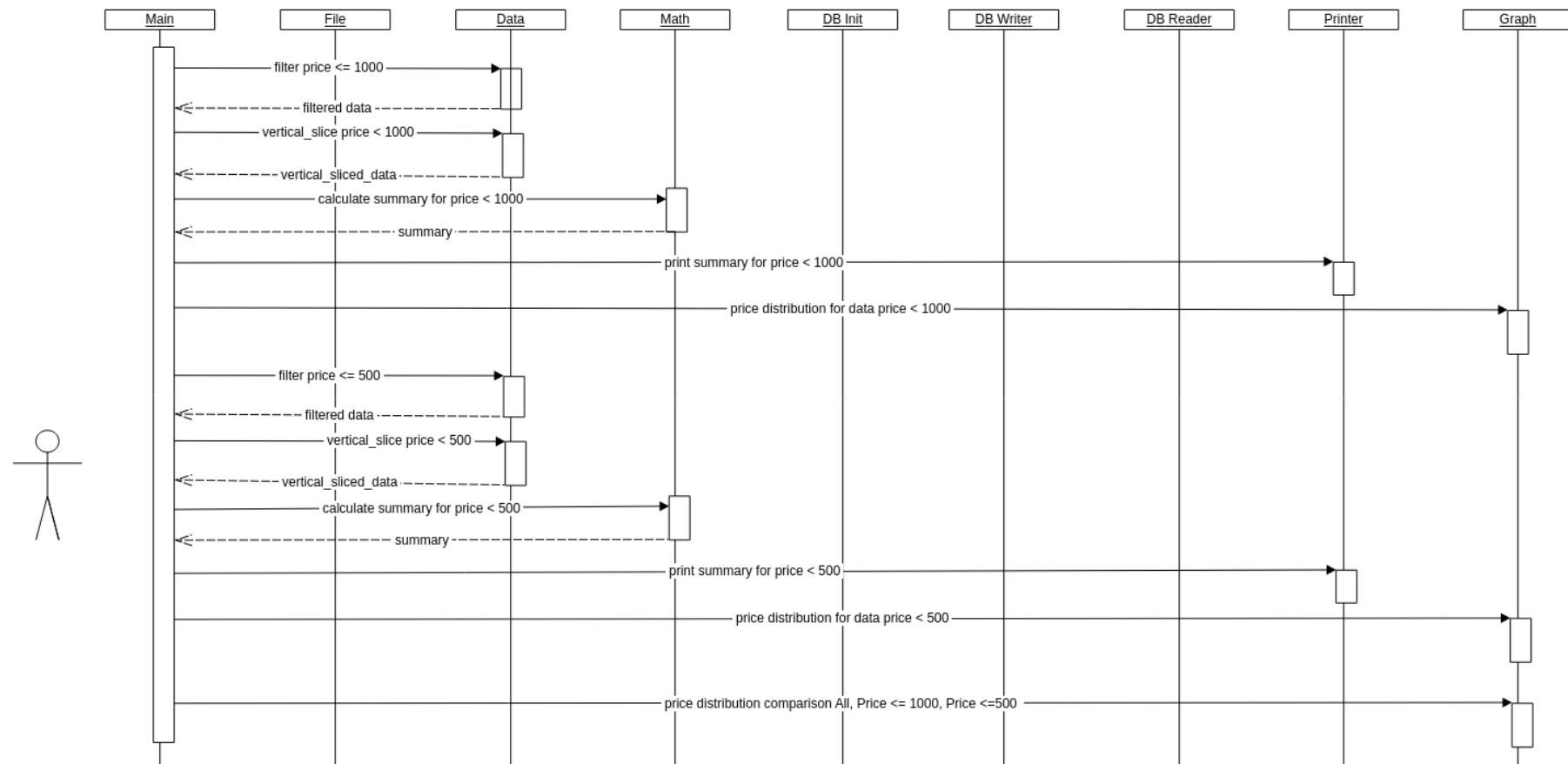


Figure 6 Sequence Diagram Selecting Price Range 2

Analysing data Price < 500

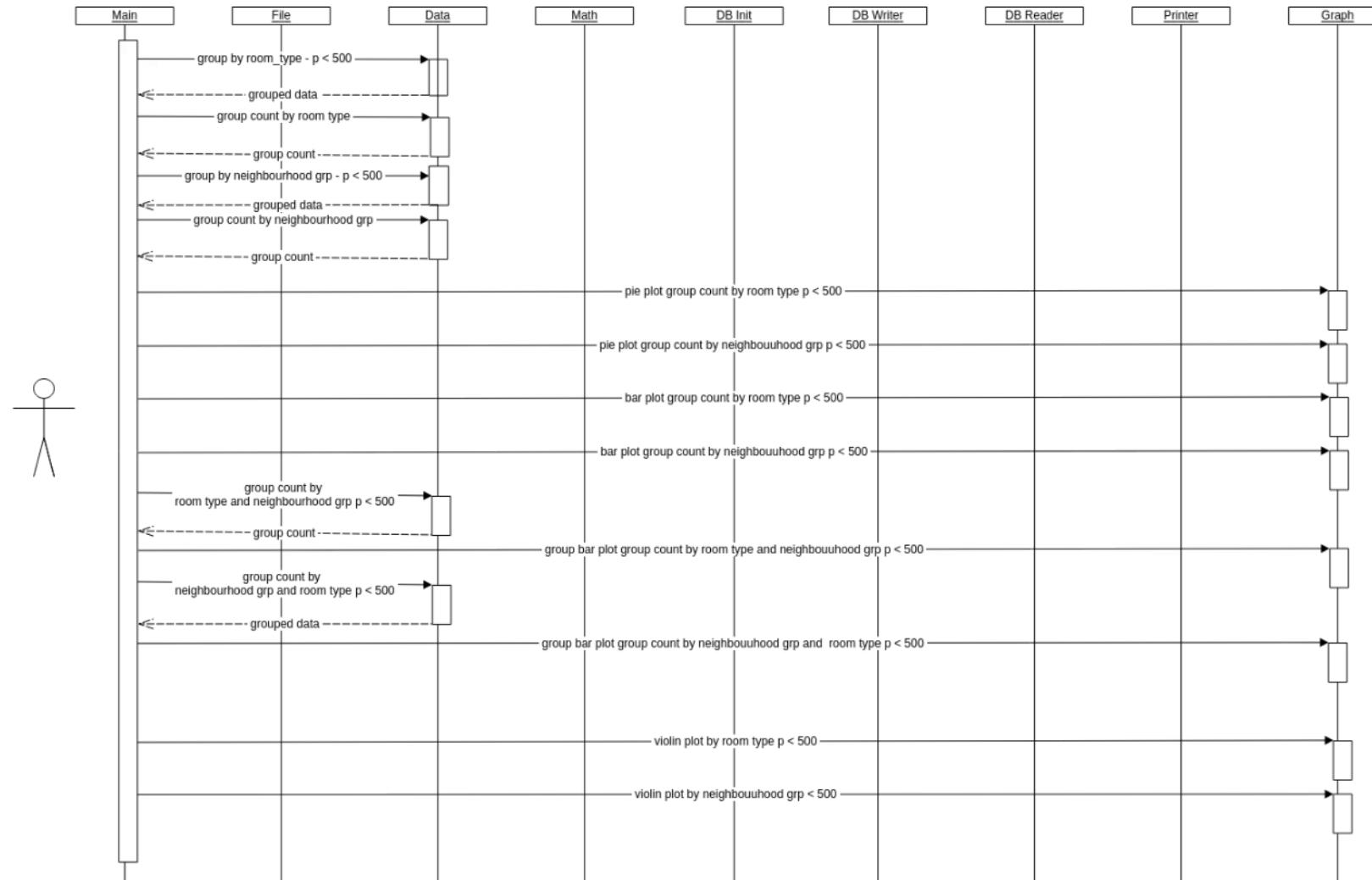


Figure 7 Sequence Diagram Analysing data Price > 500

Analysing data Price > 1000

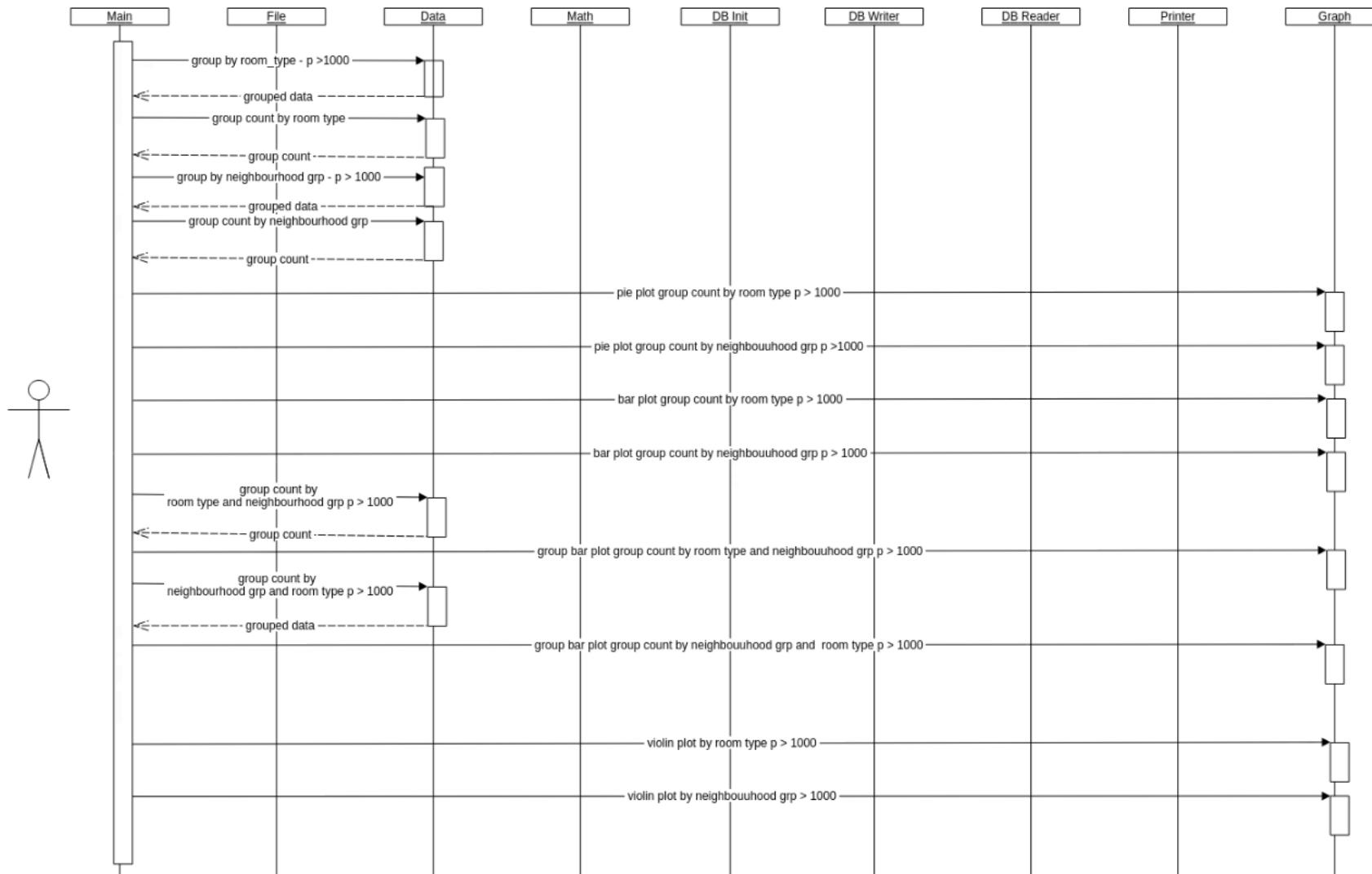


Figure 8 Sequence Diagram Analysing data Price > 1000

Analysing data Price > 5000

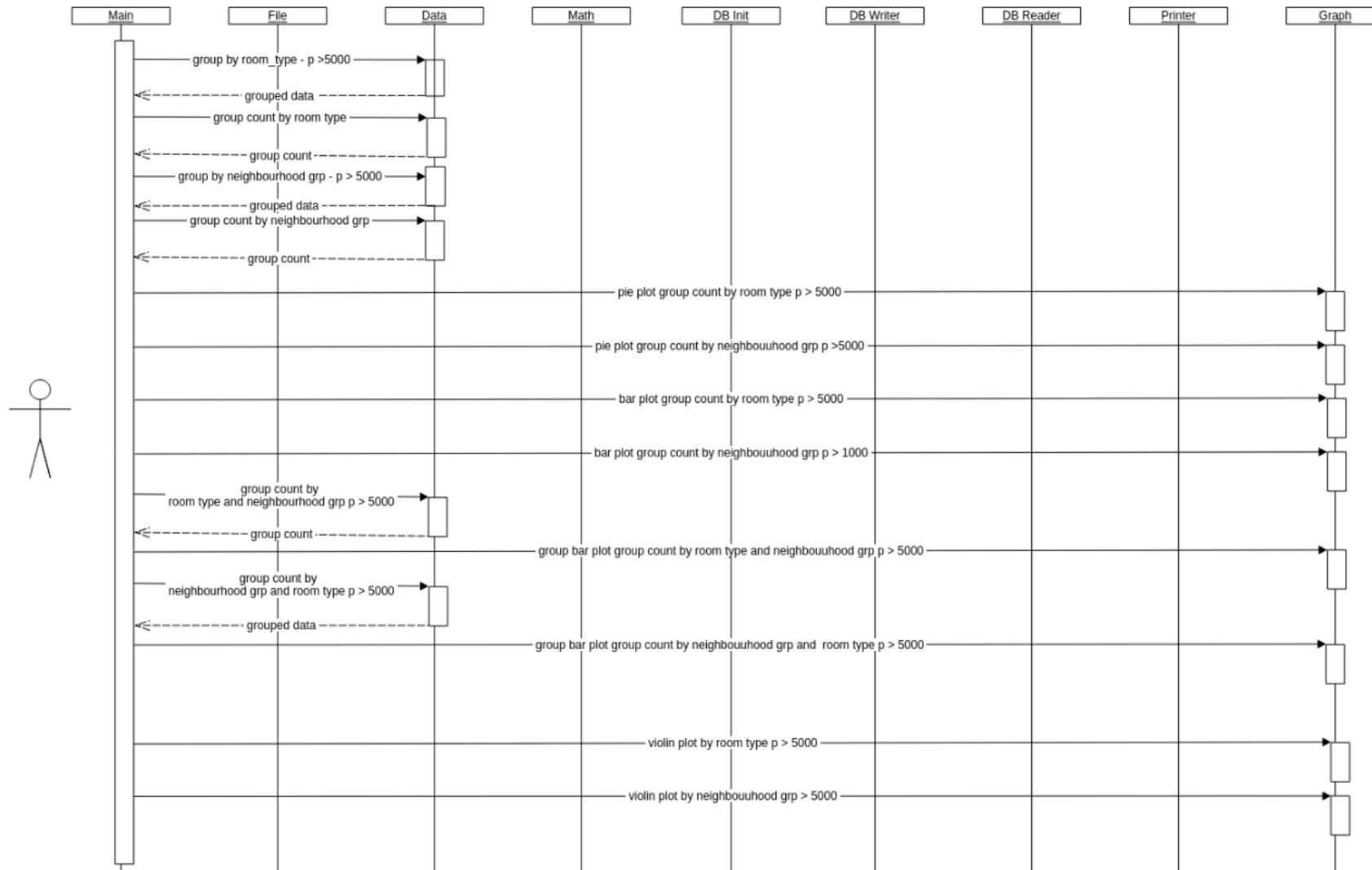


Figure 9 Sequence Diagram Analysing data Price > 5000

Availability = 0, Availability > 0

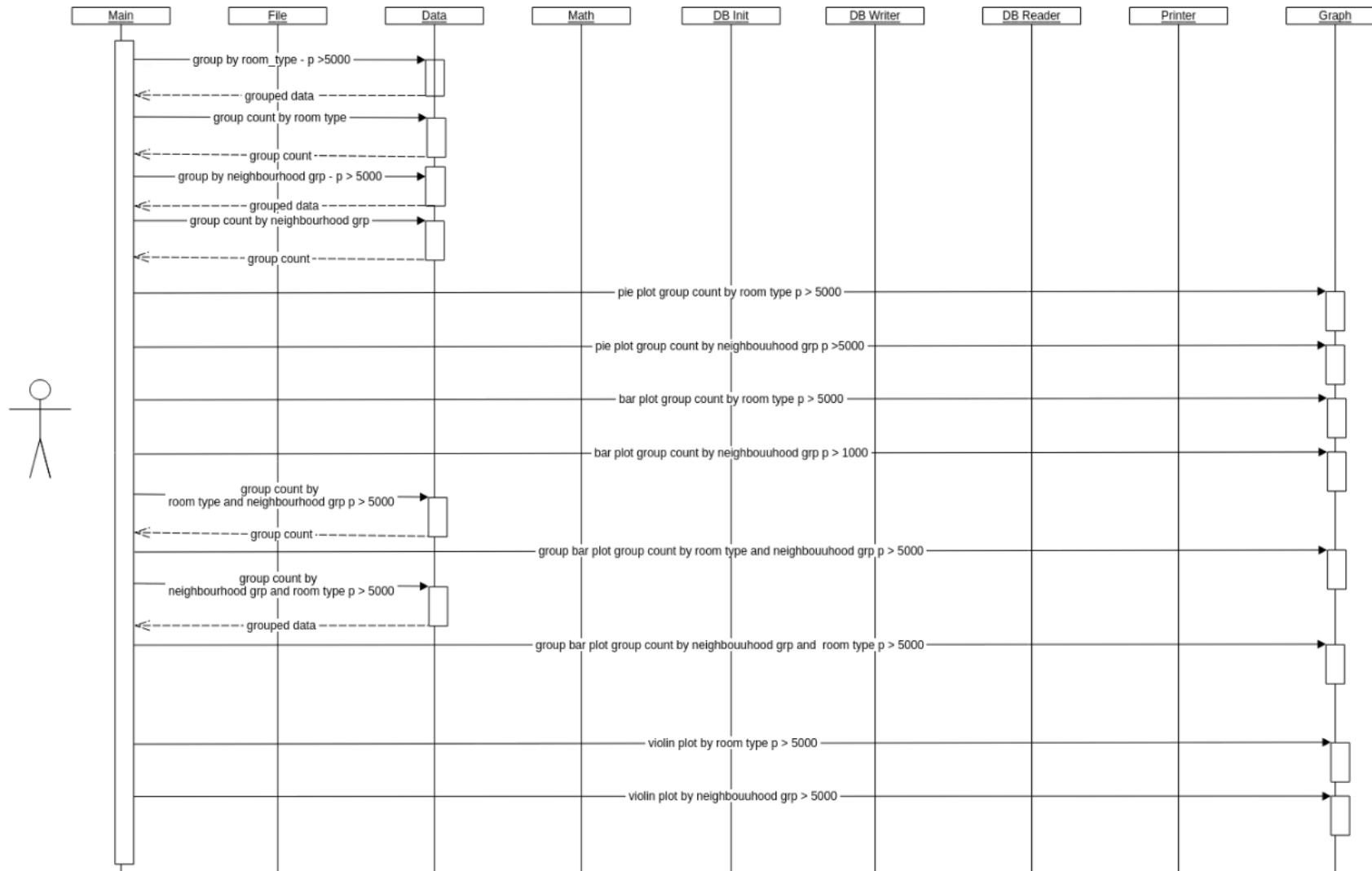


Figure 10 Sequence Diagram Availability = 0, Availability >

Testing

Unittest has been used for testing main modules. 100%-line coverage has been recorded.



