DSA5104 Project

Lan Rong	A0169501H
Sun Wei	A0244759N
Wang Yuhua	A0251301B
Yan Yucheng	A0251342R

Contents

Clean the Data

We first dropped columns that are not useful for our analysis.

```
listings.drop(columns=["scrape_id",
                       "last_scraped",
                       "bathrooms",
                       "calendar_updated",
                       "availability_30",
                       "availability_60",
                       "availability_90",
                       "availability_365",
                       "has_availability",
                       "calendar_last_scraped",
                       "number_of_reviews",
                       "number_of_reviews_ltm",
                       "number_of_reviews_l30d",
                       "first_review",
                       "last_review",
                       "license",
                       "calculated_host_listings_count",
                       "calculated_host_listings_count_entire_homes",
                       "calculated_host_listings_count_private_rooms",
                       "calculated_host_listings_count_shared_rooms",
                       "reviews_per_month",
                       "neighbourhood_group_cleansed"], inplace=True); # Drop columns that are not useful for our analysis
```

Clean the Data

We then fill in the rare null values by getting the information from the url included.

```
listings.loc[listings.host_listings_count.isna(), ['host_name']] = "Jakchai"
listings.loc[listings.host_listings_count.isna(), ['host_since']] = "2014-01-01"
listings.loc[listings.host_listings_count.isna(), ['host_location']] = "Bangkok, Thailand"
listings.loc[listings.host_listings_count.isna(), ['host_is_superhost']] = "f"
listings.loc[listings.host_listings_count.isna(), ['host_thumbnail_url']] = "https://a0.musc
listings.loc[listings.host_listings_count.isna(), ['host_picture_url']] = "https://a0.muscac
listings.loc[listings.host_listings_count.isna(), ['host_total_listings_count']] = 2
listings.loc[listings.host_listings_count.isna(), ['host_has_profile_pic']] = 't'
listings.loc[listings.host_listings_count.isna(), ['host_identity_verified']] = 't'
listings.loc[listings.host_listings_count.isna(), ['host_listings_count']] = 2
```

Clean the Data

For some numerical null values that are not important, we fill in their mean values.

```
for column in ['minimum_minimum_nights', 'maximum_minimum_nights', 'minimum_maximum_nights', 'maximum_maximum_nights', 'minim
    mean = listings[column].mean()
    listings[column].fillna(mean, inplace=True)
```

Clean the Data

Besides, we also dropped neighbourhood, since there is a cleansed column called neighbourhood_cleansed.

We also dropped room_type since it is included in the property_type.

We also dropped the price in listing since it is included in calendar.

We did not drop the properties with price abnormally high since some host may have set an abnormal price in the first place.

Data Generation

We used the review data to generate transaction data.

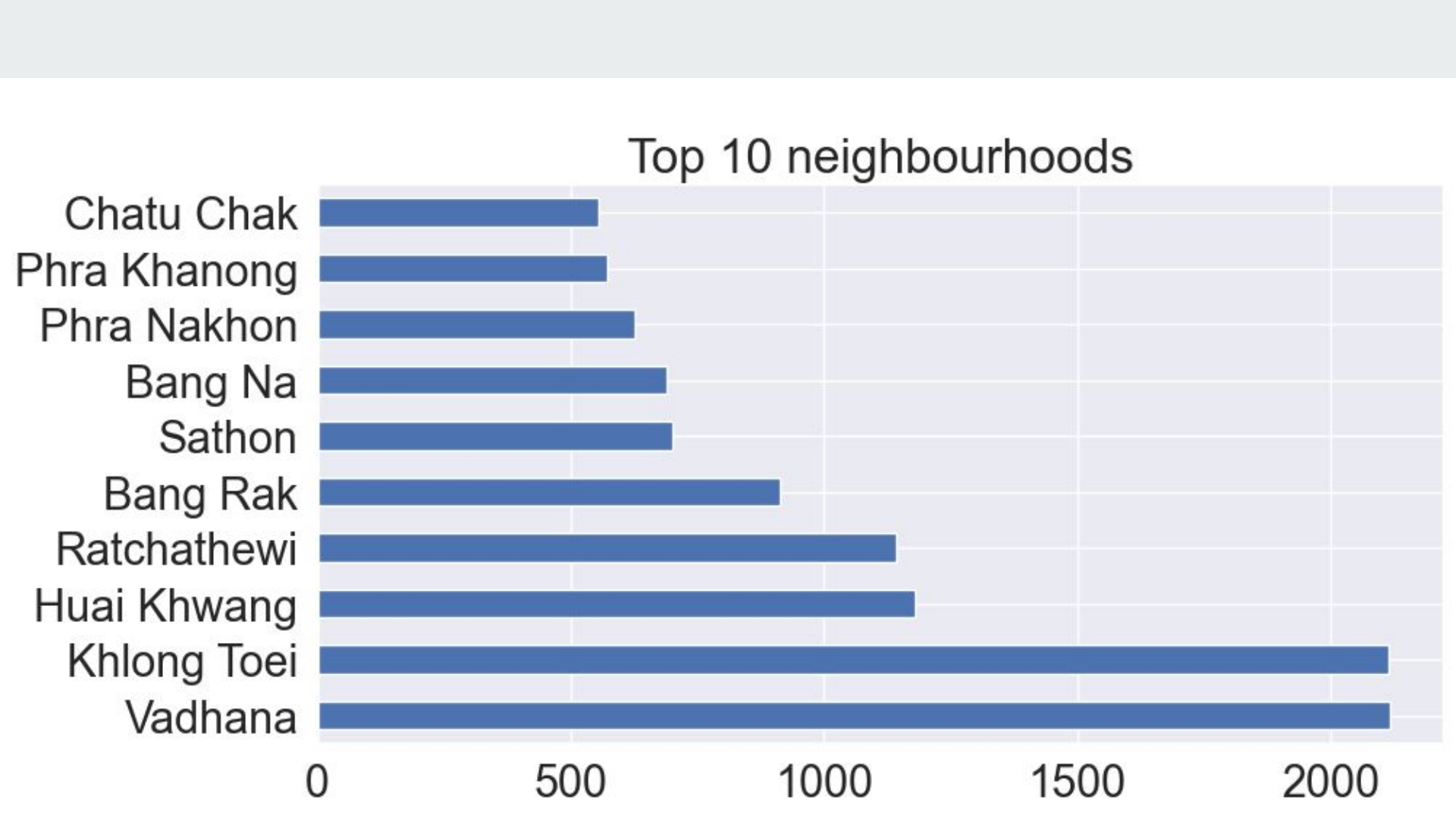
We used the date from the review table as the end_date. After generating some random number as total_days, we calculated the start_date from end_date.

