Final Project Big Data Concepts: LA Airbnb Analysis

Section 1: Introduction

The data from Los Angeles, California's Airbnb listings served as the foundation for this project. I built this project using the concepts of virtualization, data ingestion and storage, and data pipelines from the big data concepts covered in the course. The Jupyter lab runs the Python code using a Jetstream VM. An input datafile is kept in the NoSQL database from MongoDB Atlas as a collection titled "Listings." Python script is used to access this file and retrieve data from it. The data analysis and data publishing are then carried out using a model of the data pipeline. These can be broadly categorized as follows:

1. Setting up a MongoDB cluster using a MongoDB database from MongoDB Atlas.
2. Using MongoDB Atlas, I ingested the csv data for LA's Airbnb listings as collections.
3. Used previously created Jetstream VM instance to run the Jupyter lab from a docker, just like we did for the assignment.
4. Using Python's MongoDB connector, I connected MongoDB to Python to fetch the data.
5. Following data cleaning, data pre-processing, and data analysis, data pipelining was put into place.
6. Results that were visualized with Python and the MongoDB Chart Builder.

Section 2: Background

As an international student, I'm curious about the United States and its tourist attractions. Los Angeles is the second most populous city in the United States and one of the world's most visited cities. Airbnb, the popular online marketplace for short-term lodging rentals, has grown in popularity in Los Angeles in recent years, offering travellers a variety of affordable lodging options. I haven't yet had the chance to visit Los Angeles. My friends and I are planning a trip to the city this summer, so finding the best Airbnb listings in and around the city would be a fun project for me to work on. This would allow us to collect information that would help us find affordable Airbnb's. Moreover, considering the technical aspect, this project entails analysing data from Los Angeles Airbnb listings, such as the type of accommodation, location, pricing, and availability. In addition, this project provides an opportunity to design the workflow architecture to work with a large dataset stored on the cloud, performing data cleaning and pre-processing, and obtain results using popular data analysis and visualization libraries such as Pandas, Matplotlib, and Seaborn.

I'm curious about the listing's distribution in neighbourhood groups and what are the top-5 neighbourhood places where we could live, price distribution of the listings these groups and top-5 neighbourhood places, and distribution of room types to stay if we go ahead with the trip plan. Because the results of this project would provide us with information about Airbnb listings, I chose this as my project.

Section 3: Methodology

The process of designing a system that allows data to flow from multiple sources to a target system via a series of interconnected steps is referred to as data pipeline architecture. It entails locating data sources, integrating and processing the data, and finally storing or delivering the data to a destination system or application. For this course project a brief architectural diagram is designed as below, showing its lifecycle.

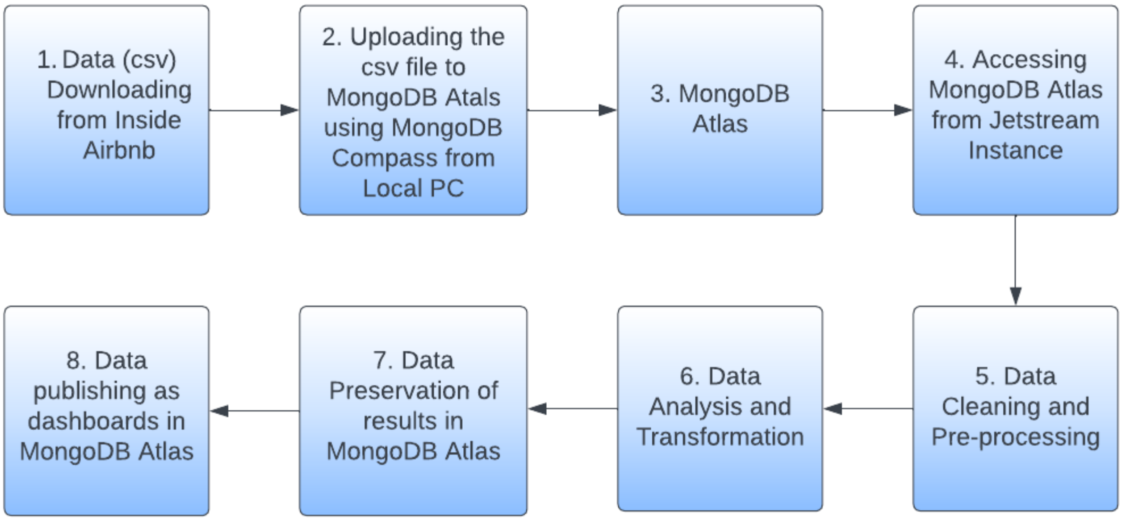


Figure 1: Data pipeline architecture for the Airbnb Analysis in L.A

Data Acquisition:

From Figure 1, all the steps from 1-4 are included in this sub-section. The required data for proceeding with the project has been collected from Inside Airbnb website http://insideairbnb.com/get-the-data and stored it in MongoDB Atlas by following the below steps. I collected the listing.csv from the Los Angeles, U.S.

Step 1: Setting up a MongoDB Cluster

For this project in order store the Airbnb listing data in LA, I used MongoDB, a NoSQL database. I set up a free shared cluster in Iowa (us-central1) with three nodes, one primary and two secondaries, and a M0 Sandbox Cluster Tier.

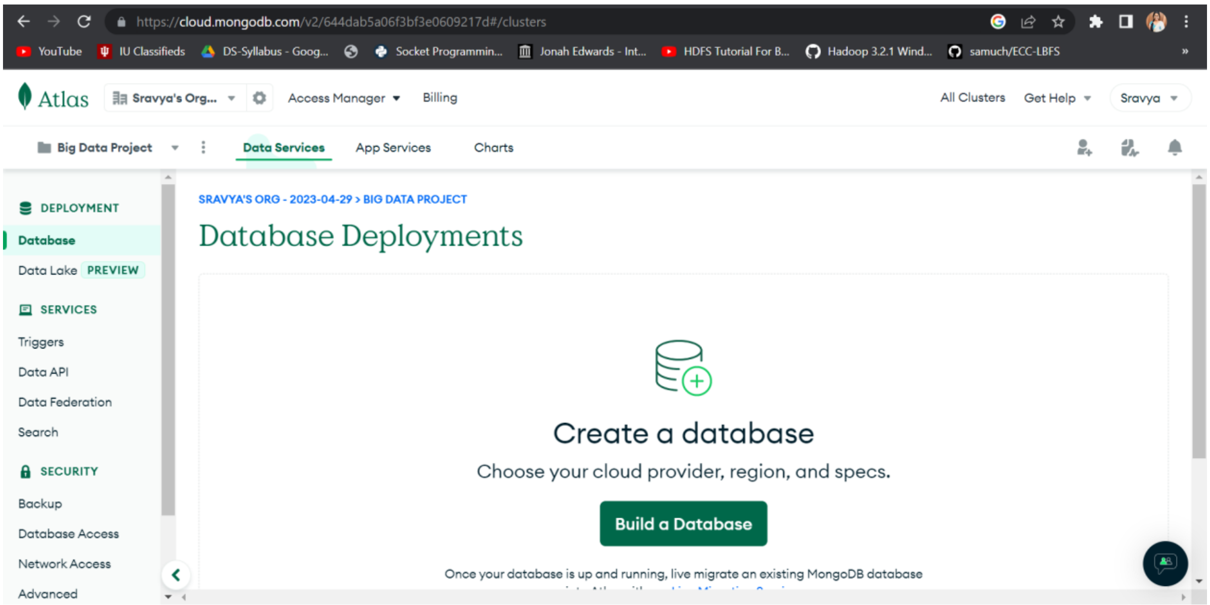


Figure 2: Opening of the MongoDB Atlas account.

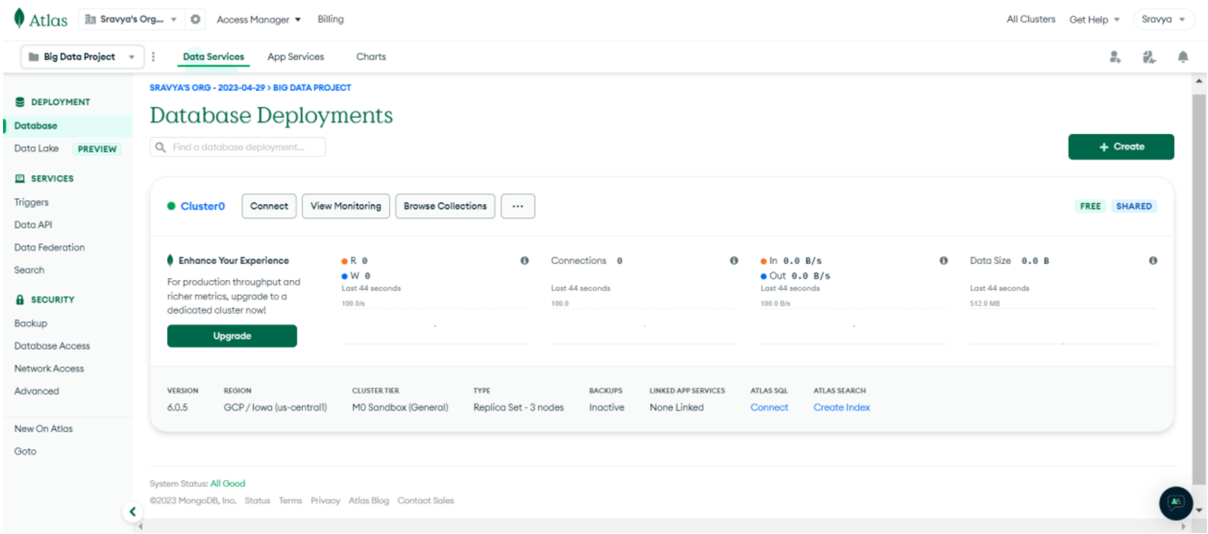


Figure 3: Setting up the Cluster0

Step 2: Importing Data into MongoDB Cluster using MongoDB Compass.

I used MongoDB Compass on my local machine to load data into the MongoDB Atlas. Later, I used the authentication to database to connect to the Cluster0 present in MongoDB Atlas. Then, within the Airbnb database, I created an input collection called "Listings" to import the listings.csv. It stores the csv files into a list of documents because it is a NoSQL database. The imported file contains approximately 42.5K documents in total.

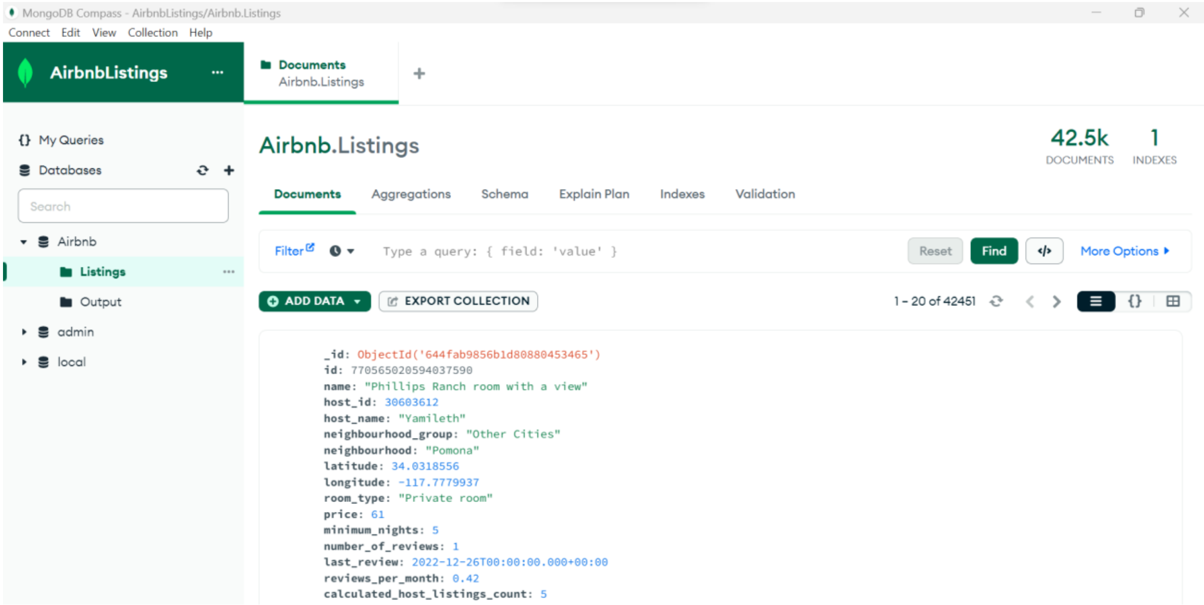


Figure 4: MongoDB Compass UI, to see the imported csv file as list of documents.

