# CM 3010 Databases and Advanced Data Techniques - Midterm Coursework 2024

## Introduction

Travelers in Europe are increasingly opting for Airbnb and other short-term rentals over traditional accommodations. According to AirDNA, the short-term rental market experienced record-breaking demand in the first half of 2023, with visitors spending over 177.8 million nights—an impressive 20% increase from the previous year (Vernon, 2023)[1]. Revenue also soared, reaching $36.1 billion, marking a staggering 40% year-on-year growth. Compared to pre-pandemic figures, there has been an 18.6% rise in demand, a 43.4% increase in average daily rates, and a 70% rise in revenue. The data highlights strong growth in Europe's short-term rental markets, presenting opportunities both in emerging tourist destinations and in investments outside crowded urban centers. As Airbnb solidifies its position as a major player in the hospitality industry, understanding guest demand is crucial in this market for achieving success in the competitive landscape.

The scope of this report is defined by the following parameters:

- **Time Frame:** The analysis and reporting will be confined to the duration of the academic term.
- **Resources:** The study will utilize only publicly available real data, excluding any confidential or artificially generated information.
- **Data Analysis:** Focus on the provided dataset on Airbnb listings in Amsterdam, including information on guest reviews, pricing, and listings attributes.
- **Research Questions:** Focus on factors influencing guest demand and satisfaction in the competitive Airbnb market in Amsterdam.
- **Methodology:** Data analysis will be conducted exclusively using SQL queries.

A well-defined scope of study is crucial as it clearly defines the boundaries and focus of the research, ensuring that efforts are concentrated on relevant areas, ultimately leading to more precise and meaningful results.

*Shareable Lab Link on Cousera*

## Stage 1: Find and Critique a Dataset

This report investigates an Airbnb dataset from Zenodo, focusing on guest reviews, pricing, and listings attributes in Amsterdam. The aim is to assess the dataset's quality, detail, documentation, and overall suitability for analyzing factors influencing guest demand and satisfaction in the Airbnb market.

### Data Source and Accessibility

The study utilizes the datasets found on Kaggle[2], which are derived from Zenodo[3], a platform renowned for its extensive repository of public data resources. The datasets encompass ten countries in Europe, each having separate datasets for weekdays and weekends. However, this study only focuses on Amsterdam weekdays and weekends datasets.

### Data Assessment

To ensure the dataset's suitability for addressing the research questions, the following criteria were evaluated:

- **Quality and Detail:** The datasets contain comprehensive information on various aspects of Airbnb listings in Amsterdam, such as room type, cleanliness rating, guest satisfaction, and distance from the city center, for both weekdays and weekends. They are no mismatched or missing values, ensuring data reliability without significant need for data cleaning and preprocessing. However, four columns (attr_index, attr_index_norm, rest_index, and rest_index_norm) are undefined and will not be used in the analysis.

- **Documentation and Use:** The datasets provide granular details about each listing, including room type, pricing, and guest reviews, which are suitable for gaining insights into many aspects of Airbnb listings across Europe, such as general trends in prices and spatial econometric analysis. However, some preprocessing may still be necessary to handle outliers.

- **Interrelation:** The dataset features well-defined relationships between different variables, such as the correlation between correlation between guest reviews and pricing, or between listing attributes and guest satisfaction. This interrelation is essential for conducting comprehensive analyses

- **Discoverability and Term of Use:** The datasets are easily accessible through Kaggle or Zenodo, easier for data acquisition and preparation. They are licensed under the CC0 1.0 Universal (CC0 1.0) Public Domain Dedication, allowing for unrestricted use in research and analysis.

## Interest and Research Questions

According to the report from Hostaway, 'satisfied guests lead to repeat bookings and positive reviews, which, in turn, attract more guests to your property. And with more guests comes a higher occupancy rate, and ultimately, higher profits.' (Hostaway, n.d.)[4]. It is interesting to investigate the factors that influence guest satisfaction and demand in Amsterdam's Airbnb market.

Based on the dataset' characteristics, the following SQL-based research questions are proposed:

1) **Establish a Foundational Understanding of the Airbnb Market Composition in Amsterdam:** Understand the total number of listings and the distribution of different room types in Amsterdam.
2) **Assess How Superhost Status May Affect Pricing Strategies:** Understandhow Superhost status influences pricing and competitive dynamics in the market.
3) **Factors Influencing Guest Overall Satisfaction in Amsterdam's Airbnb Market:** This broader question seeks to identify the key factors that affect guest satisfaction. Sub-questions for clarity include:
    a) **Superhost Status on Guest Satisfaction:** Explore the impact of Superhost status on guest satisfaction, a key metric for guest demand.
    b) **Listing Size and Guest Satisfaction:** Examine whether the size of the listing (person capacity) correlates with higher satisfaction scores, which could influence listing management strategies.
    c) **Pricing Strategies and Day Types on Satisfaction:** Investigate if pricing strategies related to day types (weekdays vs. weekends) affect guest satisfaction.

By analyzing these research questions, it will provide insights into the dynamics of Amsterdam's Airbnb market, highlighting factors that influence pricing and guest satisfaction. This understanding can help hosts optimize their listings and improve guest experiences.

# Stage 2. Model your data

This stage outlines the data modeling process for the dataset database, including the design of an Entity-Relationship Diagram (ERD) which is then translated into a relational schema. This step is crucial for ensuring that the database structure aligns with the analysis requirements.

