Abstract

For this final project we combined Kaggle datasets to create an application for Bostonian homeowners interested in becoming AirBnB hosts. This application allows users to evaluate the revenue potential of renting out their properties by allowing them to review listings, prices, reviews throughout the calendar year. The ability to view the properties based on the dates is especially useful to measure the impact of climate on rental prices in a cold weather environment like Boston. The application features five page types which provide detailed information on rental property listings and hosts. For an optimal user experience, we implemented queries that at most execute in 2.3 seconds.

Group Members: Yiyasu Paudel, Gabriela Campoverde, Purva Sheth, Yasmine Boukataya

Introduction and project goals

We created an application which evaluates the relationship between AirBnBs and the weather for the Boston area by gathering two datasets from Kaggle which are specific to Boston. There are 3 tables in the Boston Airbnb dataset: listings, calendar, and reviews. Listings contains information about a particular listing such as its name, id, summary, and a url to the listing on airbnb.com. Calendar contains the listing id and the date that it was booked.[1] There is 1 table in the Weather dataset which contains the date, temperature, dew point, and humidity.[2]

Our target user for our web application is a person who lives in Boston and is considering becoming an AirBnB host. Our application will give a user the opportunity to query through Boston listings and assist them in how this user should price their rental according to how people price their listings in their surrounding area. Moreover, the user will be able to look up the ratings and other host and listing statistics so that the user can be well informed about how competitive the area is. Boston is known for its cold, early winters, and our tool will allow these users to measure the effect of weather on AirBnB rental demand.

Data Sources and technologies used

Users can evaluate the effect of weather on the availability and price of listings in the area with our combined AirBnB and Boston weather dataset. To grasp an overview of the datasets we evaluate the Weather and AirBnB data sets separately.

1. A*irbnb Dataset* consists of 3 separate files (calendar, listing and review) and 105 columns.
2. *Boston Weather Dataset* contains a file with 24 columns.

Our full stack web application uses the following tools:

1. *Backend:* NodeJS-Express server deployed on Heroku, MySQL DB hosted on AWS RDS
2. *Frontend:* React, HTML/CSS, deployed on Netlify

Relational Schema

[1]Boston AirBnB Dataset: https://www.kaggle.com/airbnb/boston/metadata
[2]Boston Weather Dataset: https://www.kaggle.com/jqpeng/boston-weather-data-jan-2013-apr-2018

1. **Calendar**(listing_id, date, available, price)
   constraint Calendar_ibfk_1
   FK (listing_id) references Listing (id)
2. **Host**(host_id, host_name, host_since, host_location, host_about, host_response_time, host_response_rate, host_acceptance_rate, host_is_superhost, host_thumbnail_url, host_picture_url, host_total_listings_count, host_has_profile_pic)
3. **Listing**(id, listing_url, last_scraped, name, summary, space, description, experiences_offered, transit, thumbnail_url, host_id)
   constraint Listing_ibfk_1
   FK (host_id) references Host (host_id))
4. **Reviews**(id, listing_id, date, reviewer_id, reviewer_name, comments)
   FK (listing_id) references Listing(id)
5. **Weather**(date, high_temp, avg_temp, low_temp, high_humidity, avg_humidity, low_humidity)

Schema Normalization

The database schema is in 3NF as all relations only have functional dependencies such that the left hand side of the functional dependency is the primary key. To achieve this, we decomposed the original listings table in the dataset to Listing and Host.

Description of System Architecture

Below are descriptions of our applications pages with information about what the respective pages accomplish:

1. *Home Page:* This home page serves as an introduction of the project. At the top of the homepage you will find the menu bar which links to the Listings Page and Hosts Page. It is connected to the backend via the *Booking with Temperature* graph to get the frequency of bookings within certain temperature ranges which requires a join on temperature and bookings table, and grouping by temperature ranges. It also redirects users to the *Listings* and *Hosts* page.
2. *Listings Page:* This page displays, in paginated format, all of the listings in our database. Each entry is a card that links to the individual listing page. The card includes the listing name, review, number of views, number of reviews, price and a link to the listing.
3. *Hosts Page:* This page displays, in paginated format, all the hosts in our database. Each entry is a card that links to the individual host page. The card contains the host's profile image in high resolution, their total number of listings, and a portion of their about me description which is fully listed on their individual page.
4. *Listing Page:* Our Listing page includes several important pieces of information. It contains the original information of the listing on Airbnb's website including: the name of the listing, a summary, a description, and transit information which are all written by the individual host. The description and transportation information can be accessed by clicking on 'Show More' which displays a pop-up box that the user can scroll through. It also includes the host's name and picture and a button which leads directly to their profile. Most importantly, there are graphs which show the number of bookings by month and by temperature ranges (ex 60-69, 70-79, etc). Below these graphs is a paginated list of reviews.