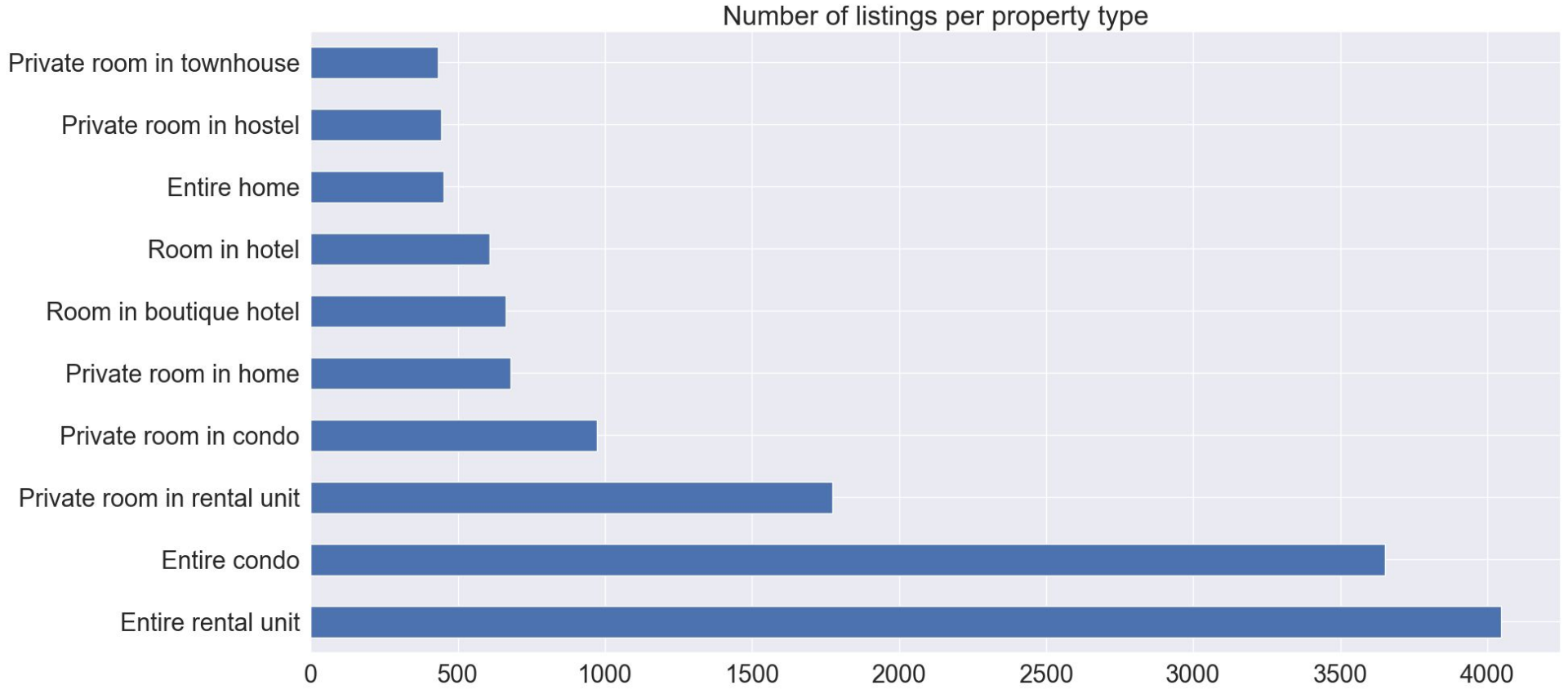
We then used the adjusted_price from calendar to generate total_price.