## Entity-Relationship Diagram (ERD)

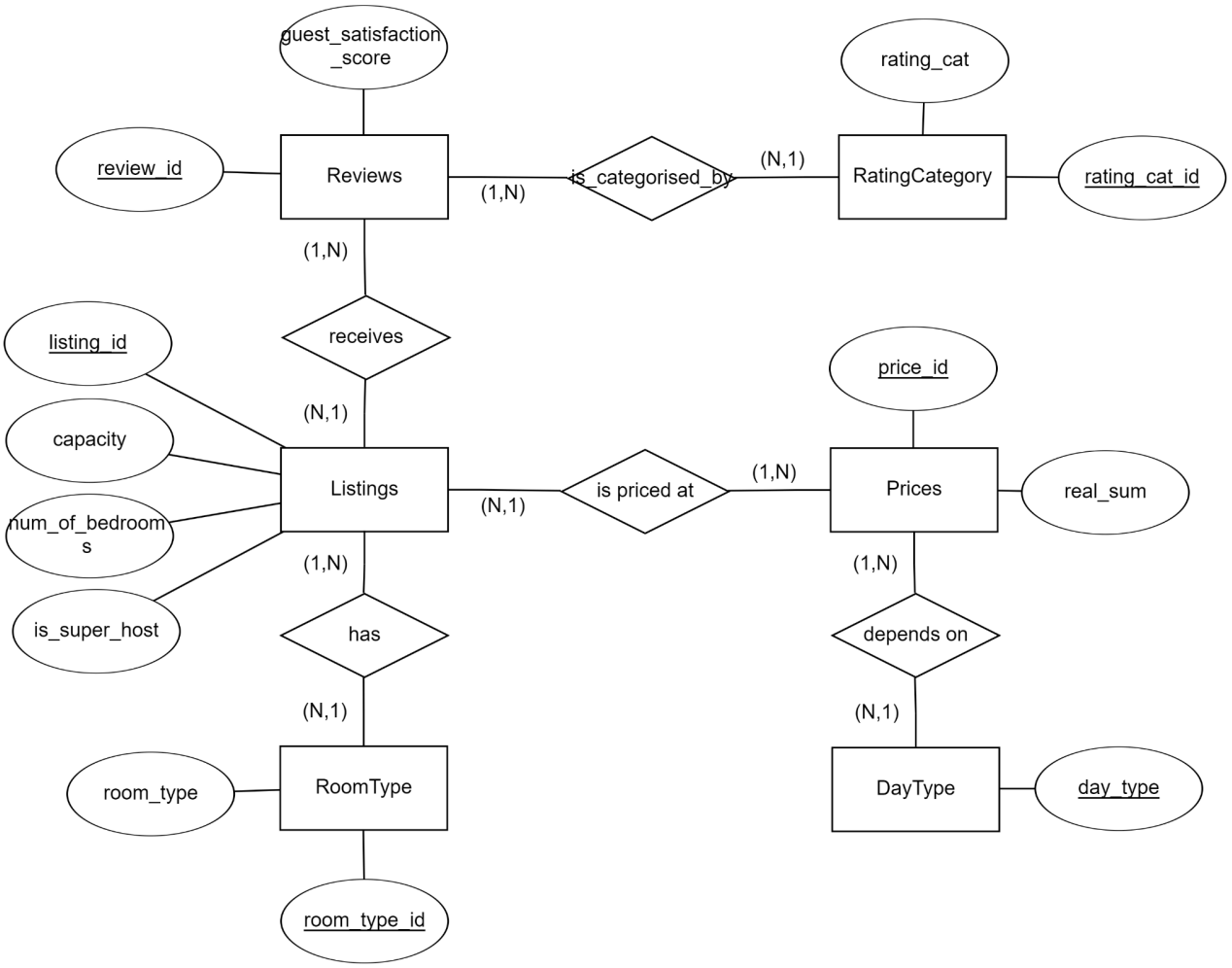



*Figure 1: ERD*

The ERD was created using the ERDPlus platform. The diagram illustrates six entities: 'Listings', 'Prices', 'Reviews', 'RoomType', 'RatingCategory', and 'DayType', along with their interrelationships. Each entity is defined with its attributes and the relationships between entities are clearly mapped out to provide a comprehensive overview of the database structure.

## Relational Schema

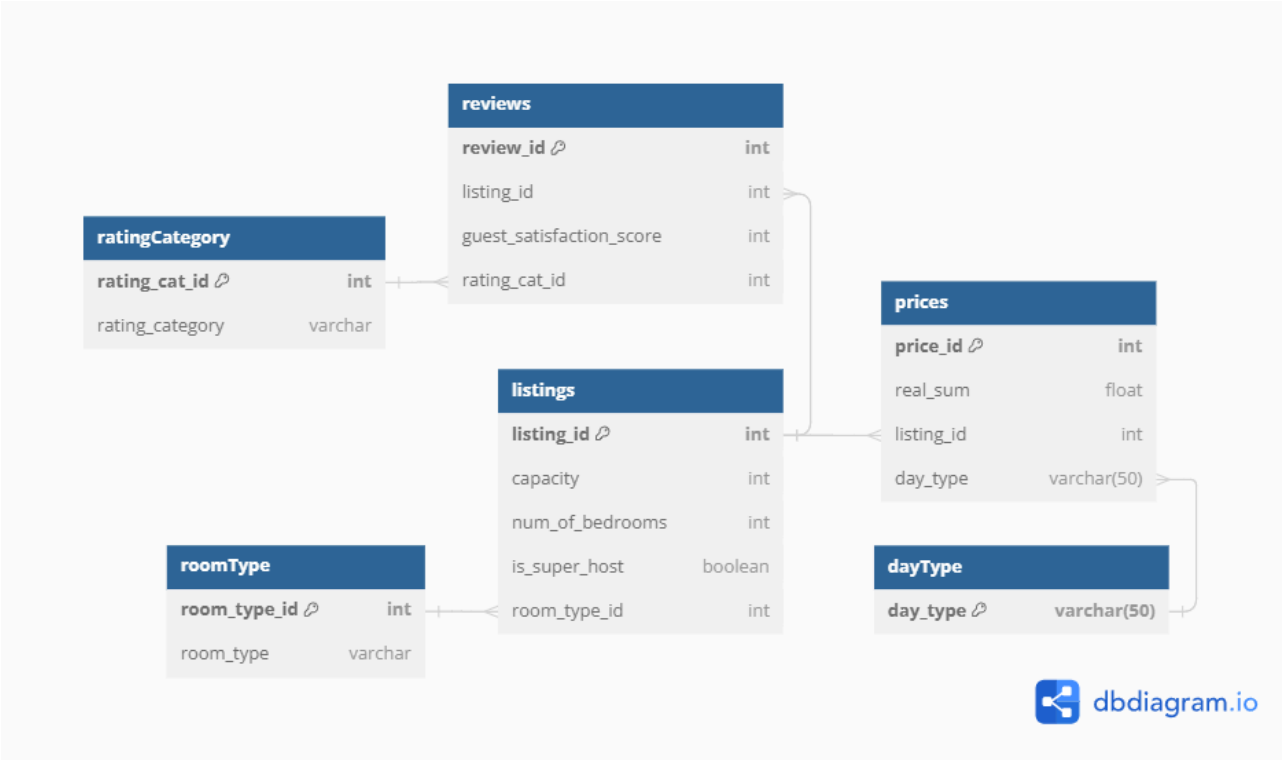



*Figure 2: Relational Schema*

To effectively understand the attributes of the tables in the database, the ER diagram was translated into a relational schema using the dbdiagram.io platform. The schema is a detailed blueprint of the database which defines primary keys and foreign keys, ensuring referential integrity and proper relational structure.

