DSA5104
Principles of Data Management and Retrieval

National University of Singapore
Data Science and Machine Learning Department
March 12, 2023

Airbnb Dataset Report

Wang Jiangyi, A0236307J
Yang Sizhe, A0236299N
Song Mingyue, A0251125U
Lou Xinyun, A0251245M

1 Framework of Report

According to the tasks in the project page, we can divide the whole report into seven parts, each of which corresponds to one task:

1. **Exploration Data Analysis**. Here, we aim to **dive into the details** of dataset so that we can deeply understand the resources we have. Beyond that, we also **dig some interesting patterns** behind data itself. This can gives us some insights and discovers problems from data.
2. **Data Cleaning**. Based on the EDA we have done, we try to **fix issues within data**, including the null values and the data types issue. We will also take multi-language issue into consideration.
3. **Data Generation**. We want to **enrich our data resources** from this part. We will generate transaction records from review records.
4. **Requirement Analysis**. Before designing the database, we do the Requirement Analysis in a **comprehensive manner**. In our report, we will conduct the analysis from, **naive user level** and **host level**. Based on this, we can further write queries to fulfill these requirements.
5. **Relational Database Storage**. According to the pipeline of designing relational database (ER diagram - schema diagram - schema refinement), we will transform **cleaning data and generating data** into ***Mysql DBMS***.
6. **Non-relational Database Storage**. To take full advantages of non-relational database, we **conduct an ad-hoc schema design for it** to store the data in ***MongoDB DBMS***.
7. **Query Answer**. In this part, for 2 different database systems, we give the queries based on our requirement anlalysis **respectively**. Besides, we will **create index in some attributes** to accelarate the data retrieving process and report the response time and **comparison will be conducted**.

2 Exploration Data Analysis

Before showing the data exploration result, we briefly discuss the **benefits of EDA**.

Firstly, in order to transform all the data into database, it is necessary for us to dive into some details of data itself. This can give us some better sense of the dataset, **which will be beneficial to the following database design process**. Moreover, we can also **gain insights** from this process, which can **help business side in application**.

Secondly, when going through the dataset in details, you can easily find some issues within the data. For example, when checking the boxplot of some attributes, then those outliers can be easily detected if existed. **This will give us some help with respect to data cleaning process**.

Lastly, from the real-life application perspective, **doing EDA carefully will throw some problems** to us. That is, we may discover some abnormal patterns from the data. After observing that, we can **pose those problems to the technical side and business side** and require their help to solve the problems, which can bring many benefits to the company.

In our report, we focus on **the insights** and **the problems** (**1** and **3**) during data exploration process. The exploration details for each dataset can be shown as follows:

2.1 calendar

In the beginning, we **make some description** to the 'calendar' dataset.

1. The 'calendar' dataset is created around July, 2022 (**same as other 4 datasets**).
2. This dataset is about, **for each listing rental**, its price, availability and order requirement information **within the following 1 year time window**.

Then, we start to show the discovery result and some intuitive comments:

2.1.1 Average price across months

Here, we visualize the ***average price*** **for all rental in different months**. This can give us some sense about the price changing trend corresponding to different period of year. The visualization result can be shown in Figure 1:

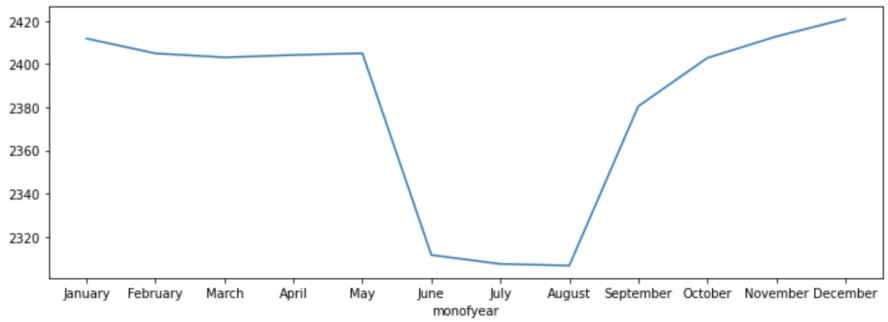


Figure 1: Average Price in Different Months

There is a sharp slope in June, July and August, which is an abnormal observation.

After searching for some information, **one reasonable reason** is: on June, July and August, students are on their holiday. Therefore, when they plan a trip to some other countries and use Airbnb app to find rentals, they are likely to prefer long-term rental. In this case, the host can give some discounts to them compared with short-term rental.

2.1.2 Average price across weekdays

Here, we visualize the ***average price*** **in different weekdays**, i.e., from Monday to Sunday. The visualization result can be shown in Figure 2:

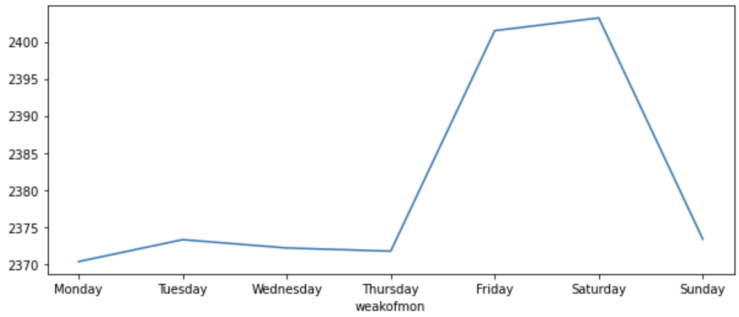


Figure 2: Average Price in Different Weekdays

The pattern here is close to our intuition: People are busy from Monday to Thursday, and when they want to have a short-term rest, they prefer to choose Friday or Saturday.