5. *Host Page:* This page includes the host name, an about section written by the host themself, and several statistics about the host such as their response and acceptance rates and how long they have been active. It also includes pictures and a link to all of their listings.

How We Addressed Required Features

We combined the AirBnB dataset with a Boston weather dataset to allow users to evaluate the effect of weather on the availability and price of listings in the area. Complex queries were implemented to obtain the graph for Bookings/Price with Temperature and Price by Month. We added a number of listings for each host and direct links from the host page to each of their listings and gathered all of the reviews for a particular listing onto a single page which required joins between a couple of tables.

Performance Evaluation

There were a couple of query optimizations that we made. The first was using an index. The second was using temporary tables in multiple endpoints including: /listing/prices_with_temp and /listings. In /listing/prices_with_temp we needed to inner join the Calendar table on the Weather table in order to get the average temperature by date. This temporary table was then used to generate a chart in the same query which grouped the temperatures in ranges of 10. In /all_listings we created a Price temporary table which consisted of a left join of Listing on Calendar. Initially, joining on all three tables directly led to crashing of the system. Finally, for /num_bookings_with_temp, adding an index on Calendar(date) improved performance. Check Appendix B and C for more information.

Technical Challenges and How They were Overcome

Heroku deployment was unsuccessful for frontend because Heroku required the frontend public folder be set up in a specific way and had no buildpack for React. Instead we deployed the frontend on Netlify.

Deploying on Netlify was another challenge because we could not set the base directory for the deployment. So we created a new branch which did not need the base directory to be set and deployed that.

Deciding whether or not beforehand to make certain pages just a component or a class. We saw that there was no way to display the reviews on the Listing's page with pagination without using state variables instead of useState/useEffect.

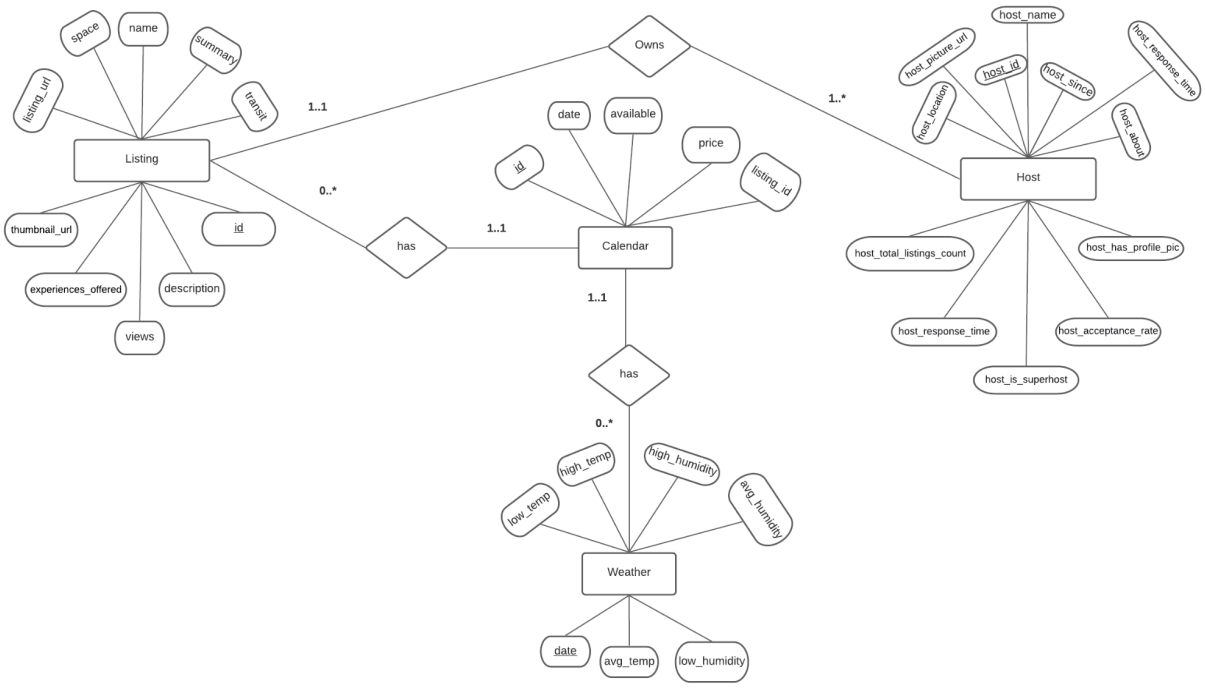
Displaying the data was also another technical challenge for us. It took some trial and error to find the right package to plot data such as prices by temp, number of bookings by month, etc. because some of them were very strict on the format of the input data and some took longer to render than others

Extra Credit Features

- Deployment (See Technical Challenges for details).