## Database Normalization Analysis

My database schema design focuses on achieving robust and efficient data management by adhering to at least the Third Normal Form (3NF), with an additional aim to meet the Boyce-Codd Normal Form (BCNF). Here's how each normalization form is applied:

1) **First Normal Form(1NF):** All tables ('listings', 'prices', 'reviews', 'roomType', 'dayType', 'reviewCategory') contain primary keys('listing_id','price_id', 'review_id' , 'room_type_id', 'day_type', 'review_cat_id') to uniquely identify each row. Each column is restricted to a single data type, with no multi-valued fields or nested records, ensuring full compliance with 1NF.

2) **Second Normal Form(2NF):** Each table is in 2NF because they are in 1NF and every non-key attribute in each table fully depends on the entire primary key. For example, in the 'Reviews' table, attributes like 'guest_satisfaction_score', 'price_id', and 'rating_cat_id' depend solely on 'review_id', ensuring no partial dependencies.

3) **Third Normal Form (3NF):** Each table is in 2NF, and there are no transitive dependencies between the non-key attributes and the primary key, ensuring that all non-key attributes are dependent only on the primary key.
   To prevent transitive dependencies, attributes like 'room_type' were separated into their own 'roomType' table. Thus, attributes like 'capacity' and 'num_of_bedrooms' in the 'listings' table are dependent solely on 'listing_id'.

4) **Boyce-Codd Normal Form(BCNF):** Every table meets BCNF requirements because every determinant is a superkey, which is a set of attributes that uniquely identifies each row. For each table in the diagram, BCNF is met by the following explanation:

| **Table** | **Primary Key** | **Attributes** | **Functional Dependencies** |
|---|---|---|---|
| Listings | listing_id | capacity, num_of_bedrooms, is_super_host, room_type_id | listing_id → capacity<br>listing_id → num_of_bedrooms<br>listing_id → is_super_host<br>listing_id → room_type_id |
| Reviews | review_id | guest_satisfaction_score, listing_id, price_id, rating_cat_id | review_id → guest_satisfaction_score<br>review_id → listing_id<br>review_id → price_id<br>review_id → rating_cat_id |
| Prices | price_id | real_sum, listing_id, day_type | price_id → real_sum<br>price_id → listing_id<br>price_id → day_type |
| RoomType | room_type_id | room_type | room_type_id → (none) |
| DayType | day_type | - | day_type → (none) |
| RatingCategory | rating_cat_id | rating_cat | rating_cat_id → rating_cat |

The database schema design adheres to these normalization forms, enhancing data integrity and efficiency, prevents data redundancy, and avoids common modification anomalies, thereby ensuring the database is robust and efficient.

# Stage 3. Create the database

In this stage, we construct the 'AMS_Airbnb_Pricing' database, adhering to best practices in database design and management. This involves a Extract, Transform, Load (ETL) process to ensure the data is consistent, reliable, and accessible for analysis and reporting.

## ETL Steps with Specific Tasks

1) **Extract (E)**
   Before creating the database and inserting the data, cleaning and preprocessing are essential to ensure the extracted data is clean and usable. The entire data cleaning and preprocessing steps are accessible via the following link: https://drive.google.com/file/d/1w4TrGV6q6dcJUMdieKoEYvkOLWbeAIcm/view?usp=drive_link
   The steps taken include:

- **Data Concatenate:** Merge two datasets into one.

Add 'day_type' column to distinguish between weekdays and weekends.

```
[20] ams_weekdays['day_type'] = 'weekdays'
     ams_weekdays.head()  # Further inspect data after adding 'day_type' column
```

Show hidden output

```
[21] ams_weekends['day_type'] = 'weekends'
     ams_weekends.head()  # Further inspect data after adding 'day_type' column
```

Show hidden output

Merge the two datasets.

```
[22] ams_df = pd.concat([ams_weekdays, ams_weekends] , ignore_index = True)
```

*Figure 3: Code Snippet for Data Concatenate*

- **Drop Columns:** Remove columns that are unnamed, undefined, or irrelevant to the analysis.

Drop unnamed index column in the dataset.

```
[26] ams_df.drop(ams_df.columns[0], axis=1, inplace=True)
```

Drop undefined columns.

```
[27] undefined_features = ['attr_index', 'attr_index_norm', 'rest_index', 'rest_index_norm']
     ams_df.drop(undefined_features, axis=1, inplace=True)
```

Drop columns that are irrelevant to the research questions.

```
[28] irrelevant_features = ['room_shared', 'room_private', 'multi', 'biz', 'cleanliness_rating', 'dist', 'metro_dist', 'lng','lat']
     ams_df.drop(irrelevant_features, axis=1, inplace=True)
```

Check the df structure after columns were dropped.

```
[29] ams_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2080 entries, 0 to 2079
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   realSum                   2080 non-null   float64
 1   room_type                 2080 non-null   object
 2   person_capacity           2080 non-null   float64
 3   host_is_superhost         2080 non-null   bool
 4   guest_satisfaction_overall  2080 non-null   float64
 5   bedrooms                  2080 non-null   int64
 6   day_type                  2080 non-null   object
dtypes: bool(1), float64(3), int64(1), object(2)
memory usage: 99.7+ KB
```

*Figure 4: Code Snippet for Drop Columns*

- **Detects and Handles Missing Values:** The dataset does not contain any missing values, thus no action is needed.

```
[30] # Calculate the number of missing values in each colum
     missing_val = ams_df.isnull().sum()

     print("Missing values per column:")
     print(missing_val)
```

```
Missing values per column:
realSum                     0
room_type                   0
person_capacity             0
host_is_superhost           0
guest_satisfaction_overall  0
bedrooms                    0
day_type                    0
dtype: int64
```

*Figure 5: Code Snippet for Detecting and Handling Missing Values*

- **Detect and Handle the Outliers:** Detect outliers and decide whether to retain or remove them based on the nature of the data and analysis requirements.
  Overview the data types among the columns before checking the outlier:

```
[31] ams_df.dtypes

realSum                       float64
room_type                      object
person_capacity               float64
host_is_superhost                bool
guest_satisfaction_overall    float64
bedrooms                        int64
day_type                       object
dtype: object
```

*Figure 6: Code Snippet for the data types of the columns*

1. **Boolean Attributes: 'host_is_superhost'**
   Boolean attributes typically do not have "outliers" in the conventional sense. Therefore, no outlier detection is performed on these columns as they represent categorical data.