Step 3: Network Access

I added all the IP addresses to this Network Access for the cluster configuration. My Local PC's and also the Jetstream's IP Address. The below figure gives the list of IP addresses accessible by Cluster0 in MongoDB Atlas.

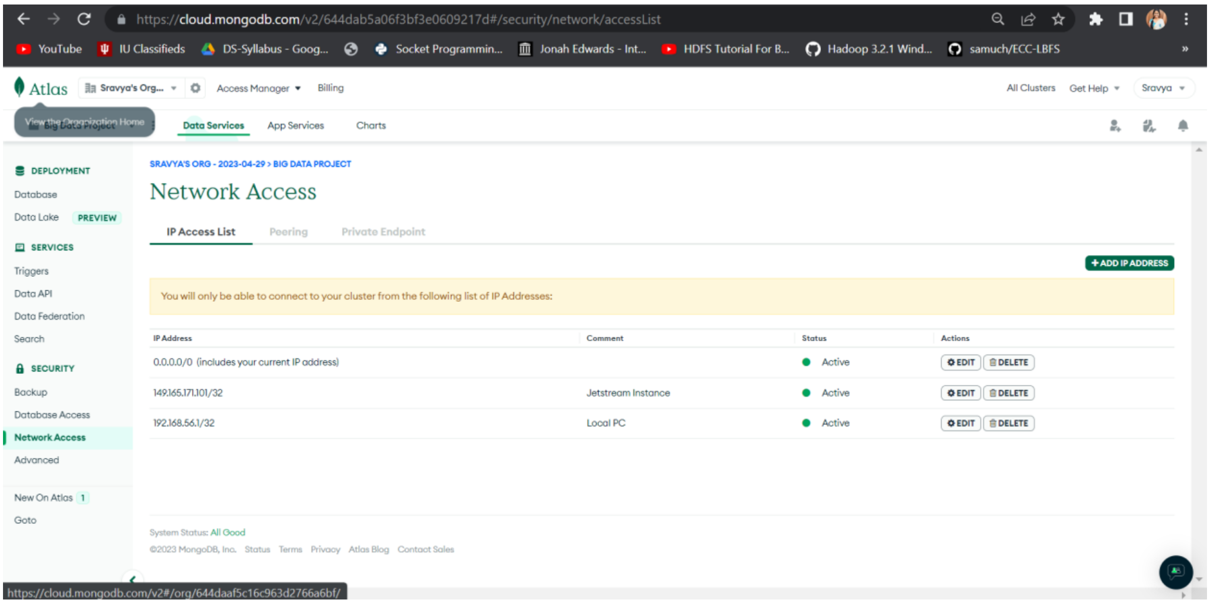


Figure 5: Network Access for the Cluster0

Step 4: Setting up the VM and Docker for Jupyter lab.

To run the Jupyter lab on a docker similar to one of our assignments, I used a Jetstream instance with a small size, two CPU cores, 6 GB of RAM, and 20 GB of root disk under our class directory. I followed the steps in the 'Analysing data with PySpark' assignment, creating a Project folder with a docker-compose.yaml file, and adding the Jupyter notebook image to the path before starting a container. I copied the URL and opened the Web Desktop browser, which launched the Jupyter lab notebook.

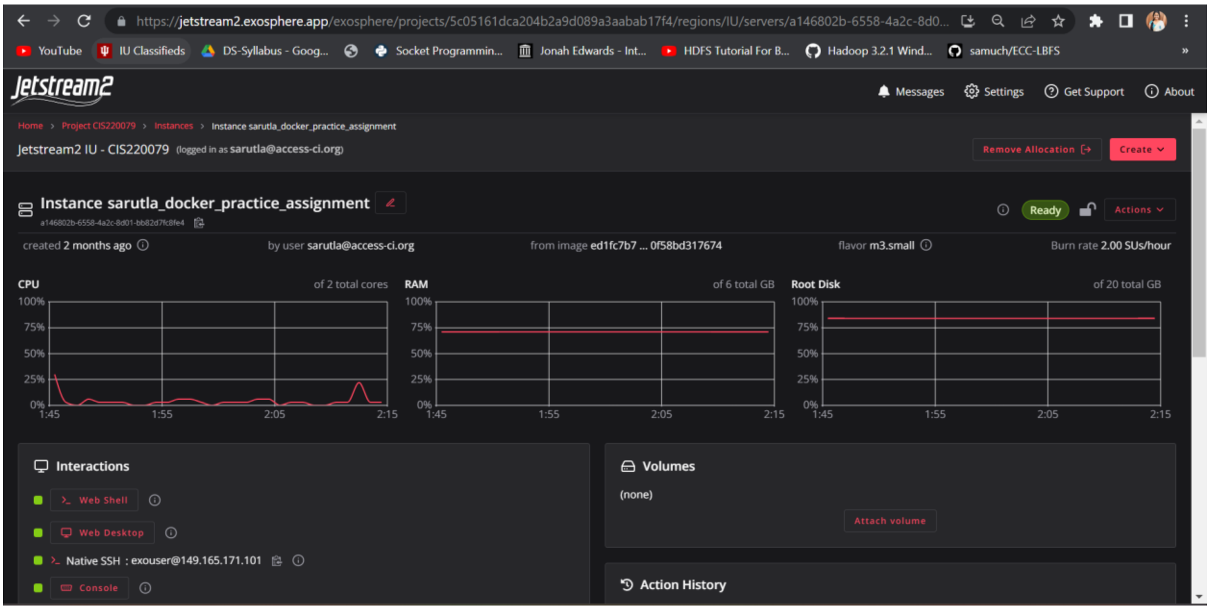


Figure 6: Jetstream Instance Overview

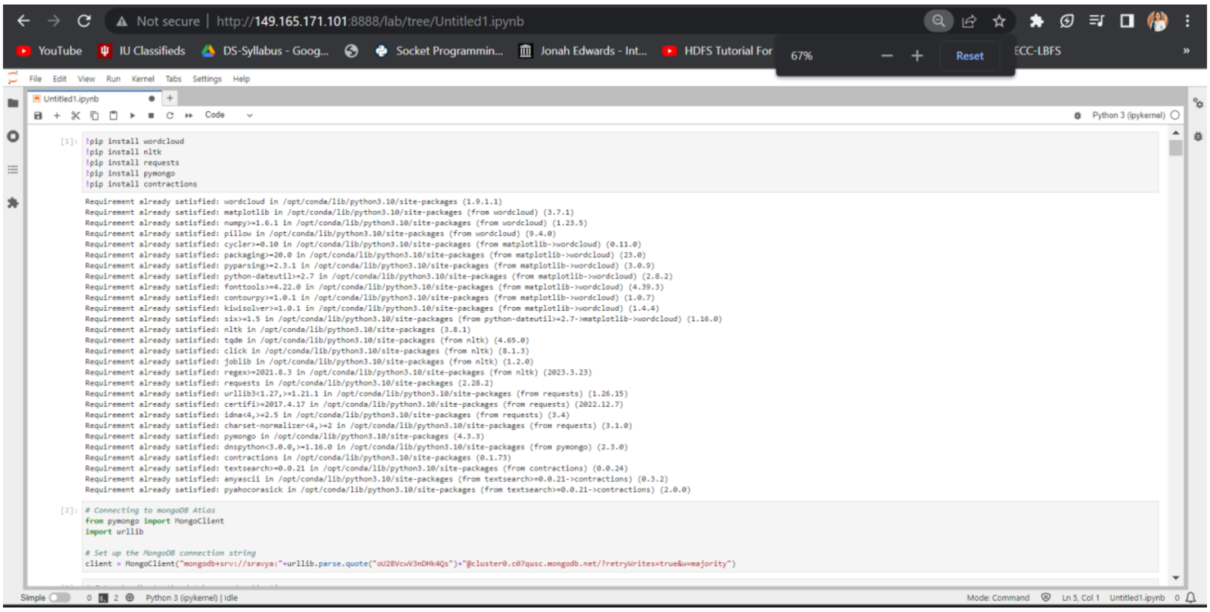


Figure 7: Jupyter Notebook on Web Desktop

Step 5: Using Pymongo to fetch the data into the Jupyter lab running on VM from MongoDB Atlas.

All the necessary packages were imported to perform the EDA in Jupyter notebook and then the with the python package pymongo, a collection has been established to fetch the data from the Cluster0 in MongoDB Atlas's as below.

```
[1]: !pip install wordcloud
     !pip install nltk
     !pip install requests
     !pip install pymongo

Requirement already satisfied: wordcloud in /opt/conda/lib/python3.10/site-packages (1.9.1.1)
Requirement already satisfied: numpy>=1.6.1 in /opt/conda/lib/python3.10/site-packages (from wordcloud) (1.23.5)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.10/site-packages (from wordcloud) (3.7.1)
Requirement already satisfied: pillow in /opt/conda/lib/python3.10/site-packages (from wordcloud) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.10/site-packages (from matplotlib->wordcloud) (3.0.9)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.10/site-packages (from matplotlib->wordcloud) (23.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.10/site-packages (from matplotlib->wordcloud) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.10/site-packages (from matplotlib->wordcloud) (1.4.4)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.10/site-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.10/site-packages (from matplotlib->wordcloud) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
Requirement already satisfied: nltk in /opt/conda/lib/python3.10/site-packages (3.8.1)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.10/site-packages (from nltk) (4.65.0)
Requirement already satisfied: joblib in /opt/conda/lib/python3.10/site-packages (from nltk) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in /opt/conda/lib/python3.10/site-packages (from nltk) (2023.3.23)
Requirement already satisfied: click in /opt/conda/lib/python3.10/site-packages (from nltk) (8.1.3)
Requirement already satisfied: requests in /opt/conda/lib/python3.10/site-packages (2.28.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.10/site-packages (from requests) (3.1.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/python3.10/site-packages (from requests) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-packages (from requests) (2022.12.7)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-packages (from requests) (3.4)
Requirement already satisfied: pymongo in /opt/conda/lib/python3.10/site-packages (4.3.3)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in /opt/conda/lib/python3.10/site-packages (from pymongo) (2.3.0)

[2]: # Connecting to mongoDB Atlas
     from pymongo import MongoClient
     import urllib

     # Set up the MongoDB connection string
     client = MongoClient("mongodb+srv://sravya:"+urllib.parse.quote("oU2BVcwV3nDHk4Qs")+"@cluster0.c07qusc.mongodb.net/?retryWrites=true&w=majority")
```

Figure 8: Establishing the pymongo connection through Jupyter lab.

