Recommendation System Development on New York City Airbnb Open Data

Mai Ngo

Raghav Chhabra

DePaul University - College of Computing and Digital Media (CDM)

DSC 478: Programming Machine Learning Applications

Carlos A Castro Herrera

June 11, 2023

Tables of Contents

Executive Summary

The main objective of this project is to create an Airbnb content-based recommendation system using the New York City Airbnb Open Data. Users can get top five most suitable Airbnb listings along with the location and distance of corresponding nearest subway station, based on three input references: neighborhood groups, price point, and reviews rating. Three datasets are used in this project, New York City (NYC) Airbnb listings with 42,931 observations, corresponding total reviews for all listings with 1,110,024 observations, and NYC subway station locations with 473 observations. In total, there are four notebooks represents each stage of this projects: Exploratory, Reviews Score, Clustering Listing Name, and Recommendation System. The logic and execution of each notebook will be explained in detail further in the report.

Several methods were applied in data preprocessing. Thorough data exploratory for Airbnb listings dataset (Exploratory notebook) using bar chart, line graph, geospatial map and scatterplot matrix to describe and visualize and the relationships between attributes. For Reviews dataset, we apply many techniques to generate reviews scores – on the scale from 1 to 5 – tqdm, transformer and nltk library for progress tracking and natural language processing; Potter Stemmer, WordNetLemmatizer and SentimentIntensityAnalyzer modules for text stemming, lemmatizing, and sentiment analysis results polarity score (Reviews Score dataset). Eventually, we chose the RoBERTa pre-trained algorithm to analyze the sentiment of reviews. The model returns three kinds of scores: negative, neutral, and positive which are used to generate a compound score for recommendation system.

Airbnb listings name attribute is grouped into 9 clusters through TF-IDF, removing stop words using Kmeans algorithm. The distance relationship between the location of each listing and each subway station's location is calculated using the geopy library, specifically geodesic module; result the longitude and latitude of the nearest subway station and corresponding distance in mile(s) between two places (Clusters Listing Name notebook). Finally, we compile the final dataset which includes listings ID, neighborhood groups, distance between listings and its nearest subway station, name of the station, Airbnb type (name clustering), and reviews ratings (1-5). In conclusion, we were successfully created a content-based recommendation system (Recommendation System notebook), yet there are still lots of room for improvement.

Abstract

New York City (NYC) is well known as a popular tourist destination in America. Indeed, it is inevitable for any tourist to visit the United States (US) without paying a visit to NYC. The diversity of this city in population, cultures, attractions, and culinary have made such huge impact to NYC economy, especially the hospitality industry, from hostel, hotel, to even couch surfing. Through the years, the traditional hospitality industry has been taken over by a company named Airbnb, Inc. - an American San Francisco-based company operating an online marketplace for short- and long-term homestays and experiences. Airbnb acts as a broker between the renter(s) and the owner of a property. The firm and their app have been evolved throughout times, starting from small business income source for common homeowners, to now mainstream income for hospitality entrepreneurs. In this project, using the New York City Airbnb Open Data, we will take a deep dive into the NYC Airbnb database and create a content-based recommendation system for NYC tourists suitable Airbnb listings based on their references, and nearby subway station for transportation efficiency.

Introduction

Datasets and Attributes

In this project, we are incorporating three datasets (.csv format): NYC Airbnb Listings and corresponding Reviews datasets retrieved from: http://insideairbnb.com/, the NYC subway station locations dataset retrieved from: https://data.ny.gov/widgets/i9wp-a4ja. The Listings dataset contains 42,931 observations of Airbnb listings and metrics in NYC. The Reviews dataset contains 1,110,024 total customer reviews for each Airbnb listing. The Subway dataset entails longitude and latitude of each subway location in NYC which will be used for recommendation system. For the main dataset, Listings, there are a total of 18 attributes: 11 numerical attributes, 1 date attribute, and 6 categorical attributes. Data schema:

- id: Listing ID (dtype = integer)
- name: Name of the Airbnb listing (dtype = string).
- host_id: ID number of the host (dtype = integer).
- host_name: First name of the host (dtype = string).
- neighbourhood_group: Borough of New York City (dtype = string).
- neighbourhood: Neighborhood (dtype = string).
- latitude: Latitude of the Airbnb listing (dtype = float).
- longitude: Longitude of the Airbnb listing (dtype = float).
- room_type: Type of rental (dtype = string).
- price: Rental price (dtype = integer).
- minimum_night: Minimum night of stay (dtype = integer).
- number_of_reviews: Total number of reviews a listing received (dtype = integer).
- last_review: Date of the latest review (dtype = date).
- reviews_per_month: Average number of reviews per month a listing received. Equal to total # of reviews / # of active months (dtype = float).
- calculated_host_listings_count: Number of listings corresponding a host has in the dataset (dtype = integer).
- availability_365: How many days per year the listing is available (dtype = integer).
- number_of_reviews_ltm: Total number of reviews last twelve months of the listing (dtype = integer).
- license: license associated with the listing (dtype = string).

Objective and Methods

The main objective is to create a simple recommendation system so that users can find the most suitable Airbnb based on their reference: neighborhood location, pricing point, and reviews rating. For methodology, several library and modules are utilized for data preprocessing of couple attributes.

RoBERTa ("Robustly Optimized BERT Approach"), a natural language processing (NLP) model, derived from the BERT (Bidirectional Encoder Representations from Transformers) model is used to create scoring system for the Reviews dataset. RoBERTa is trained on much larger amount of data compared to the original BERT model, which it captures more language patterns and ability to understand text. For each input review as a sequence of tokens, the model returns three ratings scores which are used to calculate compound rating scores from 1 to 5.

K-means cluster algorithm is used to cluster Airbnb listings name after Term Frequency-Inverse Document Frequency (TF-IDF) transformation. K-means partition Airbnb listings into nine clusters which was determined by the Elbow method, based on the similarity of data points using Euclidean distance as default. Geodesic distance metrics is used to calculate the distance in miles between listings and subway stations using longitude and latitude of each.

Finally, content-based recommendation system to suggests top five Airbnb listings and its nearest subways station using users' references. In content-based filtering system, the suggestions are made based on a set of characteristics or feature attributes of the item themselves. The system compares all available feature attributes similarities with references from users and return top significant item based on similarity score in descending order.

Data Exploratory

Exploratory Notebook

Descriptive Statistics

The main Listings dataset has 42931 observations and 18 attributes. Looking at Descriptive Statistics table (See Appendix A – Exploratory Notebook, Table 1), 'last_reviews' and 'reviews_per_month' both have missing data with 32,627 counts. Host with the most listings (526 listings) is Blueground - host ID: 107434423. Majority of Airbnb listings are in the Bedford-Stuyvesant neighborhood (3,086 listings) in Brooklyn, New York neighborhood group. Contradicting to the most shown neighborhood group in the dataset, which is Manhattan with 17,658 counts (41.13% of the dataset). Overall, most Airbnb listings in the dataset are scattered across various small neighborhoods within Manhattan area. Entire Home / Apartment is most common room type with 24,279 counts (56.55% of total Airbnb listings). Noticeably, 'license' attribute only has one value across the entire dataset. For next step, we will look further into the distribution of each attribute.