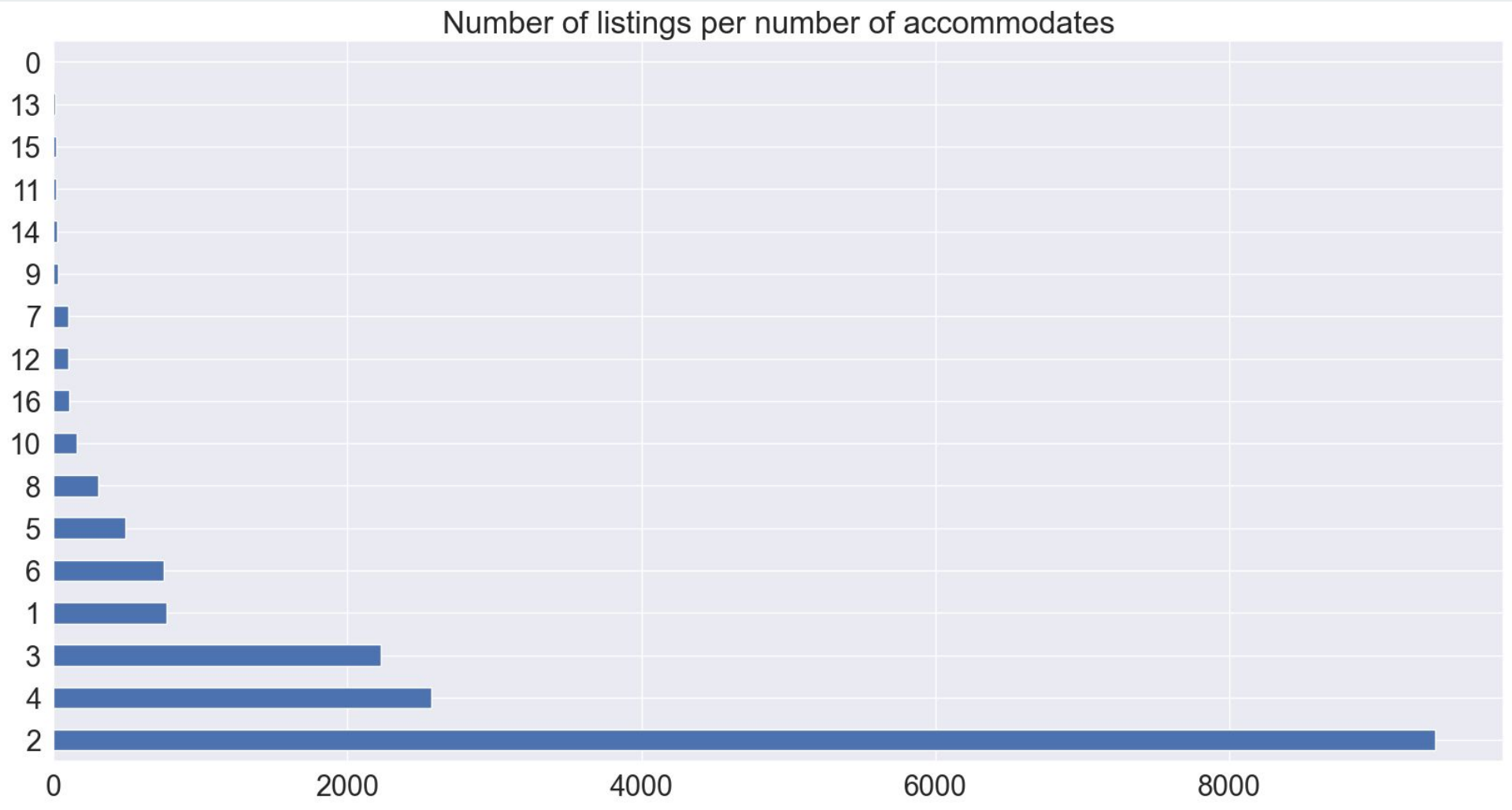
Although all of our transactions include reviews, transactions in real life may not contain a review necessarily.

Visualization

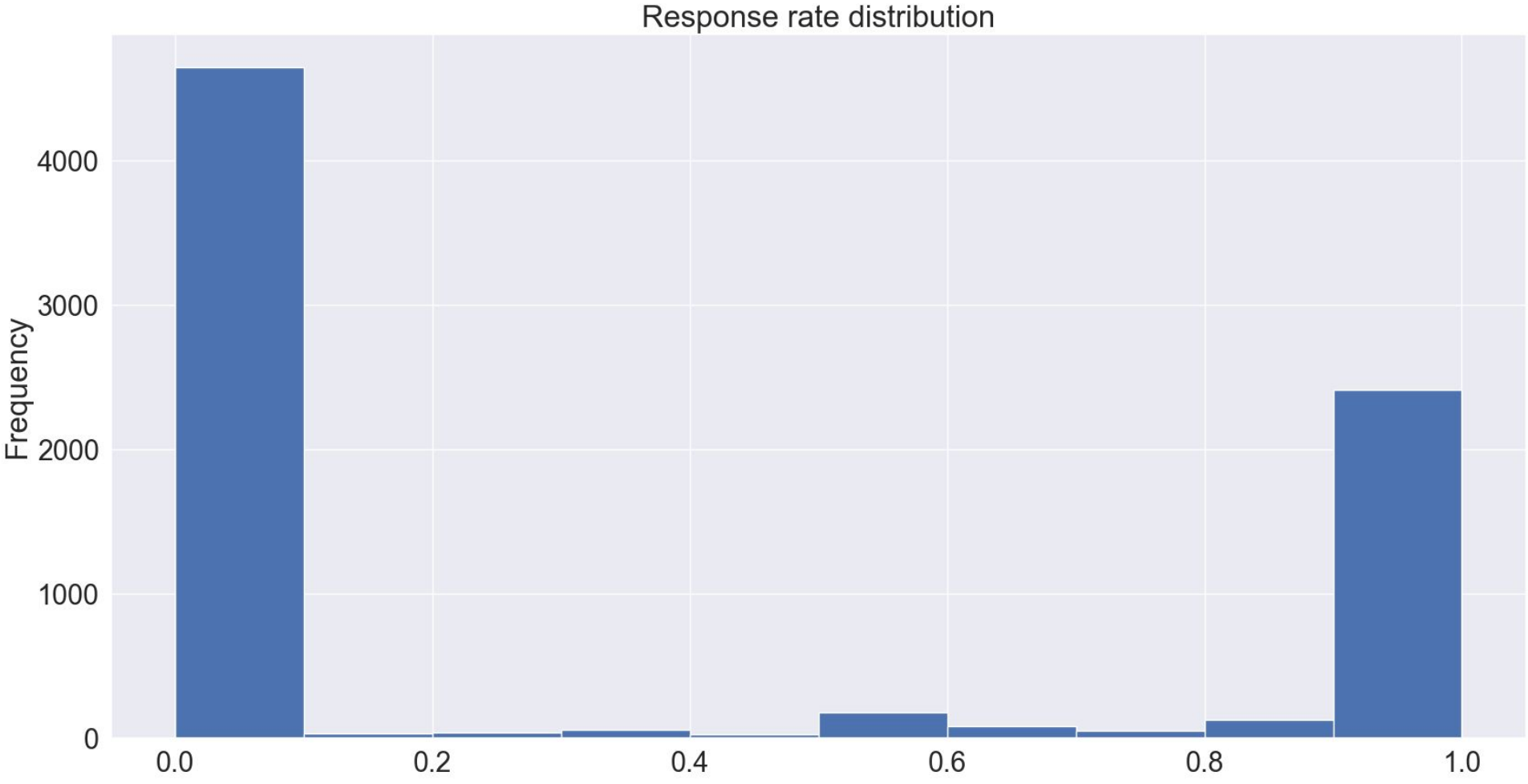
We used seaborn to visualize some categorical and numerical features in listing, host, and transaction.