2. **Categorical Attributes: 'room_type', 'day_type'**
   We review unique values and their frequency to identify any potential outliers.

```
[32] categorical_cols = ["room_type", "day_type"]

     # Get unique counts for each categorical attribute to check for rare categories which might be outliers
     unique_values = ams_df[categorical_cols].nunique()

     # frequency counts for each unique combination of values
     value_counts = ams_df[categorical_cols].value_counts()

     # Display the unique values and their frequencies
     print("Unique values:\n " ,unique_values)
     print("\nValue counts:\n", value_counts)
```

```
Unique values:
  room_type    3
day_type     2
dtype: int64

Value counts:
 room_type        day_type
Entire home/apt  weekends    588
Private room     weekdays    559
Entire home/apt  weekdays    538
Private room     weekends    385
Shared room      weekdays      6
                 weekends      4
Name: count, dtype: int64
```

*Figure 7: Code Snippet for the Outlier Detection of Categorical Attributes*

- Observation: 'Entire home/apt' and 'Private room' dominate the dataset, 'Shared room' appears less frequently, particularly in weekends bookings.
- Action: 'Shared room' is retained to ensure a comprehensive analysis, allowing us to assess pricing strategies across all listing types.

3. **Numeric Attributes: 'realSum', 'person_capacity', 'guest_satisfaction_overall', 'bedrooms'**
Numeric attributes require statistical methods to detect outliers. We use box plots and Z-scores for this purpose.

**(1) 'realSum'**

```
[35] # set the size of plot
     plt.figure(figsize=(3, 5))

     # boxplot for 'realSum'
     plt.boxplot(ams_df['realSum'])
     plt.title('Box Plot of realSum')
     plt.ylabel('RealSum')
     plt.show()
```

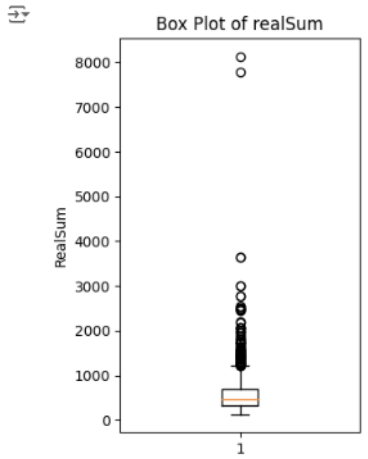



*Figure 8: Code Snippet for Box Plot of 'realSum'*

- **Observation:** The box plot shows a median slightly below 2000 with extreme outliers reaching up to 8000.

- **Action:** Handling Outliers Using Z-score.

```
[36] from scipy.stats import zscore

     # apply Z-score method
     ams_df["z_score"] = zscore(ams_df["realSum"])

     # boxplot for Z-score
     plt.figure(figsize=(3, 5))
     plt.boxplot(ams_df['z_score'])
     plt.title('Box Plot of Z-scores')
     plt.ylabel('z-score')
     plt.show()
```

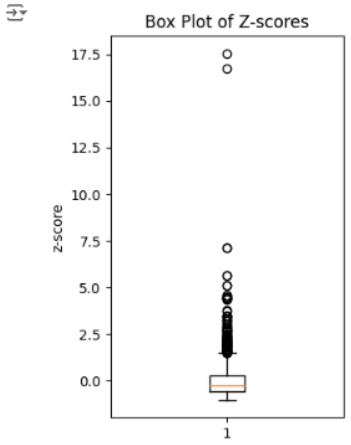



*Figure 9: Code Snippet for Box Plot of Z-score*

- **Z-score Analysis:** Majority of data points have Z-scores between -2 and 2, with extreme outliers up to 12.

- **Action:** Apply a Z-score threshold between -4 and 4 to exclude the most extreme outliers.

```
[37] # calculate Z-scores
     ams_df['z_score'] = zscore(ams_df['realSum'])

     # define the threshold for outliers detection
     threshold = 4
     outliers = ams_df[np.abs(ams_df['z_score']) > threshold]
     print("Number of Outliers:", outliers.shape[0])

     # filter the dataset to exclude outliers based on Z-score threshold of 4
     ams_df2 = ams_df[np.abs(ams_df['z_score']) < threshold]

     Number of Outliers: 16
```

Verify the size of the original and filtered dataset.

```
[38] print("Original DataFrame size:", ams_df.shape)
     print("Filtered DataFrame size:", ams_df2.shape)

     Original DataFrame size: (2080, 8)
     Filtered DataFrame size: (2064, 8)
```

*Figure 10: Code Snippet for Z-score Calculation*

The dataset is reduced from 2080 to 2074 entries, removing 6 extreme outliers.

### (2) 'guest_satisfaction_overall'

```
[42] # box plot for 'guest_satisfaction_overall'
     sns.boxplot(x=ams_df2['guest_satisfaction_overall'])
     plt.title('Box Plot of guest satisfaction overall')
     plt.show()
```

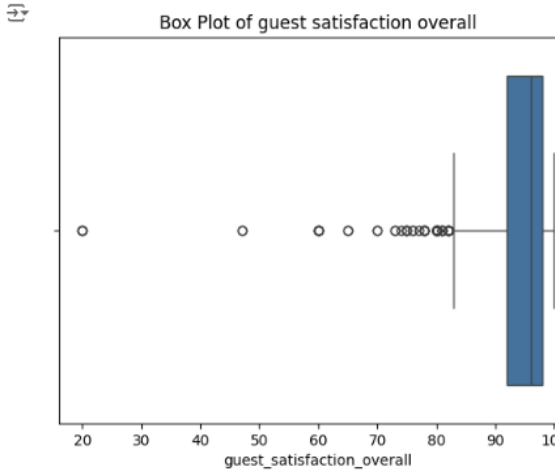



*Figure 11: Code Snippet for Box Plot of 'guest_satisfaction_overall'*

- **Observation:** Most data points are clustered between ratings of 80 and 100, with outliers below 50.