Once connection is established, I have accessed the database – Airbnb and its input collection called "Listings" into the python dictionary, later on converted that into a pandas data frame for doing Analysis.

```
[3]: # Get a handle to the database and collection
     db = client["Airbnb"]
     collection1 = db["Listings"]
     collection2 = db["Analysis_6a"]
     collection3 = db["Analysis_6b"]

     # Example query to check the connection with the client mongoDB.
     result1 = collection1.find_one({"id": 770565020594037590})
     print(result1)

     {'_id': ObjectId('644fab9856b1d80880453465'), 'id': 770565020594037590, 'name': 'Phillips Ranch room with a view', 'host_id': 30603612, 'host_name': 'Yamileth', 'neighbourhood_group': 'Ot
     rhood': 'Pomona', 'latitude': 34.0318556, 'longitude': -117.7779937, 'room_type': 'Private room', 'price': 61, 'minimum_nights': 5, 'number_of_reviews': 1, 'last_review': datetime.datetim
     'reviews_per_month': 0.42, 'calculated_host_listings_count': 5, 'availability_365': 27, 'number_of_reviews_ltm': 1}

[4]: # Storing the collection data locally using the python dictionary.
     x = collection1.find({})
     listings_dict=[]
     neighbourhoods_dict=[]

     for i in x:
         del i['_id']
         listings_dict.append(i)

[5]: import pandas as pd

     # Create a pandas DataFrame from listings_dict
     listings_df = pd.DataFrame(listings_dict)
```

Figure 9: Python code for fetching the data into a pandas data frame.

```
[6]: listings_df.head()
```

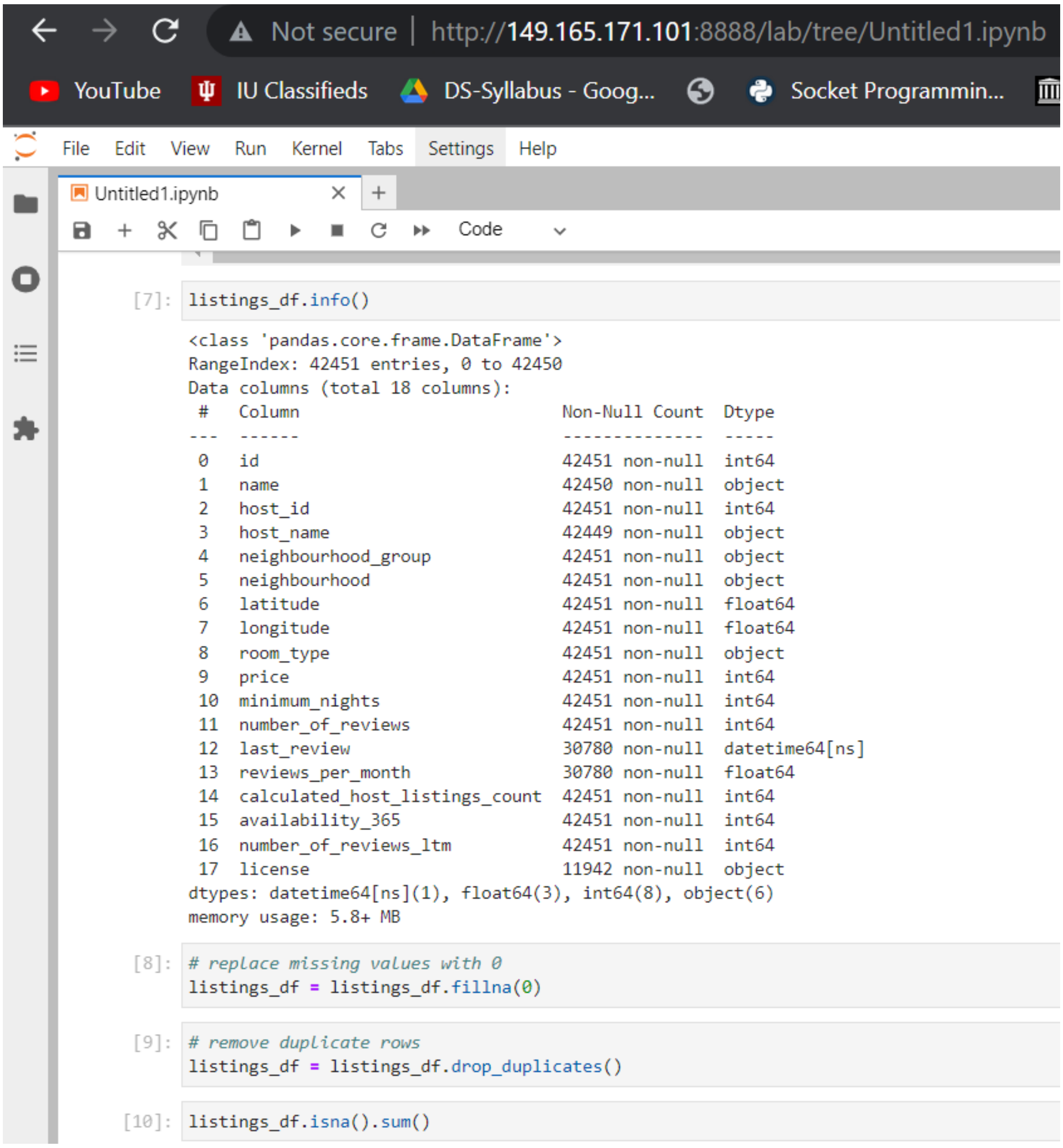
[6]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month
0	770565020594037590	Phillips Ranch room with a view	30603612	Yamileth	Other Cities	Pomona	34.031856	-117.777994	Private room	61	5	1	2022-12-26	0.42
1	777451666060243581	Family oriented home	490709958	Dana	Unincorporated Areas	Castaic Canyons	34.439250	-118.444090	Private room	399	3	0	NaT	NaN
2	698097753730921190	方便易居驿站	444692513	Hanna	Other Cities	Pomona	34.048480	-117.768870	Private room	46	3	5	2023-01-16	0.86
3	13063118	TERRANEA OCEANFRNT 1BR CASITA -203 RSRT AMENIT...	41736985	Beth	Other Cities	Rancho Palos Verdes	33.738420	-118.395800	Private room	408	3	53	2022-11-26	0.64
4	39337076	Brand New Duplex Near World Cruise Center	208375458	Jae	Other Cities	Rancho Palos Verdes	33.748420	-118.311000	Private room	80	2	44	2023-02-24	1.09

Figure 10: Listings_df content

Data Cleaning:

From the obtained listings_df, the data has been cleaned. Null values, Duplicates have been dropped. Also, the price outliers have been removed as below. This is the part of step 5 in the Figure 1.

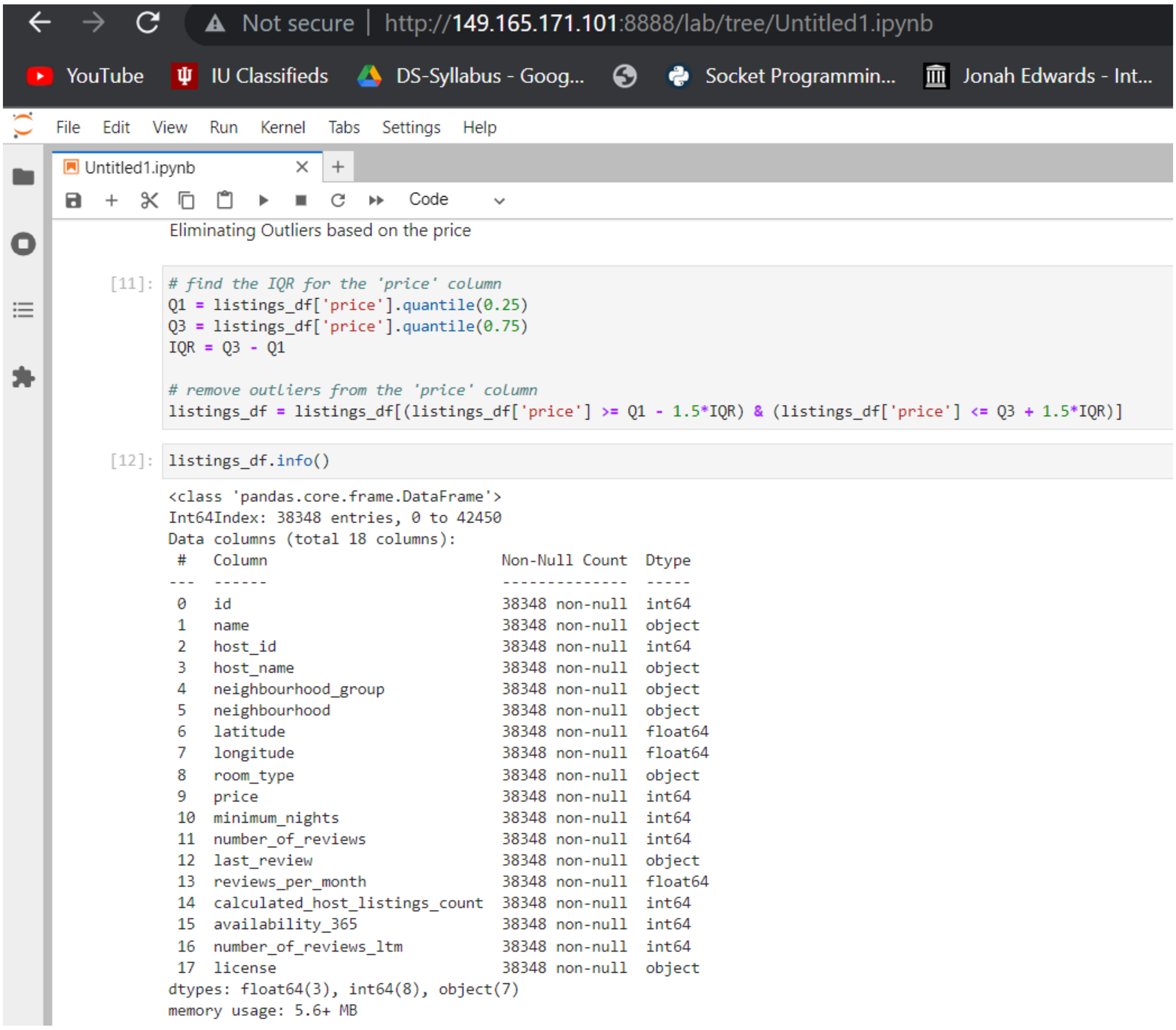



```
[7]: listings_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42451 entries, 0 to 42450
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              42451 non-null  int64
 1   name                            42450 non-null  object
 2   host_id                         42451 non-null  int64
 3   host_name                       42449 non-null  object
 4   neighbourhood_group             42451 non-null  object
 5   neighbourhood                   42451 non-null  object
 6   latitude                        42451 non-null  float64
 7   longitude                       42451 non-null  float64
 8   room_type                       42451 non-null  object
 9   price                           42451 non-null  int64
 10  minimum_nights                  42451 non-null  int64
 11  number_of_reviews               42451 non-null  int64
 12  last_review                     30780 non-null  datetime64[ns]
 13  reviews_per_month               30780 non-null  float64
 14  calculated_host_listings_count  42451 non-null  int64
 15  availability_365                42451 non-null  int64
 16  number_of_reviews_ltm           42451 non-null  int64
 17  license                         11942 non-null  object
dtypes: datetime64[ns](1), float64(3), int64(8), object(6)
memory usage: 5.8+ MB

[8]: # replace missing values with 0
     listings_df = listings_df.fillna(0)

[9]: # remove duplicate rows
     listings_df = listings_df.drop_duplicates()

[10]: listings_df.isna().sum()
```

Figure 11: Data Cleaning

Data Pre-processing:

From the original dataset, I have removed unnecessary columns host_name, last_review, reviews_per_month and license columns from original dataset. host_name is removed as it causes issues with the user's privacy. I removed the other columns as I was not using them in my analysis purpose. This is the part of step 5 in the Figure 1




```python
# find the IQR for the 'price' column
Q1 = listings_df['price'].quantile(0.25)
Q3 = listings_df['price'].quantile(0.75)
IQR = Q3 - Q1

# remove outliers from the 'price' column
listings_df = listings_df[(listings_df['price'] >= Q1 - 1.5*IQR) & (listings_df['price'] <= Q3 + 1.5*IQR)]
```

```python
listings_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38348 entries, 0 to 42450
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              38348 non-null  int64
 1   name                            38348 non-null  object
 2   host_id                         38348 non-null  int64
 3   host_name                       38348 non-null  object
 4   neighbourhood_group             38348 non-null  object
 5   neighbourhood                   38348 non-null  object
 6   latitude                        38348 non-null  float64
 7   longitude                       38348 non-null  float64
 8   room_type                       38348 non-null  object
 9   price                           38348 non-null  int64
 10  minimum_nights                  38348 non-null  int64
 11  number_of_reviews               38348 non-null  int64
 12  last_review                     38348 non-null  object
 13  reviews_per_month               38348 non-null  float64
 14  calculated_host_listings_count  38348 non-null  int64
 15  availability_365                38348 non-null  int64
 16  number_of_reviews_ltm           38348 non-null  int64
 17  license                         38348 non-null  object
dtypes: float64(3), int64(8), object(7)
memory usage: 5.6+ MB
```

Figure 12: Data Pre-processing

It is seen that, after the data cleaning and the data pre-processing the number of entries in the data frame is now 38K. Hence, this step removed the unwanted data for giving the beneficial results in our further analysis.