Figure 11 Unit Test Coverage

To run all the unit test from the root directory of the project run following command

```
python -m unittest
```

To verify the calculations are correct, numpy, panda, stat modules has been used only in unit testing verification

```
(venv) C:\Users\camara\PycharmProjects\as1>python -m unittest  
.....  
-----  
Ran 31 tests in 0.718s  
  
OK  
  
(venv) C:\Users\camara\PycharmProjects\as1>
```



Figure 12 Running all unittests

Analysis and Visualisation

1. File Reading summary

After initial file reading program generate following report to illustrate Total Valid Rows, Skipped Rows and Total number of rows. This execution is similar to pandas describe method

	Count	%
Valid Rows	41707	84.978
Skipped Rows	7373	15.022
Total Reads	49080	100
Summary of file read		

Figure 13 Summary of file read

2. Data Head

Display first 10 rows of valid data. This functionality is similar to pandas head

Printing First 10 lines of data set													
name	host_id	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	availability
Clean & quiet apt home by the park	2787	1	1	40.647	-73.972	Entire home/apt	210	1	9	2018-10-19	0.218	6	365
Skylit Midtown Castle	2845	2	2	40.754	-73.984	Entire home/apt	225	1	45	2019-05-21	0.388	2	355
THE VILLAGE OF HARLEM...NEW YORK !	4632	1	2	40.889	-73.942	Entire home/apt	150	3	0		0	1	365
Cozy Entire Floor of Brownstone	4869	2	1	40.685	-73.968	Entire home/apt	89	1	270	2019-07-05	4.648	1	194
Entire Apt: Spacious Studio/Loft by central park	7193	2	2	40.799	-73.944	Entire home/apt	80	10	9	2018-11-19	0.109	1	0
Large Cozy 1 BR Apartment In Midtown East	7322	2	2	40.748	-73.975	Entire home/apt	200	3	74	2019-06-22	0.590	1	129
AliSportsSpace!	7356	1	1	40.687	-73.956	Entire home/apt	60	45	49	2017-10-05	0.408	1	0
Large Furnished Room Near B'way	8967	1	2	40.765	-73.985	Entire home/apt	79	2	438	2019-06-24	3.478	1	228
Cozy Clean Guest Room - Family Apt	7490	1	2	40.892	-73.967	Entire home/apt	79	2	118	2017-07-21	0.998	1	0
Cute & Cozy Lower East Side 1 bdrm	7549	2	2	40.713	-73.998	Entire home/apt	150	1	168	2019-06-09	1.338	4	188

Figure 14 Data Head

3. Data Summary

Print count, min, mean, mode, median, sd, max all numerical fields.

Field	count	min	mean	mode	median	sd	max
latitude	41707	40.500	40.729	[(40.71 40.723 813, 17)]	40.723	0.054	40.913
longitude	41707	-74.244	-73.953	[(-73.9 -73.956 5427, 15), (-73.9540 5, 15), (-73.95 136, 15)]	-73.956	0.046	-73.713
price	41707	0	153.790	[(100.0 109 , 1771)]	109	243.437	10000
minimum_nights	41707	1	6.957	[(1, 2 10987)]	2	20.920	1250
number_of_reviews	41707	0	23.185	[(0, 5 8678)]	5	44.647	629
reviews_per_month	41707	0	1.090	[(0, 0.370 8678)]	0.370	1.602	58.500
calculated_host_listings_count	41707	1	6.954	[(1, 1 27459)]	1	32.559	327
availability_365	41707	0	113.778	[(0, 46 14972)]	46	132.186	365

Figure 15 Data Summary

4. Pair Scatter Plots

Pair scatter plots are nice feature of seaborn plots to quickly visualize the correlation of a variable against all other variables. But seaborn only support numpy's dataframes. Introduced an adapter to generate pair scatter plots from generic list of array.

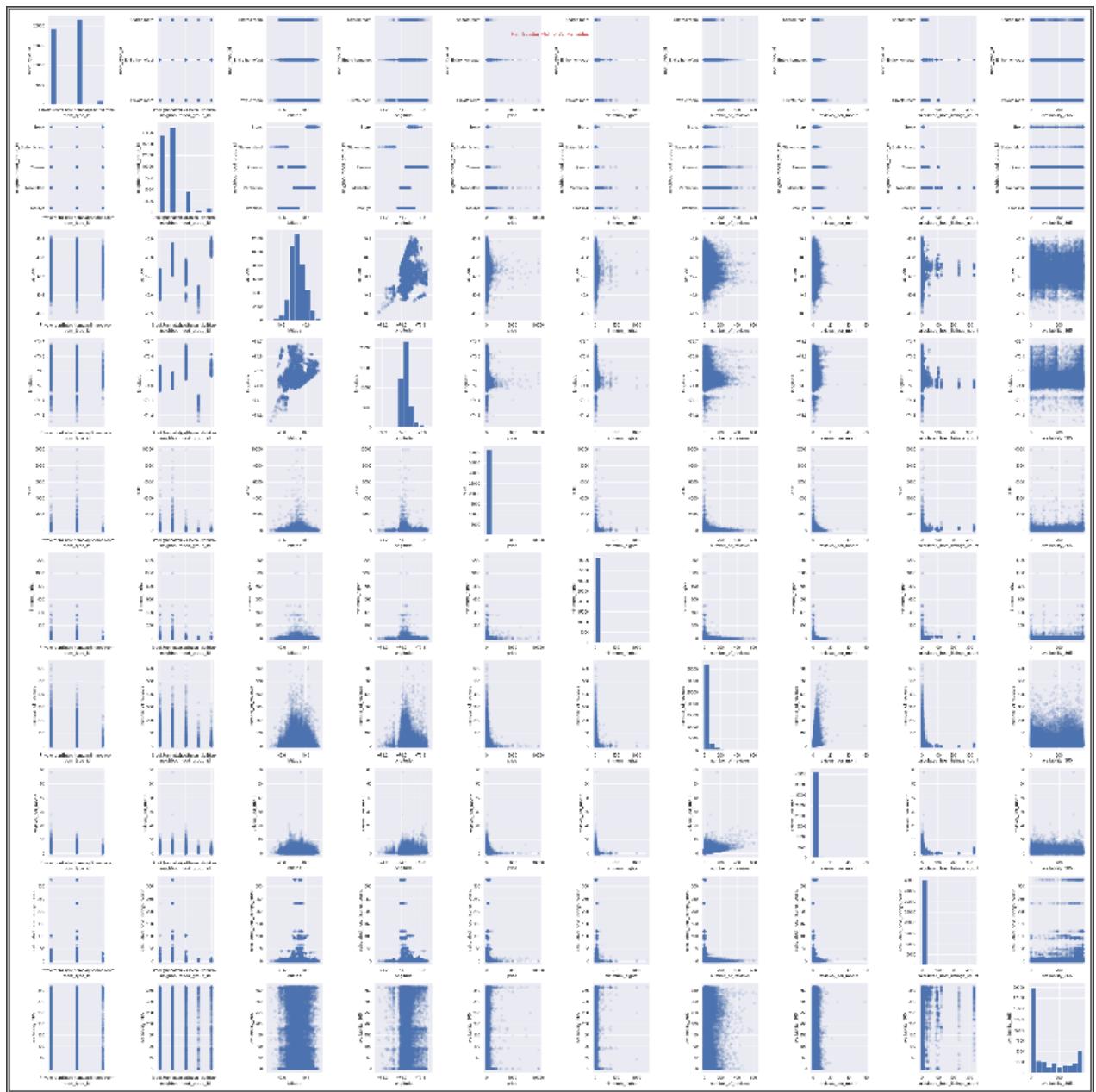


Figure 16 Pair Scatter Plots



Figure 17 Pair Scatter Plots with Correlations

5. It was decided to focus on price data. Print price summary

Field	count	min	mean	mode	median	sd	max
price	41707	0	153.790	[(100.0, 1771)]	109	243.437	10000

Figure 18 Price Summary

6. Price Distribution graph for all data



Figure 19 Price Distribution - All Data

The graph says the price distribution is between 0 and \$1000. Extreme Price listings need to be Filtered.

7. Consider Price less than or equal 1000. Price Summary

Field	count	min	mean	mode	median	sd	max
price	41501	0	142.083	[(100.0, 1771)]	106	118.224	1000

Figure 20 Price Summary. Price <=1000

Standard deviation has been reduced to 118.224

8. Price Distribution for Price <= 1000



Figure 21 Price Distribution. Price <= 1000

9. Considering listings where price <= 500

Field	count	min	mean	mode	median	sd	max
price	40789	0	131.906	[(100.0, 1771)]	103	88.388	500

Figure 22 Price Summary. Price <=500

10. Price Distribution for Price <= 500



Figure 23 Price Distribution. Price ≤ 500

11. Price Distribution comparison for All Data, and where Price $\leq \$1000$ and Price $\leq \$500$

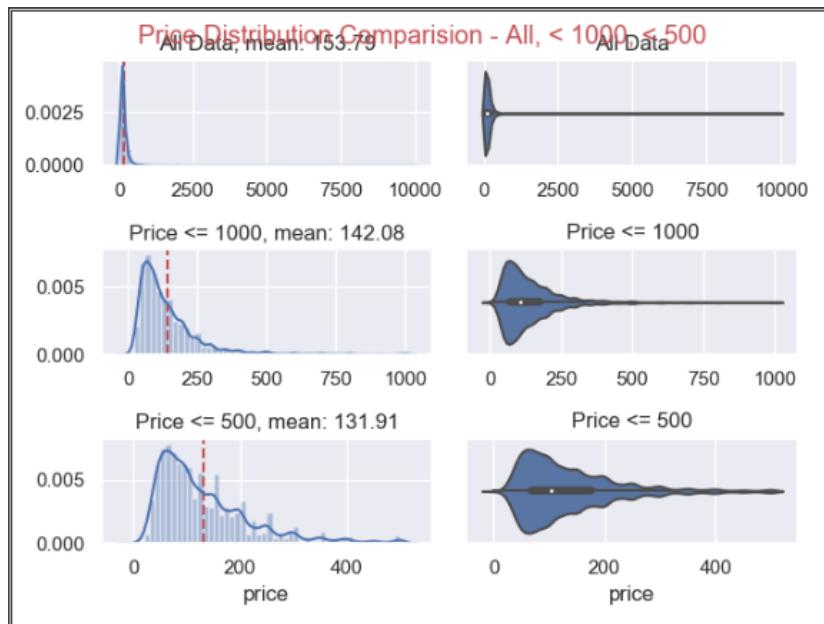


Figure 24 Price Distribution comparison

Considering Price Distribution comparison graph it was decided to analysis data separately where Listing price is 0 to \$500 and above \$500

12. Room listing by Room Type (Price <= \$500)

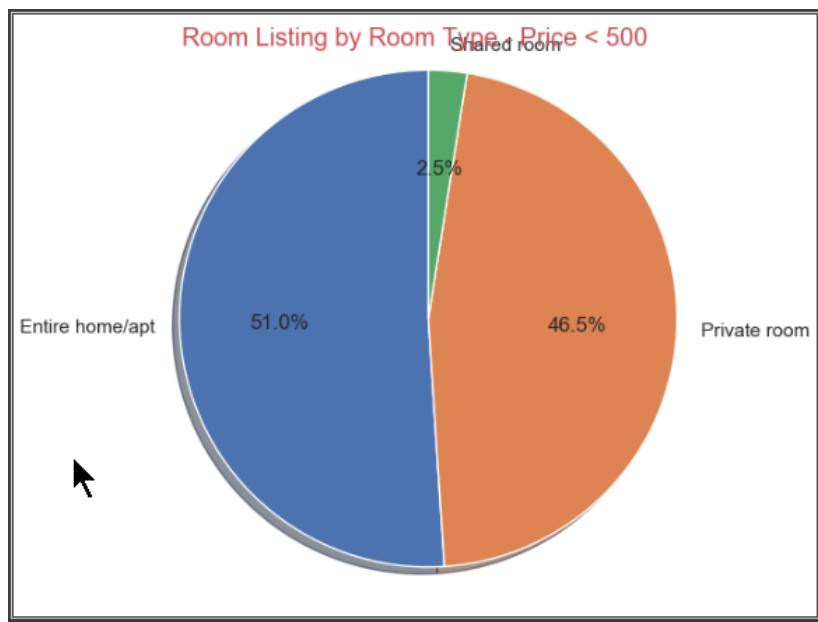


Figure 25 Room Listing by Room Type, Pie Chart (Price < 500)

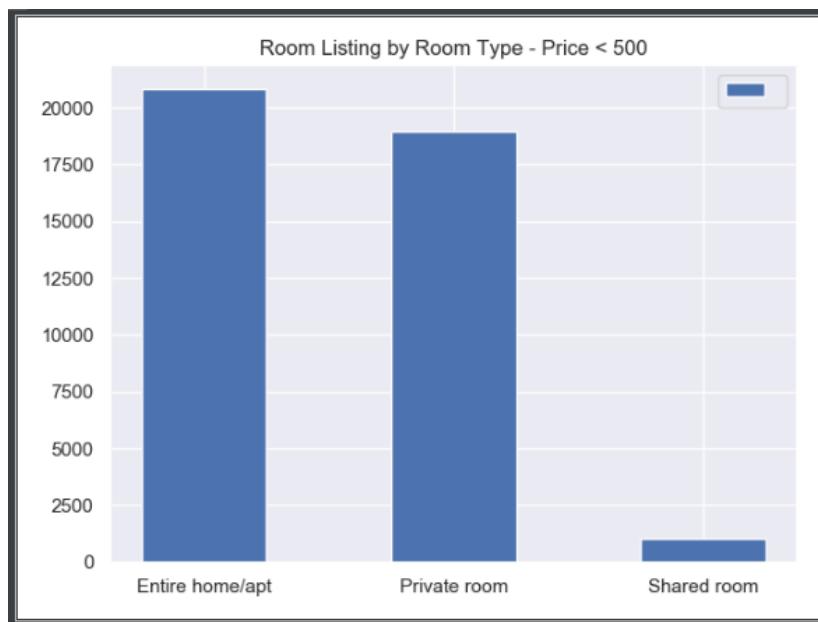


Figure 26 Room Listing by Room Type, Count Bar Chart (Price < 500)

13. Room listing by Neighbourhood Group (Price <= \$500)

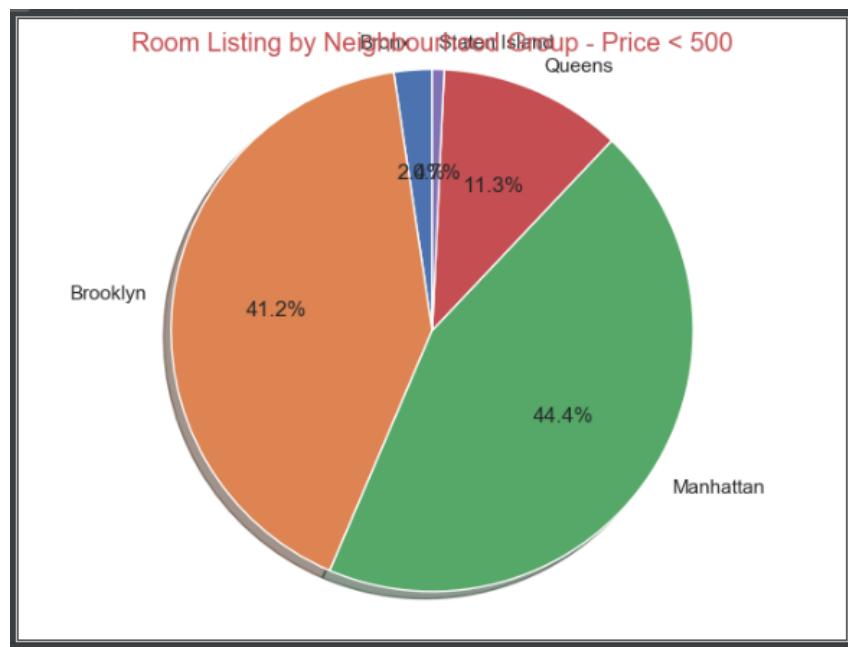


Figure 27 Room Listing by Neighbourhood Group, Pie Chart (Price < 500)

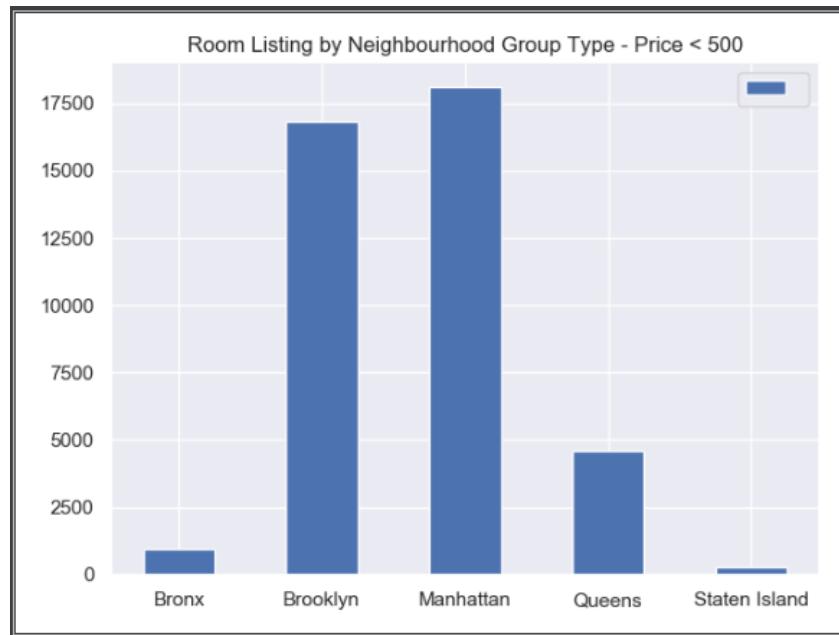


Figure 28 Room Listing by Neighbourhood Group, Count Bar Chart (Price < 500)

14. Room listing by Room Type and Neighbourhood Group (Price <=\$ 500)

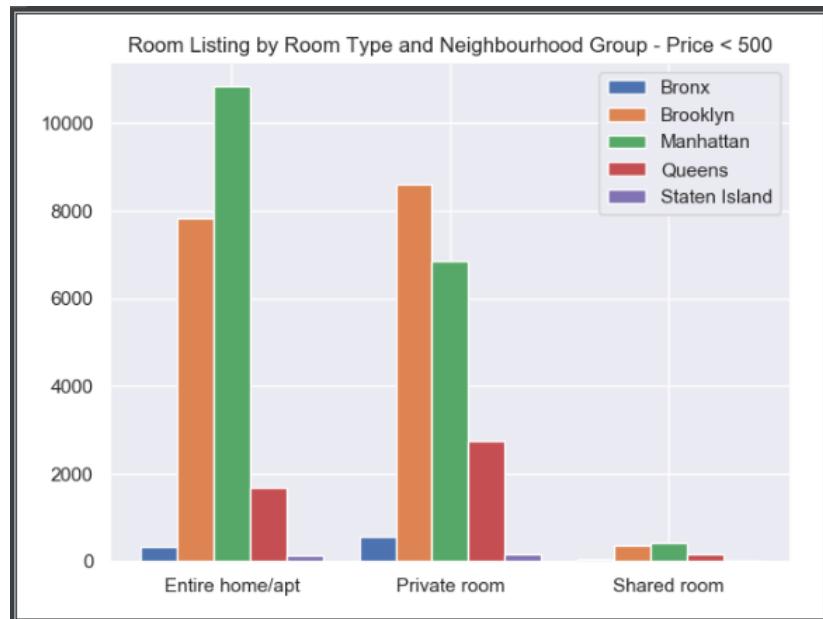


Figure 29 Room listing by Room Type and Neighbourhood Group (Price <=\$ 500)

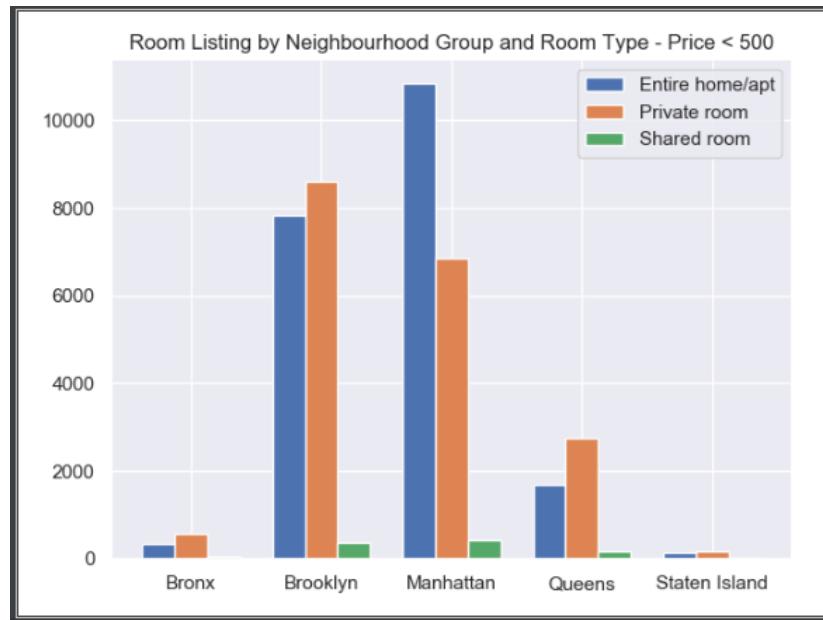


Figure 30 Room listing by Neighbourhood Group and Room Type (Price <=\$ 500)

