Crimebnb

Travel Safe, Travel More, Travel Better

Tommy Drenis
Zach Duey
Sungbum Hong
Kun Hwi Ho

Introduction

Travelers are increasingly using services like Airbnb to find short-term rentals both in the United States and abroad when traveling for both leisure and pleasure. Although Airbnb provides users with advanced search functionality for finding places to stay, one area that is lacking is that users must leverage other resources to incorporate the safety of the neighborhood where the rental is located into their decision. Our application is a proof-of-concept that fills this gap by integrating publicly-available crime and Airbnb listing data in New York City, enabling users to make smarter, safety-conscious decisions about where to stay when visiting the city.

The application allows users to identify the best listing for their needs from two angles. Users who are primarily concerned about the safety of the location of the listing can use the neighborhood path, while users who are primarily interested in features of the listing can use the listings path. Both paths ultimately allow users to drill down into the single-best listing for their stay.

Users primarily interested in safety start by navigating to the all boroughs landing page, which displays the top 100 crime hot spots based on the number of crimes committed at that location (red markers) alongside the top 100 listings with a minimum of 25 reviews ranked by average overall rating (blue markers). From this page, users can use the tabs at the top of the page to zoom in on each individual borough. If interested, users can then drill further down to the single borough page and view additional summary information about the borough such as the number of listings, the average and median price of listings, safest neighborhoods within the borough, and crime statistics since 2015. Users who need more information to make a decision

can then use the **borough comparison page** to determine which borough they should focus on in their listing search. By exploring the different borough pages, users can make a decision about which borough best fits their needs. From here, they have two options. The quick option is to use the provided map to explore highly-rated listings by clicking on the listing of interest and viewing the listing detail page (discussed later). The second option is to navigate to the listing search page.

The **listing search page** allows users to identify listings within a single borough that meet their specific criteria. For example, users can filter listings based on the highest price they are willing to pay, the number of bedrooms and bathrooms, and the cleanliness and location ratings of the listings. When users submit their preferences, they are provided with two ways to explore the results. In the map view, they can see a (hopefully familiar now) map of the listings that meet their criteria. By clicking on an icon, they can then drill down further to the listing details. Alternatively, users can view a list of results that takes into account listing features as well as crimes that have been committed near the listing. This allows users a simple way to find listings that meet their budget, amenity, and safety requirements. By clicking on a particular listing, users can navigate to the listing detail page, which provides an even finer-grained view of the listing.

The **listing detail page** gives users fine-grained information designed to help them make a final decision about whether or not to book their stay. At the top of the page, users are given summary information about the listing (room type, neighborhood) that complements the criteria they used to identify the listing. It also provides a detailed breakdown of the average ratings summarized on a 1-5 star (fractions allowed) scale. Crime information is provided in two ways. Precinct-level crime information is summarized in a plot of the top 5 crimes in the precinct in which the listing is located. Nearby crimes are displayed alongside the location of the listing (light blue marker) within the map (red markers), aggregated by location. Finally, users are provided with a list of similar listings close to the current listing so that they can easily hop around to other listings without needing to navigate to other pages.

Architecture

Our project architecture is divided into three components: data processing pipeline, application backend, and application frontend. Our data processing pipeline leverages python and SQL. The source data (described in the following section) was acquired in CSV format and stored in a MySQL database. Airbnb data was parsed and cleaned using the pandas library. Due to the size of the crime data, both dask and pandas were used for parsing and cleaning. We used sqlalchemy and pandas to write records into the database. Our application backend consists of two components: a MySQL database and an API written using node express that is responsible for interacting with the database and returning results as json. Finally, our application frontend is written using the React.js framework with charts generated using Chart.js. The frontend gathers crime and listing data using the API provided by our node-express-based backend service. Finally, we render interactive maps using mapboxgl, which provides a javascript interface for interacting with the Mapbox API.

Data

The New York City crime data comes from Kaggle. This dataset covers the time period from 2006 to 2017 and includes 35 columns and more than 6 million rows describing various aspects of reported crime incidents across the city. For each crime, demographic information is included about both the victim(s) and suspect(s). In addition, the type of crime committed is described and assigned a proper code, as well as a description of any weapons used. This data is used to render crime hot spots in map on the **all boroughs page**, is the source data for the crime statistics charts on the **single borough page**, rank boroughs based on criminal activity in the **borough comparison page**, rank lististings on the **listing search page**, and is the source data for the precinct-level crime summary on the **single listing page**.