Neighborhood and Neighborhood Groups

Generating NYC Map Boundaries (See Appendix A – Exploratory Notebook, Figure 1) for geospatial maps to visualize relationship between couple pairs of attributes later. For 'neighbourhood_group' and 'neighbourhood' attributes, majority of Airbnb listings are in Manhattan and Brooklyn neighborhood groups: 17,658 (41.13%) and 16,237 (37.82%) listings, respectively (See Appendix A - Exploratory Notebook, Figure 2. Many Airbnb listings are in popular neighborhoods, Midtown, Harlem, Upper West Side, Hell's Kitchen for Manhattan area; Williamsburg, Bushwick, Crown-Heights for Brooklyn area (See Appendix A - Exploratory Notebook, Figure 3). The two neighborhoods with dense Airbnb listings are both in Brooklyn: Bedford-Stuyvesant with 3,086 listings and Williamsburg with 2,597 listings. The other three less common area are Queens, the Bronx and Staten Island in descending order listings. Overall, we can there are Airbnb offered almost everywhere in NYC with concentration in the two well-known neighborhood groups: Manhattan and Brooklyn.

Geospatial map described the distribution of high-priced Airbnb listings in each neighborhood group (See Appendix A - Exploratory Notebook, Figure 4). High-priced Airbnb listings are determined above the 99 percentile which listing price are greater than $1,262. Almost all expensive Airbnb are in Manhattan area – Downtown, Midtown, and Chelsea neighborhoods, or in another word, most Airbnb in Manhattan area are above the average rental price, followed by Brooklyn and Queens. This makes sense since these neighborhoods in Manhattan are notorious for extreme housing prices.

Host Listing Count and Host Name

For 'calculated_host_listing_count', describes total number of listings a host possesses., and 'host_name' attributes. Among a total of 27,455 Airbnb hosts, 82.0543% of hosts own one Airbnb listing, 10.3187% own two listings, and 3.4165% own three listings, makes up for approximately 96% of total Airbnb listings. We can conclude that majority of Airbnb are still categorized as small business income model for common homeowners (See Appendix A - Exploratory Notebook, Figure 5). Then there are much bigger scales of Airbnb listings, one single host with 526; 394; 356; 222; 207; etc. listings. These are potentially hospitality entrepreneurs with mainstream income from Airbnb. Appendix A - Exploratory Notebook, Figure 6 shows host names that have more than 100 Airbnb listings, some of the host names are entrepreneur oriented such as 'Blueground' with 526 listings, 'RoomPicks' with 356 listings, or 'Urban Furnished' with 159 listings. Noticeably, the total of entrepreneur Airbnb businesses are not as significant compared to the total of homeowners, but the total listings quantity speaks volume.

Room Type

Entire home/apartment room type is the most common one with 24,279 listing (56.55%) followed by private room with 17,879 listings (41.64%) (See Appendix A - Exploratory Notebook, Figure 7). Shared room and hotel room has significantly low quantity compared to the other two types: 576 and 197, respectively. For the hotel business, there are other platforms where they can promote their business rather than Airbnb which explains for the low count of

listings. Appendix A - Exploratory Notebook, Figure 8 represents the distribution of average rental price for each room type. Hotel room has the highest average price of $309.96, followed by entire home/apartment of $249.26 , then private room and shared room. This price distribution makes sense, traditionally hotel room price is high, which steered tourists to look for hospitality accommodations in apps like Airbnb.

Using geospatial map to visualize the distribution of room types across NYC, the larger marker size, the higher rental price (See Appendix A - Exploratory Notebook, Figure 9). Entire home/apartment are offered across all neighborhood groups, private room is denser around Manhattan and Brooklyn area, scattered around other neighborhood groups. Majority of hotel are promoted in Manhattan, especially in expensive area like Midtown and Downtown. Shared room has the same distribution with private room type, majorly in the two popular neighborhood groups. Noticeably, the average prices of private and shared room are not that much different: $135,02 and $126.25, respectively. Thus, this is all come down to pricing reference, for an average of $10 extra, tourists can stay in private room, or save money and stay in shared room.