15. Price KDE by Room Type (Price \leq \$500)

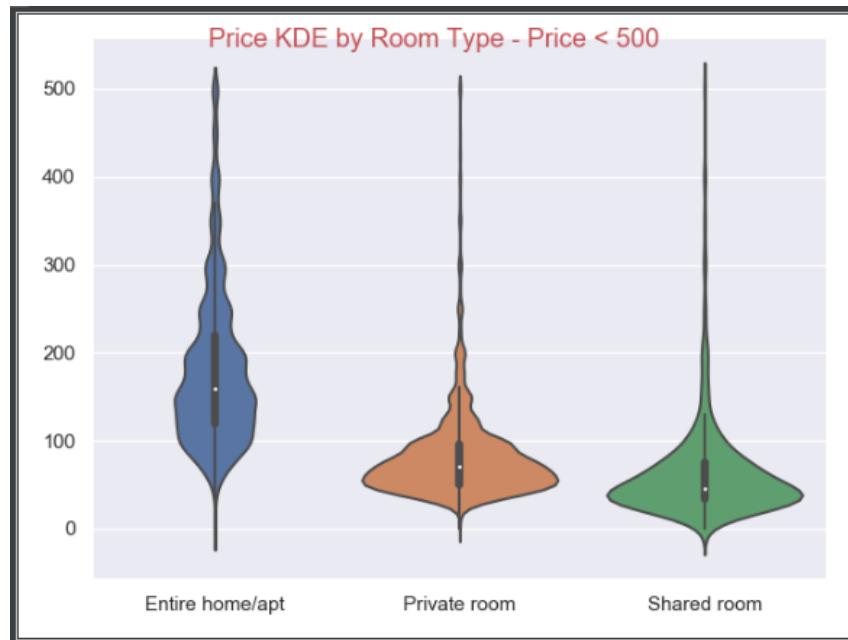


Figure 31 Price KDE by Room Type (Price \leq \$500)

16. Price KDE by Neighbourhood Group (Price \leq \$500)



Figure 32 Price KDE by Neighbourhood Group (Price \leq \$500)

17. Price Analysis. Price > 1000 and Price > 5000

Field	count	min	mean	mode	median	sd	max
price	206	1002	2512.204	[(2000.0, 23)]	1939	1895.429	10000
Field	count	min	mean	mode	median	sd	max
price	23	5000	7066.087	[(5000.0, 6)]	6500	1860.567	10000

Figure 33 Price Summary. Price > 1000, Price > 5000

Price Distribution - Price greater than \$1000 Price Distribution - Price greater than \$5000

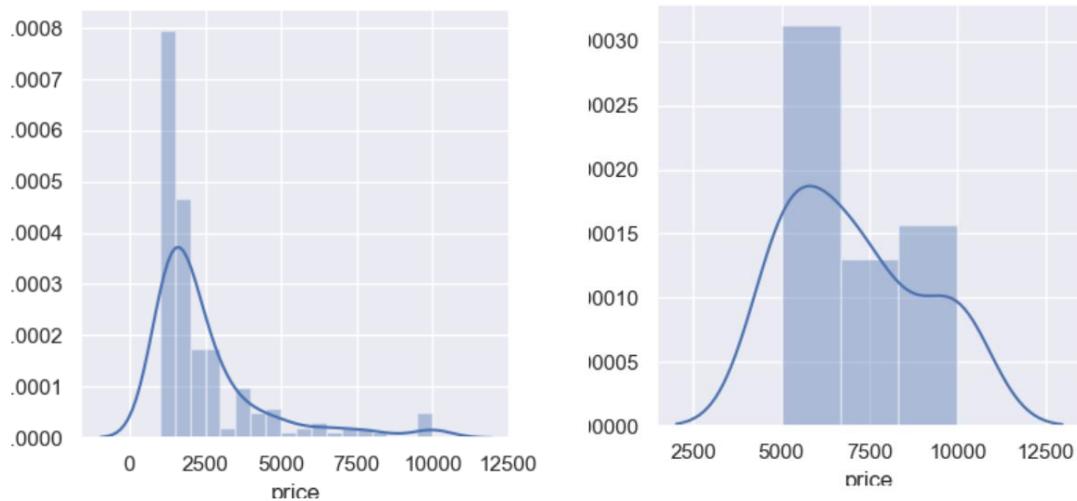


Figure 34. Price Distribution. Price > \$1000, Price > \$5000

Price Distribution Comparison - All, < 500, > 1000, > 5000

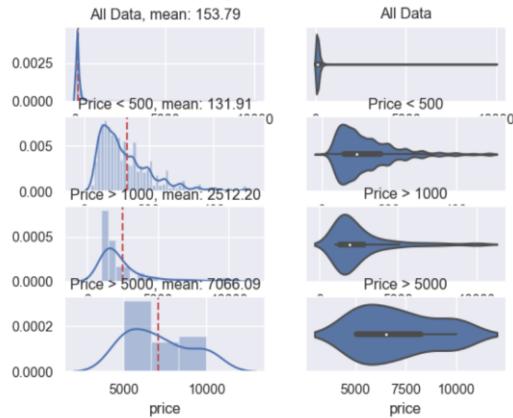


Figure 35 Price Comparison. All Price, Price <\$500, Price > \$1000, Price > \$5000

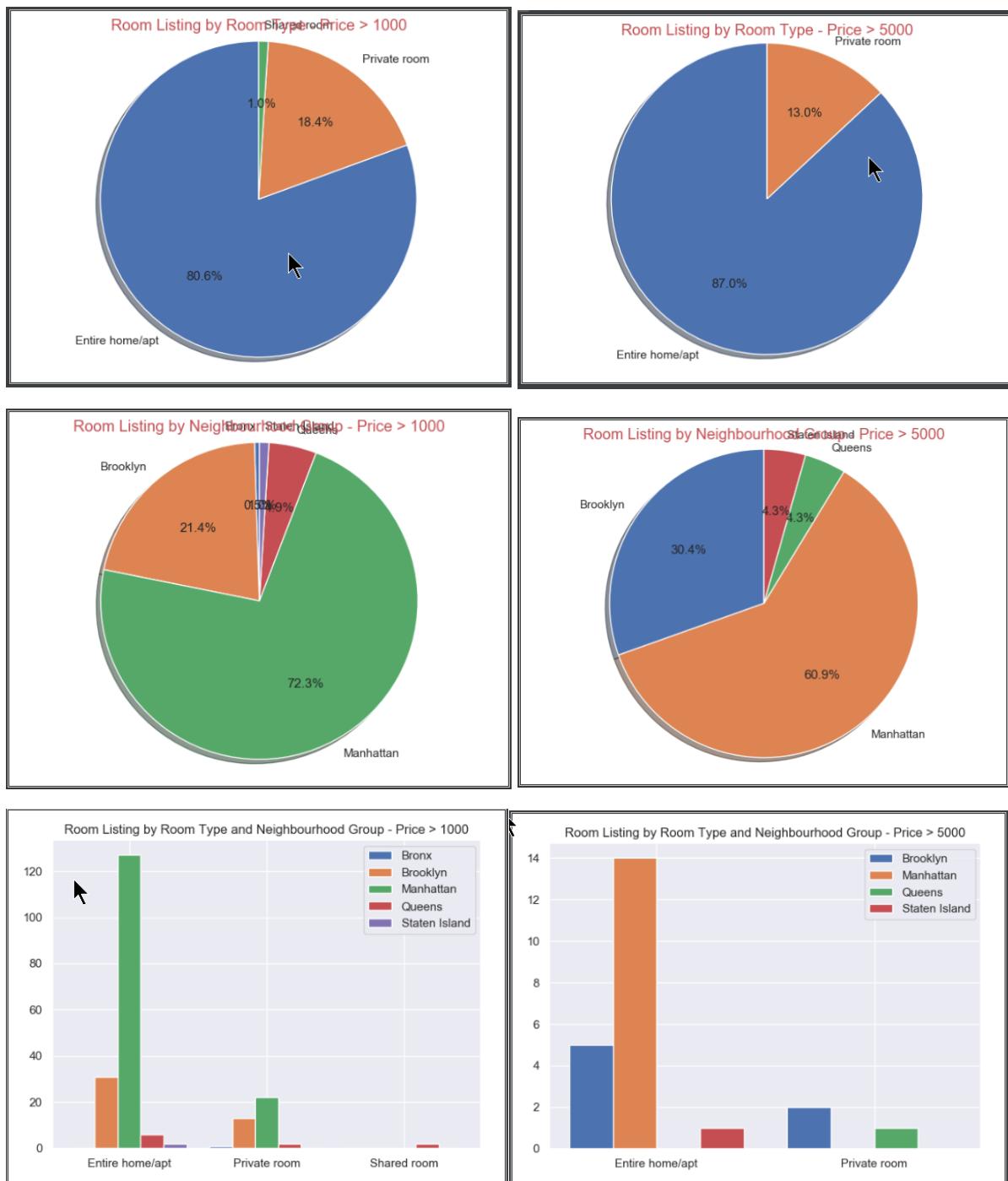


Figure 36 Price Grouping

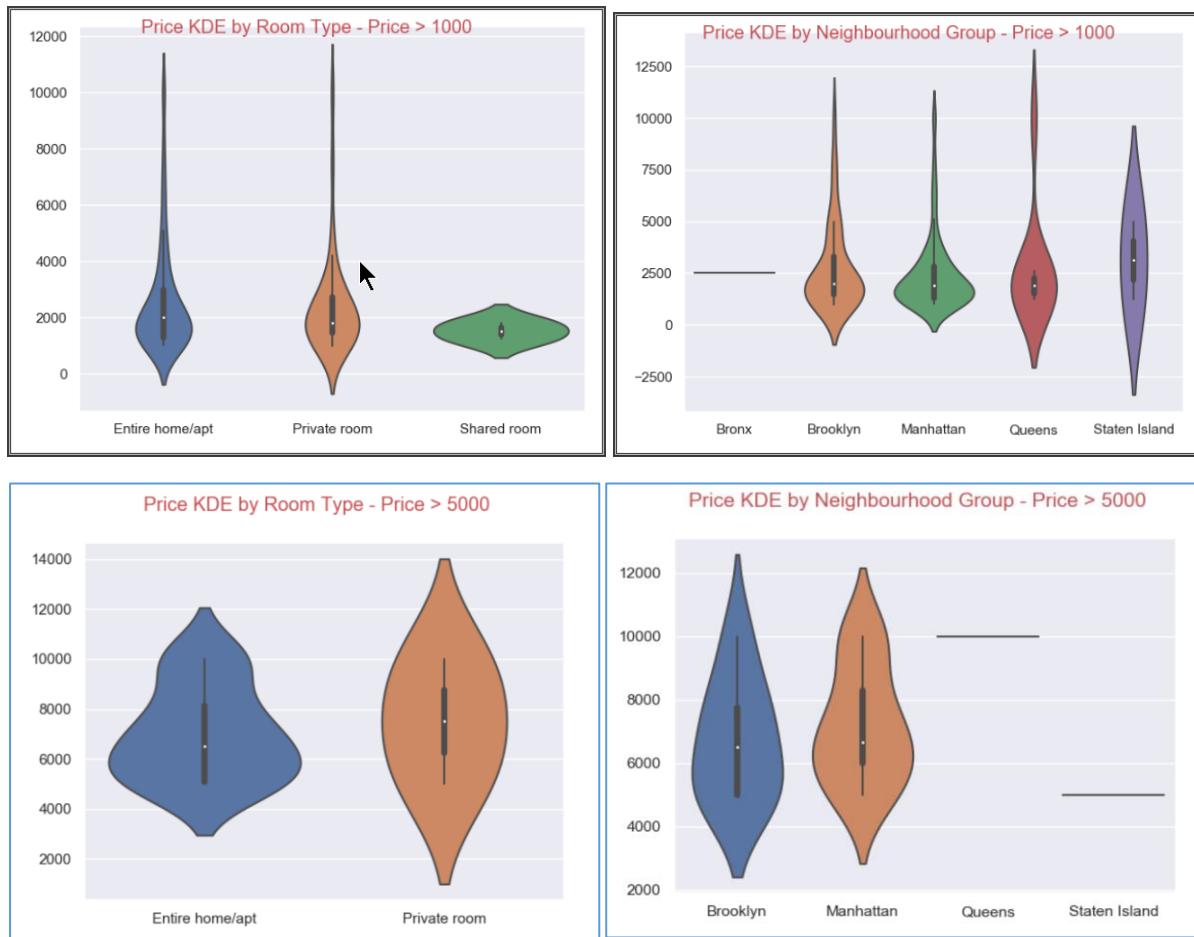


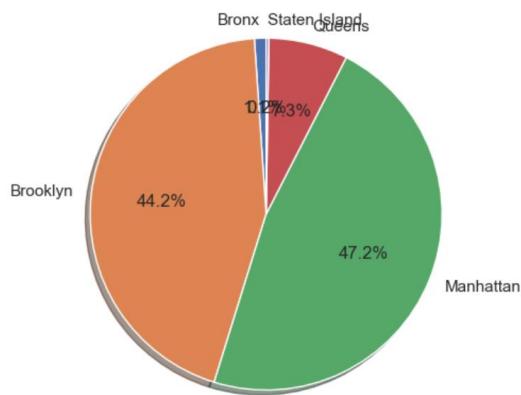
Figure 37 Price Grouping

18. Price Comparison Availability 0 vs Availability > 0

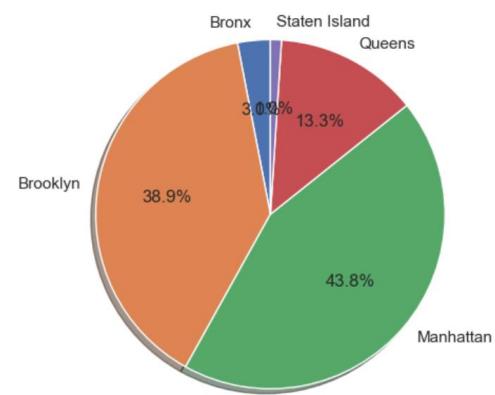
Field	count	min	mean	mode	median	sd	max
price	14972	0	137.848	[(100.0, 748)]	100	218.864	10000
price	26735	0	162.717	[(100.0, 1023)]	113	255.738	10000

Figure 38 Price Comparison for Availability

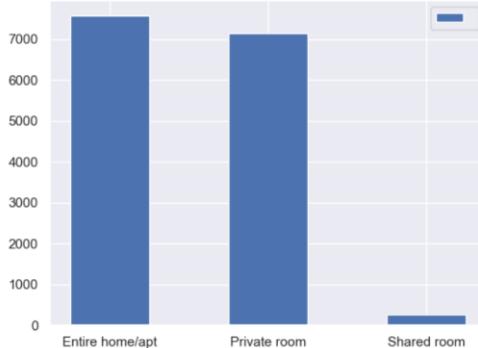
Room Listing by Neighbourhood Group - Availability = 0



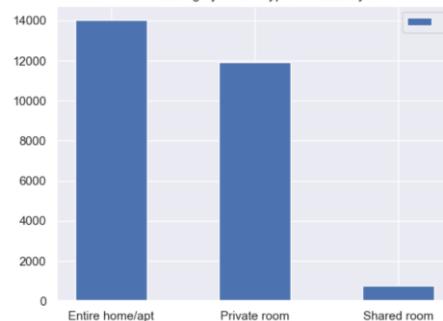
Room Listing by Neighbourhood Group - Availability > 0



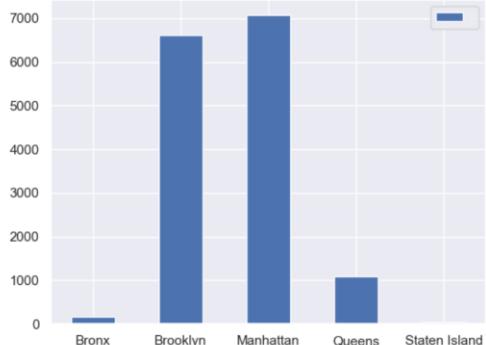
Room Listing by Room Type - Availability = 0



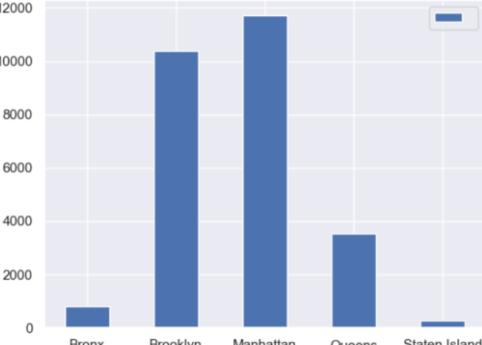
Room Listing by Room Type - Availability > 0



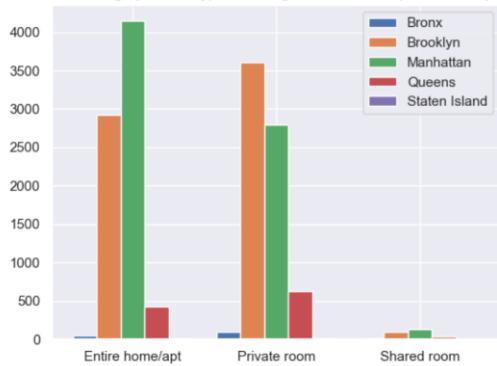
Room Listing by Neighbourhood Group Type - Availability = 0



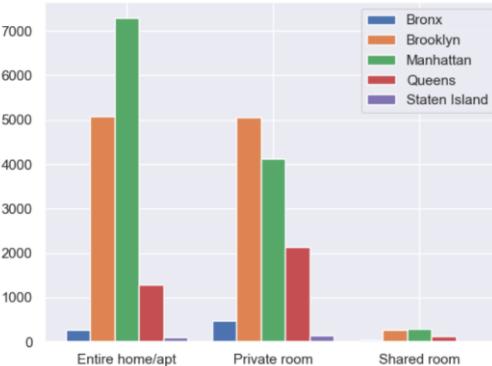
Room Listing by Neighbourhood Group Type - Availability > 0

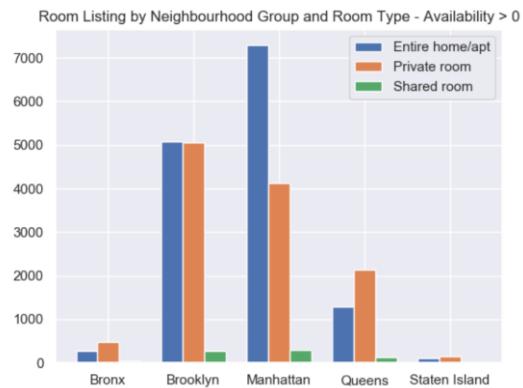


Room Listing by Room Type and Neighbourhood Group - Availability = 0

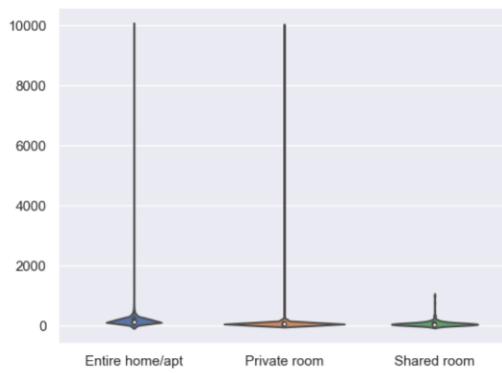


Room Listing by Room Type and Neighbourhood Group - Availability > 0

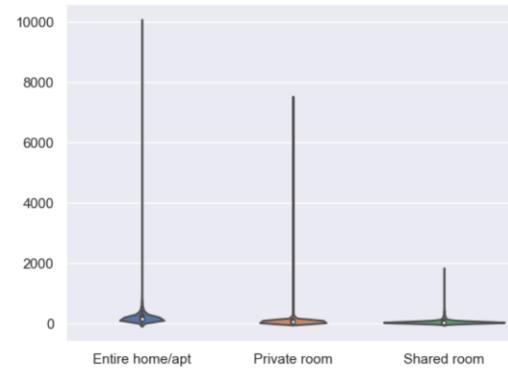




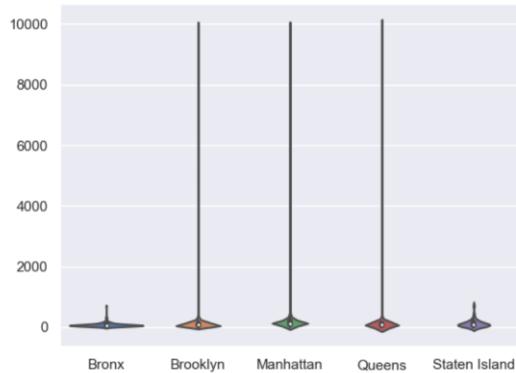
Price KDE by Room Type - Availability = 0