- **Action:** Retain outliers to provide insights into factors negatively influencing guest satisfaction, maintaining dataset integrity.

### (3) 'person_capacity' and 'bedrooms'

```
[43] # box plot for 'person_capacity'
     plt.figure(figsize=(10, 5))
     plt.subplot(1, 2, 1)
     sns.boxplot(x=ams_df2['person_capacity'])
     plt.title('Box Plot of Person Capacity')

     # box plot for 'bedrooms'
     plt.subplot(1, 2, 2)
     sns.boxplot(x=ams_df2['bedrooms'])
     plt.title('Box Plot of Bedrooms' )
     plt.show()
```

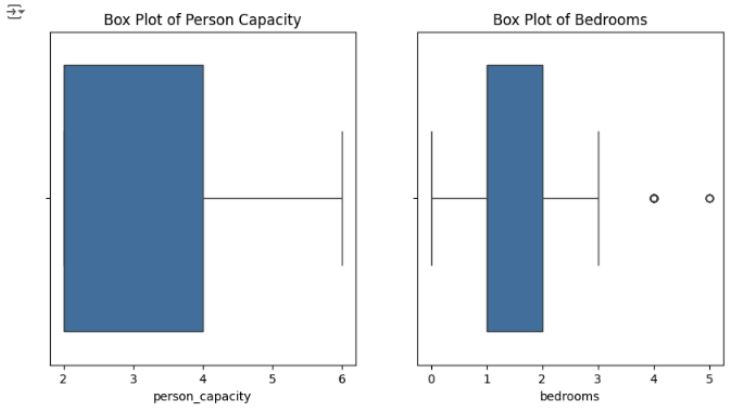



*Figure 12: Code Snippet for Box Plots of 'person_capacity' and 'bedrooms'*

**- Observation:**

- 'person_capacity': No significant outliers, indicating consistent data.
- 'bedrooms': Outliers for listings with more than four bedrooms are retained as they reflect legitimate market variations.

By systematically identifying and handling outliers, we enhance the reliability of our insights into Airbnb pricing strategies and guest satisfaction in Amsterdam, ensuring our research accurately reflects the real-world variability of the market.

- **Data Transformation:**

Categorize the values in the 'guest_satisfaction_overall' column into predefined categories and add the results to a new column named 'rating_category'.

**Categorizing numerical data**

Categorizing the score of **'guest_satisfaction_overall'** helps to simplify the analysis and interpretation of satisfaction levels. To handle the categorization of 'guest_satisfaction_overall', converting continuous numerical scores into defined categories (e.g., 'Very Dissatisfied', 'Dissatisfied', 'Neutral', 'Satisfied', 'Very Satisfied'), it becomes easier to identify patterns, trends, and relationships within the data.

```
[44] # define bins and labels for the satisfaction scores
     bins = [0, 20, 40, 60, 80, 100]
     labels = ['Very Dissatisfied', 'Dissatisfied', 'Neutral', 'Satisfied', 'Very Satisfied']

     # create 'category' column in the dataset
     ams_df2['rating_category'] = pd.cut(ams_df2['guest_satisfaction_overall'], bins=bins, labels=labels, right=True)
```

*Figure 13: Code Snippet for Categorical Numerical Data*

Convert data types to ensure compatibility with MySQL database storage.

**Converting data types**

```
[45] # view the data type of each attribute
     ams_df2.dtypes
```

```
realSum                      float64
room_type                     object
person_capacity              float64
host_is_superhost               bool
guest_satisfaction_overall   float64
bedrooms                       int64
day_type                      object
rating_category             category
dtype: object
```

```
[46] # convert float data type into integer
     ams_df2['person_capacity'] = ams_df2['person_capacity'].astype(int)
     ams_df2['guest_satisfaction_overall'] = ams_df2['guest_satisfaction_overall'].astype(int)

     # since mysql does not suppert boolean data type,
     # convert boolean True/False to 1/0
     ams_df2['host_is_superhost'] = ams_df2['host_is_superhost'].astype(int)

     # round float to 2 decimal points
     ams_df2['realSum'] = ams_df2['realSum'].round(2)

     # convert category data type into object
     ams_df2['rating_category'] = ams_df2['rating_category'].astype(object)
```

```
[47] # check data type after the conversion
     ams_df2.dtypes
```

```
realSum                     float64
room_type                    object
person_capacity               int64
host_is_superhost             int64
guest_satisfaction_overall    int64
bedrooms                      int64
day_type                     object
rating_category              object
dtype: object
```

*Figure 14: Code Snippet for Convert Data Types*

- **Save the Filtered Dataset:** Retain only the necessary data for loading and insertion.

```
[50] # save the clean, filtered data to a csv file for data loading and data insertion
     ams_df2.to_csv(f"{DATA_PATH}/amsterdam-listings-cleaned.csv", index=False)
```

*Figure 15: Code Snippet for Save Filtered Dataset*

**2) Transform (F)**

The transformation step involves the creation of the database, defining table schemas, specifying data types, and setting up user access rights to maintain data security and integrity.

- **Database Creation and User Configuration**

  Create the database and configure user access to ensure data security and efficient access.

```
1  -- drop existing database (if it exists) and create new database
2  DROP DATABASE IF EXISTS AMS_Airbnb_Pricing;
3  CREATE DATABASE AMS_Airbnb_Pricing;
4
5  -- drop existing user (if it exists) and create new user
6  DROP USER IF EXISTS 'studb'@'%';
7  CREATE USER 'studb'@'%' IDENTIFIED WITH mysql_native_password BY '10233189';
8
9  -- grant user permission to access the project
10 GRANT ALL ON AMS_Airbnb_Pricing.* TO 'studb'@'%';
11
12 -- grant additional permissions to run mysql on Colab
13 GRANT SELECT ON mysql.* TO 'studb'@'%';
```

*Figure 16: SQL script for Database Creation and User Configuration*

- **Table Structure**
  Create table schemas in the MySQL database based on the filtered data. The creation order, considering dependencies, is 'roomType', 'dayType', 'listings', 'prices', 'ratingCategory', 'reviews'.