Section 4: Results

This section focuses on the steps 6, 7 and 8 from the Figure 1.

Data Analysis:

Analysis 1: Which neighbourhood groups and neighbourhoods have the highest Airbnb Listings available in LS?

Analysis 1: Neighbourhood and Neighbourhood_group Distribution

```
[16]: listings_df.neighbourhood_group.unique()

[16]: array(['Other Cities', 'Unincorporated Areas', 'City of Los Angeles'],
            dtype=object)

[17]: Result1a_df = listings_df.neighbourhood_group.value_counts()
      Result1a_df

[17]: City of Los Angeles     19912
      Other Cities            14867
      Unincorporated Areas     3569
      Name: neighbourhood_group, dtype: int64
```

Figure 13: Python Code for Analysis 1.

From the listings data frame, unique neighbourhood groups have been identified. As shown in above figure, we have city of Los Angeles has the highest listings, followed by other cities and Unincorporated areas. Its bar plot is as given below. Python's seaborn and matplotlib has been used for visualisations.

```
Name: neighbourhood_group, dtype: int64

[18]: import matplotlib.pyplot as plt
      import seaborn as sns
      sns.set(font_scale=1.0)
      listings_df.neighbourhood_group.value_counts().plot(kind='bar',figsize=(10, 5));
      plt.xlabel("Neighbourhood groups in LA")
      plt.ylabel("Number of Airbnb's booked")
      plt.title("Neighbourhood group wise Airbnb booking distribution in LA", y=1);
```

Figure 14: Neighbourhood group wise Listings distribution.

Similarly, distribution for neighbourhood is also plotted as shown below. Here I am concentrating on top-5 neighbourhoods for the listings available.

```
[19]: print(len(listings_df.neighbourhood.unique()))

265

[20]: Result1b_df = listings_df.neighbourhood.value_counts()
      Result1b_df

[20]: Sherman Oaks          2096
      Hollywood             1608
      Long Beach            1477
      Venice                1334
      Santa Monica          1158
                            ...
      Sepulveda Basin          2
      Walnut Park              2
      Hasley Canyon            2
      Lake View Terrace        1
      Elizabeth Lake           1
      Name: neighbourhood, Length: 265, dtype: int64

[21]: top_neighborhoods = listings_df['neighbourhood'].value_counts().head(5).index.tolist()
      listings_top5 = listings_df[listings_df['neighbourhood'].isin(top_neighborhoods)]
```

Figure 15: Python Code for Analysis 1

It is seen that the places Sherman Oaks, Hollywood, Long Beach, Venice and Santa Monica are the top 5 places where there are more Airbnb's listings are distributed more comparatively, because of its famous and most populated areas.

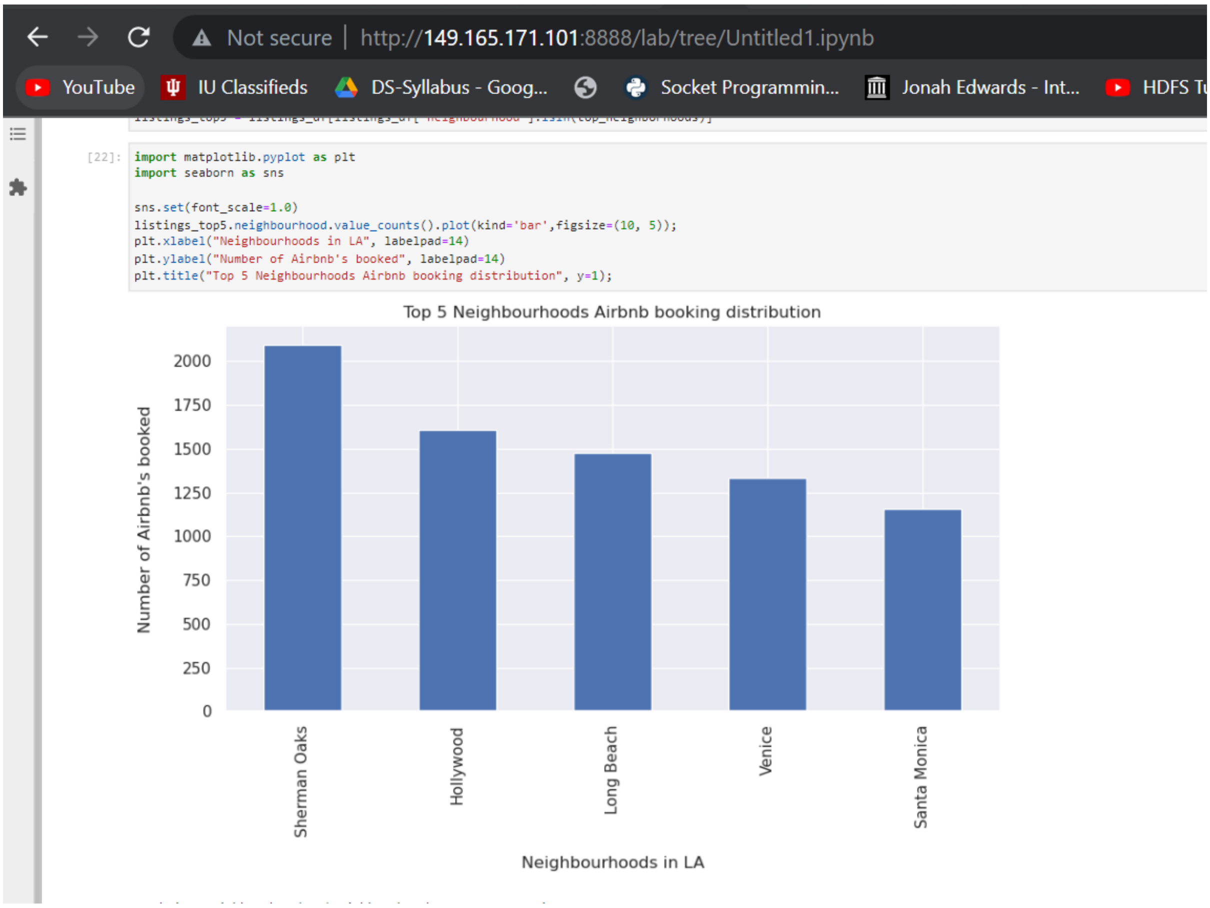


Figure 16: Top 5 Neighbourhoods Listings distribution

Analysis 2: What is the average price distribution as per the neighbourhood groups and top 5 neighbourhoods?

When neighbourhood_group is grouped by the mean of prices, it Is seen that, Average cost is highest for the Other Cities, followed by City of Los Angeles and then Unincorporated areas. The result is interesting as Other Cites are having higher average price when compared to the City of Los Angeles.

Analysis 2: Neighbourhood and neighbouthood_group average price

[23]:
```
Result2a_df = listings_df.groupby('neighbourhood_group')['price'].mean().reset_index()
Result2a_df
```

[23]:

	neighbourhood_group	price
0	City of Los Angeles	151.434562
1	Other Cities	159.348624
2	Unincorporated Areas	154.959092

[24]:
```
# Create a pie chart of mean price for each neighbourhood using Matplotlib
plt.pie(Result2a_df['price'], labels=Result2a_df['neighbourhood_group'], autopct='%1.1f%%', startangle=90)
plt.axis('equal')
# Set the plot title using Matplotlib
plt.title('Average price per neighbourhood group')
plt.show()
```

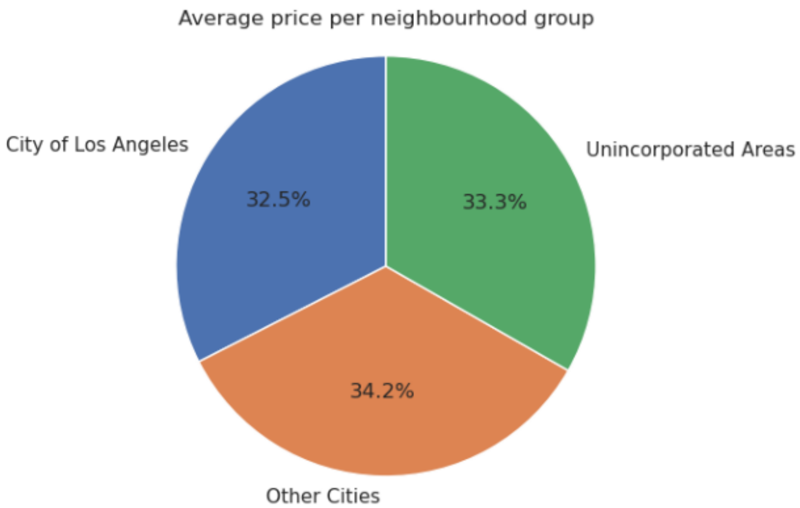


Figure 17: Average price distribution based on neighbourhood group.

Similarly, the average pricing for the top-5 neighbourhoods is analysed as below.

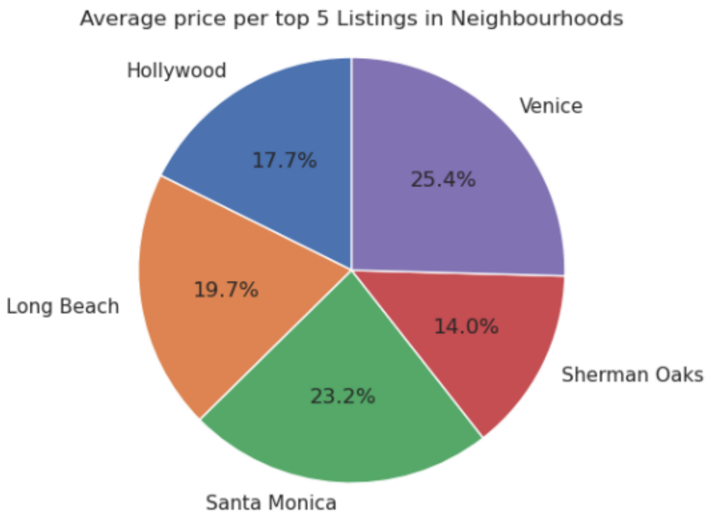
```
[25]: Result2b_df = listings_top5.groupby('neighbourhood')['price'].mean().reset_index()
```

```
[26]: Result2b_df
```

[26]:

	neighbourhood	price
0	Hollywood	144.921642
1	Long Beach	162.004062
2	Santa Monica	190.224525
3	Sherman Oaks	114.937023
4	Venice	208.700150

```
[27]: # Create a pie chart of mean price for each neighbourhood using Matplotlib
plt.pie(Result2b_df['price'], labels=Result2b_df['neighbourhood'], autopct='%1.1f%%', startangle=90)
plt.axis('equal')
# Set the plot title using Matplotlib
plt.title('Average price per top 5 Listings in Neighbourhoods')
plt.show()
```



It is seen that, the average price of the listings is high for Venice, followed by Santa Monica

Figure 18: Average price distribution based on top-5 neighbourhoods.