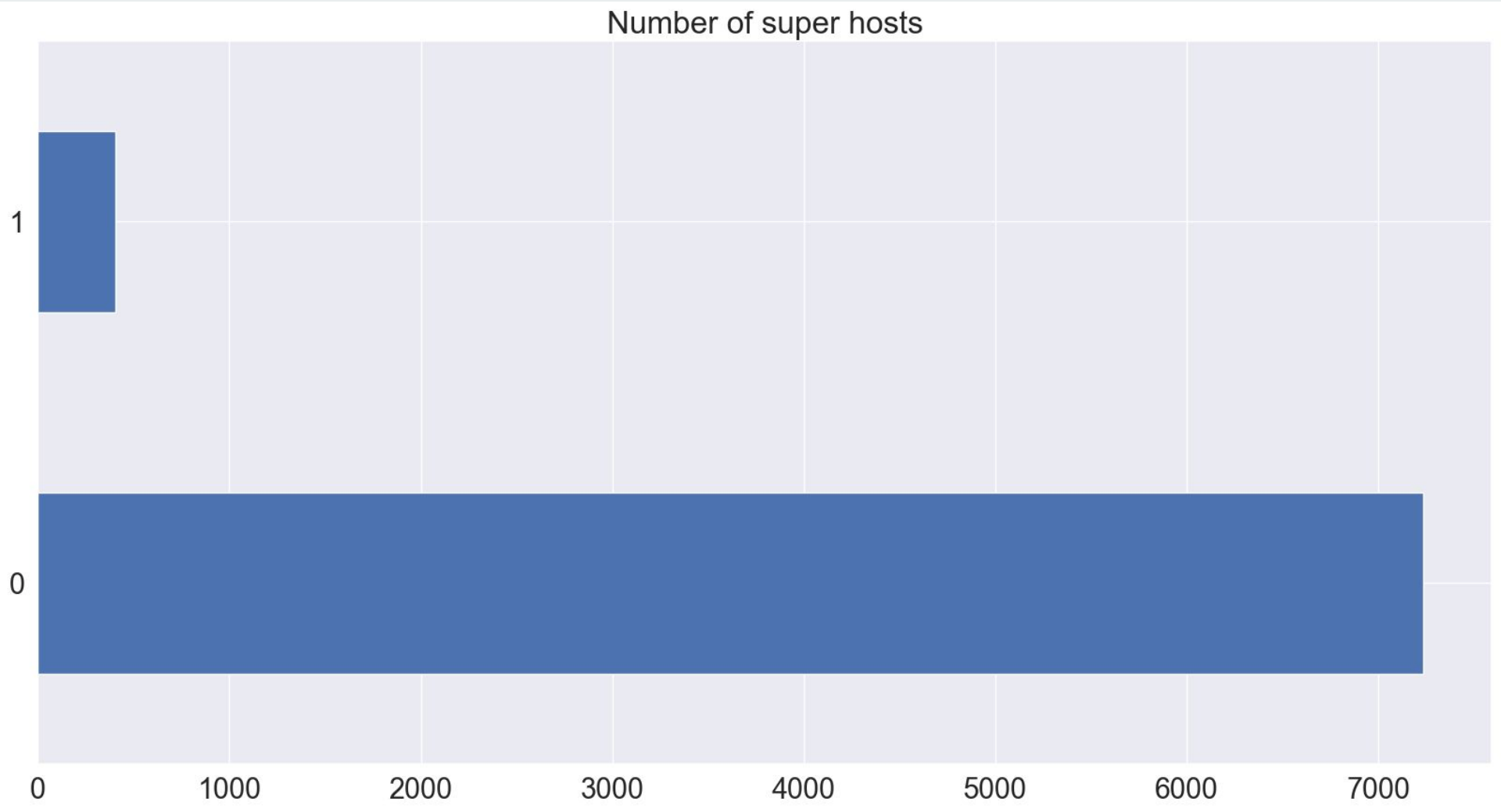

Top 10 neighbourhoods
Chatu Chak
Phra Khanong
Phra Nakhon
Bang Na
Sathon
Bang Rak
Ratchathewi
Huai Khwang
Khlong Toei
Vadhana
0
500
1000
1500
2000