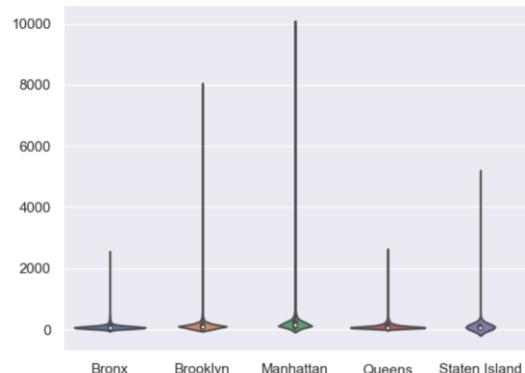
Price KDE by Room Type - Availability > 0



Price KDE by Neighbourhood Group - Availability = 0

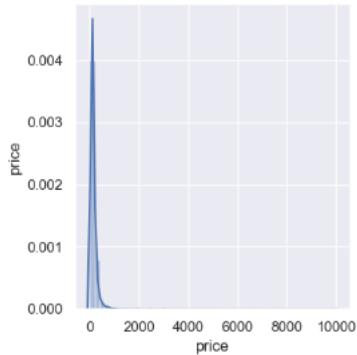


Price KDE by Neighbourhood Group - Availability > 0

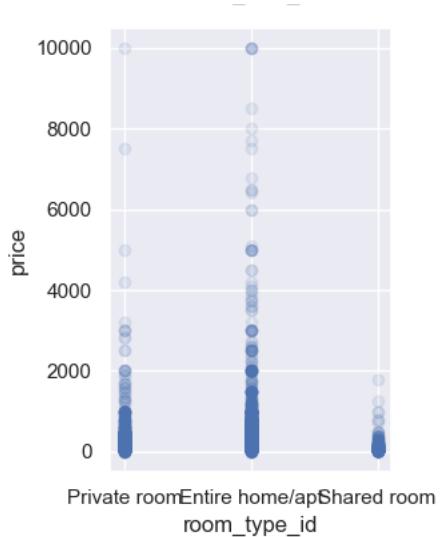


Conclusion

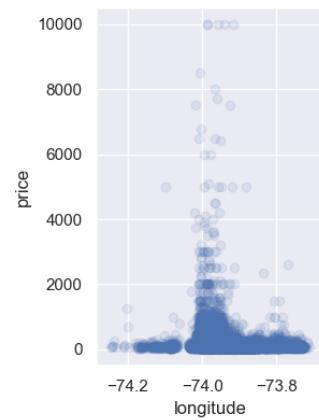
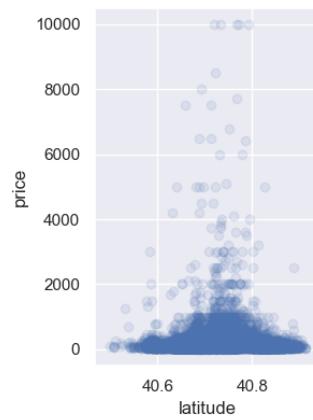
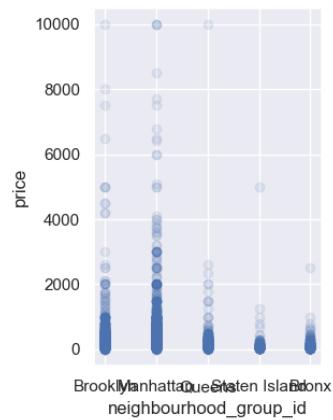
1. Most of the price listing are lower than \$2000



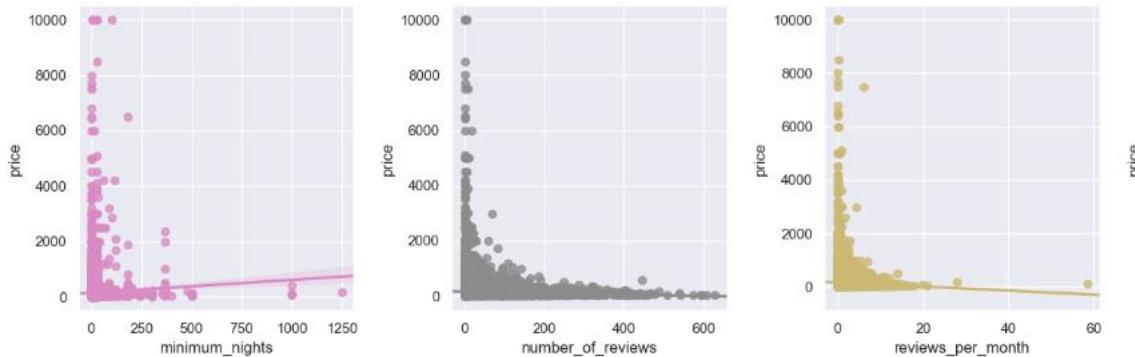
2. Room type has a direct relationship to the price. Entire Home / apt has highest price and always Shared rooms price is lower than \$2000.



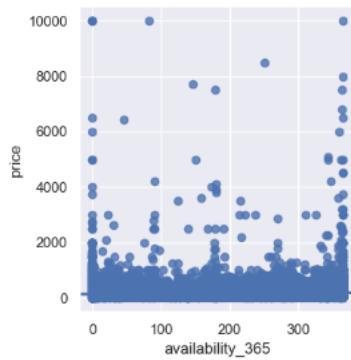
3. The location of the listing has a strong relation to the price. Manhattan and Brooklyn properties have been listed with a higher price.



4. Price has a positive correlation with minimum number of nights, while it has a negative correlation with Number of reviews and Number of reviews per month. The reason could be low price attracts higher number of tenants.



5. Below graph tries to find the relationship between Price and Availability. There is no clear relationship between Price and Availability.



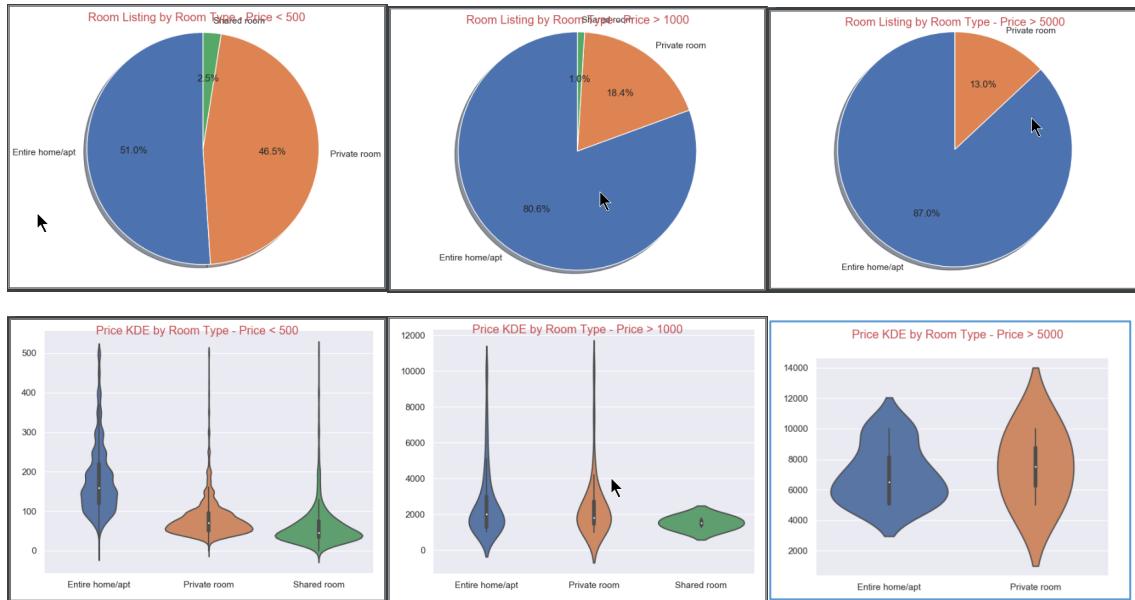
6. More than 97% listing's price is lower than \$500

Field	count	min	mean	mode	median	sd	max
price	41707	0	153.790	[(100.0, 1771)]	109	243.437	10000
price	40789	0	131.906	[(100.0, 1771)]	103	88.388	500

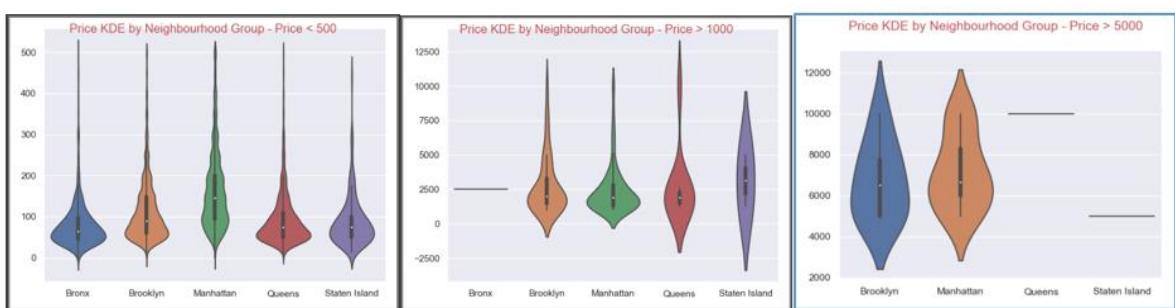
7. Only 0.00006% (23 out of 41707) listings are priced over \$5000

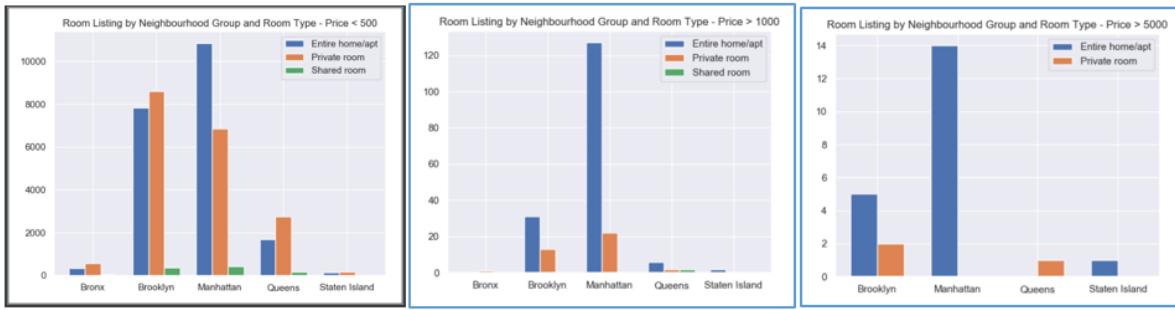
Field	count	min	mean	mode	median	sd	max
price	23	5000	7066.087	[(5000.0, 6)]	6500	1860.567	10000

8. Entire Home / apt are popular in all price categories.
9. But when price goes up Entire Home / apt are more popular.
10. No any Shared rooms which are priced over \$5000



11. Manhattan has the highest median for Price in all 3 price categories, followed by Brooklyn
12. Manhattan has the highest listing count followed by Brooklyn
13. In \$500 price category, Private rooms are much popular in all neighbourhood groups, Except Manhattan where Entire Home / Apt has highest listing.
14. For over \$5000 category, Private Room nor Shared Room available in Manhattan.





15. Nearly 40% of Total listing are not available.

Field	count	min	mean	mode	median	sd	max
price	14972	0	137.848	[(100.0, 748)]	100	218.864	10000

Appendix 1: Reflective Learning Log

Date: 25/10/2019

Work Completed: Init project. Setup git for version controlling

Understanding Achieved: Understanding python project structure.

```
(venv) C:\Users\camara\PycharmProjects\as11>git init
Initialized empty Git repository in C:/Users/camara/PycharmProjects/as11/.git/

(venv) C:\Users\camara\PycharmProjects\as11>
```

```
(venv) C:\Users\camara\PycharmProjects\as11>git log --pretty=oneline
fatal: your current branch 'master' does not have any commits yet

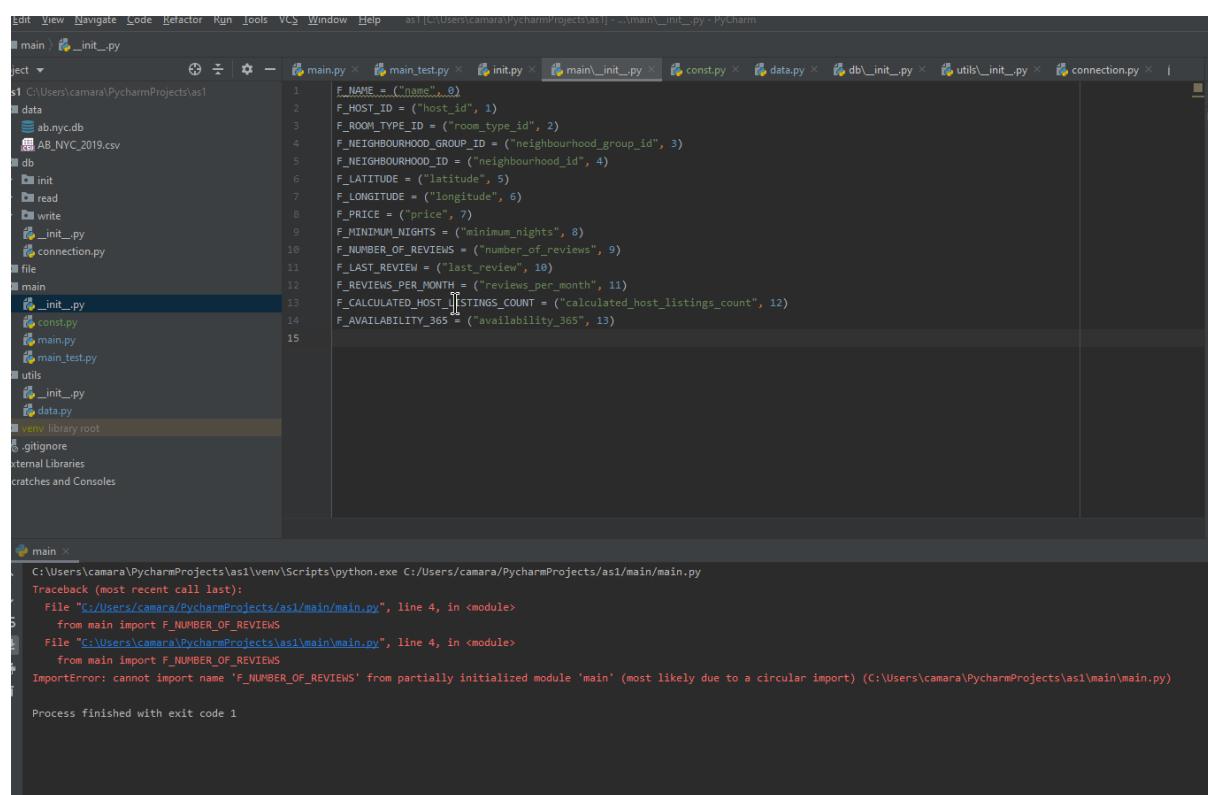
(venv) C:\Users\camara\PycharmProjects\as11>
```

Figure 39 Setting up git repository

Date: 28/10/2019

Work Completed: Module Structure

Understanding Achieved: Understanding Python Modular programming and importing and installing 3rd party modules



The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure with files like main.py, db.py, and const.py. The main editor window shows a portion of main.py with code defining constants for room features. Below the editor, a terminal window shows a stack trace for a circular import error:

```
C:\Users\camara\PycharmProjects\as1\venv\Scripts\python.exe C:/Users/camara/PycharmProjects/as1/main/main.py
Traceback (most recent call last):
  File "C:/Users/camara/PycharmProjects/as1/main/main.py", line 4, in <module>
    from main import F_NUMBER_OF_REVIEWS
  File "C:/Users/camara/PycharmProjects/as1/main/main.py", line 4, in <module>
    from main import F_NUMBER_OF_REVIEWS
ImportError: cannot import name 'F_NUMBER_OF_REVIEWS' from partially initialized module 'main' (most likely due to a circular import) (C:/Users/camara/PycharmProjects/as1/main/main.py)

Process finished with exit code 1
```

Figure 40 Facing Circular import issue

The screenshot shows the PyCharm IDE interface. The left pane displays a project structure for a Python application named 'as1'. The main file 'main.py' is open in the editor. The code attempts to import 'main.const' but fails because it is not a package. The terminal window at the bottom shows the execution of 'main.py' and the resulting traceback.

```

Project: as1
  - data
    - ab.nyc.db
    - AB_NYC_2019.csv
  - db
    - init
    - read
    - write
      - __init__.py
      - connection.py
  - file
  - main
    - __init__.py
    - const.py
    - main.py
    - main_test.py
  - utils
    - __init__.py
    - data.py
  - venv library root
    - .gitignore
  - External Libraries
  - Scratches and Consoles

main.py
1  from file.file_read import process_data_file
2  from db.init.init import create_all_tables
3  from db.write import write
4  from main.const import F_NUMBER_OF_REVIEWS
5
6  data_file = process_data_file()
7  print(F_NUMBER_OF_REVIEWS)
8  # create_all_tables()
9  # write.insert_room_types(data_file["room_types"])
10 # write.insert_neighbourhood_groups(data_file["neighbourhood_groups"])
11 # write.insert_neighbourhoods(data_file["neighbourhoods"])
12 # write.insert_price_data(data_file["price_data"])

main
C:\Users\camara\PycharmProjects\as1\venv\Scripts\python.exe C:/Users/camara/PycharmProjects/as1/main/main.py
Traceback (most recent call last):
  File "C:/Users/camara/PycharmProjects/as1/main/main.py", line 4, in <module>
    from main.const import F_NUMBER_OF_REVIEWS
  File "C:\Users\camara\PycharmProjects\as1\main\main.py", line 4, in <module>
    from main.const import F_NUMBER_OF_REVIEWS
ModuleNotFoundError: No module named 'main.const'; 'main' is not a package

Process finished with exit code 1

```

Figure 41 Fixing Module not found issue

Date: 29/10/2019

Work Completed: File Reading

Understanding Achieved: Read CVS file. Encoding issues.

```

30     with open(file) as data_file:
31         read_header = False # has header read
32         for line in data_file:
33             line = line.strip()
34             comma_separated_line = line.split(",")
35             if read_header: # reading normal data row
36                 if len(comma_separated_line) == 16: # there should be exactly 16 data items
37                     process_line_(comma_separated_line, data_items) # process line
38             else:
39                 data_items["ab_skipped"].append(comma_separated_line) # skipped otherwise
40             else: # first row as a header
41                 if not len(comma_separated_line) == 16:
42                     raise FileNotFoundError
43             data_items["ab_headers"].append(comma_separated_line)
44
45     process_data_file() > try > with open(file) as data_file

```

(venv) C:\Users\camara\PycharmProjects\as1>python main.py
Reading data file
Traceback (most recent call last):
File "main.py", line 43, in <module>
 ab_data_to_db = process_data_file() # read file
File "C:\Users\camara\PycharmProjects\as1\file\file_read.py", line 32, in process_data_file
 for line in data_file:
 line = line.strip()
 comma_separated_line = line.split(",")
UnicodeDecodeError: 'charmap' codec can't decode byte 0x90 in position 4131: character maps to <undefined>

(venv) C:\Users\camara\PycharmProjects\as1>[]

Figure 42 Fixing Encoding issue while reading file

Date: 30/10/2019

Work Completed: Test graph for project

Understanding Achieved: Installing Matplotlib, Matching versions for Matplotlib and python. Visual C++ requirement

