COMPETITIONS BETWEEN AIRBNB ACCOMMODATIONS AND HOTELS IN LOS ANGELES

FINAL REPORT OF CIT 550 FINAL PROJECT

Team: Charging Bulls

Team Members*: Liang Tang (Email: tangreb@seas.upenn.edu, Github: LiangTang888)
Dian Gu (diangu@seas.upenn.edu, Github: DianGu86),
Xiaofei Huang (xh2342@seas.upenn.edu, Github: xh2342),
Zhongyin Zhang (johnianz@seas.upenn.edu, Github: Johnianzhang),

*All team members contribute equivalently

1. Introduction and Project Purpose

Sharing economy as a shift from ownership of goods to temporary rental of them has emerged since 2009 due to global economic recession, cumulative trust of world wide web, and development of online payment system (Dillahunt & Malone, 2015). The lodging industry is probably one of the sectors most impacted by the meteoric development of sharing economy (Johnson & Neuhofer, 2017). Airbnb has risen as a dominant player of P2P accommodations with the net worth of $67.6B in 2023, which is higher than almost any of the top hotel corporations, including Marriott (53.54B), Hilton (45.03B), and Hyatt (13.43B). Accordingly, it is crucial to investigate the competitions between Airbnb accommodations and hotel providers in a district. Thus, our team was motivated to develop a website for both Airbnb hosts and hoteliers who are interested to improve their competitiveness in the local market. Los Angeles was selected as an investigation site for the website.

The website title is AirInsights. “Air” is a cool abbreviation of Airbnb. By referring a significant number of academic publications in digital marketing, especially in hospitality industry, we identified the following factors which determine the competitions between sharing accommodations and traditional hotels, including reviews (e.g., Schamel, 2012), pricing (e.g., Espinet et al, 2003), functional (e.g., room type, number of customers hosted, number of bedrooms and beds) and host information (e.g., Hung, Shang, & Wang, 2010), as well as spatial dependency (i.e., geographic locations of Airbnb offerings and hotels in a specific area) (e.g., Dube & Legros, 2014). Accordingly, these attributes were used as filters of the website. And Xiaofei Huang, a team member who is professional designer with plentiful years of experience, provided the aesthetic design of the website.

2. Architecture

We processed and cleaned data with Pandas. MySQl hosted on AWS serves our Database. We adopted React for front-end, since it allows users to design simple views for each state in the application and efficiently update and render just the right components when data changes. NodeJs Express was used for the Back-end since Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

3. Data

We adopted the combination of secondary dataset and data scraping for the project. The secondary dataset was gained from http://insideairbnb.com/. The website collects data from primary cities on the Airbnb website on a regular basis and makes data available to the public. The purpose of the website is to collect evidence to support the website owners to win in a debate against Airbnb. The website owners believe that the majority of Airbnb listings in most cities are entire homes, many of which are rented all year around, disrupting housing and communities. We used the dataset of Los Angeles scraped on Mar 7, 2023, which is available on http://insideairbnb.com/get-the-data/. The dataset includes four tables titled with listings, calendar, reviews, and neighborhoods. The raw tables are not cleaned. Some attributes in different tables are duplicated and some of the attribute titles can’t tell well the meaning of the attributes. And especially the table of calendar includes the 365 records for one single listing id.

After communicating with the dataset owner and exploring the insiderairbnb website, we figured out that the listings in the table of “calendar” are the long-term rental properties, which are part of the properties in “General_listings”. We cleaned data with a diversity of techniques, including replacing categorical variables with indicators (replace different room types with numbers), replacing missing data for most of attributes, but kept null values for host information according to the TA’s guidance on ed to practice complex queries. The sample summary statistics is included in the appendix A.

4. Database

Among the nearly 100 attributes in the raw databases, we carefully filtered the useful attributes which are consistent with the purpose of the website by referring to a number of academic publications related to sharing accommodation and hospitality businesses. Based on the purpose of the website, we generated three tables from the raw dataset. The first table is “General_listings” (e.g., property information, reviews, price, geographic information) with 42,450 records and 18 attributes. The second table is “LongtermRental” (e.g., daily price of one year for the long-term rental properties) with 1,048,575 records and 6 attributes. The third table is “Host_information_distinct” (e.g., host response time, host is superhost or not) with 21,247 records and 6 attributes. Besides the secondary dataset, we also scraped hotel information from www.hotels.com. The table is titled with “hotels_clean” with 2,135 records and 12 attributes. All the tables are BCNF since the raw dataset provides listing_id as the listing key and host_id for the host key and for every relationship $X \rightarrow A$, X is the superkey for the table. And furthermore, by referring to the mentor’s suggestions, we didn’t decompose the table of “General_listings” upon different categories of attributes corresponding to different webpages. The ER diagram is illustrated in Figure 1 in Appendix D, and the relational Schema is included in Appendix B.

5. Web App Description

A general structure of every webpage has three components, including bar, main page and footer. The navigation bar contains the hyperlinks to all 5 pages, which are the homepage, property features, host information, Airbnb vs. hotels, and top listings. They redirect a user to the corresponding page when clicked. In addition, when the mouse hovered on the link, it will highlight up by changing to a larger padding and background color from #569DAA to a darker one #577D86. The footer contains the team members’ names. The detailed functions of individual webpages are described below:

Home page: background information of sharing accommodations and purpose of the website.