It is seen that, the average price is maximum for the Venice, followed by Santa Monica. Even though the greatest number of listings is present in Sherman Oaks, Venice has the higher average price for the Airbnb Listings.

Analysis 3: What is the Distribution of Room Types with the Listings in L.A

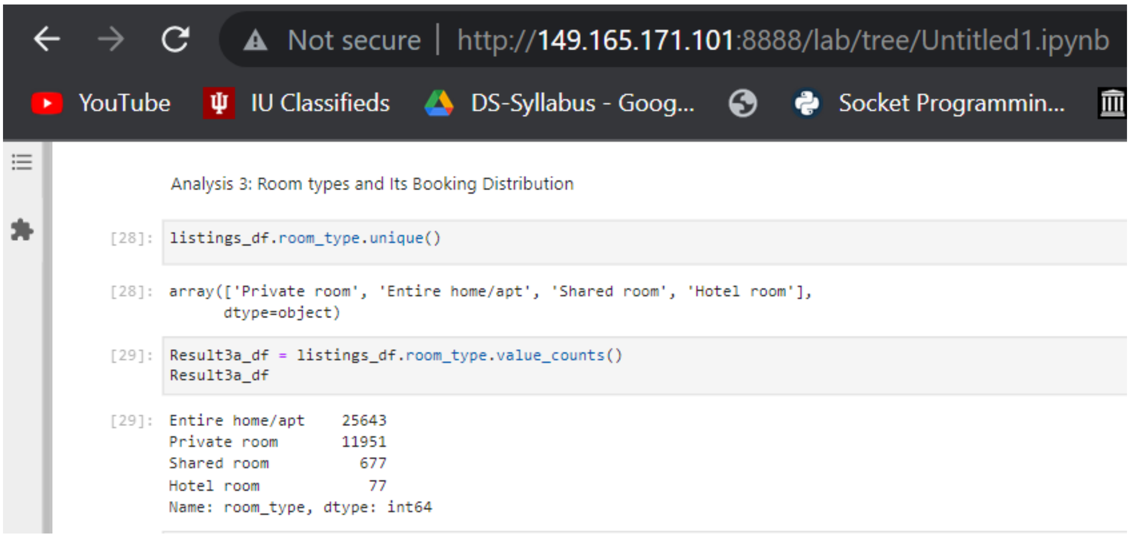


Figure 19: Python code for Analysis 3

More Airbnb Listings are of home/apt type, followed by the Private room, Shared rooms and Hotel room. As more people do visit this place with family/friends, they tend to book home/apt more when compared to others types. Its bar plot can be plotted as below.

```
[30]: listings_df.room_type.value_counts().plot(kind='bar',figsize=(10, 5));
plt.xlabel("Room Types")
plt.ylabel("Number of Airbnb's booked")
plt.title("Room Type wise booking distribution", y=1.0);
```

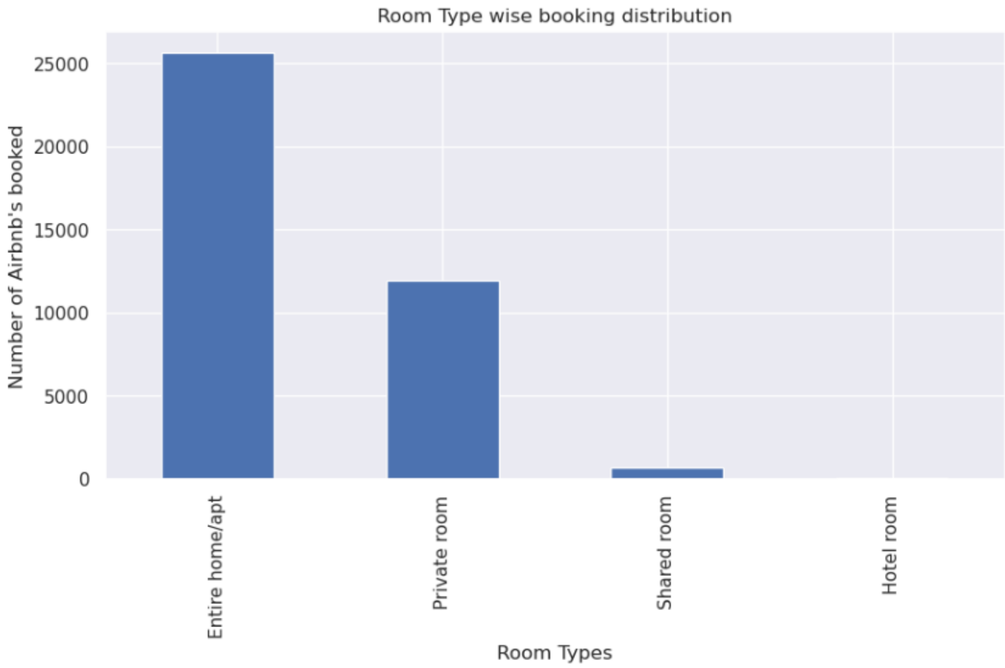


Figure 20: Distribution of room type booking in Listings.

Also, when grouped against the neighbourhood groups, the below plot is obtained. In all the 3 groups, we see that mostly Airbnb listings has Entire home/apt types of rooms available. In Unincorporated areas, it is seen that, there are no shared room available, which would be an interesting finding.

```
[31]: plt.figure(figsize=(10, 5))
      sns.set_theme(style="whitegrid")
      sns.countplot(x='neighbourhood_group', hue='room_type', data=listings_df)
```

```
[31]: <Axes: xlabel='neighbourhood_group', ylabel='count'>
```

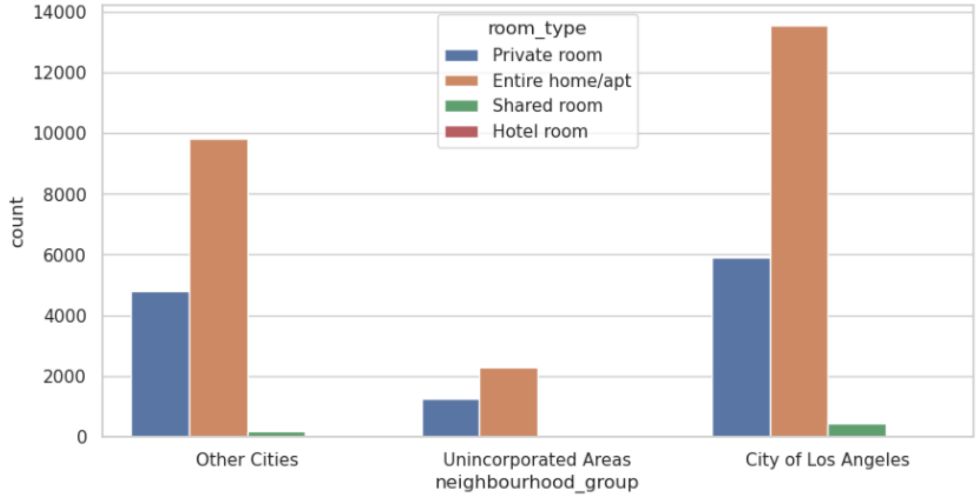


Figure 21: Distribution of room type booking in neighbourhood groups.

Similarly, the distribution for top-5 neighbourhood in the listings is also plot and the results are as below. Sherman Oaks has highest number of Private rooms than the other types. But for the other top-5 places, we have the regular Entire home/apt as the top one.

```
[32]: plt.figure(figsize=(10,5))
      sns.set_theme(style="whitegrid")
      sns.countplot(x='neighbourhood', hue='room_type', data=listings_top5)
```

```
[32]: <Axes: xlabel='neighbourhood', ylabel='count'>
```

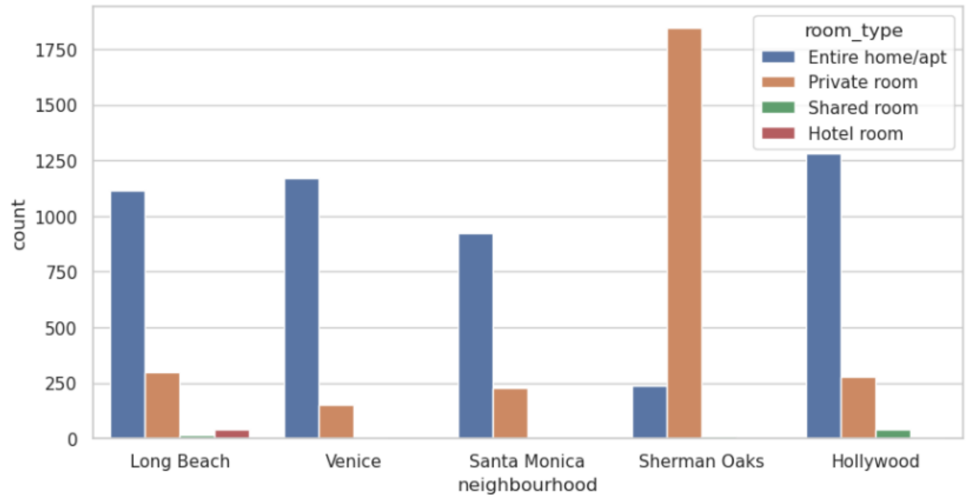


Figure 22: Distribution of room type booking in top-5 neighbourhoods.

Analysis 4: Who are the Top Listings hosts?

In order to protect the privacy of the users, the distribution has been plotted against the host_id instead as below. The top most host_id has 1001 bookings registered.

```
[33]: top_hosts=listings_df.host_id.value_counts().head(10)
      top_hosts
```

```
[33]: 144214204    1001
      107434423     717
      401130632     663
      891818        136
      48005494      134
      101537031     129
      464261743     109
      134267499     102
      263524662      87
      271118401      81
      Name: host_id, dtype: int64
```

```
[34]: sns.set(font_scale=1.4)
      top_hosts.plot(kind='bar',figsize=(10, 5));
      plt.xlabel("Top Host IDs")
      plt.ylabel("Number of listings")
      plt.title("Top 10 Hosts with most number of listings", y=1);
```

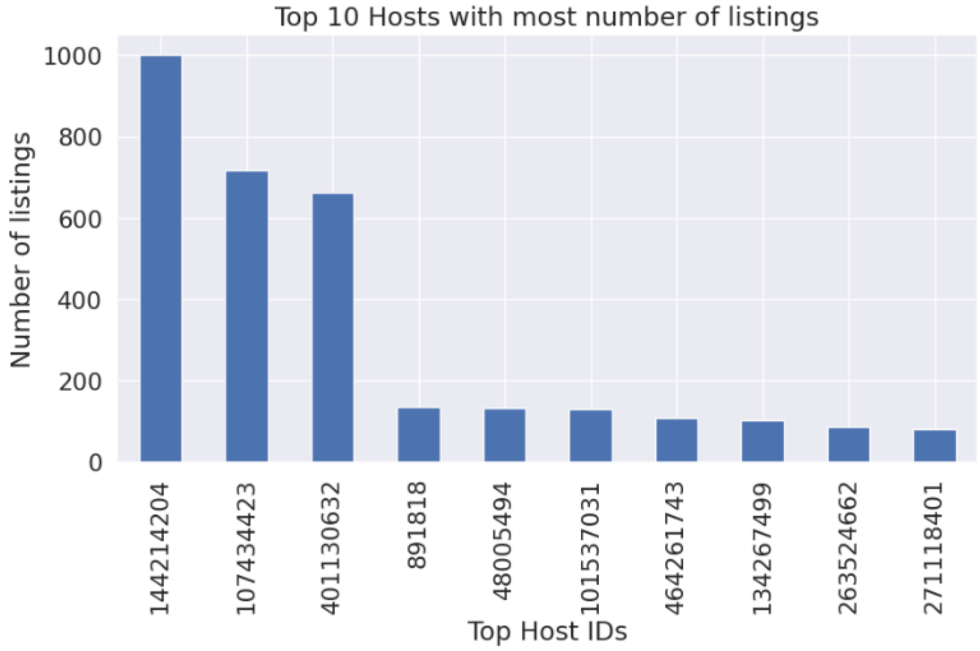


Figure 23: Top Host IDs distribution.