Price

Price attribute has an extremely right skewed distribution (See Appendix A - Exploratory Notebook, Figure 10). The average Airbnb price is $200.307 with a standard deviation (S/D) of $895.083. Three S/Ds top the right gives value at $2,885.556, way less than the maximum value at $99,000. Furthermore, looking at the percentile chart of the price attribute (See Appendix A - Exploratory Notebook, Figure 11), the 99 percentile gives value at $1,262.00. We can conclude that there are outliers on the upper range. With this, 429 listings that has price higher than $1,262.00 are considered as high priced. Distribution is shown in geospatial map described the distribution of high-priced Airbnb listings in each neighborhood group (See Appendix A - Exploratory Notebook, Figure 4).

Minimum Nights

Minimum nights attribute has a bimodal distribution and very right skewed. From the distribution plot (See Appendix A - Exploratory Notebook, Figure 12), there are extreme outliers on the upper range. The two peaks in the distribution are for 1 minimum night with 7,946 counts (18.5%), and 30 minimum nights with 18,235 counts (42.47%) (See Appendix A - Exploratory Notebook, Figure 13). Majority of hosts are comfortable with short-term stay, Looking at the percentile distribution (See Appendix A - Exploratory Notebook, Figure 14), the 99 percentile gives value at 90 days which is 3 months, the 100 percentile is 1,250 days which is three and a half years. We can say that for listings beyond the 99 percentile thresholds is considered as a long-term stay; or Airbnb hosts that offer long-term rental. There are a total of 278 listings that have minimum nights of staying more than three months, with corresponding average price of $250.78 – higher than average Airbnb price of $200.307.

Availability 365

Availability 365 shows how many days per year the listing is available with minimum value of 0 day, and maximum value of 365 days. Looking at the distribution of availability day (See Appendix A - Exploratory Notebook, Figure 15), the two extreme values 0 and 365 have the highest frequency count: 13,990 and 2,377 respectively (See Appendix A - Exploratory Notebook, Figure 16). The high frequency count for 0 availability days suggests that there are listings that are rarely available for booking. This could be due to various reasons, such as the listing being permanently unavailable, always fully booked in advance, or being blocked off by the host for personal use.

Number of reviews, Last Reviews, Reviews per Month, and Number of Reviews Last Twelve Months.

Appendix A - Exploratory Notebook, Figure 17 visualizes the relationship between full range of price and number of reviews. The plot does not really depict enough the changes in either attribute or its effect toward the other. Therefore, we will re-plot with price less than 99 percentile value of $1,262 with corresponding number of reviews (See Appendix A - Exploratory

Notebook, Figure 18). We can see that majority of total reviews are within 0 to 500 reviews, with corresponding listing prices between 0 and $500. A pattern we can see here is that the cheaper the listing price, the higher number of reviews they received. Tourist prefers lower and reasonable listing price with resulting high number of reviews. In addition, total number of reviews for Airbnb also increases throughout times (reviews (See Appendix A - Exploratory Notebook, Figure 19). There is an interesting peak at the beginning of 2020 which was when COVID-19 started. This potentially happens due to tourists were more cautious with hygiene and safety measures, thus leave reviews for that matter.

Looking at the percentile distribution of total number of reviews (See Appendix A - Exploratory Notebook, Figure 20), the 99 percentile gives reviews value at 270 reviews and 100 percentile gives maximum value of 1,842 reviews. The listing (ID = 44799007) with highest total number of reviews is in Manhattan, entire home/apartment room type with a total of 1,842 reviews (See Appendix A - Exploratory Notebook, Figure 21). Geospatial map represents total of reviews distribution with pints at listings have high number of reviews greater than 270 (See Appendix A - Exploratory Notebook, Figure 22). Majority of listings on average has 25 reviews, Airbnb with high reviews is also in popular neighborhood groups like Manhattan and Brooklyn, some scattered in Queens. Reviews per months and number of reviews in the past 12 months are highly correlated to each other. The listing (ID = 53843545) with highest total number of reviews per month also with the highest total number of reviews in the last twelve months. Also located in Manhattan, has private room type with a total of 1,097 reviews.