Property feature page and host information page: These two pages show similar functionalities and design, but include different content with distinct queries and routes. On the property feature page, a user could filter the information of the Airbnb listings, including room type (e.g., entire home/apt, private room, etc.), accommodates (i.e., number of guests hosted), number of bedrooms, and number of beds. The search result is demonstrated as a chart with x-axis as the price ranges (under 50, 50-99, 100-149, 150-199, 200-249, over 250) and y- axis as the number of properties satisfying the filter criteria. With similar structure, host information allows a user to filter information on the host’s response time, response rate, acceptance rate, whether he/she is a host, and the total number of listings. A similar bar chart is used to illustrated the relationship between the number of listings satisfying the filter criteria on hosts and price ranges.

Airbnb vs. hotels page: With the assistance of the embedded Google map API, a map is shown to demonstrate the spatial relationship between Airbnb listings and hotels in the specific area given by the user. The map contains most of the functionalities of Google Map, including zoom, drag, satellite, street-view. Below the map, a search box is to prompt the user to input a neighborhood name with the function of self-examination (i.e., report error for invalid input). The corresponding query #9 provides the comparison result of review ratings and price, and suggests the user to choose Airbnb or hotels in this area considering the composite value.

Top listing page: This page allows a user to filter review type (rating, accuracy, cleanliness, check-in, communication, location, value) and page size (10, 20, 50, 100). As a result of search, a table demonstrates the corresponding Airbnb listings. The reviews in all the categories are based on 1-5 Likert scale (1=extremely unsatisfactory; 2=very unsatisfactory; 3=neutral; 4=very satisfactory; 5-extremely satisfactory). All the columns can be sorted (e.g., ascending order, depending order) or hidden. A property card shows up when the property ID of the table is clicked, which includes textual information (i.e., neighborhood name, area, latitude, longitude and general price) and radar chart.

6. API Specification

The general information of routes is described below. The details are included in appendix C.

Page Routes
Route 1: “/” – route to the homepage with a description of the website.
Route 2: “/property_features” - route to a webpage that allows users to search property by property features.
Route 3: “/host_info”- route to a webpage that enables users to search Airbnb listings by host information.
Route 4: “/hotels”- route to a webpage that performs cross-analysis with Airbnb listings and hotels.
Route 5: “/top_listing” - route to a webpage that returns the top listings for different review metrics.

Query Routes
Route 6: “/search_features” - The route uses a GET() method to retrieve numbers of listings in different price ranges filtered by input property features.
Route 7: “/search_features_count” - The route uses a GET() method and returns the total number of retrieved listings filtered by input property features.
Route 8: “/search_features_percentage” - The route uses a GET() method and returns the percentage of the total listing that is retrieved after filtering the data with input property features.
Route 9: “/search_host_info” - The route uses a GET() method to retrieve numbers of listings in different price ranges filtered by input host information.

Route 10: “/search_host_info_count” - The route uses a GET() method and returns the total number of retrieved listings filtered by input host information.
Route 11: “/search_host_info_percentage” - The route uses a GET() method and returns the percentage of the total listing that is retrieved after filtering the data with input host information.
Route 12: “/top_ranking” - The route uses a GET() method to retrieve top rankings for the input review type.
Route 13: “/top_listing/:listing_id” - The route uses a GET() method to retrieve information for a particular listing with listing_id.
Route 14: “/rec” - The route uses a GET() method to retrieve both Airbnb and hotel data in the same neighborhood and perform cross-analysis on the price and review scores before providing recommendations on the accommodation.

Map Functionality
Route 15: “/midpoint” - get the middle point for maps by neighborhood, used for mapping.
Route 16: “/airbnb_neighborhood”- get the coordinates for the airbnb listings for the given neighborhood.
Route 17: “/hotels_neighborhoo” - get the coordinates for the hotel listings for the given neighborhood.
Route 18: “/neighborhoods” - get all the neighborhoods in the dataset.

7. Queries

The most complex queries are query #9 and query #4.

