Predicting Airbnb Listing Price | Data Wrangling
Claire Miles

Data Importing

Since I am using data sourced from many files found on the same webpage, I decided to write a script that scraped that page for the relevant URLs to my project.

To complete this task, I used the requests and BeautifulSoup packages. First, I used requests to capture the response from the url where all of the csv files are hosted: http://insideairbnb.com/get-the-data.html. After catching that response in a variable, I read the text into another variable. I turned the text variable into a BeautifulSoup object using the BeautifulSoup function. From there, I could extract the text from the HTML anchor tags, where the csv links are found in the 'href' attribute of the tag. I stored the names of the csv urls in a list. Since the links led to gzipped (.csv.gz) files, I called the list 'zipped_links'.

Next, I wrote the information from the online gzipped csv files to local files. Using a context manager, I looped through zipped_links to create custom filenames for each list item and write to that file. The filename includes the city name, the date the data was collected, and the information category (listings or reviews).

Data Combination

After the raw files were scraped, created, and stored, the data was further consolidated into the listings and reviews categories in the form of the pandas dataframe. Since the scraped data represented data collected monthly from hundred of cities and was incredibly large, I decided it would be a better idea to focus my initial analysis on just one city: Los Angeles.

Because the data from the website is collected monthly, there were several raw files to process with hundreds of thousands of lines of data. Therefore, it helped to create functions that did the heavy lifting:

- **consolidate_data** checks if the csv file for either listings or reviews data has been created for the designated city. If the file has not been created, run the combine_listings or combine_reviews function for that city, and then create the csv file for that city.
- **combine_listings, combine_reviews** goes through files in the directory and checks for the designated city listings or reviews files. The it appends the names of the listings or reviews files of that city to a list, and passes the list and the directory name to the concat_files function.
- **concat_files** creates a pandas dataframe for each file name in the list of files, then adds the date recorded as a column in that dataframe (taken from the file name). Then it appends the dataframe to a list of dataframes. After all files in the list have been converted to pandas dataframes, it concatenates the dataframes together, drops duplicate rows, and resets the dataframe index.
- **export_csv** checks if the desired csv file does not exist in the current working directory, then converts the dataframe to a csv file and moves the the desired folder in the destination directory.

Even though I am starting by analyzing one city, having functions will also allow me to scale my analysis to multiple cities in the future.

After I created the functions, I defined the directory (the pathway where the raw data is stored) and destination folder (where I wanted to store the concatenated data on the computer) and ran the functions. The outcome was a single listings file and single reviews file for Los Angeles.

Data Cleaning

After combining the data into single csv files for Los Angeles, I still needed to clean and explore the data.

Datatypes:

First, I worked with the listings data, which initially had 97 different columns of float and object datatypes. From an initial inspection of the data, there were many columns that could be converted to more suitable datatypes. For example, price data was stored in strings along with the '$' character - this data was converted to numeric datatypes with the dollar sign being added to the column name. Additionally, string data that only had a few unique values, like in the 'property_type' column, was converted into categorical data. To make the datatype conversion process easier, I created a function **change_datatypes**. After running the function, I was able to reduce the size of the dataframe from 4.9GB to 3.8GB.

Which column to predict?:

In the proposal for this project, I was still deciding whether to create an algorithm to predict Airbnb listing prices or average review scores. I ultimately decided to choose the metric that contained the least missing data. After calculating the percentage of data missing for both, I found that the price data is only missing 2% of data, while review scores is missing 23%. Therefore, I decided to move forward with predicting price data.

Inspecting Price Data:

Since it would not help to train an algorithm on data without the target metric present, I removed rows from the dataframe without price data. Then, I ran the **describe()** function on the data to get some summary statistics. With a relatively low mean and very high maximum value, it was already apparent that the price data was skewed, and that I should inspect further for outliers. After running the **value_counts()** function, I found that 207 listings had a price of $0 and 36 listings had prices above $90,000, with the next highest price being all the way down at $999. These high and low values immediately seemed suspicious.

Luckily, the data included URLs for each listing. Looking at listings whose prices were listed at over $90,000, I found that the URLs led to legitimate listings, but the data entry seemed off in the dataframe. I decided to attribute these 36 erroneous values with data entry error and removed them from the dataset. For listings that were $0, I found that only some of them existed, while the others were probably also a data entry fluke. I decided to remove this small amount of data as well.

After removing outliers, I created a histogram and boxplot of the cleaned price data. From the visualization, it's obvious that the data is very skewed. However, the previous analysis showed that the outliers are likely real listings. Expensive listings are likely more rare for a service like Airbnb, where most customers are generally looking for affordable prices.

Tidying Data:

Several columns in the dataset presented data cleaning challenges, mostly due to data entry errors like misspellings and inconsistent capitalizations of words. Categorical data is the most likely candidates for these types of errors, so I corrected for false and inconsistent data in the zipcode, property type, room type, bed type, cancellation type, city, and state columns.

I also converted columns with more than one type of information into multiple tidy columns. For example, the 'amenities' column made the dataset untidy because it contained lists of different features in a single column. Using a function I created called **has_feature()**, I separated unique values of these lists into new boolean columns that displayed whether a listing either has or doesn't have that amenity. I then repeated that process with a column with a similar structure, 'host_verifications'.

I finished the analysis by saving the data as a csv file to my computer to preserve the changes I made during the data cleaning process.

Reviews Data:
The reviews data only has 8 columns, the most important being the 'comments' column that contains the text of the review. I plan to use the text to create a score that indicates the average positivity/negativity of the reviews for a particular listing. Since most NLP algorithms are based on the English language, it's outside the scope of this project to do sentiment analysis all the languages present in the dataset. Therefore, assuming that the majority of the reviews are in English, I removed the non-English rows in the dataframe using the langdetect python package. I also converted the 'date' row to the more suitable datetime datatype before saving the dataframe to a csv file on my computer.