  Data Type and Justification:
  - **FLOAT**: Used for 'real_sum', which is the price of listings requiring decimal precision.
  - **TINYINT**: Used for the boolean value of the 'is_super_host' attribute since MySQL does not support boolean data type.
  - **INT**:
    - Used for primary keys: 'listing_id', 'review_id', 'price_id', 'room_type_id', 'rating_cat_id' to support auto-increment and identification numbers.
    - Used for 'capacity', 'num_of_bedrooms', 'guest_satisfaction_score' to store numeric values.
  - **VARCHAR(10)**: Used for 'day_type', identifying entries in the 'dayType' table and considering the length of values like weekdays and weekends is short.
  - **VARCHAR (default 255)**: Used for 'room_type' and 'rating_category' to accommodate the variability in lengths of different room types and satisfaction levels.

```sql
USE AMS_Airbnb_Pricing;

-- create normalized tables
DROP TABLE IF EXISTS hosts;
DROP TABLE IF EXISTS roomType;
DROP TABLE IF EXISTS dayType;
DROP TABLE IF EXISTS listings;
DROP TABLE IF EXISTS prices;
DROP TABLE IF EXISTS ratingCategory;
DROP TABLE IF EXISTS reviews;

CREATE TABLE roomType (
  room_type_id int PRIMARY KEY AUTO_INCREMENT,
  room_type varchar(255)
);

CREATE TABLE dayType (
  day_type varchar(10) PRIMARY KEY
);

CREATE TABLE listings (
  listing_id int PRIMARY KEY AUTO_INCREMENT,
  capacity int,
  num_of_bedrooms int,
  is_super_host tinyint,
  room_type_id int,
  FOREIGN KEY (room_type_id) REFERENCES roomType (room_type_id)
);

CREATE TABLE prices (
  price_id int PRIMARY KEY AUTO_INCREMENT,
  real_sum float,
  listing_id int,
  day_type varchar(10),
  FOREIGN KEY (listing_id) REFERENCES listings (listing_id),
  FOREIGN KEY (day_type) REFERENCES dayType (day_type)
);

CREATE TABLE ratingCategory (
  rating_cat_id int PRIMARY KEY AUTO_INCREMENT,
  rating_category varchar(255)
);

CREATE TABLE reviews (
  review_id int PRIMARY KEY AUTO_INCREMENT,
  guest_satisfaction_score int,
  listing_id int,
  rating_cat_id int,
  FOREIGN KEY (listing_id) REFERENCES listings (listing_id),
  FOREIGN KEY (rating_cat_id) REFERENCES ratingCategory (rating_cat_id)
);
```

*Figure 17: SQL script for Tables Creation*

- **Data Ingestion and Denormalization**
  Create a denormalized table and pivot tables to facilitate data manipulation and analysis. The denormalized table retains raw data in its original form, while pivot tables store data aligning with the normalized structure of the database.

```sql
USE AMS_Airbnb_Pricing;

DROP TABLE IF EXISTS denormalised_ams_data;
-- Create the denormalised table
CREATE TABLE denormalised_ams_data (
    realSum float,
    room_type varchar(255),
    person_capacity int,
    host_is_superhost tinyint,
    guest_satisfaction_overall int,
    bedrooms int,
    day_type varchar(10),
    rating_category varchar(255)
);

LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\amsterdam-listings-cleaned.csv
INTO TABLE denormalised_ams_data
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

DROP TABLE IF EXISTS pivot_listings, pivot_prices, pivot_reviews;

-- Create a pivot table for listings
CREATE TABLE pivot_listings AS (
    SELECT
        person_capacity AS capacity,
        bedrooms AS num_of_bedrooms,
        host_is_superhost AS is_super_host,
        room_type
    FROM denormalised_ams_data
);

-- Create a pivot table for prices
CREATE TABLE pivot_prices AS (
    SELECT
        person_capacity AS capacity,
        bedrooms AS num_of_bedrooms,
        host_is_superhost AS is_super_host,
        room_type,
        realSum,
        day_type
    FROM denormalised_ams_data
);

-- Create a pivot table for reviews
CREATE TABLE pivot_reviews AS (
    SELECT
        person_capacity AS capacity,
        bedrooms AS num_of_bedrooms,
        host_is_superhost AS is_super_host,
        room_type,
        guest_satisfaction_overall AS guest_satisfaction_score,
        rating_category
    FROM denormalised_ams_data
);
```

*Figure 17: SQL script for the Creation of Denormalized Table and Pivot Tables*

### 3) Load (L)

The load step involves inserting the prepared and transformed data into the database, ensuring that the data is accurately placed into the correct tables.

- **Data Insertion into Normalized Tables**

  The order of data insertion follows the order of table creation to maintain integrity and relationships.

```sql
USE AMS_Airbnb_Pricing;

-- Insert distinct room types
INSERT INTO roomType (room_type)
SELECT DISTINCT room_type
FROM denormalised_ams_data;

-- Insert distinct day types
INSERT INTO dayType (day_type)
VALUES ('weekdays'), ('weekends')
ON DUPLICATE KEY UPDATE day_type = VALUES(day_type);

-- Insert distinct rating categories
INSERT INTO ratingCategory (rating_category)
SELECT DISTINCT rating_category
FROM denormalised_ams_data;

-- Insert listings data
INSERT INTO listings (capacity, num_of_bedrooms, is_super_host, room_type_id)
SELECT
    l.capacity,
    l.num_of_bedrooms,
    l.is_super_host,
    rt.room_type_id
FROM pivot_listings l
JOIN roomType rt ON rt.room_type = l.room_type;

-- Insert prices data
INSERT INTO prices (real_sum, listing_id, day_type)
SELECT
    p.realSum,
    l.listing_id,
    p.day_type
FROM pivot_prices p
JOIN listings l ON p.capacity = l.capacity
    AND p.num_of_bedrooms = l.num_of_bedrooms
    AND p.is_super_host = l.is_super_host
    AND l.room_type_id = (SELECT room_type_id FROM roomType WHERE room_type = p.room_type)
JOIN dayType dt ON p.day_type = dt.day_type;

-- Insert reviews data using the new pivot table
INSERT INTO reviews (guest_satisfaction_score, listing_id, rating_cat_id)
SELECT
    r.guest_satisfaction_score,
    l.listing_id,
    rc.rating_cat_id
FROM pivot_reviews r
JOIN listings l ON r.capacity = l.capacity
    AND r.num_of_bedrooms = l.num_of_bedrooms
    AND r.is_super_host = l.is_super_host
    AND l.room_type_id = (SELECT room_type_id FROM roomType WHERE room_type = r.room_type)
JOIN ratingCategory rc ON r.rating_category = rc.rating_category;
```

*Figure 17: SQL script for Data Insertion*

## Critical Reflection on Database Design and Management

The overall structure of the database was well-handed, but upon reflection, I have identified several areas for improvement.