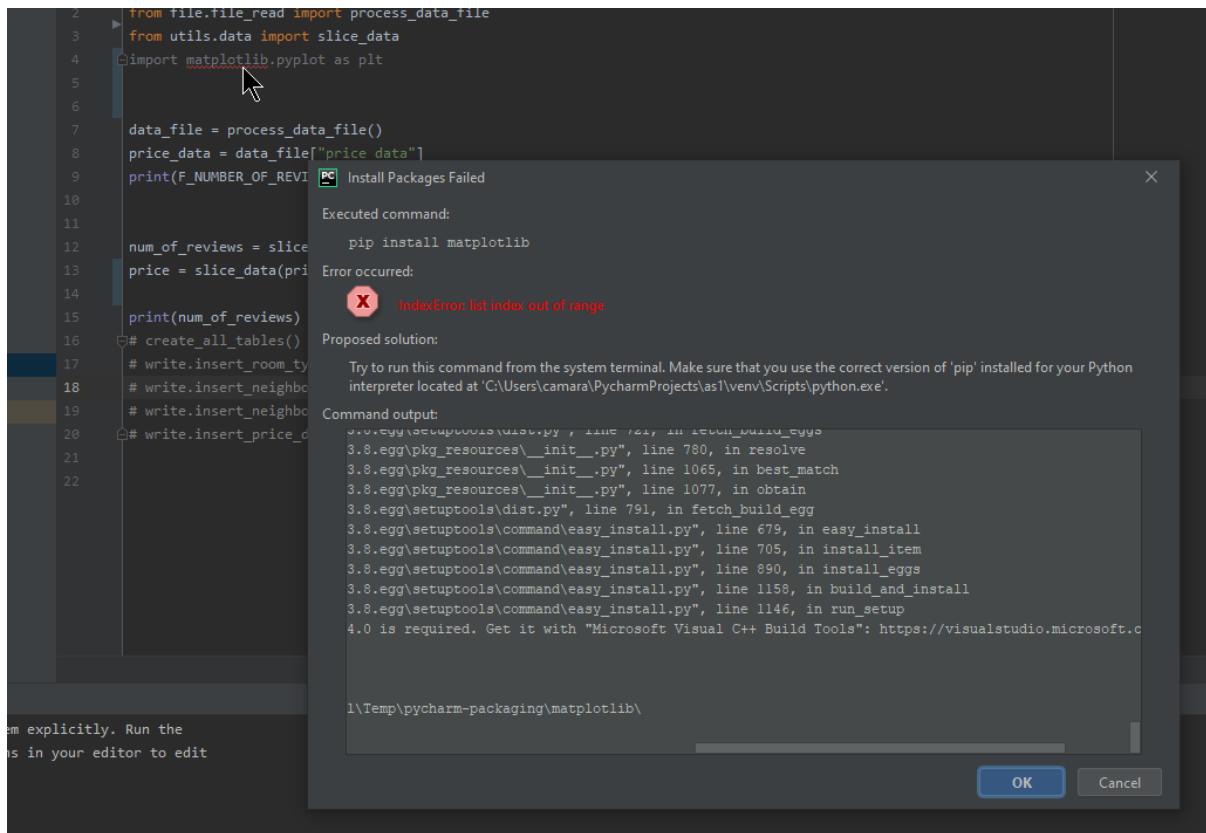


Figure 43 VC++ dependency issue while installation matplotlib

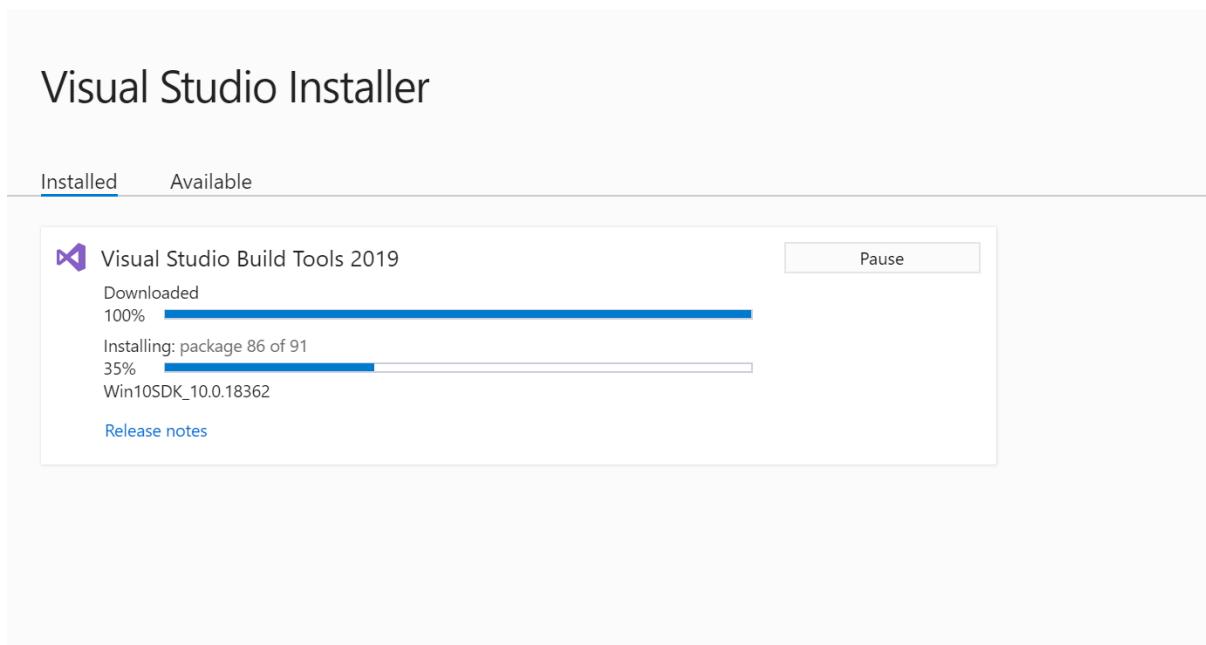


Figure 44 Installing VS Build tool as a fix

Figure 45 Python 3.8 is not supported but matplotlib

You have python 3.8, not python 3.7. But **matplotlib 3.1.1 cannot be installed on python 3.8**. So best remove python 3.8 completely and install python 3.7. When you then run `python -m pip install matplotlib` it will install the compiled version from the wheels, so there is no need to compile anything yourself or have Microsoft Studio available.

share edit

answered Oct 18 at 20:39

ImportanceOfBeingErnest

173k • 17 • 229 • 319

Figure 46 Searched and found solution for matplotlib version issue

Date: 31/10/2019

Work Completed: Basic Maths functions

Understanding Achieved: Meaning of Mean, Mode, Standard Deviation and how to calculate

The screenshot shows the PyCharm IDE interface. On the left is the project file tree:

- __init__.py
- connection.py
- file
- main
 - __init__.py
 - main.py
 - main_test.py
- utils
 - __init__.py
 - data.py
 - math.py
- venv library root
 - include
 - Lib
 - Scripts
 - pyvenv.cfg
- .gitignore
- External Libraries
- Scratches and Consoles

The code editor contains Python code for data processing and plotting:

```

15 room_type_ids = vertical_slice_data(sorted_by_reviews, F_ROOM_TYPE[1])
16 price = vertical_slice_data(sorted_by_reviews, F_PRICE[1])
17 print("hi")
18 a = [t for t in price_data if t[7]>1100 and t[7]<1200]
19 print("ok")
20 #print(a)
21 print(max(price))
22 sorted_by_reviews = sort_data(a, F_PRICE[1])
23 price = vertical_slice_data(sorted_by_reviews, F_PRICE[1])
24 plt.hist(price, density=True, bins=50)

25 print(summary(price_data, [F_PRICE, F_NUMBER_OF_REVIEWS]))
26 print(summary(a, [F_PRICE, F_NUMBER_OF_REVIEWS]))
27 #print(summary(a, [F_PRICE, F_NUMBER_OF_REVIEWS]))
28 #plt.scatter(room_type_ids, price, alpha=0.1)
29 #plt.scatter(num_of_reviews, price, alpha=0.1)
30 #plt.ylim(1, 10)
31 #plt.xlim(1, 10)
32

```

The run console at the bottom shows the output of the script:

```

Run: main_test x
C:\Users\camara\PycharmProjects\as1\venv\Scripts\python.exe C:/Users/camara/PycharmProjects/as1/main/main_
('number_of_reviews', 9)
hi
ok
10000
[('price', 7), {'count': 41707, 'min': 0, 'mean': 153.78989138513919, 'mode': 100, 'median': 109, 'sd': 1
[('price', 7), {'count': 8, 'min': 1115, 'mean': 1162.125, 'mode': 1195, 'median': 1160.0, 'sd': 27.3936
[55, 3, 93, 95, 95, 0, 2, 5, 0, 0, 2, 43, 93, 42, 14, 22, 0, 0, 4, 2, 113, 0, 0, 8, 0, 0, 9, 2, 3, 0, 0, 0
[1115, 1145, 1150, 1150, 1170, 1177, 1195, 1195]

```

Figure 47 Testing major math functions

Date: 1/11/2019

Work Completed: Testing Scatter Plot

Understanding Achieved: Usage of Scatter Plot, Plotting Matplotlib and Seaborn Scatter plots. Scaling X and Y axis

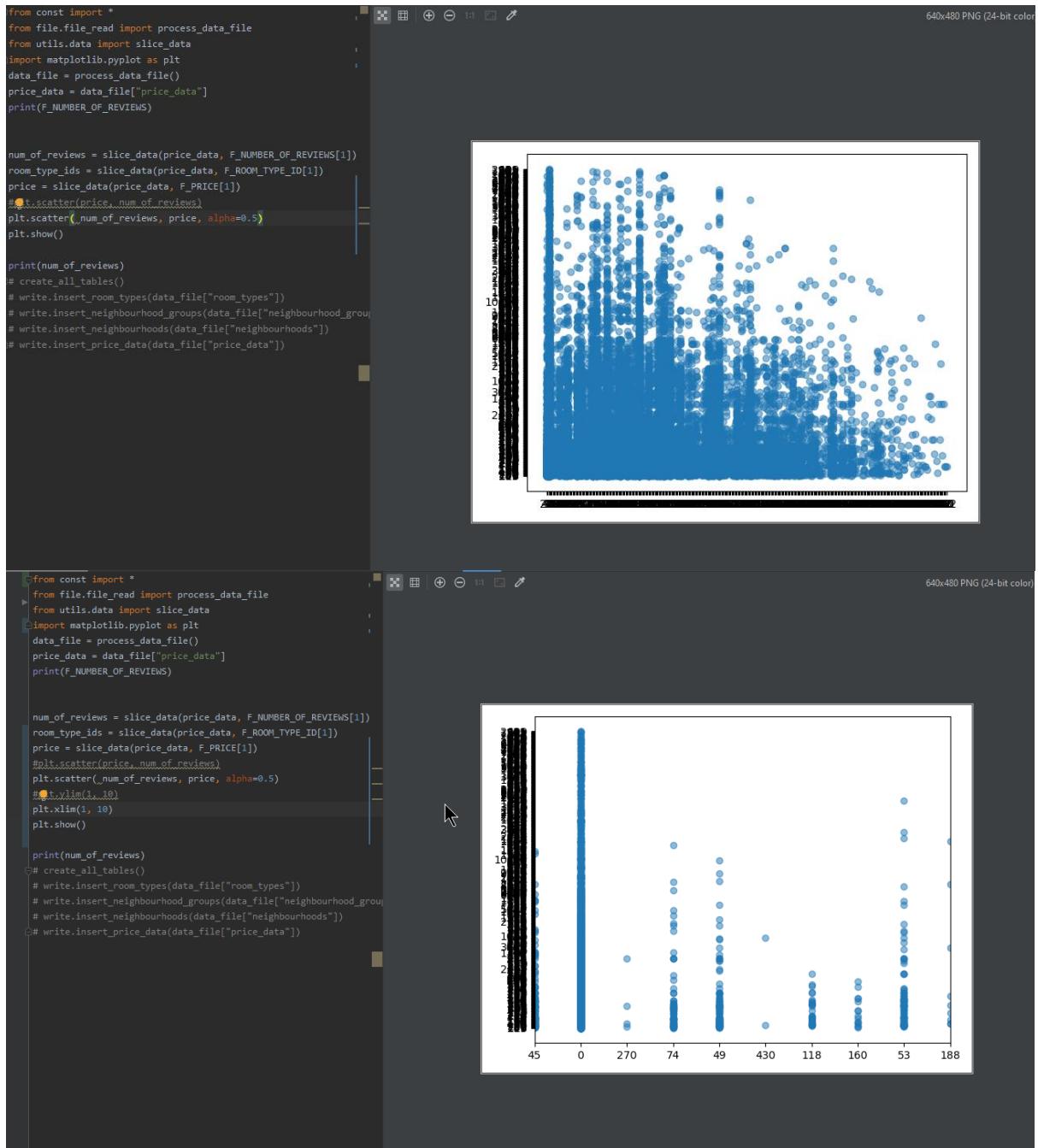


Figure 48 Fixing overlapped x, y labels

Date: 02/11/2019

Work Completed: Pair Scatter Plot

Understanding Achieved: Features of Pair Scatter Plot. DataFrames. Manipulating DataFrames for plotting Pair Scatter Plot without numpy or panda.

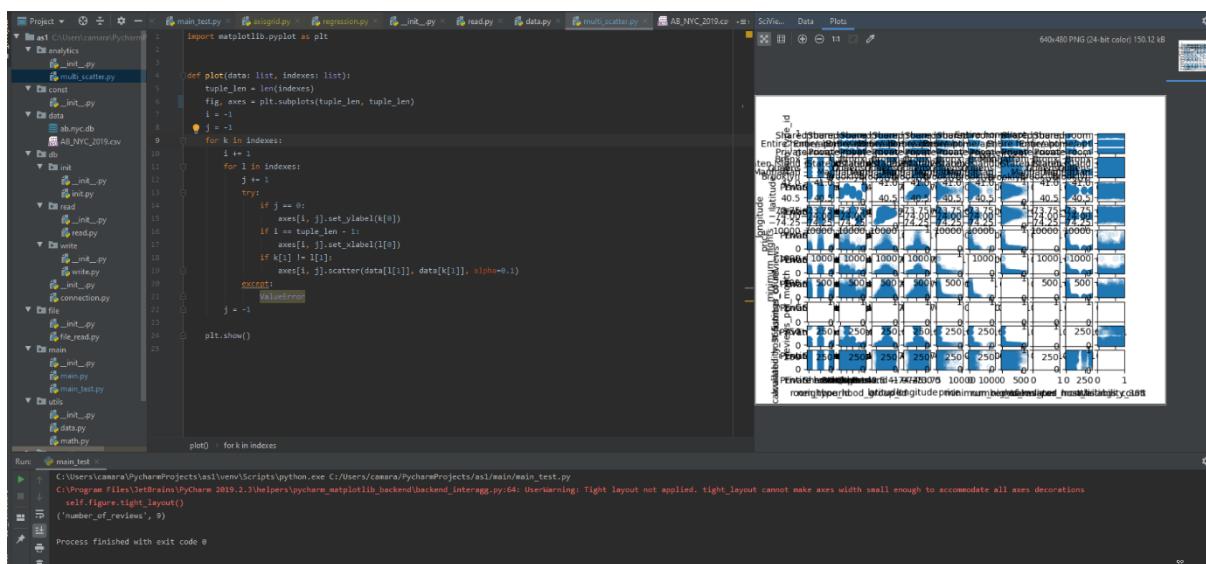
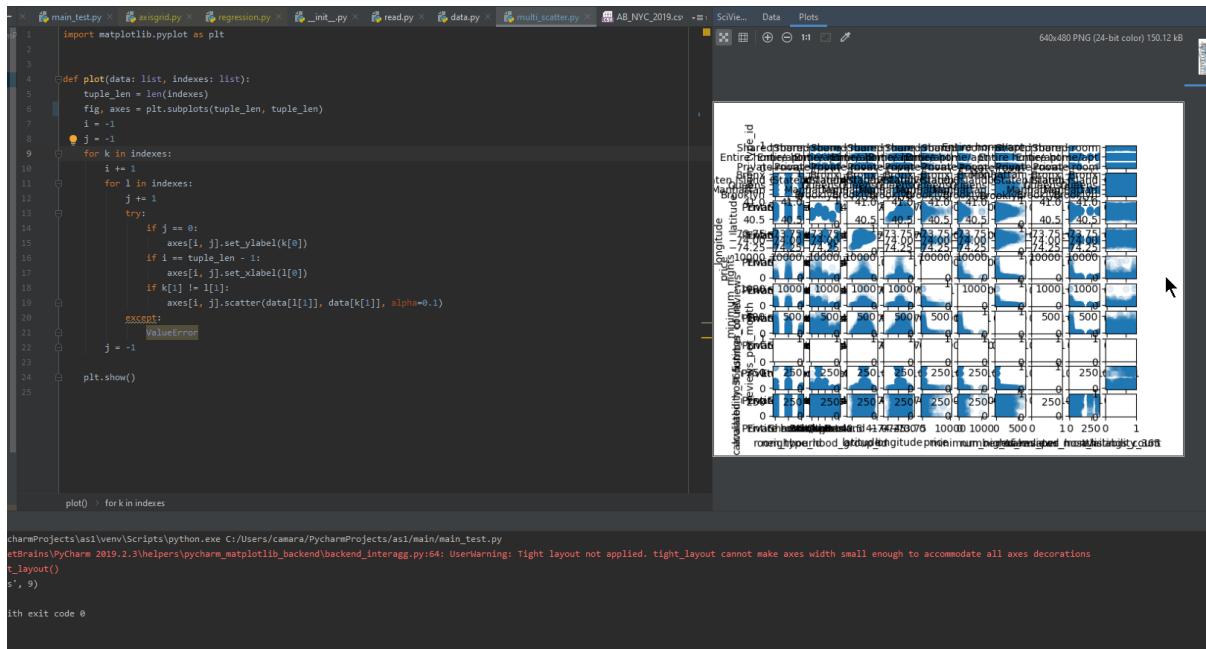


Figure 49 Faced overlapped sub plots

Date: 03/11/2019

Work Completed: Changing the image size based on number of subplots

Understanding Achieved: Change image size.