Query #9 is as follows:
```
WITH airbnbs AS (
    SELECT g.ind, g.neighborhood, g.general_price, g.accommodates,
      COALESCE(g.review_scores_accuracy, 0) +
      COALESCE(g.review_scores_cleanliness, 0) +
      COALESCE(g.review_scores_checkin, 0) +
      COALESCE(g.review_scores_communication, 0) +
      COALESCE(g.review_scores_location, 0) +
      COALESCE(g.review_scores_value, 0) AS sum_ratings,
      CASE
        WHEN g.review_scores_accuracy IS NOT NULL THEN 1 ELSE 0 END +
      CASE
        WHEN g.review_scores_cleanliness IS NOT NULL THEN 1 ELSE 0 END +
      CASE
        WHEN g.review_scores_checkin IS NOT NULL THEN 1 ELSE 0 END +
      CASE
        WHEN g.review_scores_communication IS NOT NULL THEN 1 ELSE 0 END +
      CASE
        WHEN g.review_scores_location IS NOT NULL THEN 1 ELSE 0 END +
      CASE
        WHEN g.review_scores_value IS NOT NULL THEN 1 ELSE 0 END AS
count_ratings
    FROM General_listings g
    WHERE g.ind NOT IN (
      SELECT ltr.ind
      FROM LongTermRental ltr
      WHERE ltr.date < CURDATE()
      GROUP BY ltr.ind)
  ),
  airbnbs_stats AS (
    SELECT ind, neighborhood,
      (sum_ratings / count_ratings / 5) * 100 AS percentage_ratings,
      (general_price / accommodates) AS percapita_price
    FROM airbnbs
  ),
  airbnb_aggregates AS (
    SELECT neighborhood,
      AVG(percentage_ratings) AS avg_bnbratings,
      AVG(percapita_price) AS avg_bnbprice
    FROM airbnbs_stats
    GROUP BY neighborhood
  ),
  hotels_stats AS (
    SELECT h.hotel_id, h.neighborhood, h.price, h.number_people,
      (COALESCE(h.cleanness, 0) + COALESCE(h.service, 0) +
COALESCE(h.amenities, 0) +
      COALESCE(h.facilities, 0) + COALESCE(h.ecofriendly, 0)) AS sum_ratings,
      (CASE WHEN h.cleanness IS NOT NULL THEN 1 ELSE 0 END +
      CASE WHEN h.service IS NOT NULL THEN 1 ELSE 0 END +
      CASE WHEN h.amenities IS NOT NULL THEN 1 ELSE 0 END +
      CASE WHEN h.facilities IS NOT NULL THEN 1 ELSE 0 END +
      CASE WHEN h.ecofriendly IS NOT NULL THEN 1 ELSE 0 END) AS count_ratings
    FROM hotels h
    GROUP BY h.hotel_id
  ),
  hotels_aggregates AS (
    SELECT neighborhood,
      AVG((sum_ratings / count_ratings / 10) * 100) AS avg_hotelsratings,
      AVG(price / number_people) AS avg_hotelsprice
    FROM hotels_stats
    GROUP BY neighborhood
  ),
  combined_stats AS (
    SELECT a.neighborhood, a.avg_bnbratings, a.avg_bnbprice,
      h.avg_hotelsratings, h.avg_hotelsprice,
      (CASE WHEN h.avg_hotelsratings > a.avg_bnbratings THEN 'Hotels' ELSE
'Airbnb' END) AS better_rating,
      (CASE WHEN h.avg_hotelsprice < a.avg_bnbprice THEN 'Hotels' ELSE
'Airbnb' END) AS better_price
    FROM airbnb_aggregates a
    LEFT JOIN hotels_aggregates h
    ON a.neighborhood = h.neighborhood
  )
  SELECT *
  FROM combined_stats
  WHERE neighborhood = '${neighborhood}';
```

The query #9 aims to calculate the per-capita pricing and average rating for each neighborhood for both Airbnbs and hotels and determine a better option (lower price and higher rating) for the user. It performs the following steps:

1) The Airbnbs CTE filters short-term Airbnbs by excluding long-term rentals in the LongTermRental table;

2) The airbnbs_stats CTE computes ratings and per-capita prices for the short-term Airbnbs based on reviews and price information;

3) The airbnb_aggregates CTE calculates average ratings and per-capita prices for short-term Airbnbs, grouped by neighborhood;

4) The hotels_stats CTE computes ratings and per-capita prices for hotels based on provided review and price information;

5) The hotels_aggregates CTE calculates average ratings and per-capita prices for hotels, grouped by neighborhood;

6) The combined_stats CTE joins the Airbnb and hotel aggregates and determines whether hotels or Airbnbs have better ratings and prices for each neighborhood;

7) The final SELECT statement retrieves the results from the combined_stats CTE for the specific neighborhood.

<u>Query #4 is as follows:</u>

```
 SELECT
      SUM(IF(b.price<50,1,0)) as 'UNDER 50',
      SUM(IF(b.price BETWEEN 50 AND 99,1,0)) as '50-99',
      SUM(IF(b.price BETWEEN 100 AND 149,1,0)) as '100-149',
      SUM(IF(b.price BETWEEN 150 AND 199,1,0)) as '150-199',
      SUM(IF(b.price BETWEEN 200 AND 249,1,0)) as '200-249',
      SUM(IF(b.price>=250,1,0)) as 'OVER 250',
      SUM(IF(b.price IS NULL, 1, 0)) as 'Not Filled In (NULL)'
      FROM
          (SELECT g.ind, g.host_id, g.general_price AS price
           FROM General_listings g
           WHERE g.ind NOT IN
           (SELECT a.ind
           FROM
              (SELECT l.ind, AVG(price)
               FROM General_listings g JOIN LongTermRental l ON g.ind = l.ind
               WHERE date< CURDATE()
               GROUP BY l.ind) a)) b
          JOIN Host_information h ON h.host_id=b.host_id
      WHERE h.host_response_time='${inputHost_response_time}' &&
h.host_response_rate IS NOT NULL &&
            h.host_response_rate='${inputHost_response_rate}' &&
h.host_acceptance_rate IS NOT NULL
            && h.host_acceptance_rate='${inputHost_acceptance_rate}'
            && h.host_is_superhost=='${inputHost_is_superhost}' &&
h.host_total_listings_count=‘${inputHost_total_listings_count}';
```

Query #4 follows the four steps:

1) identify the listings in the table of “General_listings”;

2) exclude the listings which are long-term rentals and calculate the average price before the current date from the table of “LongtermRental”;

3) join the table of “Host_information_distinct” and filter the host information;

4) project the number of listings with different price categories.