One issue I encountered was poor naming conventions. For example, the attribute ‘realSum’ was used in the pivot table, while ‘real_sum’ was used in the normalized table. This inconsistency can lead to confusion in data retrieval. Naming conventions should be consistent across the database to ensure clarity and ease of use.

Besides, there was ambiguity in the term “capacity” within the listings table. The “capacity” of listings" can refer to either the person capacity (the number of people a listing can accommodate) or the size capacity (the physical space or size of the listing such as square footage), which cause misunderstandings and should be clarified by using more specific term like ‘person_capcity’ or ‘size_capacity’.

Furthermore, I realized that if the database design is too normalized, it can lead to complex queries requiring multiple join operations,which can result in slower performance. For example, the SQL query for the research question 3c,I retrieved the data from the pivot table instead of the normalized tables to improve the performance. This experience highlights the importance of creating pivot tables for certain use cases to streamline data retrieval and enhance performance.

## SQL Commands for Research Questions

**Research Question 1:** Establish a foundational understanding of the Airbnb market composition in Amsterdam.

- SQL Query: Count the total number of listings and the number of listings for each room type.

```sql
-- counts the total number of listings
SELECT COUNT(*) AS total_listings FROM listings;

-- counts the number of listings for each room type
  SELECT rt.room_type, COUNT(listings.listing_id) AS count
  FROM listings
  JOIN roomType rt ON listings.room_type_id = rt.room_type_id
  GROUP BY rt.room_type;
```

*Figure 18: SQL Query for Research Question 1*

**Research Question 2:** Assessing how superhost status may affect the pricing strategies, understanding competitive dynamics in the market.

- SQL Query: Get the average price of listings for superhost and non-superhost.

```sql
SELECT
    ROUND(AVG(prices.real_sum),0) AS avg_price,
    listings.is_super_host
FROM prices
JOIN listings ON prices.listing_id = listings.listing_id
GROUP BY listings.is_super_host;
```

*Figure 19: SQL Query for Research Question 2*

**Research Question 3**

How do Factors Influence Guest Overall Satisfaction in Amsterdam's Airbnb Market?

**Research Question 3a:** Explore the impact of superhost status on guest satisfaction, a key metric for guest demand.

- SQL Query: get the average guest satisfaction score for superhost and non-superhost.

```sql
SELECT
        ROUND(AVG(reviews.guest_satisfaction_score),2) AS avgScore,
        listings.is_super_host
    FROM reviews
    JOIN listings ON reviews.listing_id = listings.listing_id
    GROUP BY listings.is_super_host;
```

*Figure 20: SQL Query for Research Question 3a*

**Research Question 3b:** Examine whether the size of the listing (capacity) has higher satisfaction scores, which could influence listing management strategies.

- SQL Query: get the average guest satisfaction score for different person capacities of listings.

```sql
SELECT
  ROUND(AVG(reviews.guest_satisfaction_score), 2) AS avgScore,
  listings.capacity
FROM reviews
JOIN listings ON reviews.listing_id = listings.listing_id
GROUP BY listings.capacity
ORDER BY listings.capacity;
```

*Figure 21: SQL Query for Research Question 3b*

**Research Question 3c:** Investigates if pricing strategies related to day types affect guest satisfaction

- SQL Query: get the average prices of listings for weekdays and weekends and the corresponding satisfaction level (rating_category).

```sql
SELECT
        p.day_type,
        r.rating_category,
        AVG(p.realSum) AS avg_price
    FROM pivot_prices p
    JOIN pivot_reviews r
    ON p.capacity = r.capacity AND
        p.num_of_bedrooms = r.num_of_bedrooms AND
        p.is_super_host = r.is_super_host AND
        p.room_type = r.room_type
    GROUP BY p.day_type, r.rating_category
    ORDER BY p.day_type, r.rating_category;
```

*Figure 22: SQL Query for Research Question 3c*

# Stage 4. Create a simple web application

In this stage, a simple web application is developed to display the results of data queries for each research question. The results are presented on a web page using visualizations such as unordered lists, tables, and charts. .

## Web App Tech Stack

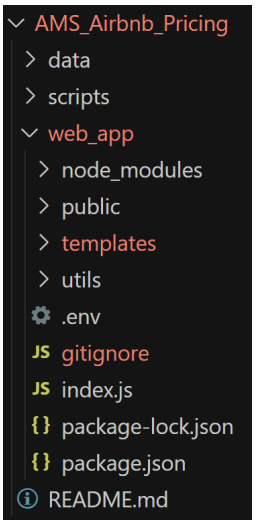
The web app is built using the following technologies:

- **Node.js:** A JavaScript runtime built on Chrome's V8 JavaScript engine, allowing for server-side scripting.
- **Express:** A minimal and flexible Node.js web application framework providing a robust set of features for web and mobile applications.

- **MySQL:** An open-source relational database management system that uses SQL for database operations.
- **body-parser:** Middleware for Express that parses incoming request bodies before handlers, making the data available under req.body.
- **Mustache:** A logic-less template engine for creating dynamic HTML, used for rendering views in web applications.
- **Dotenv:** A module that loads environment variables from a .env file into process.env, allowing for configuration management.
- **Chart.js:** A JavaScript library for creating simple yet flexible charts for data visualization on the web.

## Project Directory Structure

*Figure 23: Project Directory Structure*




- **data/:** Stores data files required by the project, such as datasets and CSV files.
- **scripts/:** Contains scripts for database connection, user configuration, and data insertion related to SQL tasks.
- **web_app/:** Contains the main application code.
- **public/:** Contains static files that are served directly by the web server.
- **utils/:** Contains utility modules and helper functions that encapsulate specific logic and database queries for each research question.
- **templates/:** Contains HTML templates used by the Mustache template engine to render the web application's front end.
- **index.js:** The main entry point of the web application. Sets up the Express server, defines routes, and integrates with the Mustache template engine.