The Airbnb New York City listings data also comes from Kaggle. The data includes listings in NYC which were last reviewed by a customer between 2017 and 2020. The data includes 35 columns and 75,750 rows containing details about each listing. Listing details include the price, number of bedrooms and bathrooms, available amenities, as well as latitude/longitude coordinates and neighborhood in which the listing is located. The data also

includes the number of reviews per month and a review score breakdown by accuracy, cleanliness, check-in, communication, location and value. This data is used render the listing locations in the map on the **all boroughs page**, is the source data for the listing statistics charts on the **single borough page**, rank boroughs by average listing rating on the **borough comparison page**, filter listings (and display them on the map) by amenity and rating on the **listing comparison page,** and is the primary source of information as well as the recommendation engine on the **single listing page**.

Database

As with most Kaggle datasets, the original data was relatively well-organized, however each data source provided different data processing and normalization challenges.

Pandas was the main tool used for our data parsing process. We began by dropping unnecessary columns that are of little use to our application, made appropriate type conversions, and removed duplicates as necessary. For example, when creating a data frame to view the unique neighborhoods within New York, keeping duplicate instances of neighborhoods makes little sense.

One difficulty for the data parsing process was the amount of string parsing that had to be involved. For instance, we wanted to create a separate csv file and table for different amenities, but this required delimiting semi-colon separated strings like “Cable TV; Wireless Internet; Kitchen; Free Parking”. In another example, we wanted to combine columns representing dates (YYYY/MM/DD) with those representing time (HH:MM:SS) into a single datetime column.

Another difficulty regarding the data parsing process was fitting the data into memory. We used an external library called Dask to handle this issue. In the end, we took 2 csv files and broke them up into 16 smaller datasets.

As for inserting the data into the MySQL database, the csv files are unfortunately too large to effectively use something like the MySQL Workbench “csv import” functionality. Instead, for the listings data, we used a Python script to generate SQL insertion statements. This proved to be slightly more efficient as SQL insert statements generally have a faster

execution speed than csv conversions. For the crimes data, we optimized the speed even more through the use of the to_sql() functionality in Pandas to inject the parsed data from the Python script straight into our AWS S3 instance. This was necessary as the crimes dataset proves to have more tuples than the listings dataset.

To keep consistency amongst the datasets, entity resolution was deemed necessary. The most effective case for entity resolution was for the "borough" table as this was a commonly used join key for the listings and crime data. We kept the borough names as all capital letters to ensure this consistency. Another case of entity resolution was for the latitude and longitude numbers. We decided to keep the precision of these values at 8 decimal points to ensure consistency when plotting these coordinates to our Mapbox API.

After a successful insertion to the AWS hosted database, we had the following tables and their corresponding number of rows:

Table Name	# of Tuples	Table Name	# of Tuples
agency	22	listing	75,746
amenities	75,746	neighborhood	240
borough	5	offense	72
coordinate	78,302	park	463
crime	387	premises	5
development	764	ratings	75,746
host	56,678	suspect	500,000
incident	499,945	victim	500,000

The resulting ER diagram is provided in the link here.

Queries And Performance Evaluation

For this project, the primary means of optimization our group used were the following : pushing selections and projections to minimize rows and columns, using smaller tables as outer joins, and reusing temp tables for duplicate queries.

Following are four complex queries that are used to evaluate the performance of our optimization. Also, note that in order to fully showcase the impact of the optimization, pre-optimization time was calculated based on queries that performed no selections until the very last moment, with zero regards to join orders. All times below are in seconds.

Name	Pre-Optimization	Post-Optimization
getListingCrimeSummary	1.77	0.94
getBoroughAllCrimeInfo	1.96	1.25
getBoroughBestListingInfo	0.21	0.11
getRatingRank	0.42	0.29
getListings	1.72	1.13

Based on various pre-optimization and post optimization trials that were performed, the most notable performance change usually occurred when selection and projection was pushed to the bottom most layer. The most logical explanation for this phenomenon is the copious size of our data table. In our current dataset, incident and listings tables host the bulk of our 2GB data set. Both of these queries hold at minimum 13 attributes and unfortunately they are the most frequently used tables in our list of queries.