<u>Query #5 is as follows:</u>

```
SELECT COUNT(b.ind)
FROM
    (SELECT g.ind, g.host_id
     FROM General_listings g
     WHERE g.ind NOT IN
        (SELECT a.ind
         FROM
             (SELECT l.ind
              FROM LongTermRental l
              WHERE date< CURDATE()
              ) a
         )
     ) b
     JOIN Host_information h ON h.host_id=b.host_id
WHERE h.host_response_time='${inputHost_response_time}' &&
h.host_response_rate IS NOT NULL &&
            h.host_response_rate='${inputHost_response_rate}' &&
h.host_acceptance_rate IS NOT NULL
            && h.host_acceptance_rate='${inputHost_acceptance_rate}'
            && h.host_is_superhost=='${inputHost_is_superhost}' &&
h.host_total_listings_count='${inputHost_total_listings_count}';
```

Query #5 is a follow-up of query #4. The difference is that query #5 returns the total number of records which satisfy the screening criteria.

<u>Query #1 is as follows.</u>

```
 SELECT
      SUM(IF(b.price<50,1,0)) as 'UNDER 50',
      SUM(IF(b.price BETWEEN 50 AND 99,1,0)) as '50-99',
      SUM(IF(b.price BETWEEN 100 AND 149,1,0)) as '100-149',
      SUM(IF(b.price BETWEEN 150 AND 199,1,0)) as '150-199',
      SUM(IF(b.price BETWEEN 200 AND 249,1,0)) as '200-249',
      SUM(IF(b.price>=250,1,0)) as 'OVER 250',
      SUM(IF(b.price IS NULL, 1, 0)) as 'Not Filled In (NULL)'
      FROM
          (SELECT g.ind, g.room_type, g.accommodates, g.bedrooms, g.beds,
g.general_price AS price
           FROM General_listings g
           WHERE g.ind NOT IN
           (SELECT a.ind
           FROM
             (SELECT l.ind, AVG(price)
              FROM General_listings g JOIN LongTermRental l ON g.ind = l.ind
              WHERE date< CURDATE()
              GROUP BY l.ind) a)) b
      WHERE b.room_type='${inputRoom_type}' &&
b.accommodates='${inputAccommodates}' &&
            b.bedrooms='${inputBedrooms}' && b.beds='${inputBeds}';
```

Query #1 follows the three steps:

1) identify the listings in the table of "General_listings";

2) exclude the listings which are long-term rentals and calculate the average price before the current date from the table of "LongtermRental";

3) filter the property information;
4) project the number of listings with different price categories.

Query #3 is as follows.

```
SELECT 1.0*
       (SELECT COUNT(b.ind)
        FROM
      (SELECT g.ind, g.room_type, g.accommodates, g.bedrooms, g.beds
       FROM General_listings g
       WHERE g.ind NOT IN
       (SELECT a.ind
       FROM
         (SELECT l.ind
          FROM LongTermRental l
          WHERE date< CURDATE()
          ) a)) b
        WHERE b.room_type='${inputRoom_type}' &&
b.accommodates='${inputAccommodates}' &&
            b.bedrooms='${inputBedrooms}' && b.beds='${inputBeds}')/
       (SELECT COUNT(g.ind)
        FROM General_listings g ) * 100 AS Percentage;
```

Query #3 follows the three steps:
1) identify the listings in the table of "General_listings";
2) exclude the listings which are long-term rentals and calculate the average price before the current date from the table of "LongtermRental";
3) filter the property information;
4) calculate the percentage of the listings satisfying the filter requirement in the total number of listings.

8. Performance Evaluation

Due to page limits, we only use the most complex query #9 as an example to explain our optimization efforts. The average running time of 10 runs for query #9 are as following: original running time: 4s 879ms, optimized query without indexing: 3s 7ms, optimized query with indexing: 2s 221ms. The optimization process is described below.

First, the optimized query avoids nested subqueries by using Common Table Expressions (CTEs) instead. CTEs make the query structure clearer and more maintainable, and they often allow the database optimizer to better optimize the query execution plan. It calculates the rating and per-capita price aggregates directly in CTEs (airbnb_aggregates and hotels_aggregates) instead of using multiple levels of subqueries as in the original query. Indexes are created on 'neighborhood' for both the Airbnbs and Hotels table. Given the Airbnb table is large, this operation alone significantly improves the running time when joining the two tables by neighborhoods since the data for each neighborhood can be retrieved faster. By using indexes, the query generates more optimized execution plans, leading to faster query execution times. Also, the optimized query joins the aggregates in the combined_stats CTE directly, rather than joining multiple subqueries. This simplifies the JOIN operation and can lead to better performance.

Second, the optimized query uses COALESCE to handle NULL values in the ratings columns. This ensures that the sum_ratings and count_ratings computations for Airbnbs and hotels are not affected by missing data. In the original query, each rating component was calculated separately using CASE statements, which could be less efficient and harder to read. And multiple passes were made through the General_listings table to filter short-term Airbnbs and compute their ratings. The optimized query combines these steps into CTEs, resulting in a more efficient single pass through the table. The optimized query reduces the number of calculations required by directly calculating the percentage_price and percapita_price values in the CTEs. This leads to fewer computations and potentially faster query execution.

Overall, the optimized query is more concise, readable, and efficient compared to the original query. Other considerations to improve the query include adding more indexes, using "left join" instead of "not in" to filter listings, and testing other ways to rewrite the current aggregate functions. Since these operations have minuscule effect on the ultimate running time, they are not included.