2.1.3 Busy rate in one year

We calculate the ***busy rate*** **for all rentals** and do the visualization. The result is shown in Figure 3:

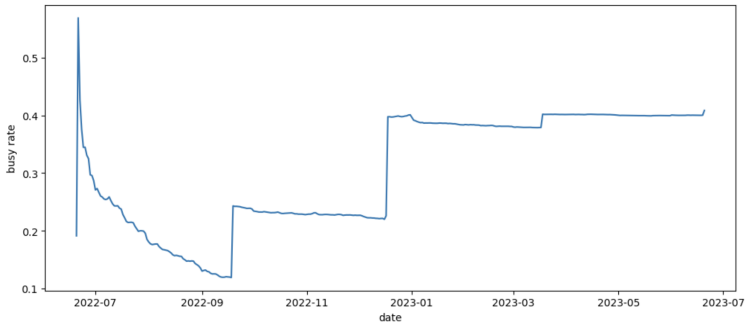


Figure 3: Busy Rate within Year, $busy\ rate := \frac{\#(non-available\ rentals)}{\#(all\ rentals)}$

To illustrate this, there are **2 main things** in Figure 3:

1. **Focus on the curve from '2022-07' to '2022-09'**, we will achieve some guideline for our customers (app users). That is, there are many rentals which are available if you reserve about one or two months before.
2. It can be observed that, there are **abnormal step-shape curve** after '2022-09'. This is mainly because, the 'availabilty' is not only because some customers book this rental on that day, but also the hosts set the rental as 'available'. If the day is far away from today, hosts are more likely to set the rental as 'non-available'.

2.2 listings

Also, we firstly make some description to the 'listings' dataset.

1. There are 2 different datasets with respect to listing records, 'listings' and 'listings_summary'. The difference is, 'listings_summary' contains important parts of attributes. Therefore, we **use 'listing_summary' for EDA** to achieve better sense of data and **use 'listings' to do data cleaning** and transform them into database.
2. The 'listings' dataset is mainly about, the listing rental and its **all types of information (we will discuss in details below).** We can see all these information when we click one specific listing rental.

Then, we start to show the discovery results and some interesting observation:

2.2.1 Host-side information exploration

Here, we mainly focus on **the distribution of *counting number* of rentals in Airbnb that one host has**. The histogram diagram can be shown in Figure 4:

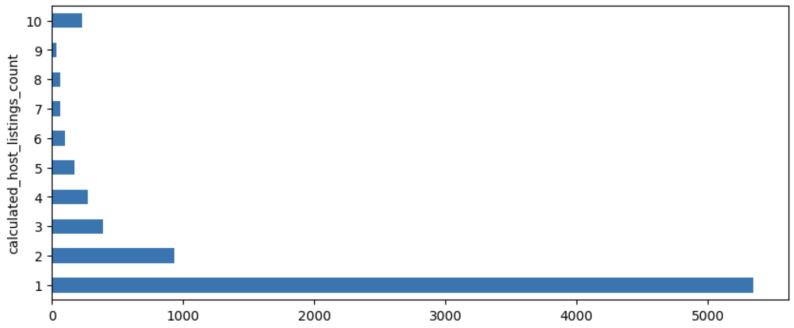
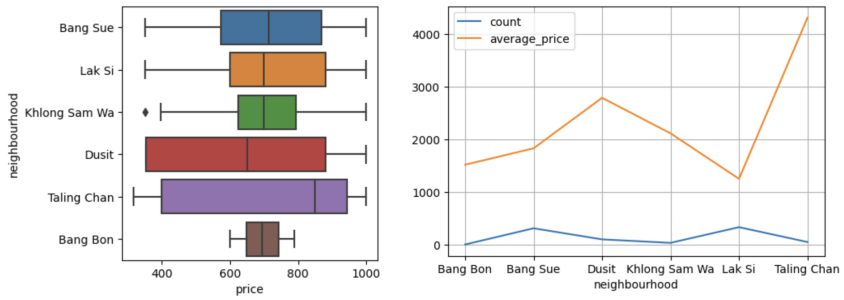


Figure 4: Distribution of Rental Number One Host Has

It can be easily seen from Figure 4 that, the majority of hosts only have 1 rental in Airbnb. This piece of information may help **business side better understands those hosts and locate anomalies**.

2.2.2 Neighbourhood-side information exploration

Here, we focus on the **distribution of *couting number* and *price* for rentals in different neighbourhoods**. The results can be shown as follows:



(a) Boxplot of Price v.s. Neighbourhood (b) Line Chart of Price v.s. Neighbourhood

Figure 5: Visualization for Neighbourhood-side Information

1. In Figure 5(a), it can be observed that, in different neighbourhoods, the variance of price is quite different. Take **'Tailing Chan'** as one example, the standard deviation of the price is quite big. This means, if you want to find one rental in that region, then **you should spend more time in comparing** because the variance of price is very big. On the other side, if you want to find rentals in **'Bang Bon'**, then you can spend less time since the price behaves quite similar. **As for the company**, these kinds of suggestions can be given to the app users to improve their app experience.
2. In Figure 5(b), we mainly focus on **different price behaviours across regions**. You can see that, **the number of rentals are quite imbalance across regions**. Buisness-side might need this information to optimize user experience.

2.2.3 Basic information exploration

Here, our interest is, the **relationship between *number of bedrooms* and *price***. Something interesting has been discovered during the analysis. The whole exploration process will be included. Firstly, we visualize the relationship through boxplot in Figure 6:

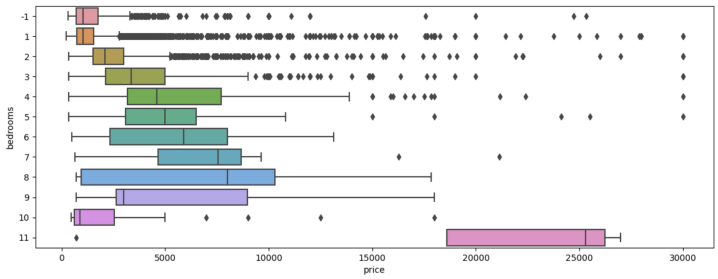
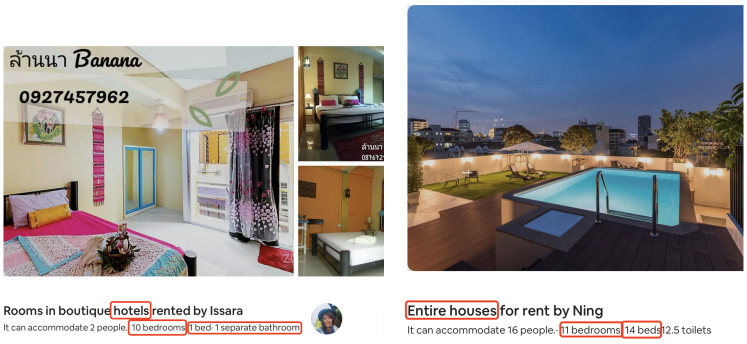