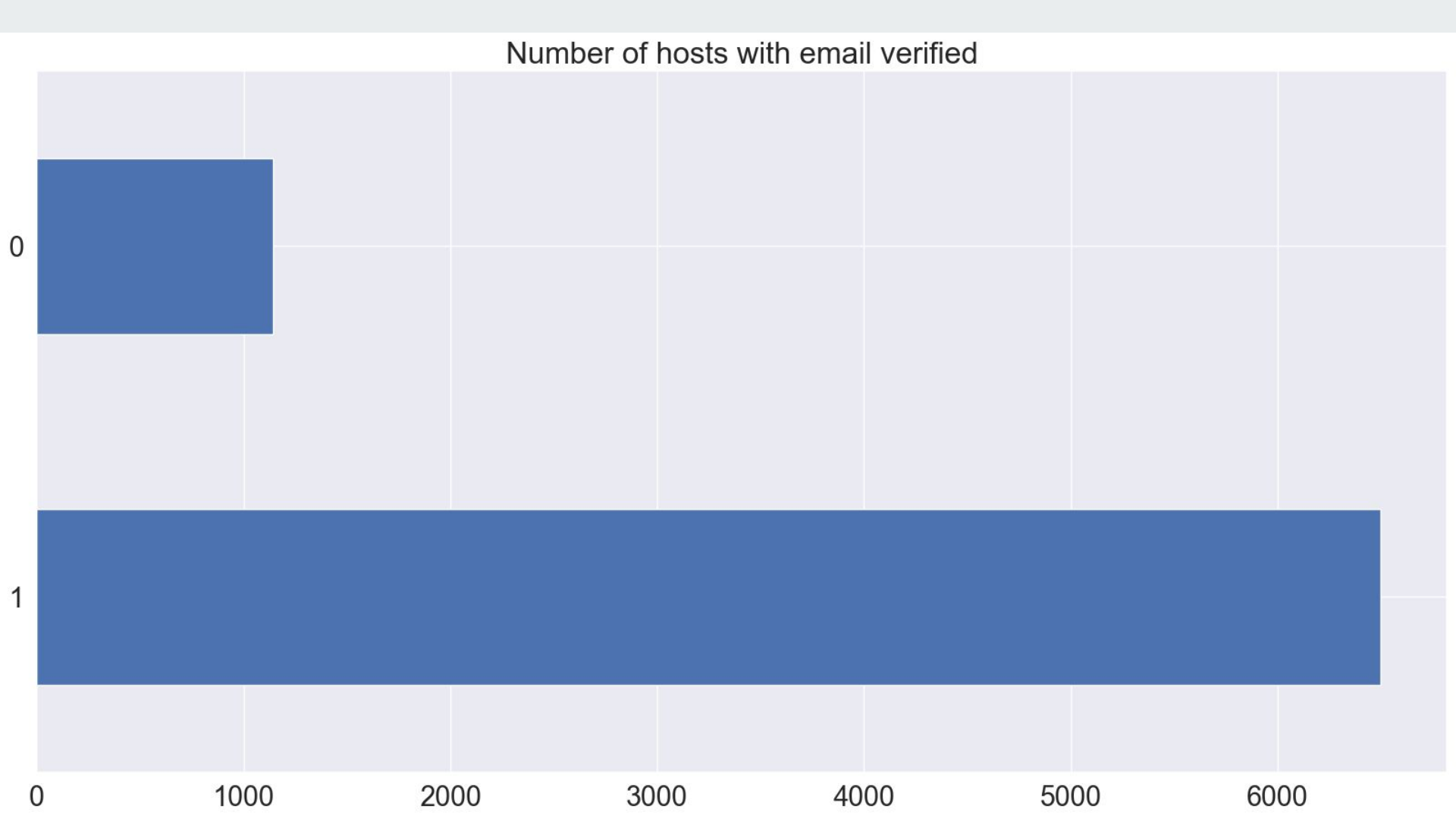

Number of listings per property type
Private room in townhouse
Private room in hostel
Entire home
Room in hotel
Room in boutique hotel
Private room in home
Private room in condo
Private room in rental unit
Entire condo
Entire rental unit
0
500
1000
1500
2000
2500
3000
3500
4000

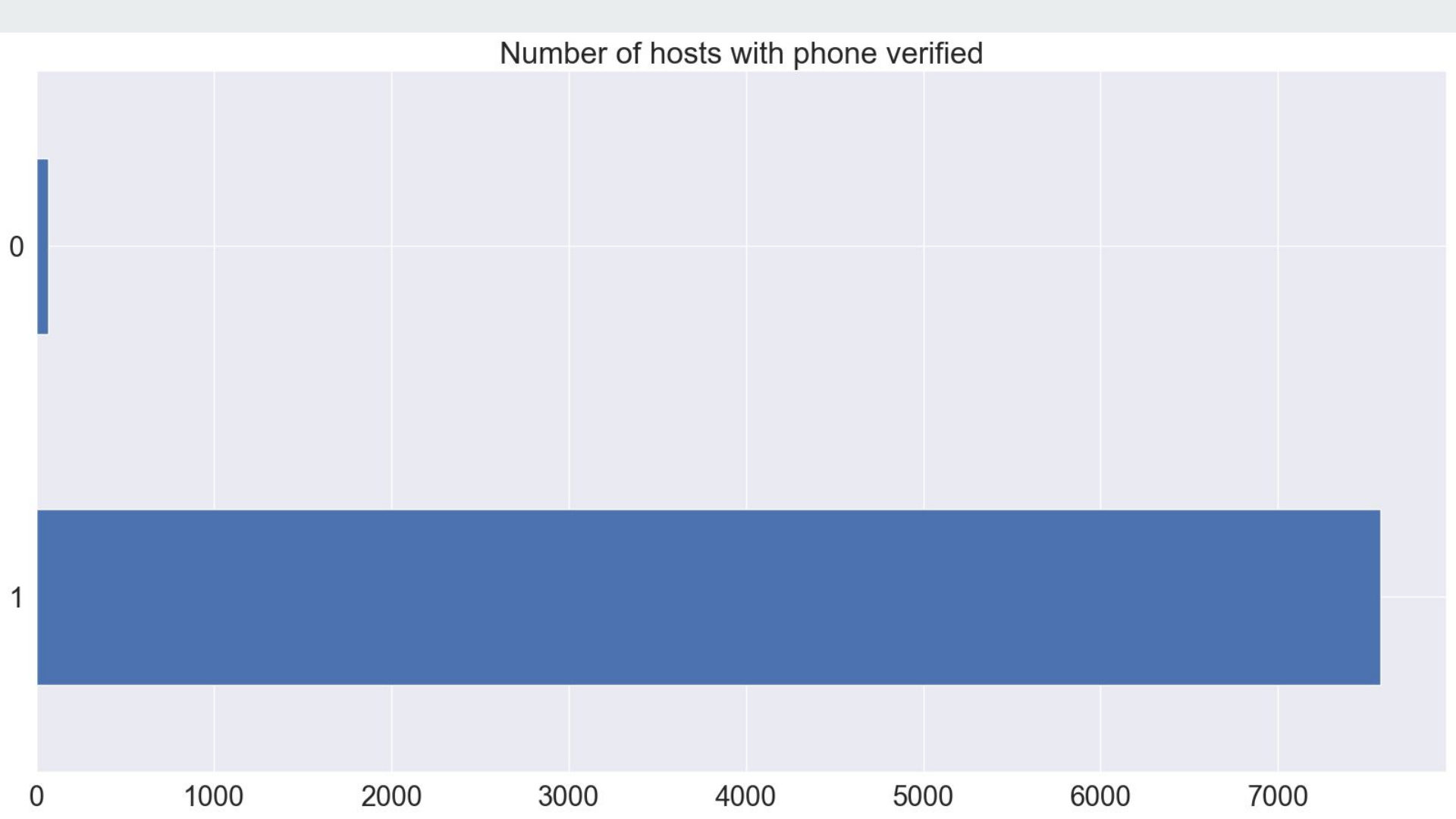

Number of listings per number of accommodates
0
13
15
11
14
9
7
12
16
10
8
5
6
1
3
4
2
0
2000
4000
6000
8000