## Database Interaction Works and Web Application Representation

The database is successfully connected, and the web application runs on ‘localhost:3000’. Each display is labeled according to the corresponding research question order.

```
PS C:\Users\wyinxuan\Downloads\DB Lecture Slides\AMS_Airbnb_Pricing\web_app> node index.js
The app is listening at http://localhost:3000.
Database connected!
```

*Figure 24: Terminal Output*

**Research Question 1:** Presented by showing the total number of listings in Amsterdam and the different room types in an unordered list.
**Research Question 2:** Presented by showing the average pricing of listings for superhosts and non-superhosts in an unordered list.
**Research Question 3a:** Visualized with a comparative bar chart showing the average prices for Superhosts vs. Non-superhosts.
**Research Question 3b:** Presented in a table showing the average guest satisfaction scores for different listing capacities, highlighting the highest average satisfaction score for listings with a specific capacity.
**Research Question 3c:** Visualized with a comparative bar chart showing guest satisfaction levels for average prices on different day types (weekdays vs. weekends).

**Web Application Screenshot for Research Questions:**
There are some certain limitations when using Chart.js to display charts within the browser preview on Coursera's lab environment. I run the wep app on the local host on my computer.

# Amsterdam Airbnb Pricing Dataset Analysis

### 1) Total Airbnb Listings in Amsterdam: 2064 Listings

- Private room: 944 listings
- Entire home/apt: 1110 listings
- Shared room: 10 listings

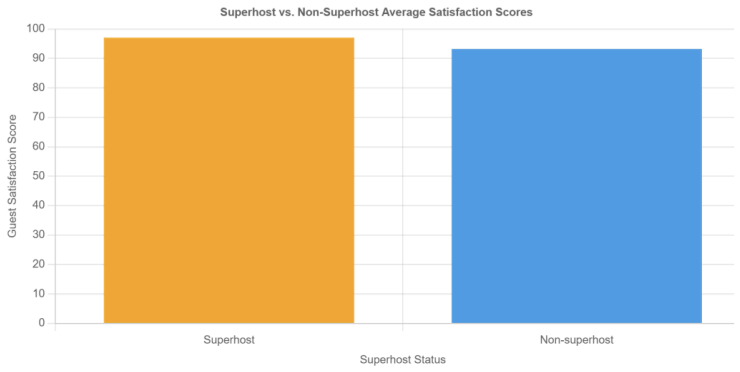
### 2) Average Pricing of Listings Managed by Superhost vs. Non-superhost listings

- Average prices for Superhost: 395
- Average prices for Non-superhost: 486

### 3) Factors Influencing Guest Overall Satisfaction in Amsterdam's Airbnb Market

Sub-questions for clarity include:

#### a. Guest Satisfaction Overall for Superhost vs. Non-superhost Listings




#### b. Guest Satisfaction Overall for Person Capacity of Listings

The average guest satisfaction scores for different listing capacities

| Listing Capacity | Average Guest Satisfaction Score |
|---|---|
| 2 Persons | 93.9 |
| 3 Persons | 94.01 |
| 4 Persons | 94.37 |
| 5 Persons | 96.29 |
| 6 Persons | 95.86 |

**The highest average guest satisfaction score: 96.29 for listings with a capacity of 5 people.**

#### c. Guest Satisfaction Trend Correlated with Pricing Differences on Weekdays vs. Weekends

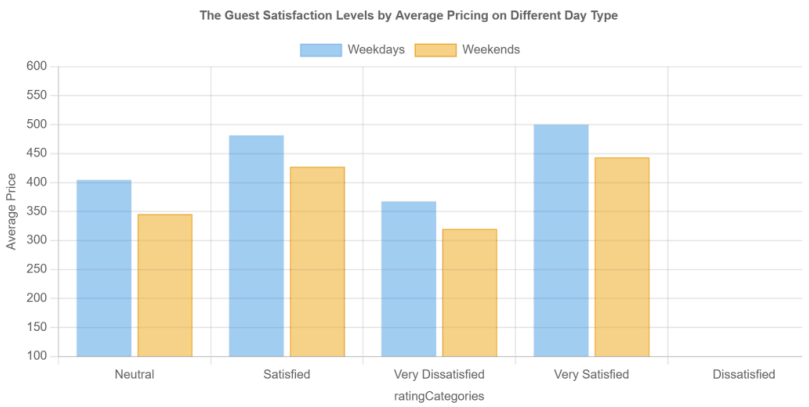



*Figure 25: Web App Screenshots*

## Conclusion

The project successfully achieved its objectives, including finding and critiquing a dataset, modeling the data, creating a database, and developing a simple web application. This effort highlights the effectiveness of combining data analysis with web technologies. Looking ahead, there is potential for integrating real-time data updates and advanced analytics to further enhance the system's capabilities.

# References


[1] Vernon, H. (2023, Aug 10). *Europe's Best Short Term Rental Markets, According to AirDNA*. b Magazine. Retrieved June 8, 2024, from https://www.benoitproperties.com/news/europes-best-short-term-rental-markets-according-to-airdna/

[2] The Devastator. (2023). *Airbnb Prices in European Cities*. Kaggle. Retrieved June 5, 2024, from https://www.kaggle.com/datasets/thedevastator/airbnb-prices-in-european-cities/data

[3] Zenodo. (2021, Jan 13). *Determinants of Airbnb prices in European cities: A spatial econometrics approach (Supplementary Material)*. Zenodo. Retrieved June 9, 2024, from https://zenodo.org/records/4446043#.Y9Y9ENJBwUE

[4] Hostaway. (n.d.). *Why Guest Satisfaction is Key to Maximizing ROI on Airbnb Properties*. Why Guest Satisfaction is Key to Maximizing ROI on Airbnb Properties. Retrieved June 9, 2024, from https://www.hostaway.com/blog/why-guest-satisfaction-is-key-to-maximizing-roi-on-airbnb-properties/#:~:text=Why%20is%20guest%20satisfaction%20so,%2C%20and%20ultimately%2C%20higher%20profits.