Figure 6: Relationship between Price and Bedroom Number

To illustrate this, you can see that, except for **'bedroom = 9, 10'**, there **exists positive relationship** between **price** and **number of bedrooms**. However, a natural question is: **What happened for those rentals which have 9 or 10 bedrooms?** We **dive into the data to answer this**. In order to find an answer, we check those listings which have 10 bedrooms in details. The result is shown as follows:



(a) 10-Bedrooms Example (b) 11-Bedrooms Example

Figure 7: Exploration Result for 10-Bedrooms Case

In Figure 7(a), you can see that, the type of this rental is hotel, and in the whole hotel, there are totally 10 rooms. **This is not what we want** with respect to the definiton of number of bedrooms. To compare, in Figure 7(b), this rental has totally 11 bedrooms, which is a villa. This number is indeed what we want to tell customers.

Besides, after further checking, almost all the '10-Bedrooms' cases belong to this senario. Therefore, we can make conclusion that, most of the **'10-Bedrooms'** cases are due to the **data issues**. That is why their price is much lower than expected. After observing this, there are **2 insights from the exploration process**:

1. Those hosts should change 9 or 10 bedrooms to 0 bedroom if it is just a hotel;
2. If we want to **make prediction** for price, then it is **better to use 'accomodates' attribute** in dataset.

2.3 reviews

Similarly, we firstly make some description with respect to the 'reviews' dataset.

1. There are 2 different datasets with respect to reivews records, 'reviews' and 'reviews_summary'. **The difference and our solution have been described in 'listings' case and we omit here**.
2. The dataset gives us information about, **all review records corresponding to each rental in given day**. This pieces of information will be displayed when we realize the 'review' function in the details of each rental.
3. The reviews information is recorded in multiple kinds of languages.

The detailed discovery results and some intuitive comments can be shown as follows:

2.3.1 Comment languages exploration

Here, we focus on **the languages which are used in *comments***. This will help **business side** better understand our customers. The histogram diagram can be shown in Figure 8:

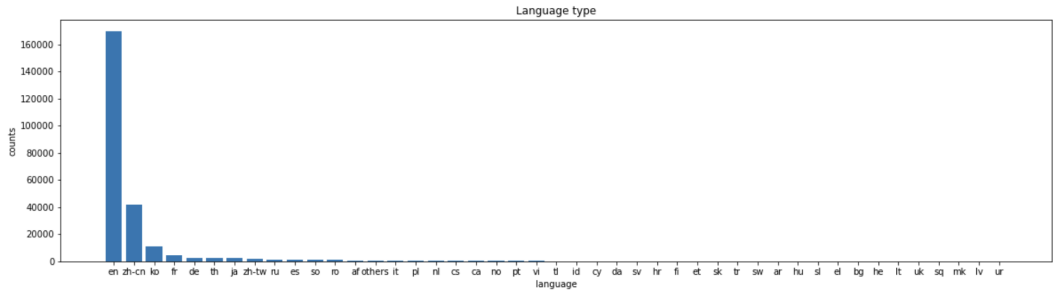


Figure 8: Different Languages Used in Comment

It can be seen from Figure 8 that, majority of our customers use English and Chinese in their daily life. This can give us **some insights about the distribution of our customers**.

Moreover, this also tells us that, **'comment'** attribute is not so clean. We may further ask tech-team to convert all languages into English and save them in database. This will bring great convenience for further NLP tasks if possible.

2.3.2 Brushing fraud exploration

Here,our interest is, **the fraud pattern hidden from *number of review records* for rental and reviewer**. We will go through the whole process of how we find this.

Firstly, we try to check the **top 20 customers with most comments**. The visualization result is shown as follows:

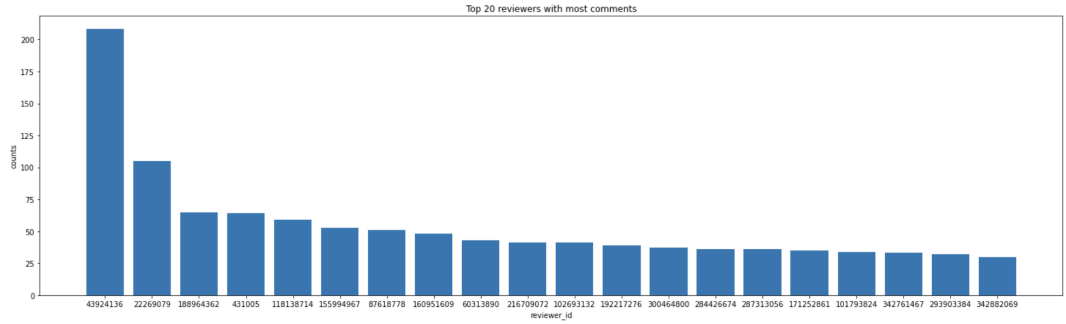


Figure 9: Top 20 Reviewers

In Figure 9, the **number of comments** from **top 1 reviewer** with most comment is extremely larger that from other reviewers. This is an abnormal observation since it is almost **impossible** for one normal customer

to **book rentals so many times**.