Appendix A

Entity Relationship Diagram



Appendix B

Optimization on query to get information on all listings

Query:

```
WITH Price as (Select l.id as id, round(avg(c.price),2) as price
    FROM Listing l LEFT JOIN Calendar c ON l.id = c.listing_id
    GROUP BY l.id)
SELECT l.id, l.num_views, l.name, l.summary, l.thumbnail_url, p.price, count(r.reviewer_id) as
'num_reviews'
FROM Listing l LEFT JOIN Reviews r
ON l.id  = r.listing_id
LEFT JOIN Price p
ON l.id = p.id
    GROUP BY l.id
    ORDER BY l.id
```

Initially, the system crashed when we tried joining all 3 tables i.e Listings, Reviews and Calendar. After creating the temporary table Price, this query runs in 1.7s

Appendix C

Optimization on query to get the frequency of bookings within certain temperature ranges

Query:

```
WITH booking_temps AS ( SELECT w.date as date, w.avg_temp as temp
    FROM Calendar c
    INNER JOIN Weather w ON c.date = w.date )
SELECT floor(temp/10)*10 AS range_floor, count(*) AS count
FROM booking_temps
GROUP BY 1
ORDER BY 1
```

Initially, this query took 3.8 seconds. After creating an index on the date column in Calendar, this query runs in 2.3s.

Appendix D

Data Repositories

Data set 1:

Airbnb Boston Dataset from Kaggle

Boston Airbnb Dataset

Data set 2:

Boston weather data from Kaggle

Boston Weather

Appendix E:

SQL Queries

Information of all listings for table

```
WITH Price as (Select l.id as id, round(avg(c.price),2) as price
   FROM Listing l LEFT JOIN Calendar c ON l.id = c.listing_id
   GROUP BY l.id)
SELECT l.id, l.num_views, l.name, l.summary, l.thumbnail_url, p.price, count(r.reviewer_id) as
'num_reviews'
FROM Listing l LEFT JOIN Reviews r
ON l.id  = r.listing_id
LEFT JOIN Price p
ON l.id = p.id
   GROUP BY l.id
   ORDER BY l.id
```

Information of all hosts for table

```
SELECT
     host_id,
```

```
    host_name,
    host_about,
    host_response_rate,
    host_response_time,
    host_acceptance_rate,
    host_total_listings_count,
    host_picture_url
FROM Host
```

Information for a listing by id

```
SELECT
    id,
    listing_url,
    name,
    summary,
    space,
    description,
    experiences_offered,
    transit,
    thumbnail_url,
    l.host_id,
    host_name,
    host_about,
    host_picture_url
FROM Listing l, Host h
WHERE h.host_id = l.host_id and id=${listingId}
```

Information for a host by id

```
SELECT
    id,
    listing_url,
    name,
    summary,
    space,
    description,
    thumbnail_url,
    h.host_id,
    host_name,
    host_since,
    host_location,
    host_about,
    host_response_time,
    host_response_rate,
    host_acceptance_rate,
```

```
    host_is_superhost,
    host_thumbnail_url,
    host_picture_url,
    host_total_listings_count,
    host_has_profile_pic
FROM Listing l, Host h
WHERE h.host_id = l.host_id and l.host_id=${hostId}
```

Get number of bookings for a listing by month

```
SELECT listing_id, MONTH(STR_TO_DATE(${date}, '%Y-%m-%d')) AS month, count(*) as count
FROM Calendar
WHERE listing_id = '${req.query.id}'
GROUP BY month
ORDER BY month
```

Increment views

```
UPDATE Listing set num_views = num_views + 1
WHERE id = ${req.query.id}
```

Average price for a listing by month (price is null when the listings is not available or booked)

```
SELECT MONTH(STR_TO_DATE(${date}, '%Y-%m-%d')) AS month, avg(price) as average_price
FROM Calendar
WHERE listing_id = ${req.query.id}
GROUP BY month
ORDER BY month
```

Reviews for a particular listing with pagination

```
SELECT l.id, r.reviewer_id, r.reviewer_name, r.comments, r.date
FROM Listing l, Reviews r
WHERE r.listing_id = l.id and l.id = ${req.query.id}
```

Frequency of bookings within certain temperature ranges (display on the main listings page not for particular listing)

```
WITH booking_temps AS ( SELECT w.date as date, w.avg_temp as temp
    FROM Calendar c
    INNER JOIN Weather w ON c.date = w.date )
SELECT floor(temp/10)*10 AS range_floor, count(*) AS count
FROM booking_temps
GROUP BY 1
ORDER BY 1
```

Change in price with average temp for a listing (5506 = id)

```
WITH booking_temps AS ( SELECT c.listing_id, c.price, w.date as date, w.avg_temp as temp
    FROM Calendar c
    INNER JOIN Weather w ON c.date = w.date )
SELECT floor(temp/10)*10 as range_floor, listing_id, round(avg(price),2) as average_price
FROM booking_temps
WHERE listing_id = ${req.query.id}
GROUP BY range_floor, listing_id
ORDER BY range_floor
```