Data Preprocessing

Reviews Score Notebook

In pre-processing the Reviews dataset, we have had many trials and errors to finally go with the RoBERTa pretrained model for sentiment analysis. First looking at common and non-common IDs between Listings and Reviews datasets, Airbnb listings data have 42,931 observations and are all unique. However, benchmarking with the Reviews dataset, there are 32,627 common IDs between these two. Therefore, at this stage, we decide that the combined dataset after generating reviews rating score will have 32,627 observations.

The original Reviews dataset has 1110024 observations and 6 attributes (See Appendix B – Reviews Score Notebook, Figure 1). We extract two important attributes "listing_id" and "comments" for scoring system calculations. First, we attempted to stem each review text to a simple version of each word; example: 'running' → 'run' using PorterStemmer module from the Natural Language Toolkit (nltk) library, then apply SentimentIntensityAnalyzer module to calculate polarity scores on all pre-processed reviews (See Appendix B – Reviews Score Notebook, Figure 2). The first sentiment analysis performs directly on stemmed text reviews, and returns four types of scores: negative, neutral, positive and compound score (See Appendix B – Reviews Score Notebook, Figure 3). In addition, we also had used the first sentiment analysis result as a prototype to create our recommendation system while running the RoBERTa algorithm.

The second sentiment analysis uses RoBERTa pre-trained model. We also attempted to lemmatize reviews to convert each input text into dictionary-based words. However, RoBERTa can take input texts as in original format, so we decided to skip lemmatization. Function polarity_scoreRoberta tokenizes the input text using RoBERTa tokenizer and encode it as PyTorch framework format tensors, then Pass encoded text to pre-trained RoBERTa model to perform sequence classification. The function returns three scores from the model's output: negative, neutral, and positive; then apply softmax() normalization to ensure scores are transformed into a probability distribution. Give that, we run polarity score using RoBERTa on the whole Reviews dataset and store in roberta_resPolarity list (See Appendix B – Reviews Score Notebook, Figure 4). The entire process took 21 hours and score results are written in csv format (reviewsScore.csv). The final reviews score dataset has 1,106,421 observations with 32,607

unique listing IDs (See Appendix B – Reviews Score Notebook, Figure 5). The reason that we chose RoBERTa for scoring system because the model was trained on extremely large dataset and able to recognize complex text patterns which we found is the most effective way for sentiment analysis on Reviews.

Clusters Listing Name Notebook

For the next step, we apply clustering on Airbnb listings 'name'. Also, process the Subway dataset and assign nearest subway station to each listing. The Subway dataset has 473 observations and 6 attributes (See Appendix C – Clusters Listing Name Notebook, Figure 1). We extract three important attributes for distance calculations which are "OBJECTID" – subway station ID, "NAME" – subway station name, and "the_geom" – corresponding longitude and latitude altogether. Using geopy library, specifically geodesic module to calculate the distance in miles between each listing and each subway station using longitude and latitude of each location. Function def nearestStation returns the longitude and latitude of the nearest subway station for each listing with total run time of 1.5 hour (See Appendix C – Clusters Listing Name Notebook, Figure 2). Function def nearestDistance returns the distance in miles between the listings and its nearest subway station (See Appendix C – Clusters Listing Name Notebook, Figure 3).

Then we perform left join between Listings and Subway station datasets into a new ‘mergedData’ data frame. At this point, there are six additional columns: ‘NearestStationLatitude’, ‘NearestStationLongitude’, ‘Distance’, ‘Subway_longitude’, ‘Subway_latitude’, and ‘NAME’.