Besides query #9 described above, we use similar approaches to optimize the query #1-6. The comparison with the average of 10 runs for each query is illustrated in Table 1 below.

Table 1. Summary of Optimization Results of Query #1 ~ #6

Query #	Initial version	Final version	Save time (%)
#1	6955.221	1430.579	79.43%
#2	7401.049	267.707	96.38%
#3	6688.271	232.487	96.52%
#4	8298.98	1376.341	83.42%
#5	6060.603	106.75	98.24%
#6	7482.579	157.752	97.89%

9. Technical Challenges

We summarized the technical challenges we met as follows. First, we experienced the most challenges when creating the spatial comparison map of Airbnb and hotels. First, the map API failed to fetch data due to multiple imports of the Google Maps JavaScript API. Thus, when we changed the specified neighborhood, unexpected errors occurred. To address this, we consolidated code dependent on the API into the window.initMap() function, ensuring it fully loads before accessing the google.maps object. Additionally, we used the useState Hook to make the map variable as a component's state, enabling easy access and modification from other parts of the component. Second, duplicated markers representing listings lead to repeated updates of point coordinates and unexpected overlays when the neighborhood changed. As a solution, we implemented proper cleanup of previous points before making new requests and updated the points in the search button's callback, which prevents overlapping points and maintains accurate and up-to-date data on the map. Third, we experienced incompatibility between the Google Maps API and React's lifecycle. To address this, we utilized the useRef hook in React to create a stable reference and store the map instance, ensuring it remains intact even if the React component re-renders. This approach resolved the problem of losing the Google Maps API's reference to the map DOM element during updates.

Second, we encountered difficulties when setting up the server. Understanding of asynchronous requests is crucial for building a responsive and efficient web application. It took time to comprehend important concepts including requests, responses, async functions, and promises. Furthermore, we need to get familiar with the development toolkits including React and Node.js frameworks in a short period of time. Those frameworks are very powerful, but some tools are not intuitive to beginners.

Third, as described in section of Data, we met different challenges of processing and cleaning the raw datasets. Unmeaningful table and attribute names made it difficult to select and identify the detailed ideas of the website. For example, we didn’t know the listings in the table of “Host_information_distinct” are part of the ones in the table of “General_listings” at the beginning. Until we experienced extremely long time period of importing tables into DataGrip, we checked back the tables in detail and found out this relationship. A statement on the website of insiderbnb gave us the idea why the owner of insiderbnb did this. Meanwhile, this finding gave us creative insight of designing complex queries.

Reference

Dillahunt, T.R., Malone, A.R., 2015. The promise of the sharing economy among disadvantaged communities. Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. ACM, New York, NY, pp. 2285-2294.

Dube, J., Legros, D., 2014. Spatial econometrics and the hedonic pricing model: what about the temporal dimension? Journal of Property Research 31(4), 333-359.

Espinet, J.M., Saez, M., Coenders, G. and Fluviàb, M. (2003). Effect on prices of the attributes of holiday hotels: a hedonic prices approach. Tourism Economics, 9(2), 165-177.

Ganu, G., Elhadad, N., & Marian, A. (2009, June). Beyond the stars: improving rating predictions using review text content. In *WebDB* (Vol. 9, pp. 1-6).

Hung, W.T., Shang, J.K., & Wang, F.C., 2010. Pricing determinants in the hotel industry: Quantile regression analysis. International Journal of Hospitality Management 29(3), 378-384.

Johnson, A., Neuhofer, B., 2017. Airbnb - an exploration of value co-creation experiences in Jamaica. International Journal of Contemporary Hospitality Management 29(90), 2361-2376.

Schamel, G., 2012. Weekend vs midweek stays: modeling hotel room rates in a small market. International Journal of Hospitality Management 31, 113-1118.

Appendix

Appendix A. Summary of Statistics

Attribute	Mean	Deviation	Max	Min
Airbnb				
ind	N/A	N/A	N/A	N/A
host_id	N/A	N/A	N/A	N/A
host_response_time	0.31	0.6697	3	0
host_response_rate	0.96	0.1210	1	0
host_acceptance_rate	0.96	0.2034	1	0
host_is_superhost	0.34	0.4734	1	0
host_total_listings_count	169.61	760.3025	8316	1
neighbourhood_cleansed	N/A	N/A	N/A	N/A
neighbourhood_group_cleansed	N/A	N/A	N/A	N/A
latitude	N/A	N/A	N/A	N/A
longitude	N/A	N/A	N/A	N/A
room_type	0.34	0.5717	3	0
accommodates	3.91	2.7312	16	0

bedrooms	1.79	1.1331	24	1
beds	2.16	1.5839	50	1
price	283.33	1081.1467	99999	0
review_scores_rating	4.70	0.5325	5	0
review_scores_accuracy	4.77	0.3945	5	0
review_scores_cleanliness	4.71	0.4203	5	0
review_scores_checkin	4.84	0.3518	5	0
review_scores_communication	4.83	0.3690	5	0
review_scores_location	4.79	0.3442	5	0
review_scores_value	4.67	0.4227	5	0
Hotels				
hotel_id	N/A	N/A	N/A	N/A
hotel_name	N/A	N/A	N/A	N/A
address	N/A	N/A	N/A	N/A
latitude	N/A	N/A	N/A	N/A
longitude	N/A	N/A	N/A	N/A
cleanness	8.13	1.05	9.90	3.70
service	8.19	0.85	9.90	4.30
amenities	7.49	1.11	9.40	2.00
facilities	7.80	1.11	9.90	3.50
ecofriendly	7.78	1.06	10.00	3.20
price	272.26	247.86	6000.00	26.00
number_people	3.16	1.14	12.00	2.00