Review scores rating distribution
Frequency
8000
7000
6000
5000
4000
3000
2000
1000
0
0
1
2
3
4
5

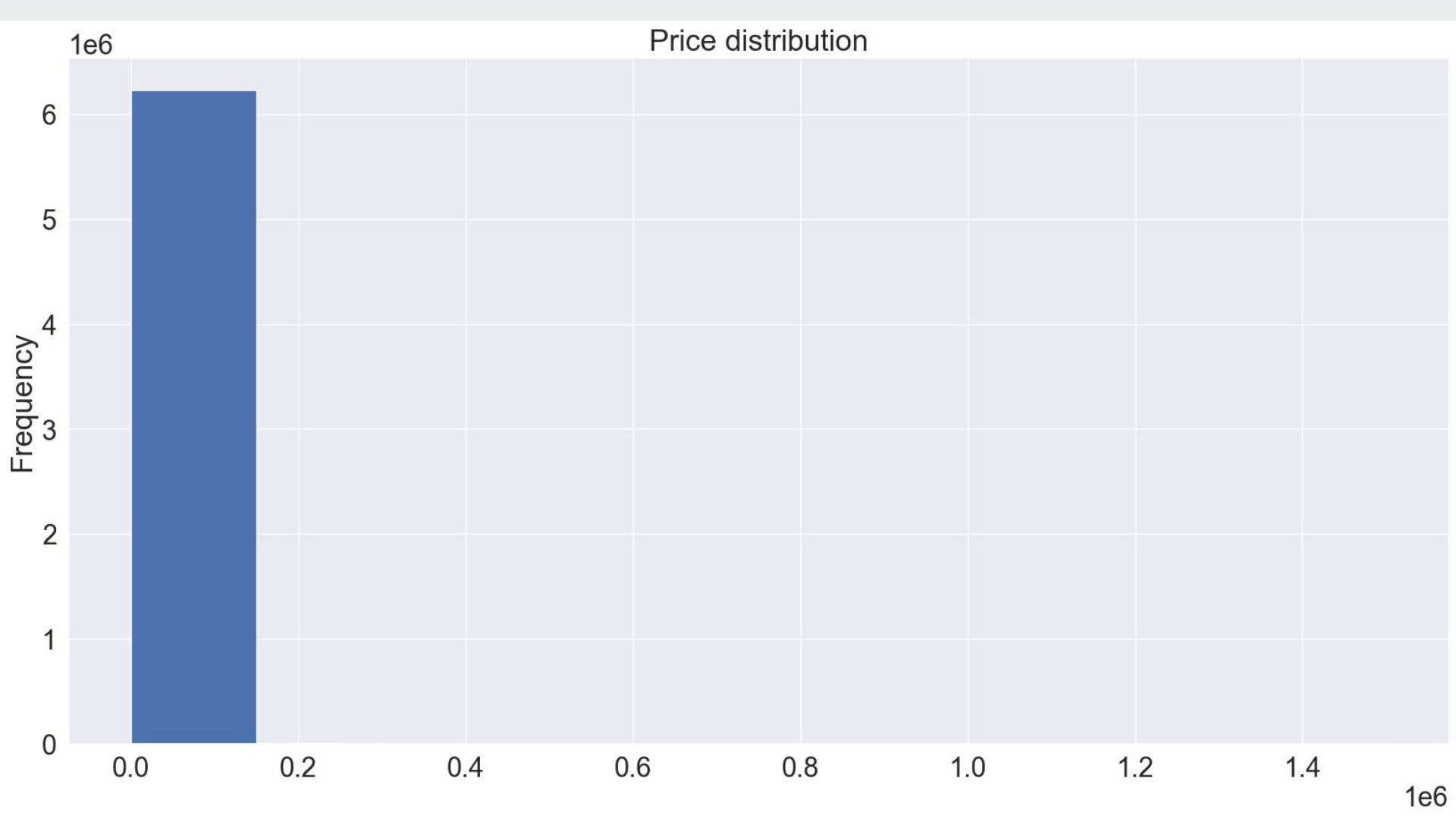

Response rate distribution
Frequency
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