The second step is to apply clustering Listings ‘name’. Using sci-kit learn library, especially TfidfVectorizer and Kmeans modules for clustering. Start with extracting "id" and "name" attributes from Listings dataset into ‘description’ data frame (See Appendix C – Clusters Listing Name Notebook, Figure 4), we apply function def stem to stem each ‘name’ attribute text to simple version of each word. Function def customTokenizer returns word tokens from input text by filtering out tokens that represent numeric values and numbers followed by a single quote; example: '101cozy'. Perform TF-IDF on the text data in 'name' attribute with the English stop words excluded (condition set in TF-IDF vectorizer instance). Obtain a list of feature names

corresponding to the columns in the TF-IDF matrix and convert to pandas data frame named ‘featuresData’. Then cross check with stop word list ‘removingPattern’ to remove any redundant or stop words in ‘featuresData’ with run time of 30 minutes. Finally, ‘featuresData’ data frame is TF-IDF format of relevant words in Listings ‘name’, rows of 42,931 observations and columns of 6,245 words (See Appendix C – Clusters Listing Name Notebook, Figure 5). We will use ‘featuresData’ to perform K-means clustering.

For clustering, we try different numbers of clusters from 1 to 14 and store Within-Cluster Sum of Squares (WCSS) results to generate Elbow method plot. WCSS measures the tightness of a cluster using Euclidean distance, the smaller WCSS value, the closer data points surround cluster centroid. Finally, choosing total number of clusters k=9, threshold with clear Elbow curve (See Appendix C – Clusters Listing Name Notebook, Figure 6 and Figure 7). The clustering results are stored in the ‘description’ data frame - written in csv format (listingsCluster.csv). Appendix C – Clusters Listing Name Notebook, Figure 8 shows each cluster in frequency descending order. Then we perform left join between ‘mergedData’ and ‘description’ data frames into a new ‘mergedData2’ data frame with one additional column ‘cluster’.

The reviewsScore dataset contains three scores resulted from the RoBERTa pre-trained model stored altogether as dictionary. Using Abstract Syntax Trees (ast) library to parse ‘polarity_score’, the reviewsScore dataset finally has 4 attributes: ‘id’ – listings ID, ‘neg’ - negative score, ‘neu’ – neutral score, and ‘pos’ – positive score (See Appendix C – Clusters Listing Name Notebook, Figure 9). Since reviewsScore has 1,106,421 observations and 32,607 unique listings ID, we group Reviews scores with same listing ID using mean value and assign to a new data frame ‘groupedIDs’. Then perform left join between original Listings and ‘groupedIDs’ (now with 32,607 observations) data frames into a new ‘mergedData3’ data frame with three additional column ‘neg’, ‘neu’, and ‘pos’.

At this point, there are processed data frames. ‘mergedData2’ has full Listings dataset, nearest subways station features, and clustered Listings ‘name’; only missing reviews rating scores. ‘mergedData3’ has full Listings dataset and reviews rating scores. We will combine these two for final dataset.

Final Dataset

Clusters Listing Name Notebook

Perform left join on ‘mergedData2’ and ‘mergedData3’ data frames, drop duplicate attributes: 'NearestStationLatitude_x', 'NearestStationLongitude_x', and 'Distance_x', the ‘finalData’ dataset has 32,607 observations and 28 observations. Specifically, 18 attributes from original Listings dataset detailed from the **Introduction** section. The additional 10 attributes are:

- neg: Negative polarity score for Reviews (dtype = float).
- new: Neutral polarity score for Reviews (dtype = float).
- pos: Positive polarity score for Reviews (dtype = float).
- NearestStationLatitude: Nearest subway station latitude (dtype = float).
- NearestStationLongitude: Nearest subway station longitude (dtype = float).
- Distance: Distance in mile(s) between the listing and its nearest subway station (dtype = float).
- Subway_longitude: Longitude of subway station ‘NAME’, should be similar to ‘NearestStationLongitude’ (dtype = float).
- Subway_latitude: Latitude of subway station ‘NAME’, should be similar to ‘NearestStationLatitude’ (dtype = float).
- NAME: Name of the nearest subway station (dtype = string).
- cluster: cluster number based on listings name/description (dtype = integer).