To make it clear, we dive into more details about this 'abnormal' customer, saying that those review records from **'43924136 customer'**. The following Figure 10 tries to **compare** his review records with another customer who also has many review records (**'431005 customer'**):

	Unnamed: 0	listing_id	id	date	reviewer_id	reviewer_name	comments	l_type
226564	226564	39539462	556994726	2019-10-31	43924136	Bo-Ther & Paa Maa	This is a great place if you are traveling wit...	en
226565	226565	39539462	559903199	2019-11-05	43924136	Bo-Ther & Paa Maa	This is a great discovery - in the heart of Sa...	en
226566	226566	39539462	560507630	2019-11-07	43924136	Bo-Ther & Paa Maa	i love staying here! and with the extension of...	en
226567	226567	39539462	560839870	2019-11-08	43924136	Bo-Ther & Paa Maa	I have stayed in a lot of places in Bangkok an...	en
226568	226568	39539462	561785155	2019-11-10	43924136	Bo-Ther & Paa Maa	Perfection - newly and smartly remodeled and ...	en
226569	226569	39539462	563506709	2019-11-13	43924136	Bo-Ther & Paa Maa	very large - big room and large bathroom (with...	en
226571	226571	39539462	564508982	2019-11-16	43924136	Bo-Ther & Paa Maa	I have stayed here many times, in this room, a...	en
226572	226572	39539462	566322003	2019-11-19	43924136	Bo-Ther & Paa Maa	another great stay with Evan and his many opti...	en
226573	226573	39539462	567253333	2019-11-22	43924136	Bo-Ther & Paa Maa	I enjoyed this place a lot. Fast internet and ...	en
226574	226574	39539462	569284659	2019-11-26	43924136	Bo-Ther & Paa Maa	I enjoyed this place a lot. Fast internet and ...	en
226575	226575	39539462	569597422	2019-11-27	43924136	Bo-Ther & Paa Maa	I enjoyed this place a lot. Fast internet, sma...	en
226576	226576	39539462	569901356	2019-11-28	43924136	Bo-Ther & Paa Maa	I enjoyed this place. the internet was fast an...	en

(a) Customer '43924136'

	Unnamed: 0	listing_id	id	date	reviewer_id	reviewer_name	comments	l_type
3826	3826	1887544	9121130	2013-12-08	431005	W	Perfect place to stay in Bangkok. Salvatore's ...	en
3827	3827	1887544	9206725	2013-12-13	431005	W	Superb place to stay. The host Salvatore and h...	en
14021	14021	858311	9990992	2014-01-23	431005	W	I spent almost a week at Don's place and i lov...	en
9184	9184	2166902	10703874	2014-03-03	431005	W	I am glad that Jay is in Airbnb host community...	en
1718	1718	1041976	11475410	2014-04-04	431005	W	The host is so wonderful. She picked me up fro...	en
1720	1720	1041976	15114387	2014-07-02	431005	W	Lovely host and the space is very pretty. Host...	en
3896	3896	1887544	16754353	2014-08-01	431005	W	My second time staying in this place and i lov...	en
408	408	993085	21261518	2014-10-13	431005	W	The place is in good location, very quiet, coz...	en
1006	1006	1744248	22928695	2014-11-17	431005	W	No! is a great host! No! is very polite, hospi...	en
30506	30506	6164354	33558168	2015-05-29	431005	W	The unit was very nice and clean with a pretty...	en
38262	38262	5969642	33878924	2015-06-01	431005	W	This place is newly build and furnished with m...	en
38263	38263	5969642	33973138	2015-06-02	431005	W	I extent my stay here for another night. The p...	en

(b) Customer '431005'

Figure 10: Reviews Comparison between 2 customers

To illustrate, in Figure 10(a), it can be apparently observed that, the **frequency of comments is much higher**, almost 2 or 3 times per week. Moreover, all the comments **contribute to the same rental**, which is very abnormal. In Figure 10(b), the comment behavior is much more normal. That is, the **frequency of comments is much slower**, about six times per year. Besides, all the comments **contribute to different rentals**.

Therefore, we can conclude that, there may exist **Brushing Fraud** within review comments, which is similar to what **'43924136 customer'** does in the previous discussion. This piece of information can be sent to business side to consider further steps, including setting rules or building models to do detection.

2.4 relationship across 3 datasets

Lastly, we will briefly discuss the relationship among 3 datasets, especially the referential constraints.

1. In 'calendar' dataset, the 'listings_id' attribute references to the 'id' attribute in 'listings';
2. In 'reviews' dataset, the 'listings_id' attribute references to the 'id' attribute in 'listings'.

Remark: Besides above 4 parts, we also **do other exploration** with respect to the 3 datasets. Moreover, **analysis on the data types** have also been done during the exploration process. For simplicity, the details of these results will be attached in the **Appendix**. Here, we only display those meaningful exploration.

3 Data Cleaning

In this section, we will clean three datasets, i.e., 'calendar', 'listings', and 'reviews' step by step. We will mainly focus on dataset 'listings' since it is most sophisticated and troublesome.

3.1 Null Value

First we count the null values grouped by columns and show the results as follows.

Column Name	Null Value Counts
bathrooms	17074
license	17074
calendar_updated	17074
neighbourhood_group_cleansed	17074

Table 1: Null value counts in dataset listings where all values are missing.

We find that all data in column 'bathrooms', 'license', 'calendar_updated' and 'neighbourhood_group_cleansed' are missing. Therefore, we drop all these four columns out of the database.

While for the result columns, only partial data is missing. We keep all these data for sake of data integrity.

Column Name	Null Value Counts
neighborhood, neighborhood_overview	7615
review_scores_value	7296
review_scores_cleanliness, review_scores_accuracy	7288
host_neighbourhood	7262
reviews_per_month, first_review	7062
review_scores_rating, last_review	7062
host_acceptance_rate	6779
host_response_time, host_response_rate	5629
bedrooms	1703
description	905
host_location	44
name	9
host_listings_count, host_picture_url	2
host_thumbnail_url, host_is_superhost	2
host_total_listings_count, host_has_profile_pic	2
host_identity_verified, host_since, host_name	2
maximum_minimum_nights, maximum_nights_avg_ntm	1

Table 2: Null value counts in dataset listings where partial data is missing (part).

3.2 Find/delete special characters

Since the data is crawled down from the website, it inevitably contains some dirty information. We detailed check every column, and conduct series of operations on the corresponding column:

1. 'host_verifications'. We drop quotation marks and brackets and only keep the items like 'email', 'phone' and 'work_email'.
2. 'amenities'. We decode every string with 'utf-8' and find something error caused by escape sequence like \n and \t. Besides, we also find some noise information like xx\\, where xx are two consecutive numbers. We drop these characters by regular expression.
3. 'description', 'neighborhood_overview', 'neighbourhood', and 'host_about'. For these four columns, we detect blank unicode like \xa0 and \u3000 and some tags like <\br >or similar formats from HTML files. We also drop all these characters by regular expression.