Analysis 5: Neighbourhood group vs availability of rooms in 365 days.

For this analysis, I have used the box plot distribution to get the acquired results. A box plot is a graphical representation of a continuous variable's distribution. It displays a five-number summary of a dataset, including the minimum value, first quartile (Q1), median (Q2), third quartile (Q3), and maximum value. Seaborn is a Python data visualization library that offers a high-level interface for producing informative and visually appealing statistical graphics.

Reference: https://practicaldatascience.co.uk/data-science/how-to-visualise-data-using-boxplots-in-seaborn

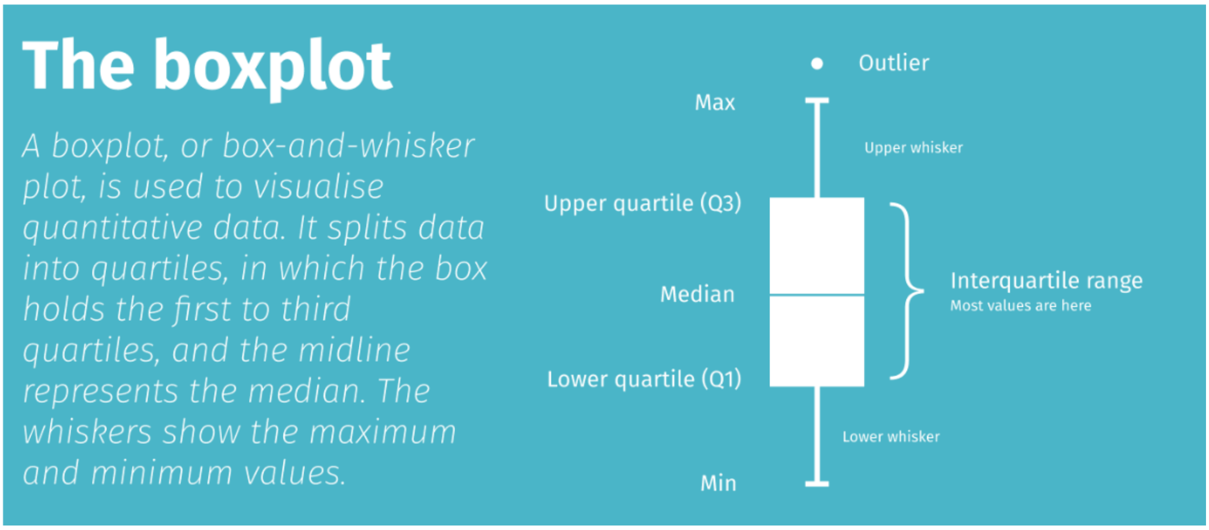


Figure 24: Understanding Box Plot.

Analysis 5: Neighbourhood group vs availability of rooms

```
[35]: plt.figure(figsize=(10,5))
sns.set_style('white')

ax = sns.boxplot(data=listings_df, x='neighbourhood_group',y='availability_365',palette='pastel')

ax.set_title('Relation between Neighbourhood group & Availability of rooms', fontsize=15)

ax.set_ylabel('Availability 365 Days', fontsize=15)
ax.set_xlabel('Neighbourhood Group', fontsize=15)

#Adjusting Bar Labels
ax.set_xticklabels(ax.get_xticklabels(), fontsize=15)
```

```
[35]: [Text(0, 0, 'Other Cities'),
 Text(1, 0, 'Unincorporated Areas'),
 Text(2, 0, 'City of Los Angeles')]
```

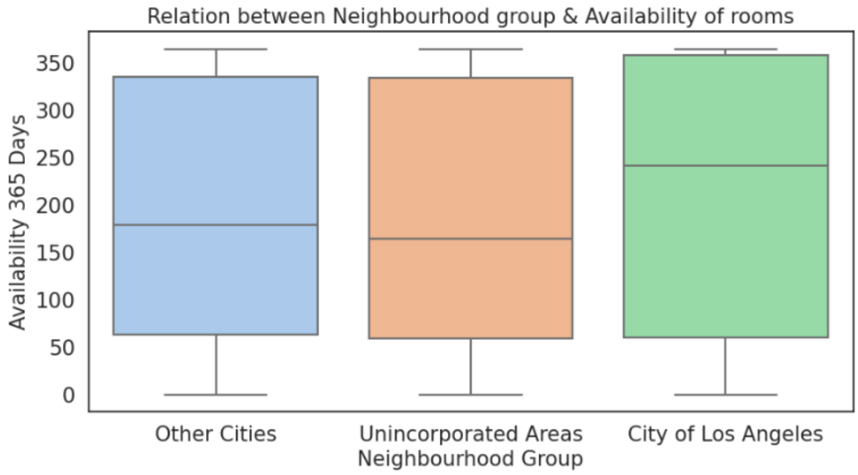


Figure 25: Box plot distribution for Neighbourhood groups and availability_365

It is seen that, out of the 3 neighbourhood groups available, the City of Los Angeles has more availability across all the 365 days when compared to the other groups. Also, this could be the reason why it has more listings too.

Similarly, the box has been plotted against the top-5 neighbourhood whose results are as below.

```
[36]: plt.figure(figsize=(10,5))
      sns.set_style('white')

      ax = sns.boxplot(data=listings_top5, x='neighbourhood',y='availability_365',palette='pastel')

      ax.set_title('Relation between Neighbourhood group & Availability of rooms', fontsize=15)

      ax.set_ylabel('Availability 365 Days', fontsize=15)
      ax.set_xlabel('Neighbourhood Group', fontsize=15)

      #Adjusting Bar Labels
      ax.set_xticklabels(ax.get_xticklabels(), fontsize=15)
```

```
[36]: [Text(0, 0, 'Long Beach'),
       Text(1, 0, 'Venice'),
       Text(2, 0, 'Santa Monica'),
       Text(3, 0, 'Sherman Oaks'),
       Text(4, 0, 'Hollywood')]
```

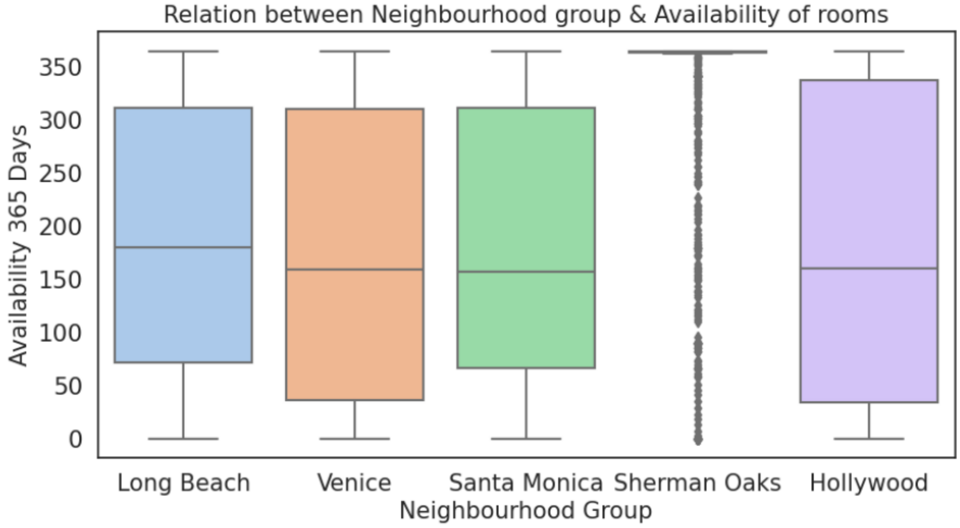
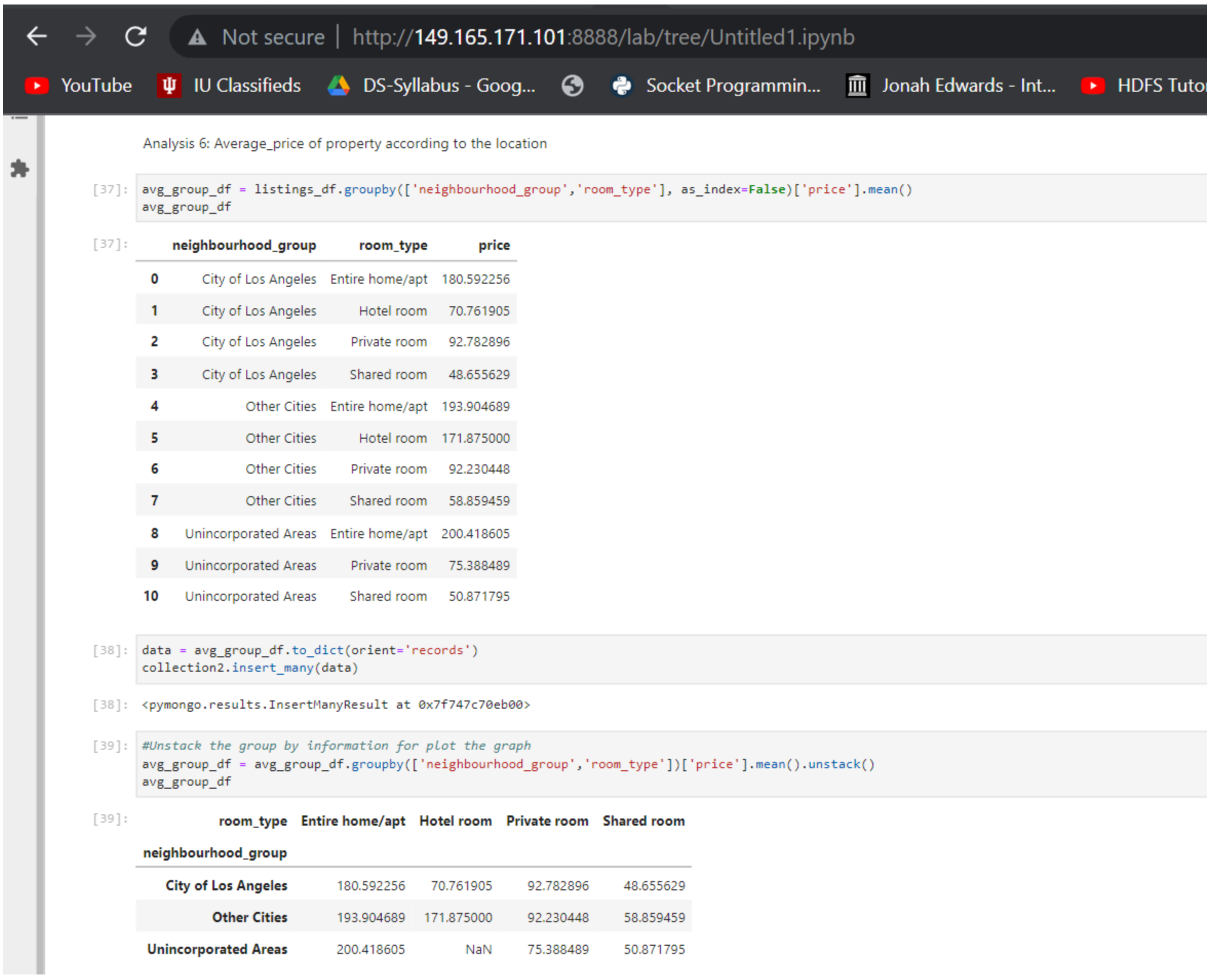


Figure 26: Box plot distribution for top-5 Neighbourhoods and availability_365

This is an interesting observation that can made, where there is no box present for Sherman Oaks, as its available on all of the days throughout a year.