The final dataset is written in csv format (finalDataset.csv) (See Appendix C – Clusters Listing Name Notebook, Figure 10).

Content-Based Recommendation System

Recommendation System Notebook

Using the final dataset ‘finalDataset.csv’ moving forward, remove nineteen attributes that are not used in the reco9mmendation system and assign to a new dataset ‘Final_df’ (32,607 observations and 9 attributes) (See Appendix D – Recommendation System Notebook, Figure 1):

- 'name', 'host_id', 'host_name', 'calculated_host_listings_count', 'room_type', 'minimum_nights', 'availability_365': Not using for recommendation system, used for exploratory and geospatial map.
- 'latitude', 'longitude', 'NearestStationLatitude', 'NearestStationLongitude', 'Subway_longitude', 'Subway_latitude': Replaced by ‘Distance’ and subway station ‘NAME’.
- 'number_of_reviews', 'last_review', 'reviews_per_month', 'number_of_reviews_ltm' : Replaced by Reviews scores ‘neg’, ‘neu’, ‘pos’.
- 'license' : Removed due to only 1 observation has value.
- 'neighbourhood': too many values, replaced by ‘'neighbourhood_group'.

Compound Reviews Score

Create a compound rating score (on a scale of 1 to 5) that represents the three polarity scores ‘neg’ , neu’, and ‘pos’ from the RoBERTa pretrained model. Start with extract the scores to a new data frame ‘ratings’, rename and assign score weight to each score feature: 'Rating Negative': 1, 'Rating Neutral': 3, and 'Rating Positive': 5. Multiply each rating with corresponding weights and sum up for original compound score (dtype = float). Then there are two options to convert compound score to integer: scale the original compound scores to a normalized range of 1-5; or rounding the original compound score to the nearest integer value (See Appendix D – Recommendation System Notebook, Figure 2 and Figure 3). We decide to go with the second option. With final compound score, ‘recommendData’ is used for recommendation system has 32,607 observations and 7 attributes with ‘Rating’ represents compound Reviews score.

Recommendation System

Create function def preprocessText to return users' input text to all lowercase. User will be asked to give references of three things: price point, neighborhood groups (Manhattan, Brooklyn, Queens, Bronx, and Staten Island), and reviews ratings (1-5). From 'recommendData' data frame, the system will obtain listings with price less than user's input, in neighborhood group with user's references, and has rating higher than user's input. Then transform user input into a vector representation to calculate the similarity between the user input and listings features using cosine similarity. Finally, sort the listings based on similarity in descending order; and display the top recommended listings which we choose five in this case, and corresponding nearest subway station name and distance in mile(s) (See Appendix D – Recommendation System Notebook, Figure 4).

Conclusion

Throughout the project, we were able to utilize different datasets and techniques to successfully create a content-based recommendation system. For data exploratory, we applied techniques from the course like descriptive statistics and boxplots, to further practice like geospatial map. For Reviews rating score, we learn in-depth about natural language processing and came across RoBERTa algorithm, also experience with long time processing. With Listings 'name', we use class materials like K-means and TF-IDF, also explore new distance calculation library like geopy for Subway dataset. At the end, combine everything and create a recommendation system, we have reached our main goal for the project, and also as learning outputs from the course.

There are still much more room for improvements throughout data pre-processing and enhance the recommendation system; for example: processing users input. Indeed, this is not the best model can be built from these three datasets. However, using all techniques and knowledge that we have learned so far. We proudly to present this recommendation system and associated processes.