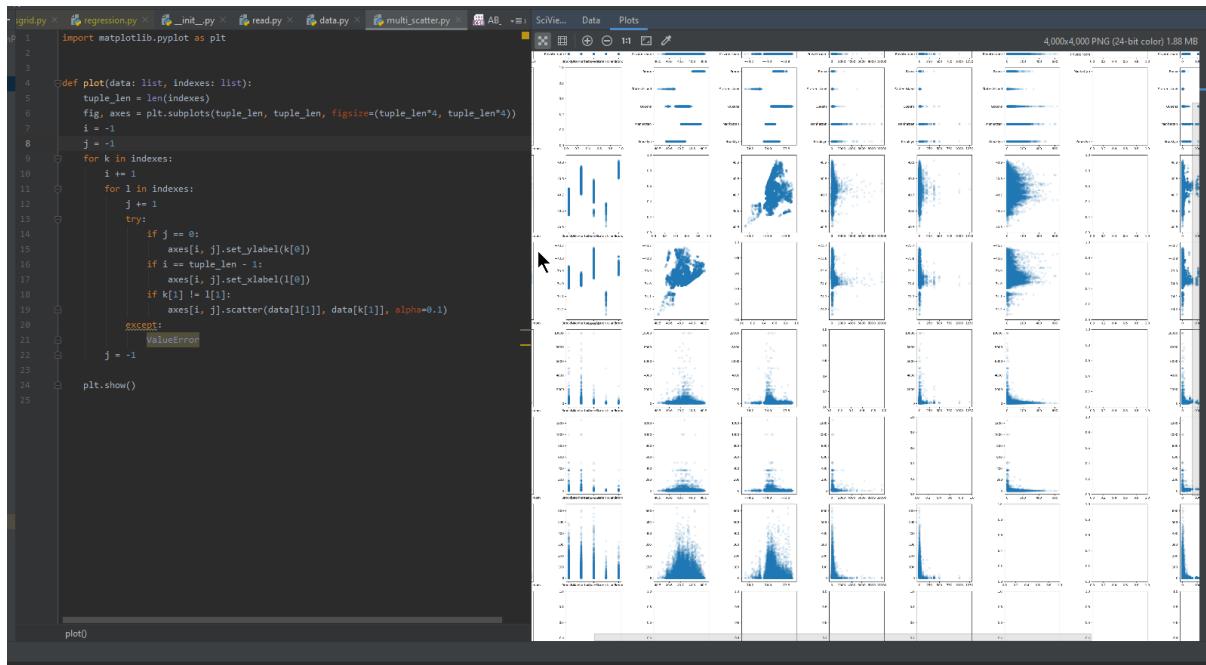


Figure 50 Fixing overlapped subplots by introducing figsize

Date: 04/11/2019

Work Completed: Correcting pair scatter plot for a variable against same variable

Understanding Achieved: Understand how seaborn achieved this

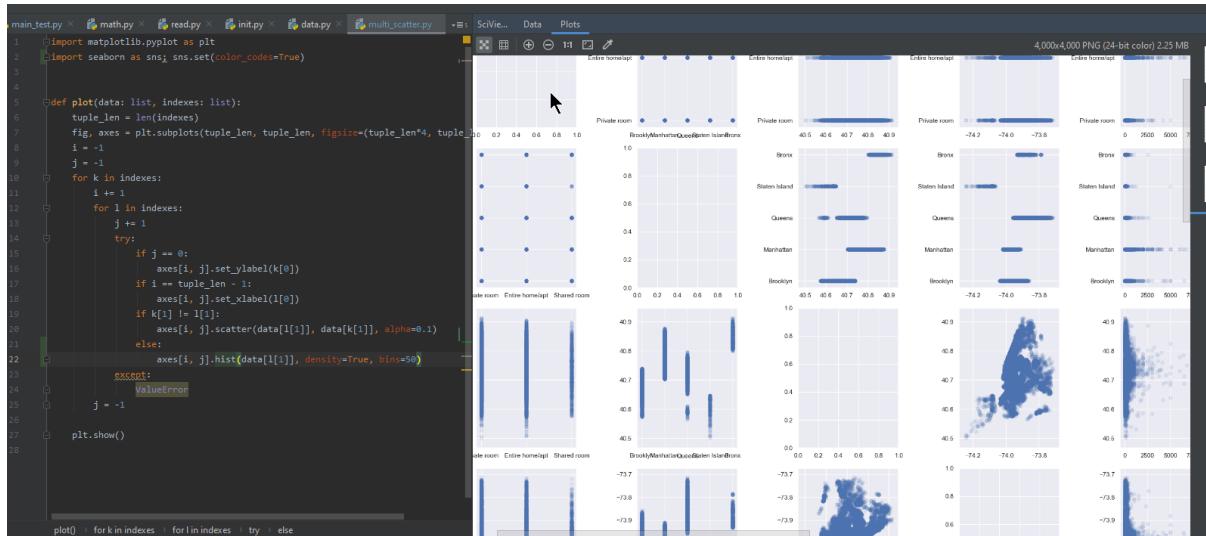


Figure 51 Facing empty subplot for variable to same variable scatter plot

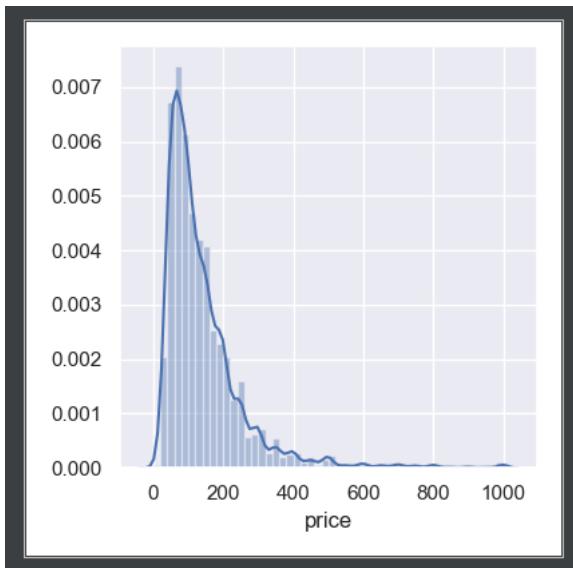


Figure 52 Decided to fill empty subplot by a distribution graph

Date: 05/11/2019

Work Completed: Table Printing

Understanding Achieved: Investigated various modules available for table printing

Screenshot of PyCharm IDE showing code editor and run output for table printing.

Code Editor:

```

Project: as1
File: print.py
1  from texttable import Texttable
2
3  def print_summary(summary: list):
4      t = Texttable()
5      headers = []
6      for i in summary:
7          if len(headers) == 0:
8              headers.append("Field")
9              headers.extend(dict(i[1]).keys())
10             print(headers)
11             t.header(headers)
12
13             row = [i[0][0]]
14             row.extend(list(dict(i[1]).values()))
15             print(row)
16             t.add_row(row)
17
18             print(t.draw())

```

Run Output:

```

Process finished with exit code 0

```

Field	count	min	mean	mode	median	sd	max
latitude	41707	40.500	40.729	[(40.71 40.723 0.054 40.913			
				813,			
				17)])			
longitude	41707	-74.244	-73.953	[(-73.9 -73.956 0.046 73.713			
e				5427,			
				15), (-			
				73.9540			
				5, 15),			
				(-73.95			
				136,			
				15)])			
price	41707	0	153.790	[(100.0 109 243.440 10000			
				,			
				1771)])			
minimum_	41707	1	6.957	[(1, 2 20.920 1250			
nights				10987)])			
number_o	41707	0	23.185	[(0, 5 44.647 629			
f_review				8678)])			
s							
reviews_	41707	0	1.090	[(0, 0.370 1.602 58.500			
per_mont				8678)])			
h							
calculat	41707	1	6.954	[(1, 1 32.559 327			
ed_host_				27459)])			
listings							
_count							
availabi	41707	0	113.778	[(0, 46 132.187 365			
lity_365				14972)])			

Figure 53 Printing summary in a table

Date: 06/11/2019

Work Completed: Categorized data based on Price

Understanding Achieved: Investigating different looping techniques for python. Understand more about Standard Deviation.

```

16     point_summary(summary(ab_data, F_NUMERIC_FIELDS))
17

```

Run: main ×

longitude	41707	-74.244	-73.953	[(-73.9, -73.956]	0.046	-73.713
latitude	41707	5427,	15), (-	73.9540		
price	41707	0	153.790	[(100.0, 109	243.440	10000
minimum_nights	41707	1	6.957	[(1, 2	20.920	1250
nights	41707	0	10987)]			
number_of_reviews	41707	0	23.185	[(0, 5	44.647	629

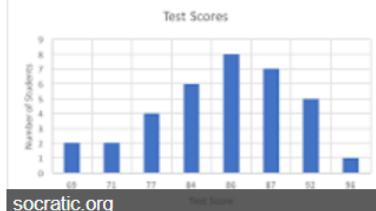
standard deviation is high what does it means



All Images Videos News Shopping More Settings Tools

About 155,000,000 results (0.80 seconds)

A low **standard deviation** indicates that the values tend to be close to the **mean** (also called the expected value) of the set, while a **high standard deviation** indicates that the values are spread out over a wider range.



[Standard deviation - Wikipedia](#)

https://en.wikipedia.org/wiki/Standard_deviation

? About Featured Snippets ! Feedback

Figure 54 Understanding Standard Deviation

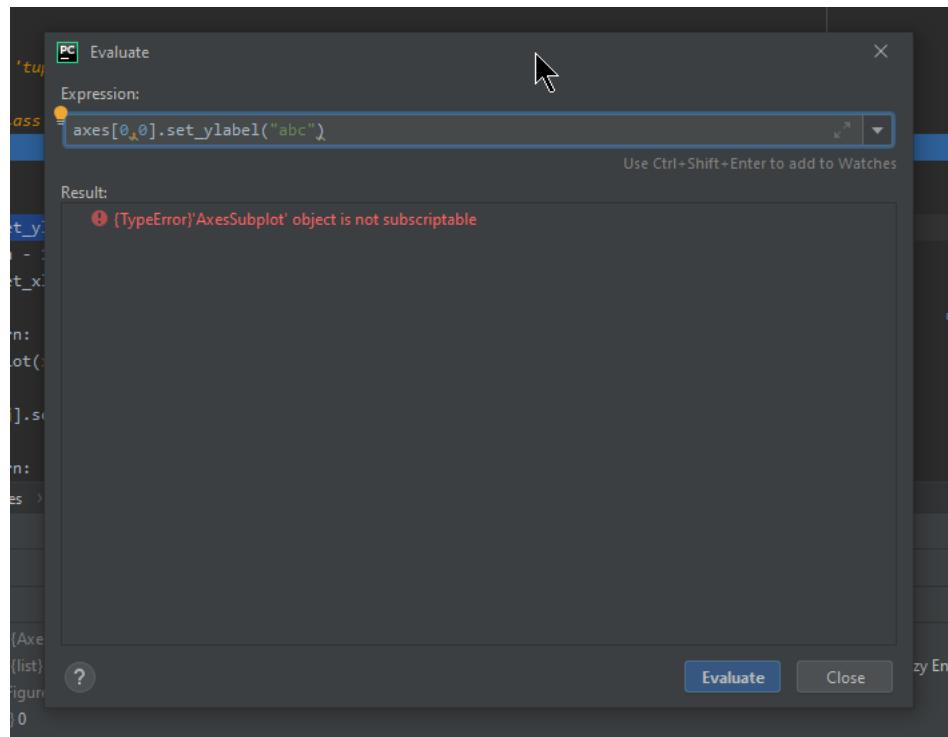
Date: 07/11/2019

Work Completed: Sub plotting

Understanding Achieved: Understand Sub Plotting, figure and axes of matplotlib

The screenshot shows a PyCharm IDE interface. The top navigation bar includes tabs for 'utils_init_.py', 'print.py', 'builtins.py', 'texttable.py', 'const_init_.py', 'main_test.py', 'math.py', 'data.py', and 'multi_scatter.py'. On the right, there are tabs for 'SciView...', 'Data', and 'Plots'. A status bar at the top right indicates '400x400 PNG (24-bit color) 0.828'. The main area has two panes: one on the left containing the Python code for 'multi_scatter.py' and one on the right showing a scatter plot. The code uses matplotlib's pyplot and seaborn libraries to create a scatter plot with axes labeled from 0.0 to 1.0. The plot shows several data points as small circles. Below the code editor is a toolbar with various icons. At the bottom, there is a 'Variables' tool window showing the current state of variables like 'axes', 'data', 'fig', 'i', 'j', 'indexes', 'k', 'tuple', 'tuple_len', and 'use_seaborn'.

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns; sns.set(color_codes=True)
3
4
5 def plot(data: list, indexes: list, use_seaborn=False):
6     tuple_len = len(indexes[0])
7     fig, axes = plt.subplots(tuple_len, tuple_len, figsize=(tuple_len*4, tuple_len*4))
8     i = -1; i += 0
9     j = -1; j += -1
10    for k in indexes: k: <class 'tuple'>: ('price', 7)
11    i += 1
12    for l in indexes: l: <class 'tuple'>: ('price', 7)
13    j += 1
14    try:
15        if j == 0:
16            axes[i, j].set_ylabel(k[0])
17        if i == tuple_len - 1:
18            axes[i, j].set_xlabel(l[0])
19        if k[1] != l[1]:
20            if use_seaborn:
21                sns.regplot(x=data[l[1]], y=data[k[1]], ax=axes[i, j])
22            else:
23                axes[i, j].scatter(data[l[1]], data[k[1]], alpha=0.1)
24        else:
25            if use_seaborn:
26
27    plot() >>> for k in indexes >>> for l in indexes
```



```
fig, axs = plt.subplots()
```

returns a figure with only one single subplot, so `axs` already holds it without indexing.

Figure 55 Sub plotting without index causing issues

The screenshot shows a Jupyter Notebook environment with several open cells and plots.

Code Cell 1:

```
j += 1
    try:
        set_axis_labels(axes, i, j, k, l, tuple_len)
        if k[1] != l[1]:
            if use_seaborn_calculated:
                sns.regplot(~data[l[1]], y=data[k[1]], ax=get_axes(axes, i, j, tuple_len))
            else:
                get_axes(axes, i, j, tuple_len).scatter(data[l[1]], data[k[1]], alpha=0.1)
        else:
            if use_seaborn_calculated:
                sns.distplot(data[l[1]], color="b", ax=get_axes(axes, i, j, tuple_len))
            else:
                get_axes(axes, i, j, tuple_len).hist(data[l[1]])
    except:
        print("Oops!", sys.exc_info()[0], "occurred.")
j = -1

plt.show()
```

Code Cell 2:

```
def set_axis_labels(axes, i, j, k, l, tuple_len):
    if j == 0:
        if tuple_len != 1:
            set_axes(axes, i, j, tuple_len).set_ylabel(k[0])
    set_axis_label(axes, i, j, k, l)
```

Code Cell 3:

```
analytics
const
data
db
file
main
utils
venv library root
__init__.py
main.py
main_test.py
utils
venv library root
__init__.py
main.py
main_test.py
utils
venv library root
__init__.py
main.py
main_test.py
utils
```

Code Cell 4:

```
ab_data_to_db = process_data_file()
create_all_tables()
# create_all_tables()
# ab_data.insert_all_data(ab_data)
ab_data = read.read_ab_data()
vertical_slice_all_data_list = vertical_slice_all_data(ab_data)

print_summary(summary(ab_data, F_NUMERIC_FIELDS))
plot(vertical_slice_all_data_list[ab_data], F_ALL_VARIABLES)

print_summary(summary(ab_data, [F_PRICE]))
plot(vertical_slice_all_data_list, [F_PRICE])
plot(vertical_slice_all_data_list[ab_data], [F_PRICE])

ab_data_p_lt_1000 = [i for i in ab_data if i[F_PRICE] <= 1000]
vertical_slice_ab_data_p_lt_1000 = vertical_slice_all_data(ab_data_p_lt_1000)
print_summary(summary(ab_data_p_lt_1000, [F_PRICE]))
plot(vertical_slice_ab_data_p_lt_1000, [F_PRICE])
```

Data View:

Field	count	min	mean	mode	median	sd	max
price	41707	0	153.790	[(100.0, 1771)]	109	243.440	10000
Field	count	min	mean	mode	median	sd	max
price	41501	0	142.083	[(100.0, 1771)]	106	118.226	1000

Output:

```
Process finished with exit code 0
```

Plots:

- A histogram of the 'price' column from the main dataset, showing a sharp peak at 100.
- A histogram of the 'price' column for items costing less than or equal to 1000, showing a broader distribution peaking around 150.

Table View:

Field	count	min	mean	mode	median	sd	max
price	41707	0	153.790	[(100.0, 1771)]	109	243.440	10000
Field	count	min	mean	mode	median	sd	max
price	41501	0	142.083	[(100.0, 1771)]	106	118.226	1000

Date: 09/11/2019

Work Completed: Grouped bar chart

Understanding Achieved: Data input formats for grouped bar charts.

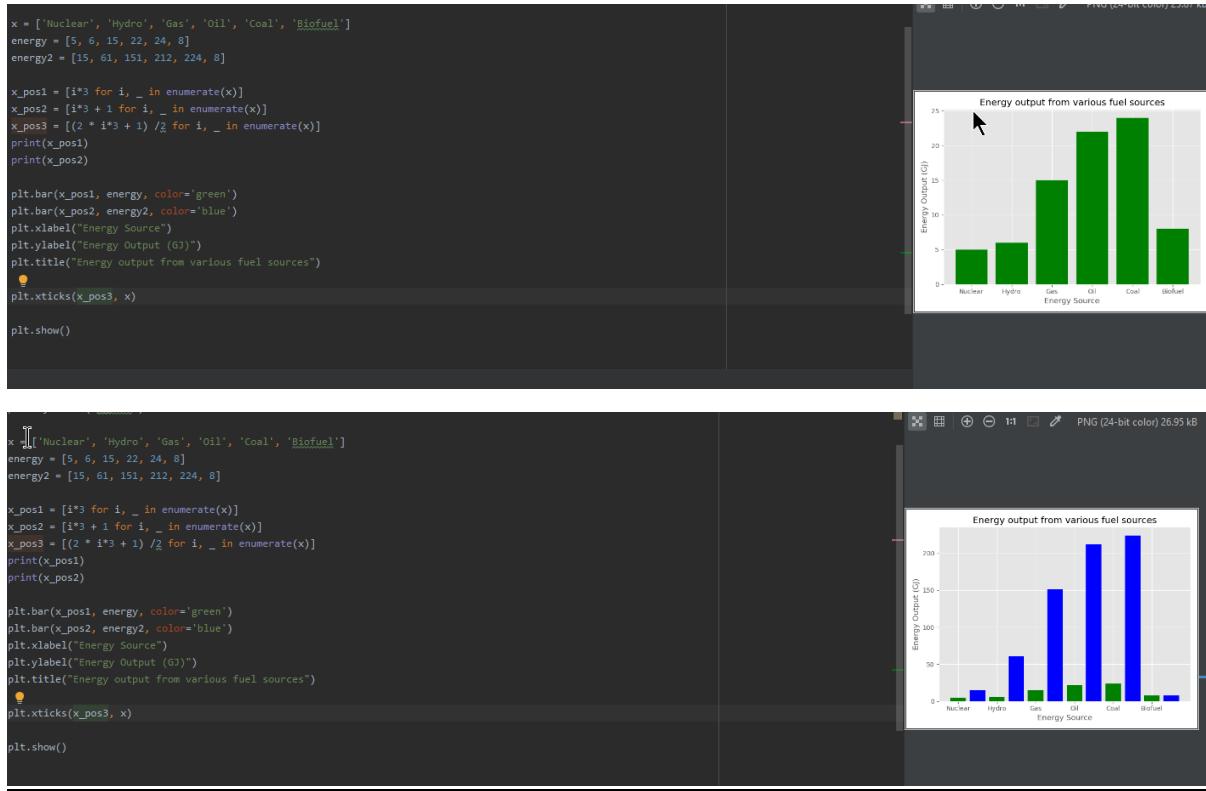


Figure 56 Testing grouped bar charts

Date: 10/11/2019

Work Completed: Pie Charts

Understanding Achieved: Understanding Data format for Pie Charts.

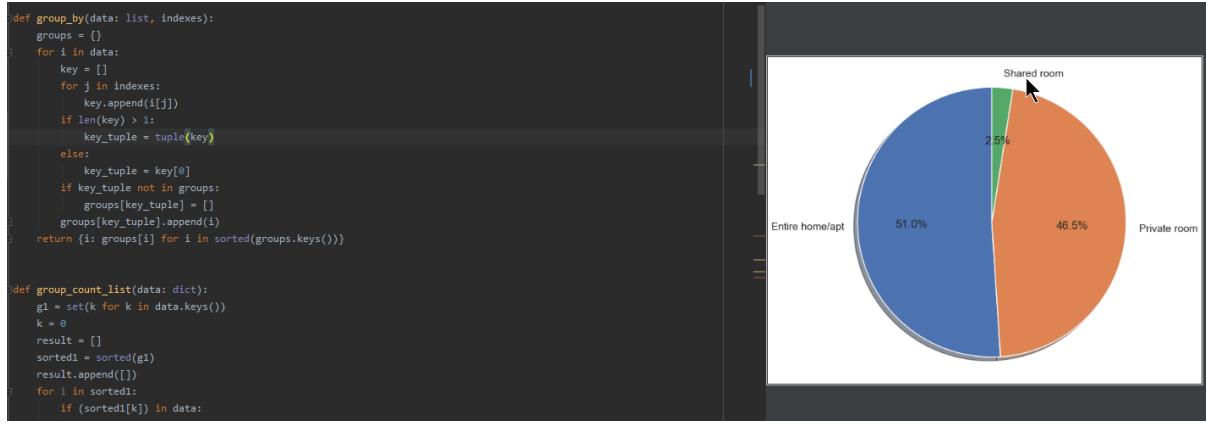


Figure 57 Testing pie charts

Date: 11/11/2019

Work Completed: Graph Saving

Understanding Achieved: Configurations available for graph saving

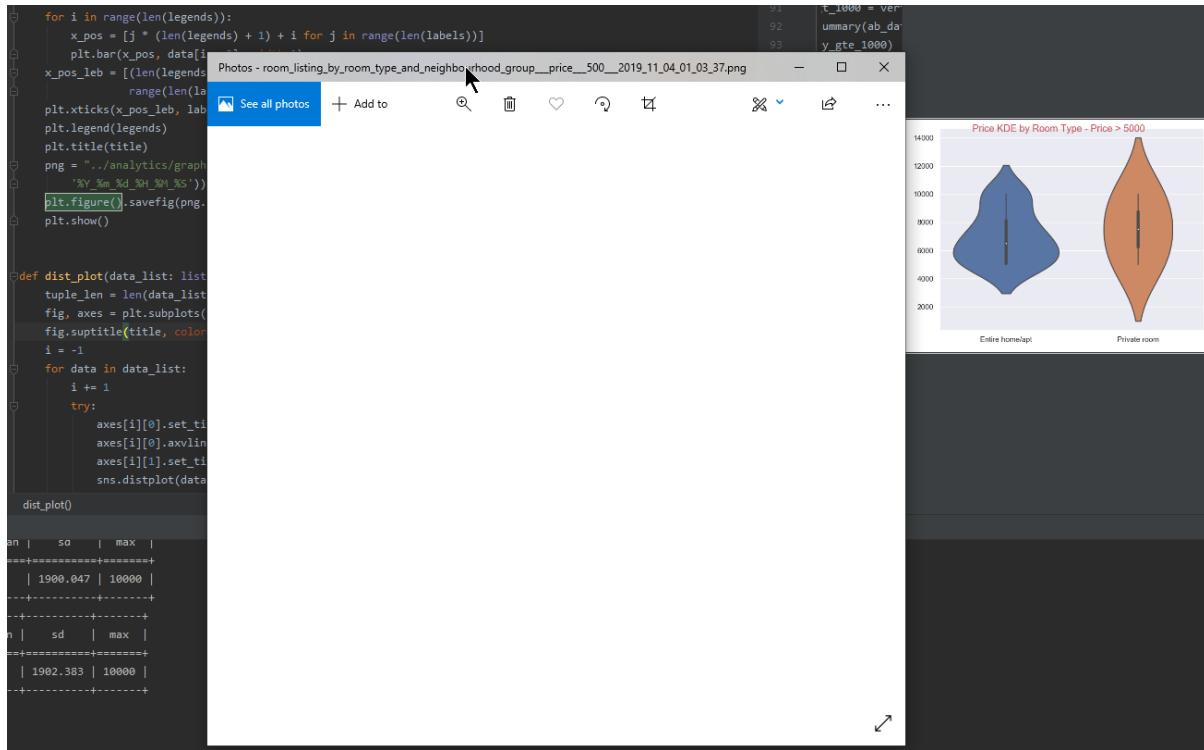


Figure 58 Image saving issue

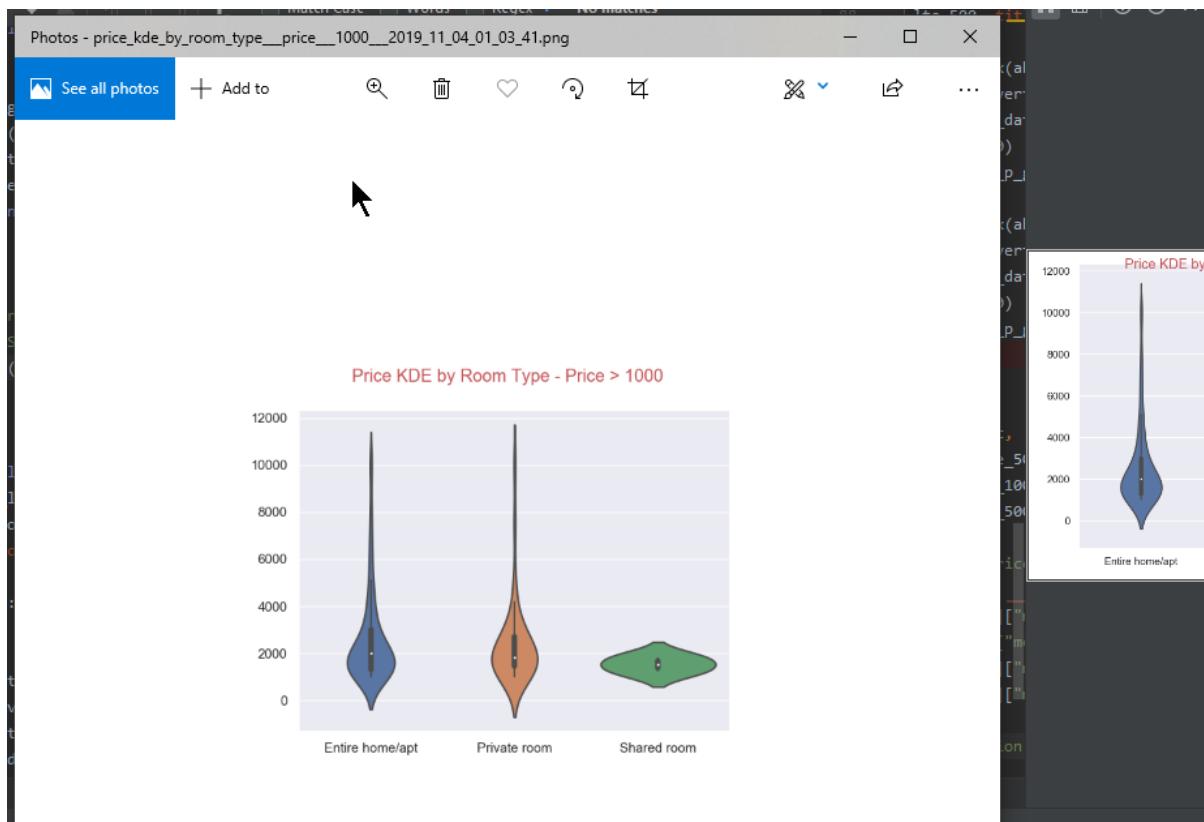


Figure 59 Fixing image save

Date: 13/11/2019

Work Completed: Initial Setup for Unit Testing

Understanding Achieved: Research on available unit testing modules. Unit Test Coverage

The screenshot shows the PyCharm IDE interface. On the left is the code editor with Python test files. In the center is the terminal window displaying the execution of tests and coverage results. On the right is the Coverage tool window.

Code Editor:

```
def test_vertical_slice_data_wrong_index(self):
    from utils.data import vertical_slice_data
    data = [(1, "A", 1.1), (2, "B", 2.2), (3, "C", 3.3)]
    self.assertRaises(IndexError, vertical_slice_data, data, 4)

def test_two_group_count_list(self):
    from utils.data import two_group_count_list
    data = [
        TestData(test_vertical_slice_data_wrong_index)
    ]
```

Terminal:

```
Tests passed: 4 of 4 tests - 2 ms
Testing started at 12:36 PM ...
C:\Users\camara\PycharmProjects\as1\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm 2019.2.3\helpers\pycharm\_jb_unittest_runner.py" --path C:/Users/camara/PycharmProjects/as1/utils -t C:/Users/camara/PycharmProjects/as1/utils in C:/Users/camara/Pychar
Launching unittests with arguments python -m unittest discover -s C:/Users/camara/PycharmProjects/as1/utils -t C:/Users/camara/PycharmProjects/as1/utils in C:/Users/camara/Pychar
Ran 4 tests in 0.0004s
OK
Process finished with exit code 0
```

Coverage:

Element	Statistics, %
__init__.py	16% lines covered
data.py	27% lines covered
math.py	not covered
print.py	not covered
test_data.py	not covered
test_vertical_slice_data.py	not covered

Figure 60 Unittesting and test coverage

Date: 14/11/2019

Work Completed: Unit Testing for Exceptions

Understanding Achieved: Parameter passing for Exception testing

The screenshot shows the PyCharm IDE interface during unit testing. At the top, a terminal window displays test results:

```

✖ Tests failed: 1, passed: 3 of 4 tests – 3 ms
Traceback (most recent call last):
  File "C:\Users\camara\AppData\Local\Programs\Python\Python37\lib\unittest\case.py", line 59, in testPartExecutor
    yield
  File "C:\Users\camara\AppData\Local\Programs\Python\Python37\lib\unittest\case.py", line 628, in run
    testMethod()
  File "C:/Users/camara/PycharmProjects/as1/utils/test_data.py", line 19, in test_vertical_slice_data_wrong_index
    vertical_slice_data(data, 4)
  File "C:/Users/camara/PycharmProjects/as1/utils/data.py", line 6, in vertical_slice_data
    return [item[index] for item in data]
  File "C:/Users/camara/PycharmProjects/as1/utils/data.py", line 6, in <listcomp>
    return [item[index] for item in data]
IndexError: tuple index out of range

Ran 4 tests in 0.007s

```

Below the terminal, the code editor shows the test function:

```

def test_vertical_slice_data_wrong_index(self):
    from utils.data import vertical_slice_data
    data = [(1, "A", 1.1), (2, "B", 2.2), (3, "C", 3.3)]
    vertical_slice_data(data, 4)
    self.assertRaises(IndexError, vertical_slice_data, data, 4)

```

To the right of the code editor is a coverage analysis tool window:

Element	Statistics, %
__init__.py	16% lines covered
data.py	27% lines covered
math.py	not covered
print.py	not covered
test_data.py	not covered
test_vertical_slice_data.py	not covered

At the bottom of the interface, the terminal shows the test results again:

```

Tests passed: 4 of 4 tests – 2ms
Testing started at 12:36 PM ...
C:\Users\camara\PycharmProjects\as1\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm 2019.2.3\helpers\pycharm\_jb_unittest_runner.py" --path C:/Users/camara/PycharmProjects/as1
Launching unittests with arguments python -m unittest discover -s C:/Users/camara/PycharmProjects/as1/utils -t C:/Users/camara/PycharmProjects/as1/utils in C:/Users/camara/Pychar
Ran 4 tests in 0.004s
OK
Process finished with exit code 0

```

Figure 61 Unit testing for exceptions

Date: 17/11/2019

Work Completed: Initial steps for Pydoc

Understanding Achieved: Research on available documentation patterns

Example Google Style Python Docstrings

See also

[Example NumPy Style Python Docstrings](#)

Download: [example_google.py](#)

```
# -*- coding: utf-8 -*-
"""Example Google style docstrings.

This module demonstrates documentation as specified by the `Google Python
Style Guide`_. Docstrings may extend over multiple lines. Sections are created
with a section header and a colon followed by a block of indented text.

Example:
    Examples can be given using either the ``Example`` or ``Examples``
    sections. Sections support any restructuredText formatting, including
    literal blocks::

        $ python example_google.py
```

Figure 62 Python doc Google Standard

Date: 14/11/2019

Work Completed: Pydoc

Understanding Achieved: Pydoc

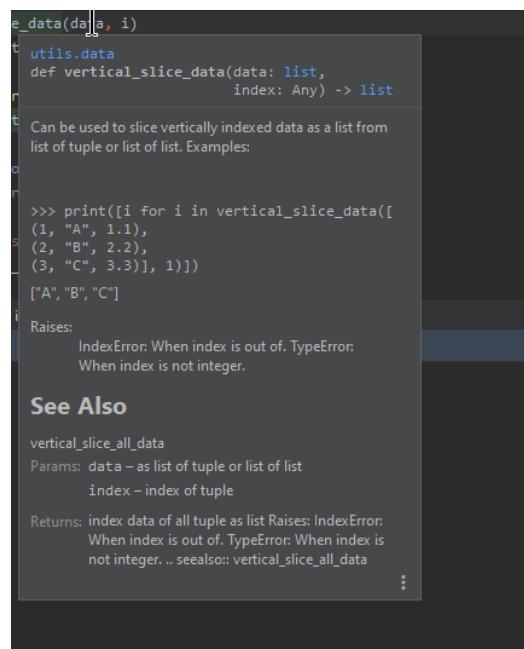


Figure 63 Testing Python doc. See Also links are not working

Date: 18/11/2019

Work Completed: Fixed static code analysis issues

Understanding Achieved: Understanding python coding best practises

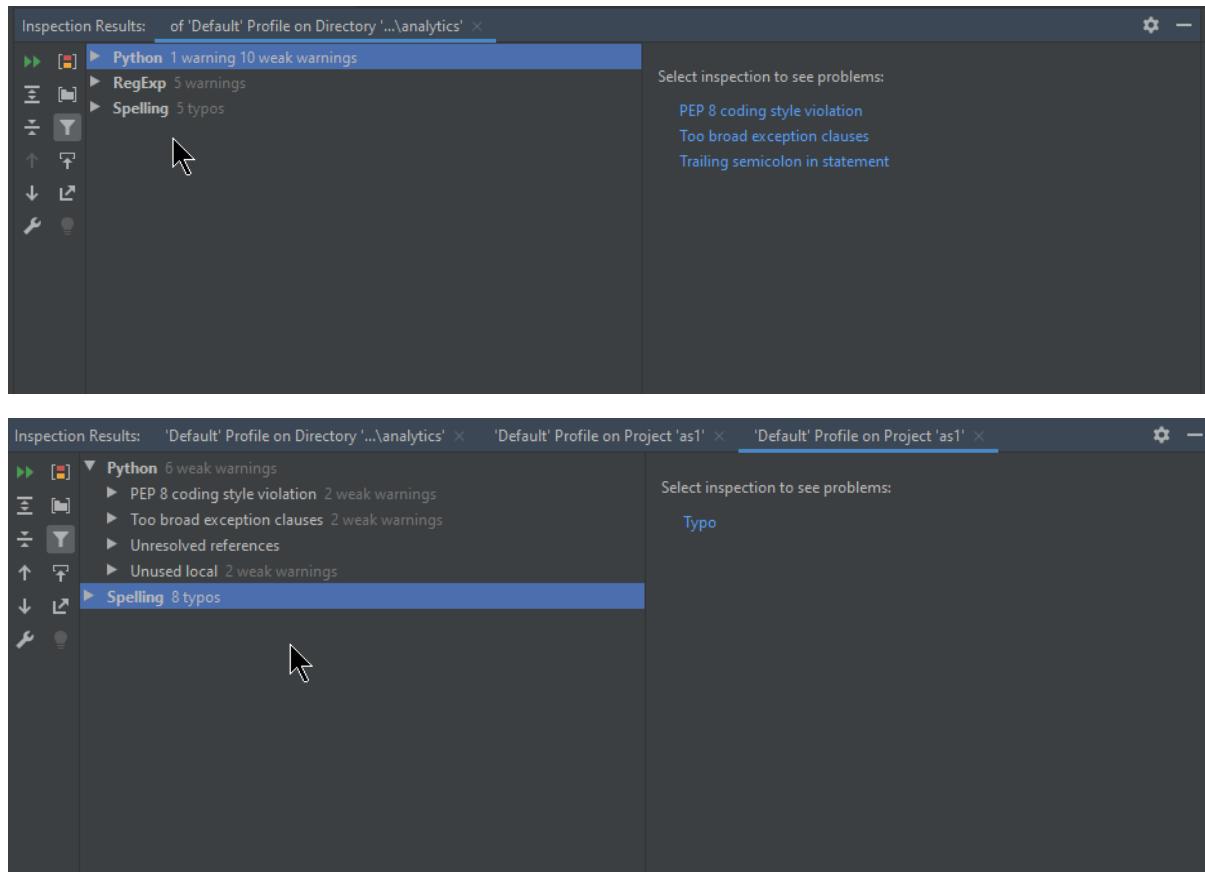


Figure 64 Fixing Static code analysis warning and errors

Date: 20/11/2019

Work Completed: Refactoring code

Understanding Achieved: Research more on Coding best practises

The screenshot shows a web browser window with the URL https://linux.die.net/divenropython/html/object_oriented_framework/private_functions.html. The page content discusses Python's concept of private elements, specifically private functions, class methods, and attributes. It notes that unlike in most languages, Python does not have a concept of private or public elements. Instead, it uses underscores to indicate visibility. A note from PEP 8 states that `__parse` and `__setitem__` are special methods. The page also includes a sidebar with a question about hiding internal methods.

5.9. Private Functions

Like most languages, Python has the concept of private elements:

- Private functions, which can't be called from outside their module
- Private class methods, which can't be called from outside their class
- Private attributes, which can't be accessed from outside their class.

Unlike in most languages, whether a Python function, method, or attribute is private or public is determined entirely by its name.

If the name of a Python function, class method, or attribute starts with (but doesn't end with) two underscores, it's private; everything else is public. Python has no concept of *protected* class methods (accessible only in their own class and descendant classes). Class methods are either private (accessible only in their own class) or public (accessible from anywhere).

In `__parse`, there are two methods: `__parse` and `__setitem__`. As you have already discussed, `__setitem__` is a *special method*; normally, you would call it indirectly by using the dictionary syntax on a class instance, but it is public, and you could call it directly (even from outside the `fileinfo` module) if you had a really good reason. However, `__parse` is private, because it has two underscores at the beginning of its name.

In Python, all special methods (like `__setitem__`) and built-in attributes (like `__doc__`) follow a standard naming convention: they both start with and end with two underscores. Don't name your own methods and attributes this way, because it will only confuse you (and others) later.

907 It's cultural. In Python, you don't write to other classes' instance or class variables. In Java, nothing prevents you from doing the same if you *really* want to - after all, you can always edit the source of the class itself to achieve the same effect. Python drops that pretence of security and encourages programmers to be responsible. In practice, this works very nicely.

If you want to emulate private variables for some reason, you can always use the `_` prefix from [PEP 8](#). Python mangles the names of variables like `_foo` so that they're not easily visible to code outside the class that contains them (although you *can* get around it if you're determined enough, just like you *can* get around Java's protections if you work at it).

By the same convention, the `_` prefix means **stay away even if you're not technically prevented from doing so**. You don't play around with another class's variables that look like `_foo` or `_bar`.

share edit edited Feb 2 '15 at 5:16 answered Oct 29 '09 at 2:01
 hpotter92 62.5k • 23 • 107 • 143 Kirk Strauser 24.6k • 5 • 43 • 59

Figure 65 Finding a way to hide internal methods

Date: 23/11/2019

Work Completed: Fixed intermittently failing test cases because of order of data set

Understanding Achieved: Understanding order of list, tuple and dict

The screenshot shows a terminal window with a test results tree on the left and a code editor on the right. The test results tree has nodes: Test Results, test_file_read, TestData, test_process_data_file_10_lines_2_skip (selected), and test_process_data_file_main_file. The code editor displays a JSON-like structure:

```
[[{"id":  
    "name":  
    "host_id":  
    "host_name":  
    "neighbourhood_group":  
    "neighbourhood":  
    "latitude":  
    "longitude":  
    "room_type":  
    "price":  
    "minimum_nights":  
    "number_of_reviews":  
    "last_review":  
    "reviews_per_month":  
    "calculated_host_listings_count":  
    "availability_365"},  
 [{"id":  
     "name":  
     "host_id":  
     "host_name":  
     "neighbourhood_group":  
     "neighbourhood":  
     "latitude":  
     "longitude":  
     "..."}]]
```

Figure 66 Unit tests are failing intermittently because of output order

Date: 24/11/2019

Work Completed: Fixed relative path and absolute path issue for data files

Understanding Achieved: Understanding how relative path and absolute path works in python

```

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\camara\PycharmProjects\as1\file\test_file_read.py", line 36, in test_process_data_file_10_lines_2_skip
    data = process_data_file("../data/tmp/AB_NYC_10_lines_2_skip.csv")
  File "C:\Users\camara\PycharmProjects\as1\file\file_read.py", line 46, in process_data_file
    raise FileNotFoundError
exceptions.file_exceptions.FileNotFoundError

=====
ERROR: test_process_data_file_main_file (file.test_file_read.TestData)

-----
Traceback (most recent call last):
  File "C:\Users\camara\PycharmProjects\as1\file\file_read.py", line 29, in process_data_file
    with open(file, encoding="utf8") as data_file:
FileNotFoundError: [Errno 2] No such file or directory: '../data/AB_NYC_2019.csv'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\camara\PycharmProjects\as1\file\test_file_read.py", line 67, in test_process_data_file_main_file
    data = process_data_file()
  File "C:\Users\camara\PycharmProjects\as1\file\file_read.py", line 46, in process_data_file
    raise FileNotFoundError
exceptions.file_exceptions.FileNotFoundError