Appendix B. Relational Schema

Host_information (host_id, host_response_time, host_response_rate, host_acceptance_rate, host_is_superhost, host_total_listings_count)
PRIMARY KEY host_id
General_listings (ind, host_id, room_type, accommodates, bedrooms, beds, review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_scores_location, review_scores_value, general_price)
PRIMARY KEY ind
FOREIGN KEY host_id REFERENCES Host_information(host_id)

LongTermRental (longterm_listing_id, ind, date, available, price, minimum_nights, maximum_nights)
PRIMARY KEY longterm_listing_id
FOREIGN KEY ind REFERENCES General_listings(id)
Hotels (hotel_id, hotel_name, address, latitude, longitude, cleanness, service, amenities, facilities, ecofriendly, price, number_people)
PRIMARY KEY hotel_id

Appendix C. Routes

ROUTE 1
Functionality: Page Route to Home Page
Request Path: "/"
Request Parameter:
- This route is a page route and does not require any parameters.

Query Parameter:
- This route is a page route and does not have any query parameters.

Response Parameter:
- This route is a page route and does not have any response parameters.

ROUTE 2
Functionality: Page Route to Property Features Page
Request Path: "/property_features"
Request Parameter:
- This route is a page route and does not require any parameters.

Query Parameter:
- This route is a page route and does not have any query parameters.

Response Parameter:
- This route is a page route and does not have any response parameters.

ROUTE 3
Functionality: Page Route to Host Information Page
Request Path: "/host_info"
Request Parameter:
- This route is a page route and does not require any parameters.

Query Parameter:
- This route is a page route and does not have any query parameters.

Response Parameter:
- This route is a page route and does not have any response parameters.

ROUTE 4
Functionality: Page Route to Hotel Page
Request Path: "/hotels"

Request Parameter:

- This route is a page route and does not require any parameters.

Query Parameter:

- This route is a page route and does not have any query parameters.

Response Parameter:

- This route is a page route and does not have any response parameters.

ROUTE 5

Functionality: Page Route to Top Listing Page

Request Path: “/top_listing”

Request Parameter:

- This route is a page route and does not require any parameters.

Query Parameter:

- This route is a page route and does not have any query parameters.

Response Parameter:

- This route is a page route and does not have any response parameters.

ROUTE 6

Functionality: The route uses a GET() method to retrieve numbers of listings in different price ranges, filtered by input property features.

Request Path: “/search_features”

Request Parameter:

- roomType[String]: type of room
- accomodates[String]: a range for the number of accommodates

- bedrooms[String]: a range for the number of bedrooms
- beds[String]: a range for the number of beds

Query Parameter:

- roomType[int]: index of room type in the table
- accomodatesLow[int]: the minimum number of accommodates
- accomodatesHigh[int]: the maximum number of accommodates
- bedroomsLow[int]: the minimum number of bedrooms
- bedroomsLow[int]: the maximum number of bedrooms
- bedsLow[int]: the minimum number of beds
- bedsHigh[int]: the maximum number of beds

Response Parameter:

- data[[List[int]]: numbers of listings in different price ranges filtered by input property features.

ROUTE 7

Functionality: The route uses a GET() method and returns the total number of retrieved listings filtered by input property features.

Request Path: “/search_features_count”

Request Parameter:

- roomType[String]: type of room
- accomodates[String]: a range for the number of accommodates
- bedrooms[String]: a range for the number of bedrooms
- beds[String]: a range for the number of beds

Query Parameter:

- roomType[int]: index of room type in the table
- accomodatesLow[int]: the minimum number of accommodates
- accomodatesHigh[int]: the maximum number of accommodates
- bedroomsLow[int]: the minimum number of bedrooms
- bedroomsLow[int]: the maximum number of bedrooms
- bedsLow[int]: the minimum number of beds
- bedsHigh[int]: the maximum number of beds

Response Parameter:

- count[int]: the total number of retrieved listings filtered by input property features.

ROUTE 8

Functionality: The route uses a GET() method and returns the percentage of the total listing that is retrieved after filtering the data with input property features.

Request Path: "/search_features_percentage"

Request Parameter:

- roomType[String]: type of room
- accomodates[String]: a range for the number of accommodates
- bedrooms[String]: a range for the number of bedrooms
- beds[String]: a range for the number of beds

Query Parameter:

- roomType[int]: index of room type in the table
- accomodatesLow[int]: the minimum number of accommodates
- accomodatesHigh[int]: the maximum number of accommodates
- bedroomsLow[int]: the minimum number of bedrooms
- bedroomsLow[int]: the maximum number of bedrooms
- bedsLow[int]: the minimum number of beds
- bedsHigh[int]: the maximum number of beds

Response Parameter:

- percentage[float]: the percentage of the total listing that is retrieved after filtering the data with input property features.

ROUTE 9

Functionality: The route uses a GET() method to retrieve numbers of listings in different price ranges
filtered by input host information.