Analysis 6: What is the average price of property according to the location and also room type.

In this, I have classified the average price listing for the neighbourhood groups and top 5 neighbourhood places with the room type as below. Unlike above results, I have used the Collections in MongoDB to store the output tables accordingly for performing the visualizations in MongoDB Atlas. These visualisations have been stored in the form of dashboards in MongoDB Atlas for future referencing purposes.



Analysis 6: Average_price of property according to the location

```
[37]: avg_group_df = listings_df.groupby(['neighbourhood_group','room_type'], as_index=False)['price'].mean()
avg_group_df
```

	neighbourhood_group	room_type	price
0	City of Los Angeles	Entire home/apt	180.592256
1	City of Los Angeles	Hotel room	70.761905
2	City of Los Angeles	Private room	92.782896
3	City of Los Angeles	Shared room	48.655629
4	Other Cities	Entire home/apt	193.904689
5	Other Cities	Hotel room	171.875000
6	Other Cities	Private room	92.230448
7	Other Cities	Shared room	58.859459
8	Unincorporated Areas	Entire home/apt	200.418605
9	Unincorporated Areas	Private room	75.388489
10	Unincorporated Areas	Shared room	50.871795

```
[38]: data = avg_group_df.to_dict(orient='records')
collection2.insert_many(data)

[38]: <pymongo.results.InsertManyResult at 0x7f747c70eb00>

[39]: #Unstack the group by information for plot the graph
avg_group_df = avg_group_df.groupby(['neighbourhood_group','room_type'])['price'].mean().unstack()
avg_group_df
```

room_type neighbourhood_group	Entire home/apt	Hotel room	Private room	Shared room
City of Los Angeles	180.592256	70.761905	92.782896	48.655629
Other Cities	193.904689	171.875000	92.230448	58.859459
Unincorporated Areas	200.418605	NaN	75.388489	50.871795

Figure 27: Python code for distribution of neighbourhood groups with the room types.

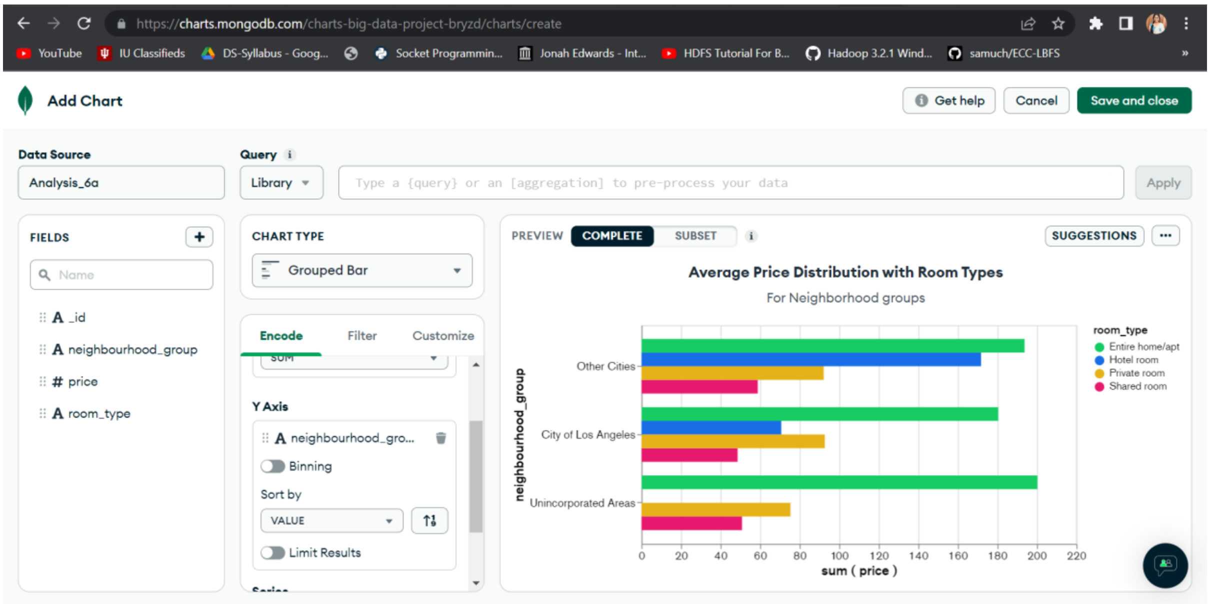


Figure 28: Visualization in MongoDB Atlas (Neighbourhood groups).

Similarly, for the top-5 neighbourhoods the grouping is done.

```
[40]: #Average_price of property according to the Location
avg_df = listings_top5.groupby(['neighbourhood','room_type'], as_index=False)['price'].mean()
avg_df
```

[40]:

	neighbourhood	room_type	price
0	Hollywood	Entire home/apt	159.937549
1	Hollywood	Hotel room	77.800000
2	Hollywood	Private room	89.982014
3	Hollywood	Shared room	62.500000
4	Long Beach	Entire home/apt	178.992819
5	Long Beach	Hotel room	198.818182
6	Long Beach	Private room	97.730897
7	Long Beach	Shared room	95.388889
8	Santa Monica	Entire home/apt	196.397616
9	Santa Monica	Hotel room	66.000000
10	Santa Monica	Private room	168.378261
11	Santa Monica	Shared room	48.666667
12	Sherman Oaks	Entire home/apt	200.924370
13	Sherman Oaks	Private room	104.212121
14	Sherman Oaks	Shared room	50.400000
15	Venice	Entire home/apt	220.936914
16	Venice	Hotel room	0.000000
17	Venice	Private room	121.253247
18	Venice	Shared room	95.666667

```
[41]: data = avg_df.to_dict(orient='records')
collection3.insert_many(data)
```

```
[41]: <pymongo.results.InsertManyResult at 0x7f747c70cd90>
```

```
[42]: #Unstack the group by information for plot the graph
avg_preffered_price_df = avg_df.groupby(['neighbourhood','room_type'])['price'].mean().unstack()
avg_preffered_price_df
```

[42]:

room_type	Entire home/apt	Hotel room	Private room	Shared room
neighbourhood				
Hollywood	159.937549	77.800000	89.982014	62.500000
Long Beach	178.992819	198.818182	97.730897	95.388889
Santa Monica	196.397616	66.000000	168.378261	48.666667

Figure 29: Python code for distribution of top-5 neighbourhoods with the room types.

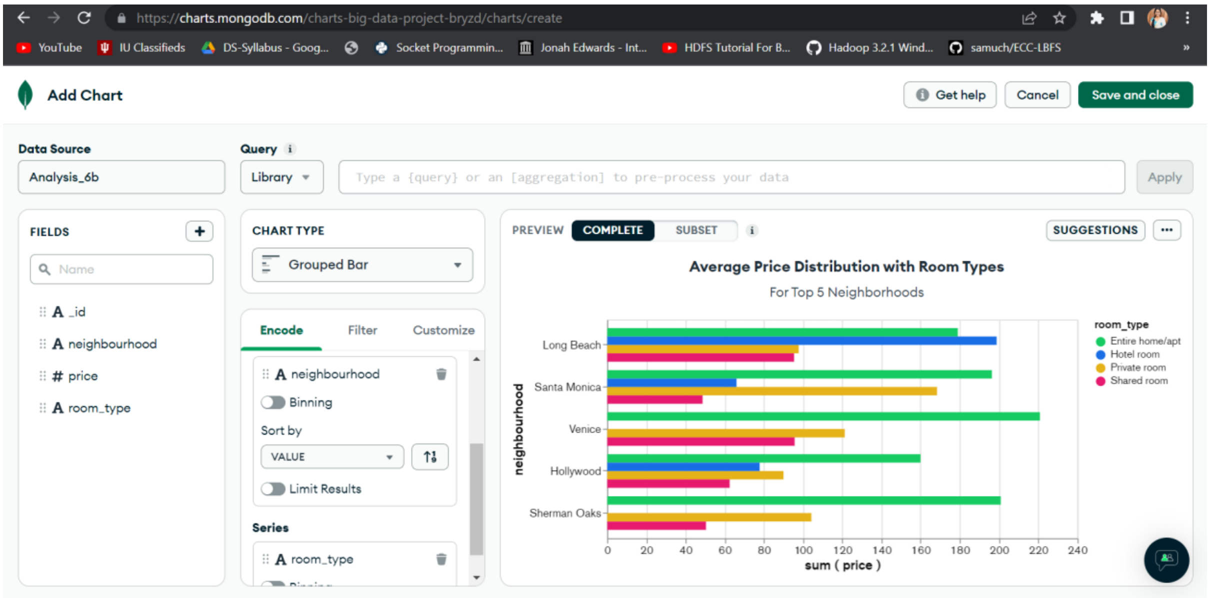


Figure 30: Visualization in MongoDB Atlas (Top-5 Neighbourhoods).

These visualizations are saved as dashboards and have been made public and anyone can view them, without having to run the code.

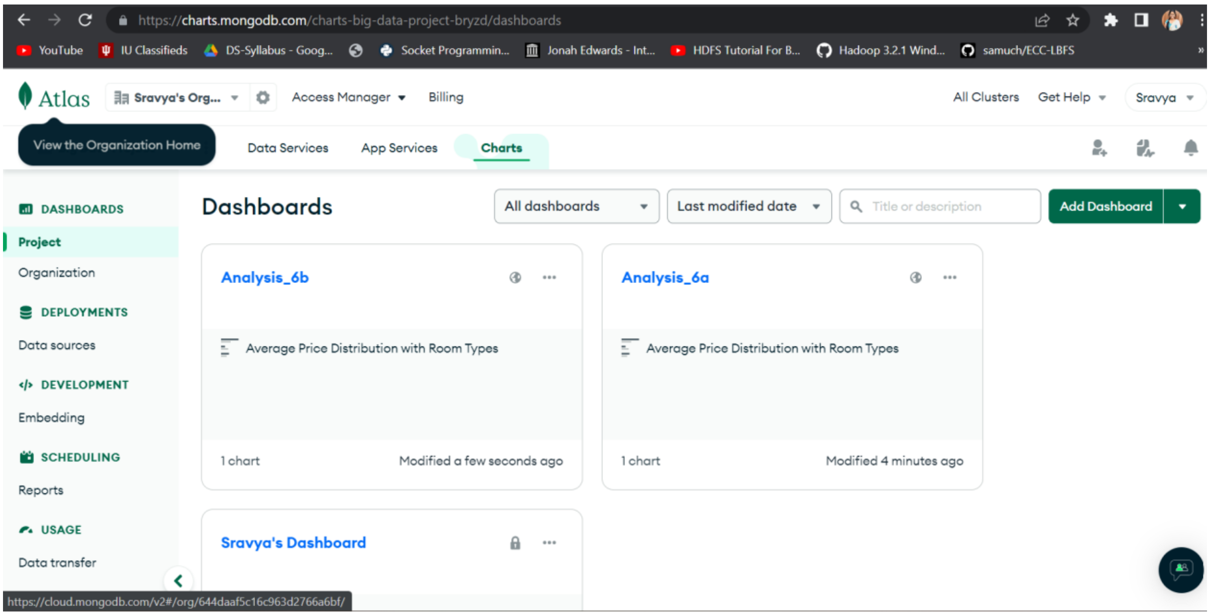


Figure 31: Published Dashboards in MongoDB Atlas.

Analysis 7: Most Frequent words used in the Airbnb Listings names.

In this, I analysed the most 100 frequent named listed in Airbnb listings to get to the top words. For this necessary action nltk python package is used for cleaning to remove stop words, short words (<= 3), non-alphanumeric characters.