3.3 Change format from string to boolean, float, and int

We convert the following columns to corresponding data types. We show the results in table 3.

Column	Data type
host_is_superhost, host_has_profile_pic	Boolean
host_identity_verified, has_availability, instant_bookable	Boolean
price, host_response_rate, host_acceptance_rate	Float
latitude, longitude,review_scores_rating	Float
review_scores_cleanliness, review_scores_checkin	Float
review_scores_communication, review_scores_location	Float
review_scores_value, reviews_per_month	Float
minimum_nights_avg_ntm, maximum_nights_avg_ntm	Float
the rest	Int

Table 3: Columns and corresponding data types.

3.4 Set ';' as separator

Usually, people set comma as separator in csv file. However, in our project we have many descriptive columns containing text information and comma. Therefore, when loading the data to database, the system will mistake the comma inside the text for the separator. In our project, we solve this problem by using ';' as the only separator. For the same reason, an very important step is to convert other ';' in text to ',' such that ';' in text will not influence data loading process.

3.5 Summary for data cleaning

We also do the similar operations for the other two datasets. Such process is much simpler comparing to dealing with dataset 'listing'. Despite the loss and change of some characters and punctuation, we try our best to uniform the format and keep the original data including, but not limited to, more than 40 languages, and emoji.

4 Data Generation

In this section, we will generate some artificial transaction data based on dataset reviewer. We do this in the following way.

1. First, We tend to generate 20% of the total comments as source of transaction data. Each comment is associated with a host, a listing, and time stamp. We fix the random state and sample from the original data.
2. Second, we generate end date for each transaction. For simplicity, we set the reviewing date as the start date for each transaction. The end date is determined by the stay time that each client spent. Based on actual needs, we generate the stay time using a specified distribution in which the probability for one-day stay is 30% and two-week stay is 1%.
3. Last, we generate booking price for each transaction.That is, we check the corresponding average booking price in the following year and set it as the cost per day index and thus generate the total cost.

5 Requirement Analysis

As we mentioned, we will conduct the analysis from 2 different scenarios, **naive user level** and **host level**.

5.1 Naive user level

From the real-life perspective, naive users, i.e., those Airbnb app users, they will focus on things like, searching for the rental that satisfies some condition and viewing the information about one speicific rental etc..

To conclude, there are **4 main requirements** for naive users:

1. **Searching function**.
That is, given some condition like price and room type, **return those rental satisfying these conditions**. Moreover, the return can be sorted by some attributes of the rental like review scores.
Moreover, someone might be interested in some super-hosts for their aesthetic. Then, it is necessary for us to **return the list of most popular** super-hosts if users want.

2. **Viewing function for rental**.
In this part, users want to see **all kinds of information about one specific rental**, including its neighbourhood, price, amenities and so on.
For example, one may want to **check whether one rental is available** during some time period in order to make further decision about their trip. Also, to compare between 2 rentals, users can view the review records to make their decision. We think the latter situation is very important in every user's experience life.

3. **Viewing function for host**.
Here, users can **check all the details of one specific host**, including his/her basic information, all review records and rental information to make better choices. This will give users a easier way to find more rentals.

4. **Viewing function for transaction.**
Customers may want to check their **previous transaction records**, which should be one basic function for users' experience.

5.2 Host level

As for hosts, for each of their rental, they need to assign one price. Thus, they may want some guideline from our side. Also, they might be interested in the function which can clearly display the orders of their rentals. We can show them the details of these information to support them.

To conclude, the **2 main requirements** for hosts are:

1. **Pricing-suggest function**.
In this part, when hosts fill in their basic information about rental, they want some suggestions from us about the pricing. We can return prices from other hosts who have rental with similar basic information. Moreover, we can further **build Machine Learning model about price-prediction** to give better suggestions for their pricing decision.

2. **Dashboard function**.
Here, hosts might be interested in, **a dashboard about his privous/future rental transaction records**. Take one simple example, many hosts will want to know that, in the previous 3 months, all the **reservation results** for each of his rental. By achieving this, it can be of some help for hosts to arrange their time.

6 Relational Database Storage

After finishing the requirement analysis, we start to build up our relational database model. Here, **we use *Mysql DBMS* to store our data**. The whole prcoess can be decomposed into, ER Diagram Deisgn, Schema Diagram Translation and Schema Refinement.

6.1 ER Diagram Deisgn

Before giving the ER Diagram, we first conduct analysis about Entity and Relationship for these 3 datasets:

Entity:
hosts, reviewers, calender slots, listing rooms, transaction, comment;

Relationship:
1. Binary: (host) holds (listing room), (listing room) has (calendar slot);
2. 3-ary: CRL(comment, reviewer, listing room);
3. 4-ary: TRHL(trasaction, reviewer, host, listing room);

Based on the above analysis, the ER Diagram can be shown in Figure 11:

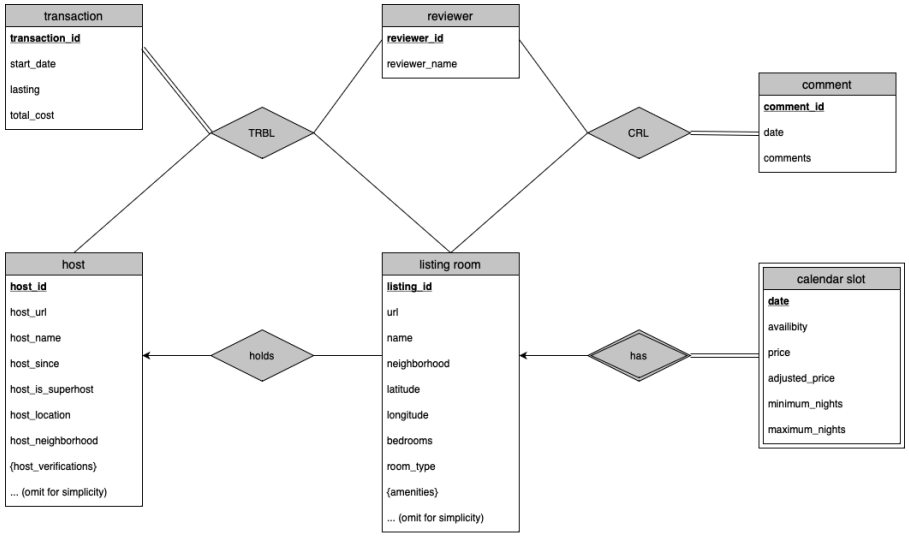


Figure 11: Entity-Relationship Diagram

There are **several explanations** with respect to Figure 11, which can be listed as follows:

1. In **'listing room' entity set**, the **'amenities'** attribute is **multi-valued type**. Therefore, to convert

this to schema diagram, we should create a new relation (table) to store (listing_id, amenity) pairs (same as **'host_verfications' in 'host' entity set**);

2. In 4-ary 'TRHL' relationship, we can view this as one **many-to-one relationship** and **'transaction' entity set is in the many side**. That is to say, **each transaction entity corresponds to exactly one (reviewer_id, host_id, listing_id) pair** (coming from the fact that 'transaction' entity set is totally participated).
Therefore, when we convert this into schema diagram, we just need to **store the primary keys of 'reviewer', 'host' and 'listing room' in the 'transaction' side** instead of creating new tables.

3. Similarly, in 3-ary 'CRL' relationship, we can also view it as **many-to-one relationship** and **'comment' entity set is in the many side**. **Each 'comment' entity corresponds to exactly one (review_id, listing_id) pair**.
Therefore, when converting to schema diagram, we only need to **store the primary keys of 'reviewer' and 'listing room' in the 'comment' side**

6.2 Schema Diagram Translation

Then, follow the above explanation, we can start to convert the ER Diagram into Schema Diagram. The converting result is shown in Figure 12:

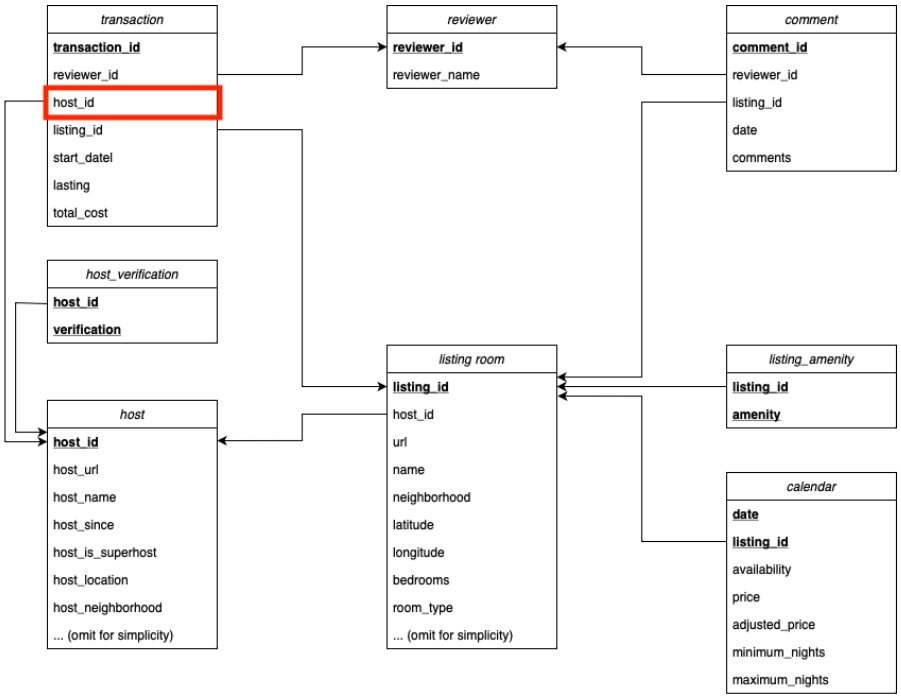


Figure 12: Schema Diagram

There are some illustration with respect to the red frame in Figure 12:

1. There are **some non-trivial Functional Dependencies** within **'transaction'** relation. That is, **'listing_id' will determine 'host_id'**, i.e.,

$$'listing\_id' \implies 'host\_id'$$

This is what we should consider when we make Schema Refinement.

6.3 Schema Refinement

In Figure 12, we can be sure that, in 'host', 'reviewer', 'listing room', 'comment', 'listing_amenity' and 'calendar' relation, there are no non-trivial FDs. The **only non-trivial FD for the whole database** is in **'transaction'**

relation, which is:

$$'listing\_id' \implies 'host\_id'$$

In order to achieve BCNF, we should decompose 'transaction' relation based on this FD. Moreover, observe that, in **'listing room'** relation, **this FD has already been included**. Therefore, we can **simply drop the 'host_id' attribute** in 'transaction' relation to achieve BCNF.

The **Refined Schema Diagram** can be shown in Figure 13:

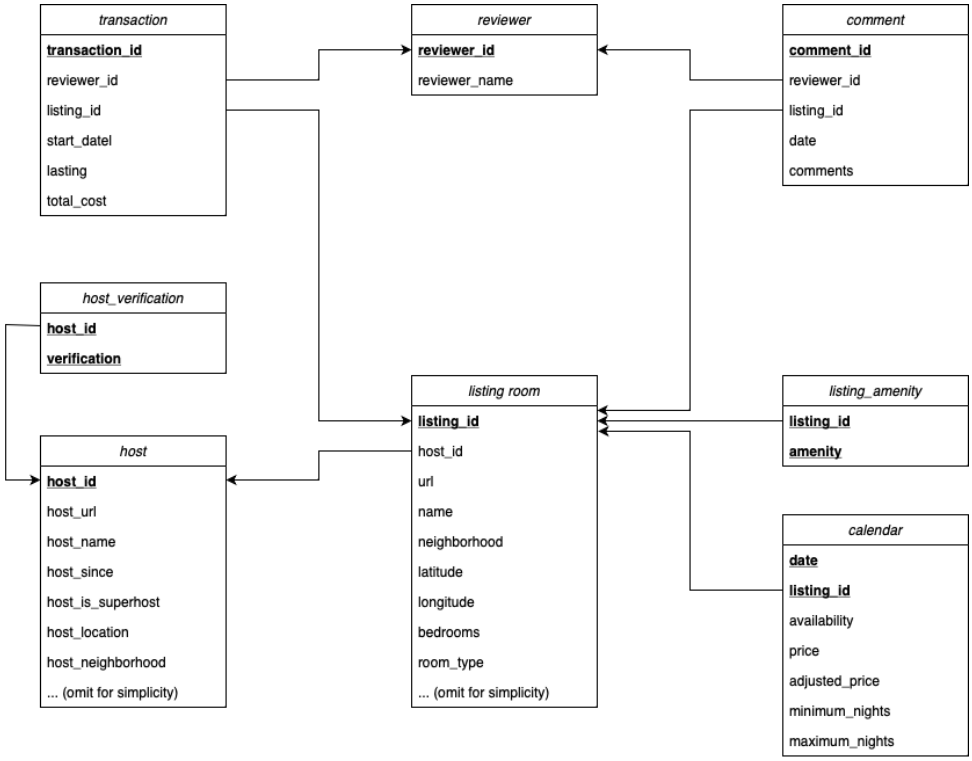


Figure 13: Refined Schema Diagram (BCNF)

Based on all above analysis, **this database is in BCNF**! Then, we split the **cleaned data (4 .csv files)** into the form of Figure 13 and store them in ***Mysql DBMS***, which can be shown in Figure 14:

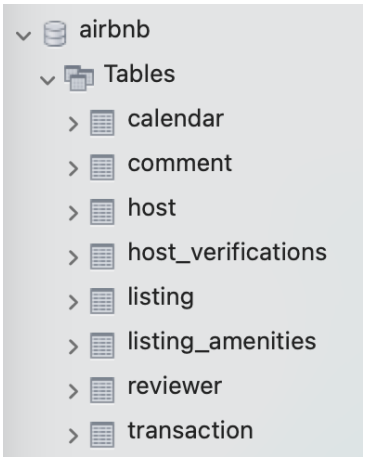


Figure 14: Mysql Database

7 Non-relational Database Storage

Firstly, we choose ***MangoDB*** to implement the **Non-relational Database** since this is what we have learnt in lecture. Again, we start from ER Diagram and then transform it into MongoDB Schema. Based on the form defined in schema, we store the data into ***MongoDB***.

7.1 ER Diagram Recap

Firstly, let's recap the ER Diagram so that we can further build **well-organized** schema **acoording to the advantages** of ***MongoDB***. The diagram is shown as follows:

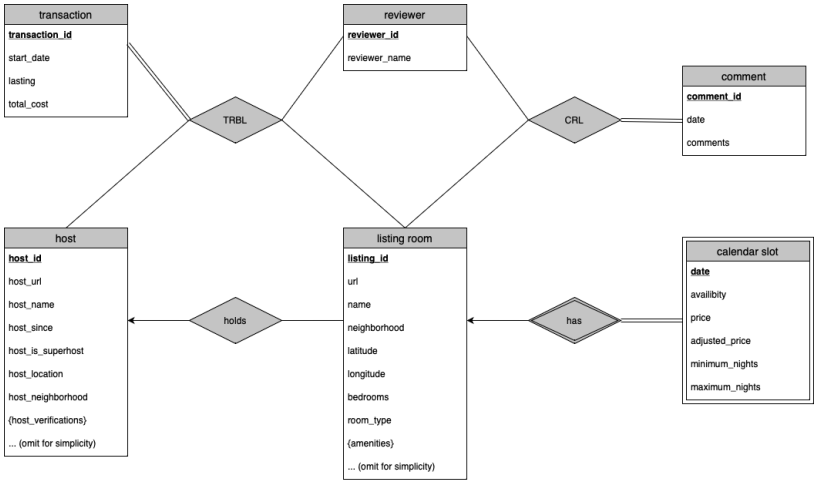


Figure 15: Entity-Relationship Diagram

Actually, what we should do is to make a choice between **'embedding' and 'reference'** based on the property of each relationship. To conclude, there are 3 kinds of relationships in ER Diagram (Figure 15):

1. 'TRBL' and 'CRL': n-ary "one-to-many" relationship with $n \geq 3$;
2. 'holds' and 'has': 2-ary one-to-many relationship;
3. 'host_verfications' and 'amenities': multi-valued attributes which can be modelled as one-to-many relationship;

7.2 Schema Analysis

In the following discussion, we will give different recipes to each kind of relationship:

1. As for these n-ary relationships, it should be noticed that, each 'transaction_id' ('comment_id') will only correspond to unique foreign key pairs, i.e., ('host_id', 'listing_id', 'reviewer_id'). In this fashion, it can be viewed as, 'transaction_id' is on the 'many' side.
Moreover, if we want to use **'embedding'** to model these 2 relationships, then **the nested term will be extremely long**. This suggests us that, **we should model these 2 relationships with respect to 'reference'**.

2. As for these 2-ary relationships, due to similar reason, we **recommend that these should be modelled as 'reference'**. Briefly speaking, **'one' side has too many attributes**, which **makes the 'embedding' formulation very 'heavy'**.

3. As for these 2 multi-valued attributes, **it should be modelled as 'embedding' because the 'one' side is very 'light'**. It can be easily nested into the json element.

7.3 Conclusion

According to previous discussion, now we can eventually determine the ***MongoDB*** **Schema** as:

1. **'TRBL' and 'CRL', 'holds' and 'has': 'reference';**
2. **'host_verfications' and 'amenities': 'embedding'.**

Moreover, then we split the **cleaned data (4 .csv files) into 6 collections (no host_verfications and listing_amenity)** and store them in ***MongoDB DBMS***, which can be shown in Figure 16:

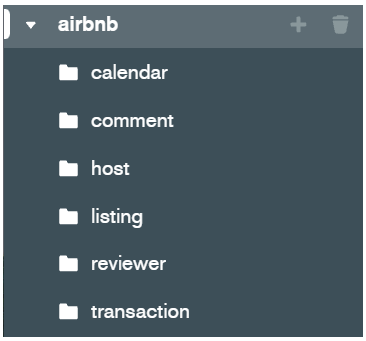


Figure 16: MongoDB Database

8 Query Answer

Before showing the results of queries, we say that, in order to achieve **better performance of query** (with respect to **efficiency**), **we set index for those important attributes**. That is, those attributes which are more likely to use for query.