Request Path: "/search_host_info"

Request Parameter:

- responseTime[String]: response time of a particular host

- responseRate[String]: response rate of a particular host
- acceptanceRate[String]: acceptance rate of a particular host
- superhost[String]: whether the host is a superhost
- totalListing[String]: a range of total listing a particular host has

Query Parameter:

- responseTimeLow[int]: the minimum response time of a particular host
- responseTimeHigh[int]: the maximum response time of a particular host
- responseRateLow[float]: the minimum response rate of a particular host
- responseRateHigh[float]: the maximum response rate of a particular host
- acceptanceRateLow[float]: the minimum acceptance rate of a particular host
- acceptanceRateHigh[float]: the maximum acceptance rate of a particular host
- superhost[int]: whether the host is a superhost [1/0]
- totalListingLow[int]: the minimum total listing a particular host has
- totalListingHigh[int]: the maximum total listing a particular host has

Response Parameter:

- data[[List[int]]: numbers of listings in different price ranges filtered by input host information

ROUTE 10

Functionality: The route uses a GET() method and returns the total number of retrieved listings filtered by input host information.
Request Path: "/search_host_info_count"
Request Parameter:

- responseTime[String]: response time of a particular host
- responseRate[String]: response rate of a particular host
- acceptanceRate[String]: acceptance rate of a particular host
- superhost[String]: whether the host is a superhost
- totalListing[String]: a range of total listing a particular host has

Query Parameter:

- responseTimeLow[int]: the minimum response time of a particular host
- responseTimeHigh[int]: the maximum response time of a particular host
- responseRateLow[float]: the minimum response rate of a particular host
- responseRateHigh[float]: the maximum response rate of a particular host
- acceptanceRateLow[float]: the minimum acceptance rate of a particular host
- acceptanceRateHigh[float]: the maximum acceptance rate of a particular host
- superhost[int]: whether the host is a superhost [1/0]
- totalListingLow[int]: the minimum total listing a particular host has
- totalListingHigh[int]: the maximum total listing a particular host has

Response Parameter:

- count[int]: the total number of retrieved listings filtered by input host information

ROUTE 11

Functionality: The route uses a GET() method and returns the percentage of the total listing that is retrieved after filtering the data with input host information.
Request Path: "/search_host_info_percentage"

Request Parameter:

- responseTime[String]: response time of a particular host
- responseRate[String]: response rate of a particular host
- acceptanceRate[String]: acceptance rate of a particular host
- superhost[String]: whether the host is a superhost
- totalListing[String]: a range of total listing a particular host has

Query Parameter:

- responseTimeLow[int]: the minimum response time of a particular host
- responseTimeHigh[int]: the maximum response time of a particular host
- responseRateLow[float]: the minimum response rate of a particular host
- responseRateHigh[float]: the maximum response rate of a particular host
- acceptanceRateLow[float]: the minimum acceptance rate of a particular host
- acceptanceRateHigh[float]: the maximum acceptance rate of a particular host
- superhost[int]: whether the host is a superhost [1/0]
- totalListingLow[int]: the minimum total listing a particular host has.
- totalListingHigh[int]: the maximum total listing a particular host has.

Response Parameter:

- percentage[float]: the percentage of the total listing that is retrieved after filtering the data with input host information.

<u>ROUTE 12</u>

Functionality: The route uses a GET() method to retrieve top rankings for the input review type
Request Path: "/top_ranking"
Request Parameter:

- reviewType[String]: type of review that will be used to sort the available listings
- listingSize[int]: number of top listings to return from the query

Query Parameter:

- reviewType[int]: the index of the review type
- listingSize[int]: number of top listings to return from the query

Response Parameter:

- data[list[Object]]: a list of top listings sorted by the input review type with a size of input listing size

<u>ROUTE 13</u>

Functionality: The route uses a GET() method to retrieve information for a particular listing with listing_id
Request Path: "/top_listing/:listing_id"
Request Parameter:

- Param/type/description
- Listing_id[int]: a unique listing id that can be used to retrieve the associated listing information

Query Parameter:

- Listing_id[int]: a unique listing id that can be used to retrieve the associated listing information

Response Parameter:

- Param/type/description
- Listing[Object]: a listing object that contains information about the listing location, neighborhood, price, and scores for various review metrics.

<u>ROUTE 14</u>

Functionality: The route uses a GET() method to retrieve both Airbnb and hotel data in the same neighborhood and perform cross-analysis on the price and review scores before providing recommendations on the accommodation.
Request Path: “/rec”
Request Parameter:

- Neighborhood[String]: the neighborhood that the user would like to accommodate

Query Parameter:

- Neighborhood[String]: the neighborhood that the user would like to accommodate

Response Parameter:

- better_rating[String]: recommendation for a better rating between ‘Airbnb’ and ‘Hotel’
- better_price[String]: recommendation for a better price between ‘Airbnb’ and ‘Hotel’

<u>ROUTE 15</u>

Functionality: The route uses a GET() method to retrieve the middle point for maps by neighborhood,
used for mapping
Request Path: “/midpoint”
Request Parameter:

- Neighborhood[String]: the neighborhood that the user would like to accommodate

Query Parameter:

- Neighborhood[String]: the neighborhood that the user would like to accommodate

Response Parameter:

- neighborhood[String]: recommendation for a better rating between ‘Airbnb’ and ‘Hotel’
- avg_latitude[Float]: average latitude of the Airbnb listings in the neighborhood
- avg_longitude[Float]: average longitude of the Airbnb listings in the neighborhood