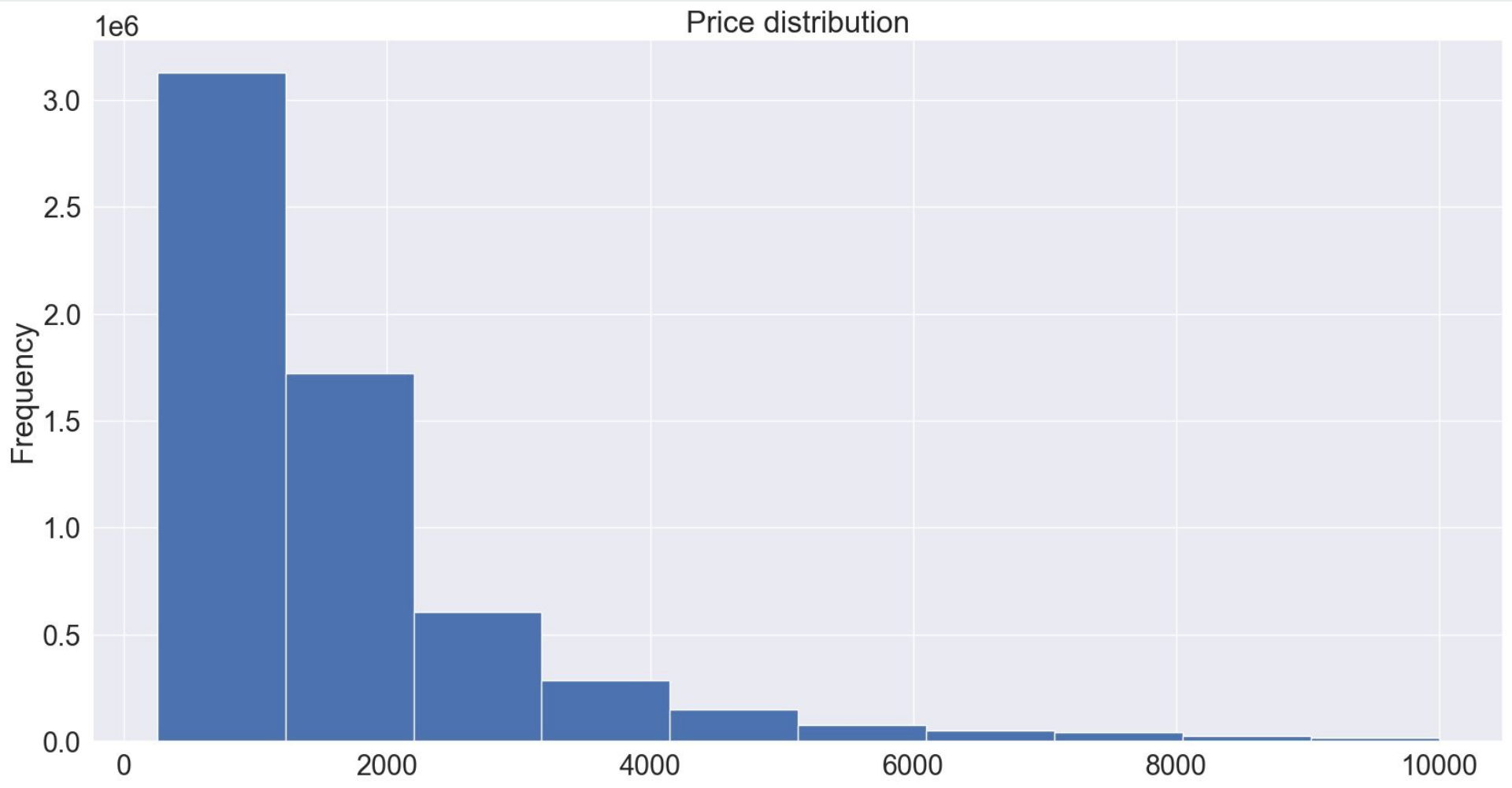

Number of super hosts
1
0
0
1000
2000
3000
4000
5000
6000
7000


Number of hosts with email verified
0
1
0
1000
2000
3000
4000
5000
6000


Number of hosts with phone verified
0
1
0
1000
2000
3000
4000
5000
6000
7000


Number of hosts with identity verified
0
1
0
500
1000
1500
2000
2500
3000
3500
4000


Price distribution
1e6
Frequency
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e6


Price distribution
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Frequency
0
2000
4000
6000
8000
10000

Relational Database - Requirement Analysis

- Check property information (availability/comments/...)
- Book available property
- Search property by name/availability/type/area/host/...
- Give comments and reviews after stay
- Search host by ratings/name/...
- Check user information