Therefore, based on the Requirement Analysis as we talked about before, we display all natural language queries and their corresponding database queries as follows (**for simplicity, in all queries, we return 5 records**):

8.1 Relational Database

1. **Natural Language Query**: return 5 listings that are suitable for 3 people.
2. **Natural Language Query**: return 5 listings whose prices are between 500 and 700, combining with room type is private room.
3. **Natural Language Query**: return 5 hosts who belong to super hosts and their all listing rooms.
4. **Natural Language Query**: return 5 reservation records for the listing room whose id = '39539462' and date between '2022-07-31' and '2022-08-07'.
5. **Natural Language Query**: return 5 review records for the listing room whose id = '39539462'.
6. **Natural Language Query**: return 5 review records for the user whose name = 'Nathan'.
7. **Natural Language Query**: return the basic information for the host whose name = 'Nathan'.
8. **Natural Language Query**: return 5 previous transaction records for the user whose name = 'Nathan'.
9. **Natural Language Query**: return 5 most up-to-date comments for the host whose name = 'Nathan'.
10. **Natural Language Query**: return 5 previous transaction records for the host whose name = 'Nathan'.

8.2 Non-relational Database

1. **Natural Language Query**: return 5 listings that are suitable for 3 people.
2. **Natural Language Query**: return 5 listings whose prices are between 500 and 700, combining with room type is private room.
3. **Natural Language Query**: return 5 hosts who belong to super hosts and their all listing rooms.
4. **Natural Language Query**: return 5 reservation records for the listing room whose id = '39539462' and date between '2022-07-31' and '2022-08-07'.
5. **Natural Language Query**: return 5 review records for the listing room whose id = '39539462'.
6. **Natural Language Query**: return 5 review records for the user whose name = 'Nathan'.
7. **Natural Language Query**: return the basic information for the host whose name = 'Nathan'.

8. **Natural Language Query**: return 5 previous transaction records for the user whose name = 'Nathan'.
9. **Natural Language Query**: return 5 most up-to-date comments for the host whose name = 'Nathan'.
10. **Natural Language Query**: return 5 previous transaction records for the host whose name = 'Nathan'.

8.3 Query application

Since we have created some queries and analyze the actual needs, now we are going to show how strong it is by comparing to the results from airbnb APP.
We simply choose one listing with id = "39539462" and compare the reviews, amenities, host_info, etc. Here, we take reviews as an example.

Figure 17: Reviews from Airbnb app where "listing_id" = "39539462".

id	listing_id	date	reviewer_id	name	comments
401793369795738975	39539462	2021-07-08	22269079	Chusit	...
421285732976850167	39539462	2021-08-04	22269079	Chusit	☹☹☹
422844713348124301	39539462	2021-08-06	22269079	Chusit	☹☹☹
489515783147237582	39539462	2021-11-06	22269079	Chusit	☺☺☺
522863487884820239	39539462	2021-12-22	15527167	Daniel	Great place in a great location! Lots of restaura...

Figure 18: Reviews from SQL database where "listing_id" = "39539462".

Figure 19: Reviews from MongoDB database where "listing_id" = "39539462".

From the above three results, we can conclude that our databases can perfectly simulate different functions while using Airbnb app. There is no significant difference in response time between two databases; both of them reacts very quickly within a second.

8.4 Comparison between 2 Databases

Actually, for these 2 databases, **each of them has its own edges**. We will briefly discuss as follows:

1. **Schema**. As for **Relational Database** (Mysql in our report), it has the property that **RDBMS will try to eliminate Redundancy issues through the storage** if we obey BCNF decomposition.
Moreover, RDBMS can **better handle Functional Dependency issues**, in the sense that when we need to update with respect to FD, we will have fewer rows to update.
As for **Non-relational Database** (MongoDB in our report), it is totally differenct from RDBMS. It has the adavantage that, the **schema is more flexible**, which can handle the storage of data stream.
Additionally, **all the data can be stored in a nested form**, which will bring great benefit to query through join operation (nested can be viewed as join implicitly). Therefore, **it will make our query requests more efficient**.

2. **Running Time** For the application in Chapter 8.3, the running time for sql query is **0.115** second while for mongoDB query, the running time is **0.229** second. We test more cases and find that the running time of two types of queries is short and generally sql query responses a little bit faster. In MongoDB, we **favor CP, which means we will lose availability**. Therefore, the system may not be able to respond very quickly to the requirements (**but actually this scenario occurs rarely**).

3. **Safety**. RDBMS is ACID compliant since we use **InnoDB** engine, which **can better handle OLTP**. That is, the whole database system will be safer.

MongoDB also supports ACID transactions after new version released in 2018, but such function is not applied by default. We do not use this function and whether a database is ACID compliant actually makes no difference in our project.

9 Workload Description

Song Mingyue: mainly focus on EDA and Requirement Analysis;

Yang Sizhe: mainly focus on Data Cleaning and Data Generation;

Wang Jiangyi: mainly focus on Relational Database and Report Writing;

Lou Xinyun: mainly focus on Non-relational Database and Report Writing.

10 Summary

Lastly, we make a short summary about what we have done in this project:

1. To **understand data deeply**, we do Exploration Data Analysis on 3 whole datasets. During this process, we get some insights about the **data distribution** and figure out the **data types**. Besides that, we observe something interesting around data, e.g., the **suspicious review records** (highly likely to be Brushing Fraud Cases).

2. To successfully convert 3 datasets into DBMS, we do some data cleaning jobs, especially for the characters in **text data**. Moreover, we **do analysis on the null values** and take different actions according to the specific situation.

3. To enrich our data, we **generate transaction records** based on exisiting data. We try our best to make these 'fake' data similar to the real-world data.

4. We follow the standard pipeline to translate cleaned datasets into relational database and non-relational database. For the **relational one**, we go step-by-step carefully to **make sure it is in BCNF**. For the **non-relation one**, we **conduct comprehensive schema analysis** to guarantee its performance.

5. According to the Requirement Analysis of database translation, we give some **natural language queries** which will **occur frequently in real-life application**. For the 2 databases, we write the corresponding queries using different database languages. After that, we **conduct comparison analysis between the 2 databases**.