-----
Ran 17 tests in 0.008s

```

Figure 67 File reading unit test are failing based on the location of test execution.

Date: 25/11/2019

Work Completed: Fixed static code analysis issue

Understanding Achieved: Understanding python coding best practises

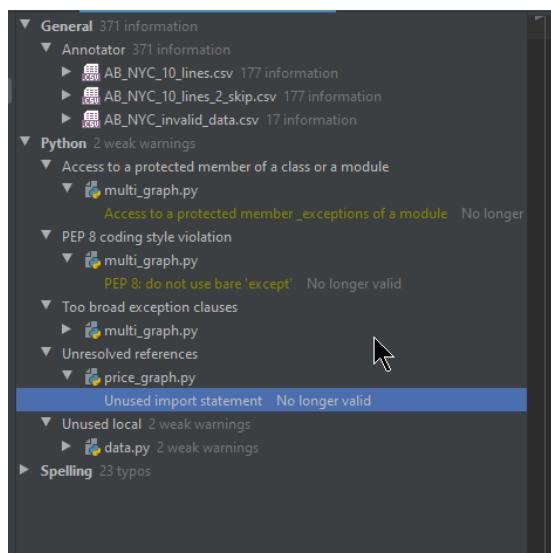


Figure 68 Code refactoring

Date: 26/11/2019

Work Completed: Fixed large table printing issues

Understanding Achieved: Understand how to change the table size for texttable

The screenshot shows a terminal window titled "main (1) ×". The command run is "C:\Users\camara\PycharmProjects\as1\venv\Scripts\python.exe C:/Users/camara/PycharmProjects/as1/main.py". The output displays a large table where the column widths are not properly aligned, causing many columns to be truncated by underscores (_). The first few rows of the table are:

id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	availability
1	Clean & quiet apt home by the park	2787	1	1	1	40.647	-73.972	Entire home/apt	40	149	1	2018-10-19	0.210	6	365
2	Skyline Midtown Castle	2845	2	2	2	40.754	-73.984	Entire home/apt	40	225	1	2019-05-21	0.380	2	395
3	THE VILLAGE OF HARLEM...NEW YORK !	4632	1	2	3	40.869	-73.942	Entire home/apt	40	158	3	2018-07-01	0.000	1	365
4	Cozy Entire Floor of Brownstone	4869	2	1	4	40.685	-73.960	Entire home/apt	40	89	1	2019-07-05	4.640	1	194
5	Entire Apt: Spacious Studio/Loft by central park	7192	2	2	5	40.799	-73.944	Entire home/apt	40	80	10	2018-11-19	0.100	1	0
6	Large Cozy 1 BR Apartment In Midtown East	7322	2	2	6	40.748	-73.975	Entire home/apt	40	208	3	2019-06-22	0.590	1	129
7	BlissArtsSpace!	7356	1	1	7	40.687	-73.956	Entire home/apt	40	68	45	2017-10-05	0.400	1	0
8	Large Furnished Room Near 8th/way	8967	1	2	8	40.765	-73.985	Entire home/apt	40	79	2	2019-06-24	3.470	1	220
9	Cozy Clean Guest Room - Family Apt	7490	1	2	9	40.882	-73.967	Entire home/apt	40	79	2	2017-07-21	0.990	1	0
10	Cute & Cozy Lower East Side 1 bdrm	7549	2	2	10	40.713	-73.990	Entire home/apt	40	158	1	2019-06-09	1.330	4	188

Figure 69: Large table print casing alignment issues

The screenshot shows a terminal window titled "Printing first 10 lines of data set". The command run is "C:\Users\camara\PycharmProjects\as1\venv\Scripts\python.exe C:/Users/camara/PycharmProjects/as1/main.py". The output displays a table with correctly aligned columns. The first few rows of the table are:

id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	availability
1	Clean & quiet apt home by the park	2787	1	1	1	40.647	-73.972	Entire home/apt	40	149	1	2018-10-19	0.210	6	365
2	Skyline Midtown Castle	2845	2	2	2	40.754	-73.984	Entire home/apt	40	225	1	2019-05-21	0.380	2	395
3	THE VILLAGE OF HARLEM...NEW YORK !	4632	1	2	3	40.869	-73.942	Entire home/apt	40	158	3	2018-07-01	0.000	1	365
4	Cozy Entire Floor of Brownstone	4869	2	1	4	40.685	-73.960	Entire home/apt	40	89	1	2019-07-05	4.640	1	194
5	Entire Apt: Spacious Studio/Loft by central park	7192	2	2	5	40.799	-73.944	Entire home/apt	40	80	10	2018-11-19	0.100	1	0
6	Large Cozy 1 BR Apartment In Midtown East	7322	2	2	6	40.748	-73.975	Entire home/apt	40	208	3	2019-06-22	0.590	1	129
7	BlissArtsSpace!	7356	1	1	7	40.687	-73.956	Entire home/apt	40	68	45	2017-10-05	0.400	1	0
8	Large Furnished Room Near 8th/way	8967	1	2	8	40.765	-73.985	Entire home/apt	40	79	2	2019-06-24	3.470	1	220
9	Cozy Clean Guest Room - Family Apt	7490	1	2	9	40.882	-73.967	Entire home/apt	40	79	2	2017-07-21	0.990	1	0
10	Cute & Cozy Lower East Side 1 bdrm	7549	2	2	10	40.713	-73.990	Entire home/apt	40	158	1	2019-06-09	1.330	4	188

Figure 70: Fixed table size issue

Date: 27/11/2019

Work Completed: User friendly logging for python

Understanding Achieved: Understanding available logging options for python

```

1 import logging
2
3 from analytics.common.multi_graph import pie_plot, group_bar_plot, violin_plot
4 from const import F_PRICE, F_ROOM_TYPE_ID, F_NEIGHBOURHOOD_GROUP_ID
5 from utils.data import group_count_list, group_by, \
6     two_group_count_list, vertical_slice_data
7
8
9 def visualize_price(price_data, logger, title=None):
10     """
11         Reusable graph generator for Price Data. Following 8 graphs will be generated.
12         1. Pie Chart by Room Type
13         2. Pie Chart by Neighbourhood_Group
14         3. Count Bar Chart by Room Type
15         4. Count Bar Chart by Neighbourhood_Group
16         5. Count Bar Chart Group by Room Type for Neighbourhood_Group
17         6. Count Bar Chart Group by Neighbourhood_Group for Room Type
18         7. Violin (KDE) Chart by Room Type
19         8. Violin (KDE) Chart by Neighbourhood_Group
20
21     :param price_data: price data
22     :param title: main title for graph
23
24     visualize_price()
25
26
27     Generating Count Bar Chart [Room Listing by Neighbourhood Group Type - Price > 1000]. After Graph appeared please close to proceed
28     Generating Count Bar Chart [Room Listing by Room Type and Neighbourhood Group - Price > 1000]. After Graph appeared please close to proceed
29     Generating Count Bar Chart [Room Listing by Neighbourhood Group and Room Type - Price > 1000]. After Graph appeared please close to proceed
30     Generating Violin Chart [Price KDE by Room Type - Price > 1000]. After Graph appeared please close to proceed
31     Generating Violin Chart [Price KDE by Neighbourhood Group - Price > 1000]. After Graph appeared please close to proceed
32     Generating Pie Plot [Room Listing by Room Type - Price > 5000]. After Graph appeared please close to proceed
33     Generating Pie Plot [Room Listing by Neighbourhood Group - Price > 5000]. After Graph appeared please close to proceed
34     Generating Count Bar Chart [Room Listing by Room Type - Price > 5000]. After Graph appeared please close to proceed
35     Generating Count Bar Chart [Room Listing by Neighbourhood Group Type - Price > 5000]. After Graph appeared please close to proceed
36     Generating Count Bar Chart [Room Listing by Room Type and Neighbourhood Group - Price > 5000]. After Graph appeared please close to proceed
37     Generating Count Bar Chart [Room Listing by Neighbourhood Group and Room Type - Price > 5000]. After Graph appeared please close to proceed
38     Generating Violin Chart [Price KDE by Room Type - Price > 5000]. After Graph appeared please close to proceed
39     Generating Violin Chart [Price KDE by Neighbourhood Group - Price > 5000]. After Graph appeared please close to proceed
40     Generating Pie Plot [Room Listing by Room Type - Availability = 0]. After Graph appeared please close to proceed
41     Generating Pie Plot [Room Listing by Neighbourhood Group - Availability = 0]. After Graph appeared please close to proceed
42     Generating Count Bar Chart [Room Listing by Room Type - Availability = 0]. After Graph appeared please close to proceed
43     Generating Count Bar Chart [Room Listing by Neighbourhood Group Type - Availability = 0]. After Graph appeared please close to proceed
44     Generating Count Bar Chart [Room Listing by Room Type and Neighbourhood Group - Availability = 0]. After Graph appeared please close to proceed
45     Generating Count Bar Chart [Room Listing by Neighbourhood Group and Room Type - Availability = 0]. After Graph appeared please close to proceed
46     Generating Violin Chart [Price KDE by Room Type - Availability = 0]. After Graph appeared please close to proceed

```

Figure 71: Introducing user friendly logging

Date: 28/11/2019

Work Completed: full git commit logs

```
(venv) C:\Users\camara\PycharmProjects\as1>git log --pretty=format:"%h%x09%an%x09%ad%x09%""
de1b7b9 Chaminda Amarasinghe   Sat Dec 7 04:23:59 2019 +0000 added price summary for availability
2be2cca Chaminda Amarasinghe   Wed Dec 4 07:38:02 2019 +0000 added % for file read summary
1ad8b2a Chaminda Amarasinghe   Wed Dec 4 07:00:43 2019 +0000 fixed unittests after skipping headers
e73a169 Chaminda Amarasinghe   Tue Nov 19 00:11:48 2019 +0000 typo
0222ec7 Chaminda Amarasinghe   Mon Nov 18 23:53:50 2019 +0000 typo
b93ad9d Chaminda Amarasinghe   Fri Nov 15 14:46:40 2019 +0000 print read summary then print head
cb71fec Chaminda Amarasinghe   Fri Nov 15 14:37:17 2019 +0000 print head of data set
33332d1 Chaminda Amarasinghe   Thu Nov 14 10:35:16 2019 +0000 added pydoc for graph
496a23d Chaminda Amarasinghe   Wed Nov 13 16:22:12 2019 +0000 fixed too broad exception catch
70414ce Chaminda Amarasinghe   Wed Nov 13 16:00:50 2019 +0000 logging in graphs
95acb4e Chaminda Amarasinghe   Wed Nov 13 16:00:16 2019 +0000 logging in graphs
a083c64 Chaminda Amarasinghe   Tue Nov 12 14:26:10 2019 +0000 moved price visualization to ab
4509273 Chaminda Amarasinghe   Tue Nov 12 14:21:40 2019 +0000 logging and comments
09a5123 Chaminda Amarasinghe   Mon Nov 11 16:34:34 2019 +0000 analytics based on availability
6379a50 Chaminda Amarasinghe   Mon Nov 11 16:16:25 2019 +0000 main to root dir
cd0eb5b Chaminda Amarasinghe   Mon Nov 11 15:32:27 2019 +0000 final commit
70ff603 Chaminda Amarasinghe   Mon Nov 11 15:00:51 2019 +0000 unit test for math module
eb3e74e Chaminda Amarasinghe   Mon Nov 11 11:13:41 2019 +0000 added math test
f0ac1a2 Chaminda Amarasinghe   Mon Nov 11 00:49:35 2019 +0000 file read summary
cb59cb0 Chaminda Amarasinghe   Mon Nov 11 00:16:02 2019 +0000 unit tests for db
3c99581 Chaminda Amarasinghe   Sun Nov 10 11:56:51 2019 +0000 pydoc for db module
fc263c2 Chaminda Amarasinghe   Sun Nov 10 10:16:54 2019 +0000 db name as parameter for making testable code
6ae1e82 Chaminda Amarasinghe   Sat Nov 9 16:25:15 2019 +0000 moved test data to tmp folder
5726ec1 Chaminda Amarasinghe   Sat Nov 9 16:15:52 2019 +0000 unit testing for file read
59d16c4 Chaminda Amarasinghe   Sat Nov 9 14:58:51 2019 +0000 unit testing for file read
1136902 Chaminda Amarasinghe   Sat Nov 9 14:58:24 2019 +0000 custom exception for file operations
498228b Chaminda Amarasinghe   Sat Nov 9 13:22:23 2019 +0000 pydoc for file_read
ee0e078 Chaminda Amarasinghe   Fri Nov 8 17:16:08 2019 +0000 added unit test for test_summary_list_error_cases
3de02ad Chaminda Amarasinghe   Fri Nov 8 17:08:03 2019 +0000 fixed code inspection issues
8f7989a Chaminda Amarasinghe   Fri Nov 8 16:52:30 2019 +0000 added unit test and pydoc for test_summary_list
3f4c2da Chaminda Amarasinghe   Thu Nov 7 18:59:41 2019 +0000 typo
2cd128a Chaminda Amarasinghe   Thu Nov 7 18:45:34 2019 +0000 unit test and pydoc for filter_by_index
157aa1c Chaminda Amarasinghe   Thu Nov 7 17:55:42 2019 +0000 group by doc correction
b95bd28 Chaminda Amarasinghe   Thu Nov 7 17:51:42 2019 +0000 filter_by_index should be generic to accept index instead tuple
5c891e1 Chaminda Amarasinghe   Thu Nov 7 17:46:32 2019 +0000 unit tests and doc for group_count_list and two_group_count_list
dc42ecf Chaminda Amarasinghe   Mon Nov 4 17:30:05 2019 +0000 pydoc and unit test for data group_by
b8a1d3e Chaminda Amarasinghe   Mon Nov 4 17:20:02 2019 +0000 pydoc and unit test for data group_by
bd2af8a Chaminda Amarasinghe   Mon Nov 4 16:29:03 2019 +0000 pydoc and unit test for data sort_data

```

f64017f Chaminda Amarasinghe	Mon Nov 4 15:06:51 2019 +0000	vertical_slice_all_data
e376b2f Chaminda Amarasinghe	Mon Nov 4 13:07:33 2019 +0000	unit test and py doc for vertical_slice_data
5804f9b Chaminda Amarasinghe	Mon Nov 4 01:06:14 2019 +0000	fixed bar chart saving issue
c30911e Chaminda Amarasinghe	Mon Nov 4 00:44:16 2019 +0000	graph saving
48d137f Chaminda Amarasinghe	Sun Nov 3 23:54:40 2019 +0000	added title to all graphs
82deab6 Chaminda Amarasinghe	Sun Nov 3 22:47:09 2019 +0000	title addition to graphs
9167b27 Chaminda Amarasinghe	Sun Nov 3 22:35:14 2019 +0000	refactored visualizing price
55f9bd7 Chaminda Amarasinghe	Sun Nov 3 22:08:28 2019 +0000	rename duplicate variables
e39e362 Chaminda Amarasinghe	Sun Nov 3 21:54:58 2019 +0000	analyze price greater than 1000
5e8616a Chaminda Amarasinghe	Sun Nov 3 21:51:05 2019 +0000	analyze price greater than 1000
99b7f0c Chaminda Amarasinghe	Sun Nov 3 17:32:48 2019 +0000	violin plots for price group by room type and neighbourhood group
71609ee Chaminda Amarasinghe	Sun Nov 3 17:26:34 2019 +0000	violin plot support
83fc9dc Chaminda Amarasinghe	Sat Nov 2 23:38:11 2019 +0000	mean value support for dis_plot
246077a Chaminda Amarasinghe	Sat Nov 2 23:03:56 2019 +0000	violinplot plot support for dist_plot
a56ebbf Chaminda Amarasinghe	Sat Nov 2 22:20:16 2019 +0000	group bar chart neighbourhood group id and room type
e2c8326 Chaminda Amarasinghe	Sat Nov 2 22:04:51 2019 +0000	pie chart and bar chart by room type, neighbourhood group id, group bar
chart room type and neighbourhood group id		chart room type and neighbourhood group id
f7e0c75 Chaminda Amarasinghe	Sat Nov 2 21:33:42 2019 +0000	pie plot support
c18768b Chaminda Amarasinghe	Sat Nov 2 21:33:10 2019 +0000	group by single index should use exact key instead single element tuple to avoid ('text',)
ddfdf39f Chaminda Amarasinghe	Sat Nov 2 20:28:10 2019 +0000	group bar chart support for common graphs
99f37c5 Chaminda Amarasinghe	Sat Nov 2 20:27:04 2019 +0000	group count list and two group count list
e4e950d Chaminda Amarasinghe	Sat Nov 2 01:15:57 2019 +0000	added 2 group support
9cedbf0 Chaminda Amarasinghe	Fri Nov 1 23:05:25 2019 +0000	analytics into sub modules. ab and common
264fa04 Chaminda Amarasinghe	Fri Nov 1 22:27:40 2019 +0000	corrected group by first element
15b0a3c Chaminda Amarasinghe	Fri Nov 1 17:01:45 2019 +0000	xlabel and ylabel for all pairplots
0ea44df Chaminda Amarasinghe	Fri Nov 1 15:13:08 2019 +0000	multi distribution plot support, started using logging
3f2be25 Chaminda Amarasinghe	Fri Nov 1 13:40:06 2019 +0000	multi distribution plot support
b3ebf2d Chaminda Amarasinghe	Fri Nov 1 13:39:37 2019 +0000	multi distribution plot support
72dcc6a Chaminda Amarasinghe	Fri Nov 1 12:41:33 2019 +0000	filter by index, limiting data
7de21d9 Chaminda Amarasinghe	Fri Nov 1 11:02:55 2019 +0000	multi_scatter for single field
d781e5e Chaminda Amarasinghe	Thu Oct 31 18:15:13 2019 +0000	print summary in table
a77c0bf Chaminda Amarasinghe	Thu Oct 31 16:58:58 2019 +0000	count chart and seaborn support to find regression
d9d5567 Chaminda Amarasinghe	Thu Oct 31 14:45:17 2019 +0000	refactored price_data to ab_data
8c4176f Chaminda Amarasinghe	Thu Oct 31 14:13:29 2019 +0000	multi scatter plot support for any data set
42bfdae Chaminda Amarasinghe	Thu Oct 31 14:12:06 2019 +0000	vertical_slice_all_data method added
27a5d48 Chaminda Amarasinghe	Wed Oct 30 18:13:53 2019 +0000	removed unused imports
5996899 Chaminda Amarasinghe	Wed Oct 30 17:48:01 2019 +0000	reading from db
81c0466 Chaminda Amarasinghe	Tue Oct 29 17:44:27 2019 +0000	group by index for any data set
81d0340 Chaminda Amarasinghe	Tue Oct 29 17:04:15 2019 +0000	corrected mode calculation
eb5887d Chaminda Amarasinghe	Tue Oct 29 00:09:40 2019 +0000	basic math functions count, min, mean, median, mode, sd, max and summary
for fields		
7443d14 Chaminda Amarasinghe	Mon Oct 28 20:35:10 2019 +0000	sort a tuple by given field index
16b6e77 Chaminda Amarasinghe	Mon Oct 28 02:29:28 2019 +0000	read csv data and convert to correct type
7ea02c5 Chaminda Amarasinghe	Mon Oct 28 00:23:23 2019 +0000	price data slicing
ced071f Chaminda Amarasinghe	Mon Oct 28 00:22:25 2019 +0000	const module
feb7c4f Chaminda Amarasinghe	Sun Oct 27 23:57:44 2019 +0000	added main_test for testing
fe82cb6 Chaminda Amarasinghe	Sun Oct 27 23:55:48 2019 +0000	added constants for field names
8f2b1d5 Chaminda Amarasinghe	Sun Oct 27 19:04:29 2019 +0000	add data to price_data table

Appendix 2: References

Müller, Andreas C. and Guido, Sarah. “Introduction to Machine Learning with Python”. O'Reilly Media, Inc., 2016. **ISBN:** 1449369898

Barry, Paul. “Head First Python, 2nd Edition”. O'Reilly Media, Inc., 2010. **ISBN:** 9781491919521

Yim, Aldrin. Chung, Claire. and Yu, Allen. “Matplotlib for Python Developers”. Packt Publishing Ltd. 2018. **ISBN:** 9781788625173

Boschetti, Alberto. and Massaron, Luca. “Python Data Science Essentials” Packt Publishing Ltd. 2015. **ISBN:** 9781785280429

Semi, Koen. “5 Key Principles of Software Architecture” <<https://towardsdatascience.com/5-key-principles-of-software-architecture-e5379cb10fd5>>. November 1st, 2019

Python Software Foundation. “Python Documentation” <<http://docs.python.org/3/>>. November 1st, 2019

NumPy developers. “NumPy Documentation” <<https://numpy.org/doc/>>. November 10th 2019

the Matplotlib development team. “Matplotlib Documentation” <<https://matplotlib.org/3.1.1/contents.html>>. November 15th 2019

Waskom, Michael. “Seaborn Documentation” <<https://seaborn.pydata.org/>>. November 15th 2019