<u>ROUTE 16</u>

Functionality: The route uses a GET() method to get coordinates for the airbnb listings for the given
neighborhood
Request Path: “/airbnb_neighborhood”
Request Parameter:

- Neighborhood[String]: the neighborhood that the user would like to accommodate

Query Parameter:

- Neighborhood[String]: the neighborhood that the user would like to accommodate

Response Parameter:

- Airbnb_id[String]: recommendation for a better rating between ‘Airbnb’ and ‘Hotel’

- latitude[Float]: latitudes of the Airbnb listings in the neighborhood
- longitude[Float]: longitudes of the Airbnb listings in the neighborhood

<u>ROUTE 17</u>
Functionality: get coordinates for the hotels listings in the given neighborhood
Request Path: “/hotels_neighborhood”
Request Parameter:

- Neighborhood[String]: the neighborhood that the user would like to accommodate

Query Parameter:

- Neighborhood[String]: the neighborhood that the user would like to accommodate

Response Parameter:

- Hotel_id[String]: IDs for the hotels in the neighborhood
- latitude[Float]: latitudes of the Airbnb listings in the neighborhood
- longitude[Float]: longitudes of the Airbnb listings in the neighborhood

<u>ROUTE 18</u>
Functionality: get the names of all distinct neighborhood
Request Path: “/neighborhoods”
Request Parameter: None
Query Parameter: None
Response Parameter:

- Neighborhood[String]: Distinct neighborhood from the Airbnb table

Appendix D. E-R Diagram

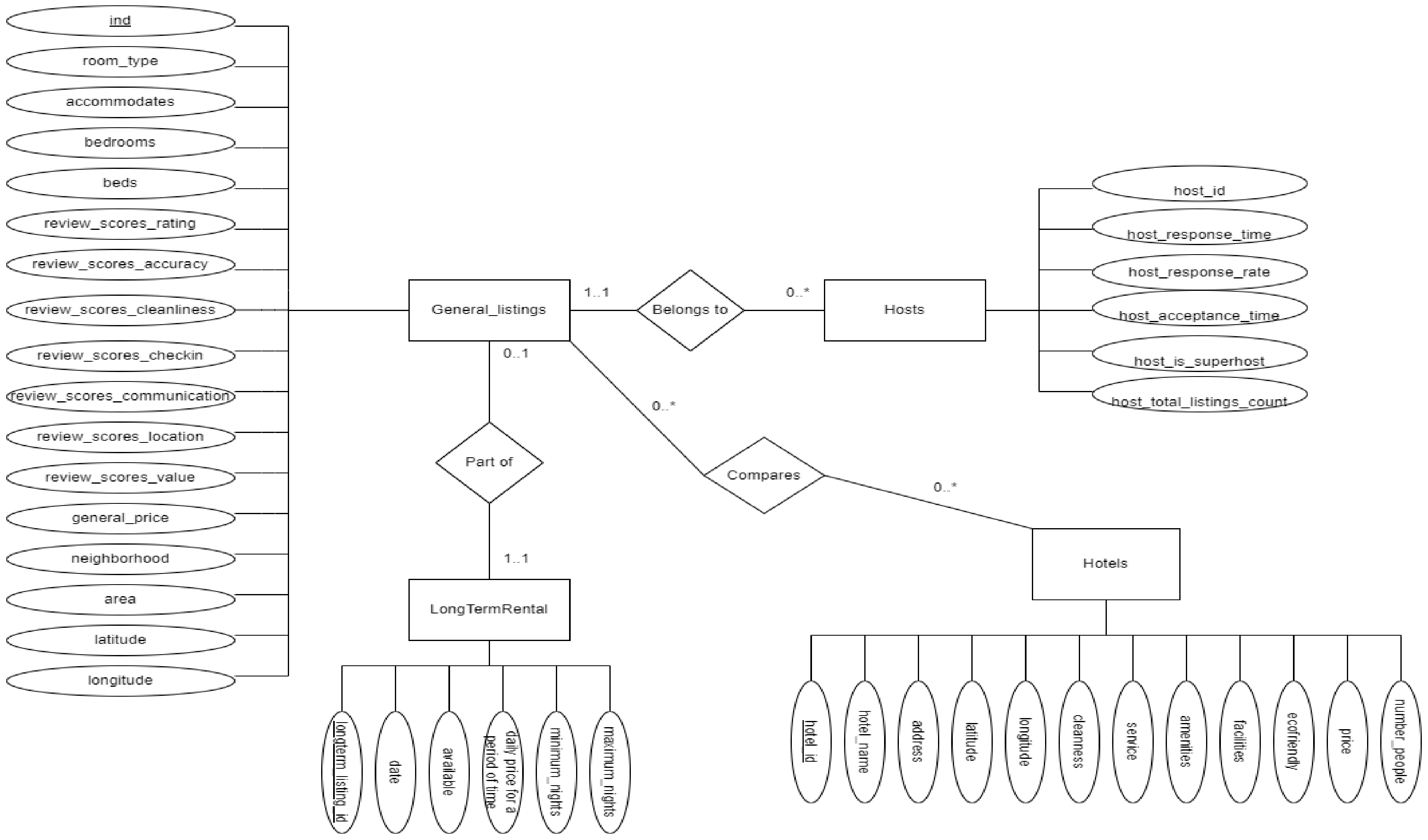

Figure 1. E-R Diagram