ER Diagram

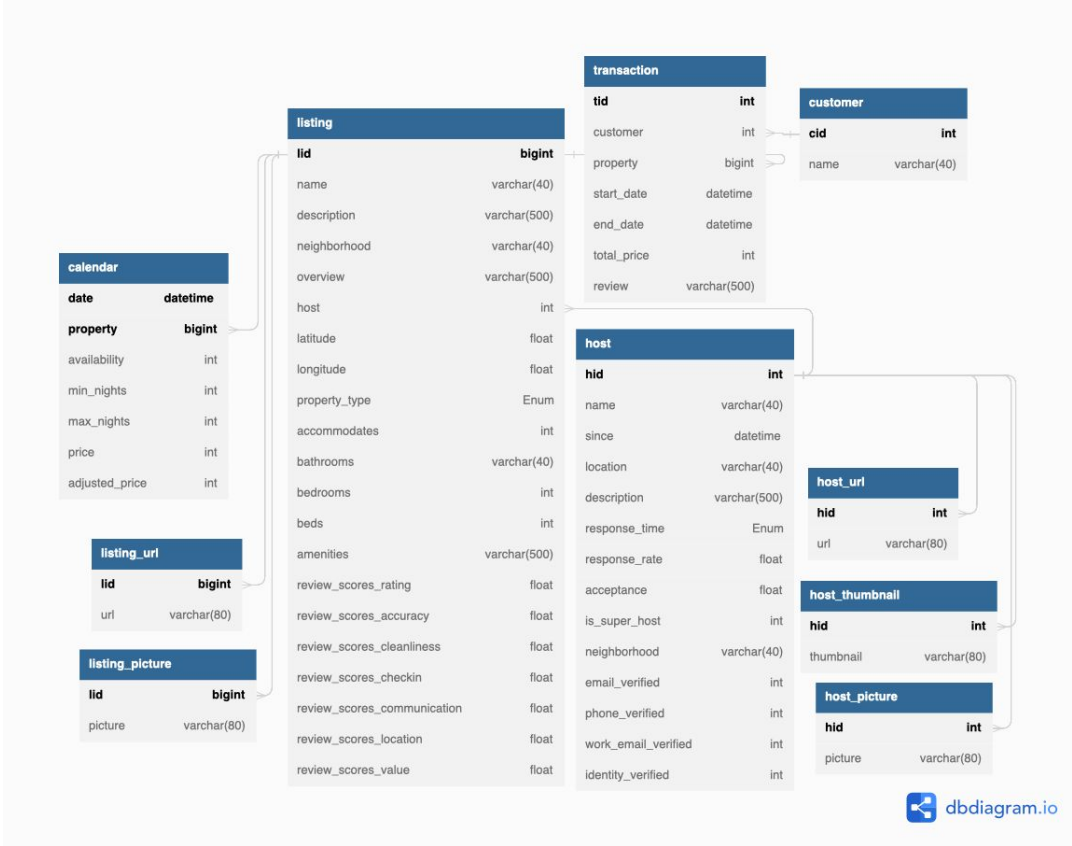

host
hid int
name varchar(40)
since datetime
location varchar(40)
description varchar(500)
response_time Enum
response_rate float
acceptance float
is_super_host int
neighborhood varchar(40)
email_verified int
phone_verified int
work_email_verified int
identity_verified int
url varchar(80)
thumbnail varchar(80)
picture varchar(80)
host_list
hid int
listing bigint
listing
lid bigint
name varchar(40)
description varchar(500)
neighborhood varchar(40)
overview varchar(500)
latitude float
longitude float
property_type Enum
accommodates int
bathrooms varchar(40)
bedrooms int
beds int
amenities varchar(500)
url varchar(80)
picture varchar(80)
review_scores_rating float
review_scores_accuracy float
review_scores_cleanliness float
review_scores_checkin float
review_scores_communication float
review_scores_location float
review_scores_value float
transaction
tid int
start_date datetime
end_date datetime
total_price int
review varchar(500)
transaction_customer
tid int
cid int
customer
cid int
name varchar(40)
transaction_listing
tid int
lid bigint
calendar
date datetime
availability int
min_nights int
max_nights int
price int
adjusted_price int
calendar_listing
date datetime
lid bigint
dbdiagram.io

Schema Diagram

When designing schema from ER diagram. We decompose host's and listing's urls and picture urls since they violate the BCNF.

We did not keep any of the relationships since all of them are one-to-many relationships.

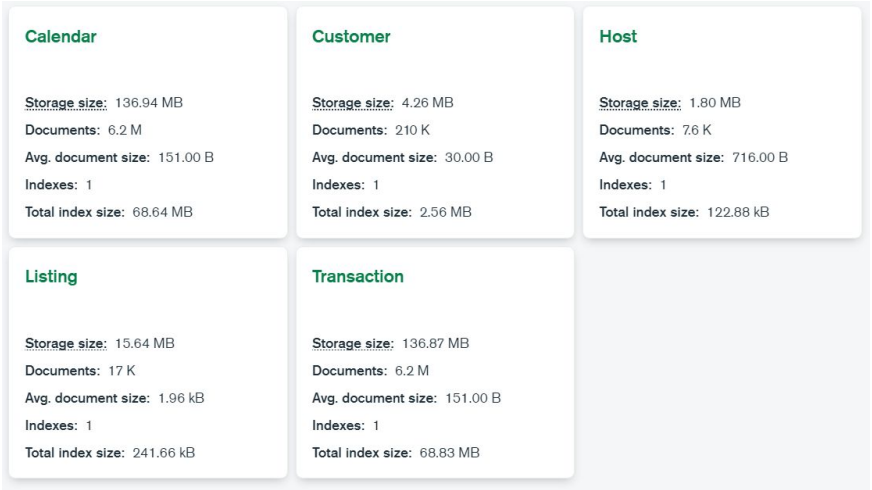
You may think that customer, property, and start_date in transaction may violate BCNF. However, one customer may have booked the room and cancelled and booked again in the same day.



Non-Relational Database

For the Non-Relational Database of this project, we chose MongoDB since it is explained in details in the lectures.

We considered various designs for the database. Similar to SQL database , we decide to use the 5 collections because this schema makes it easy to query for the frequently demanded information.



MongoDB Schema Design

For Calendars, its arguable that they could be embedded inside the corresponding Listing document. However, when considering the use case in context, we reckon there are mainly 2 scenarios when such information is queried:

1. User wants to see the available dates of a particular listing
2. User wants to find all available listings on a particular date

Embedding Calendars inside Listings made scenarios 1 easy to accomplish. For scenarios 2, on the other hand, we need to iterate through all listings and check if they contain the specified date. This, is computationally expensive. Thus, we decided to store Calendar in its own collection.


```
{
  "_id": {
    "$oid": "636b583c69c65a2eb1f7f92b"
  },
  "property": 37433623,
  "date": "2022-08-14",
  "availability": true,
  "min_nights": 7,
  "max_nights": 1125,
  "price": 900,
  "adjusted_price": 900
}
```


Similar use case analysis was performed on Transaction and Listing too.

Queries

Natural Language	SQL Query	MongoDB Query
Check property information (Find availability of a certain property on a certain day.)	SELECT availability FROM calendar JOIN listing WHERE listing.name=... AND date=...	listing=db.Listing.findOne({“name”:…}) db.Calendar.find({“property”: listing._id, “date”:...}, {“availibility”:...})
Book available property (Book a property and set its availability to 0 in booking period)	UPDATE calendar SET availability = 0 WHERE property=... AND date>=... AND date<=...	db.Calendar.updateMany({ “property” :listing._id, “date”; {“$gte”: …, “$lte”:...}}, {“$set”:{“availability”:False}})

Queries

Natural Language	SQL Query	MongoDB Query
Search property by name/type/host/ …	SELECT * FROM listing WHERE property_type=... AND neighbourhood=...	db.Listing.find({“neighbourhood”: …, “property_type”: …, “bedrooms”: …, “beds”: …})
Give comments and reviews after stay	UPDATE transaction SET review=... WHERE tid=...	db.Transaction.updateOne ({“_id”:...}, {“$set”: {“review”: …}})

Queries

Natural Language	SQL Query	MongoDB Query
Search host by ratings/name… (select a host whose response rate is large and his/her listings has high review scores)	SELECT * FROM host H WHERE response_rate>=... AND EXISTS (SELECT * FROM listing L WHERE L.host=H.hid AND L.review_score>=...)	hosts=db.Listing.find({“ review_scores_rating”: {“$gte”: …}}, {“host”:...}) db.Host.find({“_id”: {“$in”: [h.host for h in hosts]}}, “response_rate”: {“$gte:...”})
Check user information	SELECT * FROM customer WHERE name=...	db.Customer. find({“name”: …})

Thank you.