Anaysis 7: Most frquently used words in Airbnb Listings

```
[43]: import nltk
import requests
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
stop = set(stopwords.words("english"))

[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
[44]: #Getting name strings from 'name' column and splitting them into separate words
listing_names = [word.lower() for name in listings_df.name for word in str(name).split()]

#Creating a dictionary to store word counts
counting_names = {}

#Counting the occurrence of each word and storing it in the dictionary
for word in listing_names:
    if word in counting_names:
        counting_names[word] += 1
    else:
        counting_names[word] = 1
```

Figure 32: Python code for Cleaning.

```
[48]: import contractions
import re

from collections import Counter

# Creating empty list to store the count of words
counting_names = []

# Getting name string to split it into separate words and append them to names_count list
for name in listings_df.name:
    for word in str(name).lower().split():
        counting_names.append(word)

# regular expression to match symbols and numbers and non-alphanumeric characters.
pattern = re.compile('[^a-zA-Z]|([^0-9A-Za-z!?])|')

# remove symbols and numbers from each string in the list using list comprehension
cleaned_list = [pattern.sub('', s) for s in counting_names]

cleaned_list = list(filter(lambda x: x != '', cleaned_list))

# remove contractions
expanded_words = [contractions.fix(word) for word in cleaned_list]

cleaned_list = [pattern.sub('', word) for word in expanded_words]

# remove short words (less than 3 characters)
cleaned_list = [word for word in cleaned_list if len(word) > 3]

# Counting top 100 most common words
top_count_words = Counter(cleaned_list).most_common(100)
print(top_count_words)

[('private', 5971), ('bedroom', 4723), ('room', 3836), ('home', 3812), ('with', 3727), ('beach', 3372), ('suite', 3091), ('studio', 3046), ('house', 3017), ('hollywood', 2899), ('cozy', 2890), ('near', 2289), ('modern', 2096), ('apartmen
t', 1925), ('mansion', 1802), ('pool', 1723), ('beautiful', 1667), ('bath', 1632), ('parking', 1615), ('spacious', 1611), ('luxury', 1432), ('hills', 1367), ('downtown', 1199), ('heart', 1146), ('guest', 1134), ('venice', 1095), ('locatio
n', 1081), ('from', 1053), ('view', 1040), ('dtla', 1025), ('condo', 1013), ('lovely', 1011), ('charming', 942), ('city', 875), ('views', 828), ('bungalow', 825), ('angeles', 823), ('unit', 817), ('bathroom', 810), ('west', 778), ('loft',
762), ('close', 754), ('quiet', 746), ('beverly', 713), ('santa', 712), ('walk', 701), ('bright', 696), ('monica', 671), ('large', 650), ('park', 630), ('oasis', 627), ('free', 615), ('designer', 612), ('guesthouse', 591), ('clean', 581),
('retreat', 551), ('ocean', 542), ('cottage', 536), ('queen', 522), ('amazing', 517), ('style', 511), ('king', 507), ('master', 494), ('great', 477), ('paradise', 474), ('patio', 473), ('garden', 467), ('entire', 451), ('prime', 435), ('lo
ng', 428), ('central', 425), ('stylish', 413), ('pasadena', 411), ('newly', 405), ('shared', 399), ('entrance', 396), ('stay', 392), ('furnished', 387), ('tropical', 380), ('remodeled', 373), ('sofi', 362), ('best', 362), ('sunny', 360),
('steps', 359), ('getaway', 353), ('renovated', 351), ('family', 348), ('chic', 347), ('bdrm', 341), ('place', 340), ('beds', 339), ('yard', 339), ('area', 336), ('presidential', 331), ('comfy', 330), ('townhouse', 327), ('gorgeous', 321),
('wifi', 317), ('balcony', 314), ('center', 311)]
```

```
[51]: stop_words = stopwords.words('english')

# Creating a list of the top 100 words, without stop_words
top_100 = [word for word, count in top_count_words if word.lower() not in stop_words][:100]
print(top_100)

['private', 'bedroom', 'room', 'home', 'beach', 'suite', 'studio', 'house', 'hollywood', 'cozy', 'near', 'modern', 'apartment', 'mansion', 'pool', 'beautiful', 'bath', 'parking', 'spacious', 'luxury', 'hills', 'downtown', 'heart', 'guest',
'venice', 'location', 'view', 'dtla', 'condo', 'lovely', 'charming', 'city', 'views', 'bungalow', 'angeles', 'unit', 'bathroom', 'west', 'loft', 'close', 'quiet', 'beverly', 'santa', 'walk', 'bright', 'monica', 'large', 'park', 'oasis', 'f
ree', 'designer', 'guesthouse', 'clean', 'retreat', 'ocean', 'cottage', 'queen', 'amazing', 'style', 'king', 'master', 'great', 'paradise', 'patio', 'garden', 'entire', 'prime', 'long', 'central', 'stylish', 'pasadena', 'newly', 'shared',
'entrance', 'stay', 'furnished', 'tropical', 'remodeled', 'sofi', 'best', 'sunny', 'steps', 'getaway', 'renovated', 'family', 'chic', 'bdrm', 'place', 'beds', 'yard', 'area', 'presidential', 'comfy', 'townhouse', 'gorgeous', 'wifi', 'balco
ny', 'center']
```

Figure 33: Python code for fetching top 100 words.

Now the visualization of the word cloud is performed using the word cloud python package.

```
[52]: # WordCloud
from wordcloud import WordCloud

wordcloud = WordCloud(width=800, height=500, random_state=52, background_color='black',  max_font_size=110).generate(str(top_100))

plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

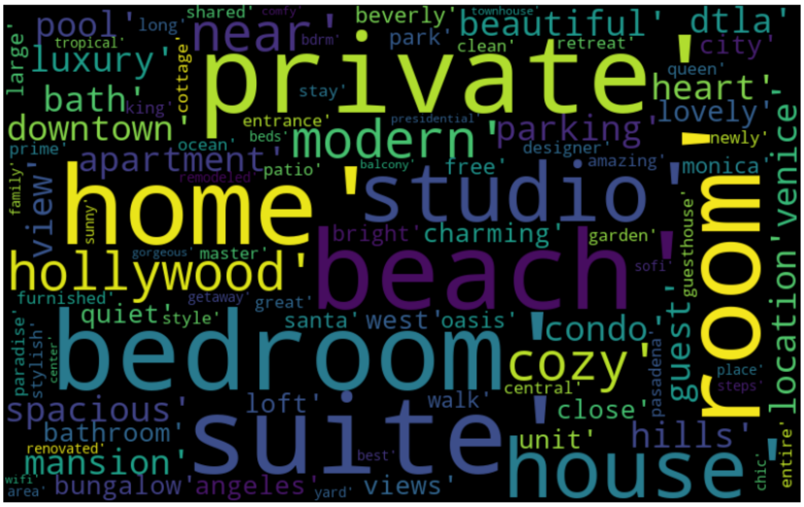


Figure 34: Word Cloud Visualization

Section 5: Discussion

I designed a pipeline architecture to implement this project and followed the steps as mentioned in Figure 1. I fetched the data from the MongoDB Atlas into the pandas data frame, as explained in detail above. There were approximately 42.5K entries at first, but after performing data cleaning and pre-processing steps, the entries were reduced to 38K. During this stage, I removed the unwanted columns from the original dataset, such as host_name, last_review, reviews_per_month, and license. By grouping with the price column, I removed the outliers in the data. The analysis and results have been completed as a result of this. Each of the outcomes would provide the necessary information about the Airbnb Listings. The following is a summary of its observations.

1. There are three distinct neighbourhood groups in Los Angeles: City of Los Angeles, Other Cities, and Unincorporated Areas, with the top five neighbourhoods in L.A. being Sherman Oaks, Hollywood, Long Beach, Venice, and Santa Monica. According to the listing distribution, the City of Los Angeles group has the most listings, and the Sherman Oaks neighbourhood has the most listings.
2. The average price distribution among the neighbourhood groups is higher for Other Cities followed by Unincorporated Areas and City of Los Angeles. Whereas the average price distribution among the top-5 neighbourhoods is higher for Venice, followed by Santa Monica, Long Beach, Hollywood and Sherman Oaks. It is clear from (1) that if there are more listings, the average price is lower, owing to the conclusion that, number of listings is inversely proportional to the average price.

3. There are more listings available for the room types home/apt, followed by private room, shared room, which has 25643 listings, and hotel room, which has 77 listings. When these are paired with neighbourhood groups, it is discovered that home/apt is the most available in all groups, followed by private rooms. Unincorporated Areas has no shared type rooms, which is an interesting result. In addition, no hotel rooms are available in any of the neighbourhood groups. Among the top-5 neighbourhood places Sherman Oaks has the most Private rooms compared to the other types. However, the regular Entire home/apt is the top of the other top-5 places.
4. Hosts with the most listings have been highlighted. For privacy and ethical reasons, the host names column was removed and replaced with host ids to identify hosts. The most extensive listing has over 1001 entries.
5. 5. When compared to the other groups, the City of Los Angeles has more availability across all 365 days out of the three neighbourhood groups available. For top-5 neighbourhood places as there is no box present for Sherman Oaks, which is available on all days of the year.
6. All the 3 neighbourhood groups have the highest average price for Entire home/apt room types. For the top-5 neighbourhood places, Hollywood, Santa Monica, Sherman Oaks, Venice has the highest average price for Entire home/apt room types costing around 159.9$, 196.39$, 200.92$ and 220.93$ respectively. While Long Beach has the highest average price for hotel rooms which is 198.89$. These results were also published in the MongoDB Atlas Dashboards.
7. After data cleaning and stop word removals, the most frequent words available in name of the listings are 'private', 'bedroom', 'room', 'home', 'beach'.

Challenges Faced:

As a person who is using the MongoDB Atlas for the first time, I found this as an opportunity to learn a new skill. At first while ingesting the csv data into the Atlas using compass, I encountered difficulties in connecting to the Cluster0. I had to refer video tutorials for resolving the problems. Later on, accessing this using the VM's instance was again a troublesome. Proper research on the network accessing helped to get through this.

Also, I was planning to do entire analysis in the PySpark environment, but due the dependencies on jar files for the pyspark-mongodb connector, I could not able to use it. It needed additional jar files to execute using PySpark. I tired figuring out adding them for 2 days, unfortunately due to version mis match errors, I had to drop the idea of going ahead with it.

Section 6: Conclusion

This project, Airbnb analysis in L.S, US hence gives us the conclusions regarding the Airbnb Listings in L.A region. L.A being the second most populous city in the United States and one of the world's most visited cities, planning a holiday or vacation to this place includes most of the expenses. Now with the detailed analysis from the project, we now know the neighbourhood groups, top-5 neighbourhoods around L.A, the average price distribution accordingly to the neighbourhood groups and top-5 neighbourhoods, the room types distribution and average pricing for the neighbourhood groups and top-5 neighbourhoods. We also know the top 10 hosts as per the listings. Based on this information, one could actually plan for the vacation by managing the expenses and also availabilities of the Listings accordingly.

Additional Work: I would like to implement the same using the PySpark environment once the error related to the jar files is resolved. As PySpark is a distributed environment, when the data is huge, this would increase the processing of the analysis much faster. Also, I would like to publish all the results into the MongoDB. Due to the time constraint, I could not able to publish the results into the dashboards for all them.

Links in support of the work:

1. https://github.iu.edu/sarutla/Big_Data_Course_Project

Section 7: Reference

1. http://insideairbnb.com/get-the-data
2. https://www.mongodb.com/docs/
3. https://practicaldatascience.co.uk/data-science/how-to-visualise-data-using-boxplots-in-seaborn
4. https://www.datacamp.com/tutorial/wordcloud-python
5. https://pymongo.readthedocs.io/en/stable/
6. https://jakevdp.github.io/PythonDataScienceHandbook/04.14-visualization-with-seaborn.html
7. https://www.mongodb.com/languages/python
8. https://www.analyticsvidhya.com/blog/2021/10/end-to-end-predictive-analysis-on-airbnb-listings-data/