Appendix A. Exploratory Notebook

	count	unique	top	freq	first	last	mean	std	min	25%	50%	75%	max
id	42931.0	NaN	NaN	NaN	NaT	NaT	222277244526372224.0	334421302203795328.0	2595.0	19404736.0	43374815.0	630501597262169088.0	840466047136140160.0
host_id	42931.0	NaN	NaN	NaN	NaT	NaT	151601209.021919	162130107.567557	1678.0	16085328.0	74338125.0	268069240.5	503872891.0
host_name	42926	9831	Blueground	526	NaT	NaT	NaN	NaN	NaN	NaN	NaN	NaN	NaN
neighbourhood_group	42931	5	Manhattan	17658	NaT	NaT	NaN	NaN	NaN	NaN	NaN	NaN	NaN
neighbourhood	42931	223	Bedford-Stuyvesant	3086	NaT	NaT	NaN	NaN	NaN	NaN	NaN	NaN	NaN
latitude	42931.0	NaN	NaN	NaN	NaT	NaT	40.728273	0.05764	40.500314	40.687485	40.72404	40.762293	40.91138
longitude	42931.0	NaN	NaN	NaN	NaT	NaT	-73.943665	0.056627	-74.251907	-73.98175	-73.95262	-73.924035	-73.71087
room_type	42931	4	Entire home/apt	24279	NaT	NaT	NaN	NaN	NaN	NaN	NaN	NaN	NaN
price	42931.0	NaN	NaN	NaN	NaT	NaT	200.307167	895.082911	0.0	75.0	125.0	200.0	99000.0
minimum_nights	42931.0	NaN	NaN	NaN	NaT	NaT	18.111178	27.462513	1.0	2.0	7.0	30.0	1250.0
number_of_reviews	42931.0	NaN	NaN	NaN	NaT	NaT	25.856001	56.616344	0.0	1.0	5.0	24.0	1842.0
last_review	32627	2795	2023-01-02 00:00:00	839	2011-05-12	2023-03-06	NaN	NaN	NaN	NaN	NaN	NaN	NaN
reviews_per_month	32627.0	NaN	NaN	NaN	NaT	NaT	1.168988	1.789675	0.01	0.14	0.52	1.67	86.61
calculated_host_listings_count	42931.0	NaN	NaN	NaN	NaT	NaT	24.054809	80.867958	1.0	1.0	1.0	4.0	526.0
availability_365	42931.0	NaN	NaN	NaN	NaT	NaT	140.262211	142.001486	0.0	0.0	89.0	289.0	365.0
number_of_reviews_ltm	42931.0	NaN	NaN	NaN	NaT	NaT	7.736507	18.290256	0.0	0.0	0.0	7.0	1093.0
license	1	1	41662/AL	1	NaT	NaT	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 1. Original Listings dataset – Descriptive Statistics.

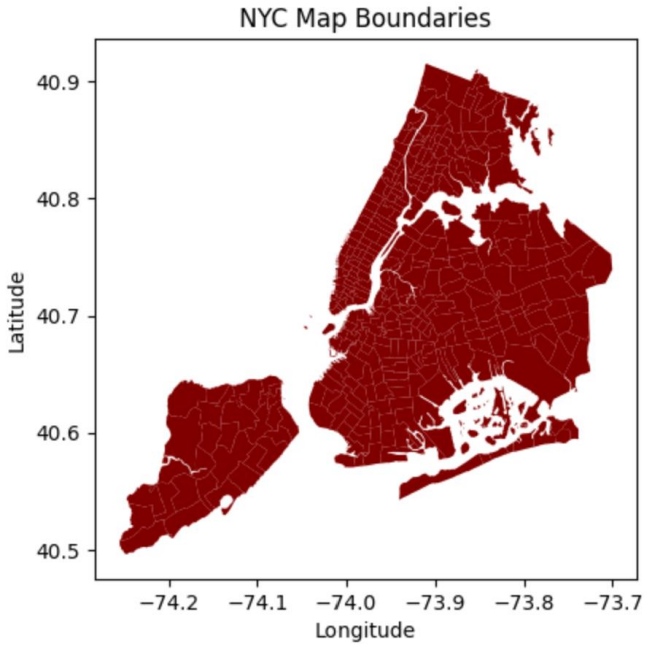

Figure 1. NYC Map Boundaries.

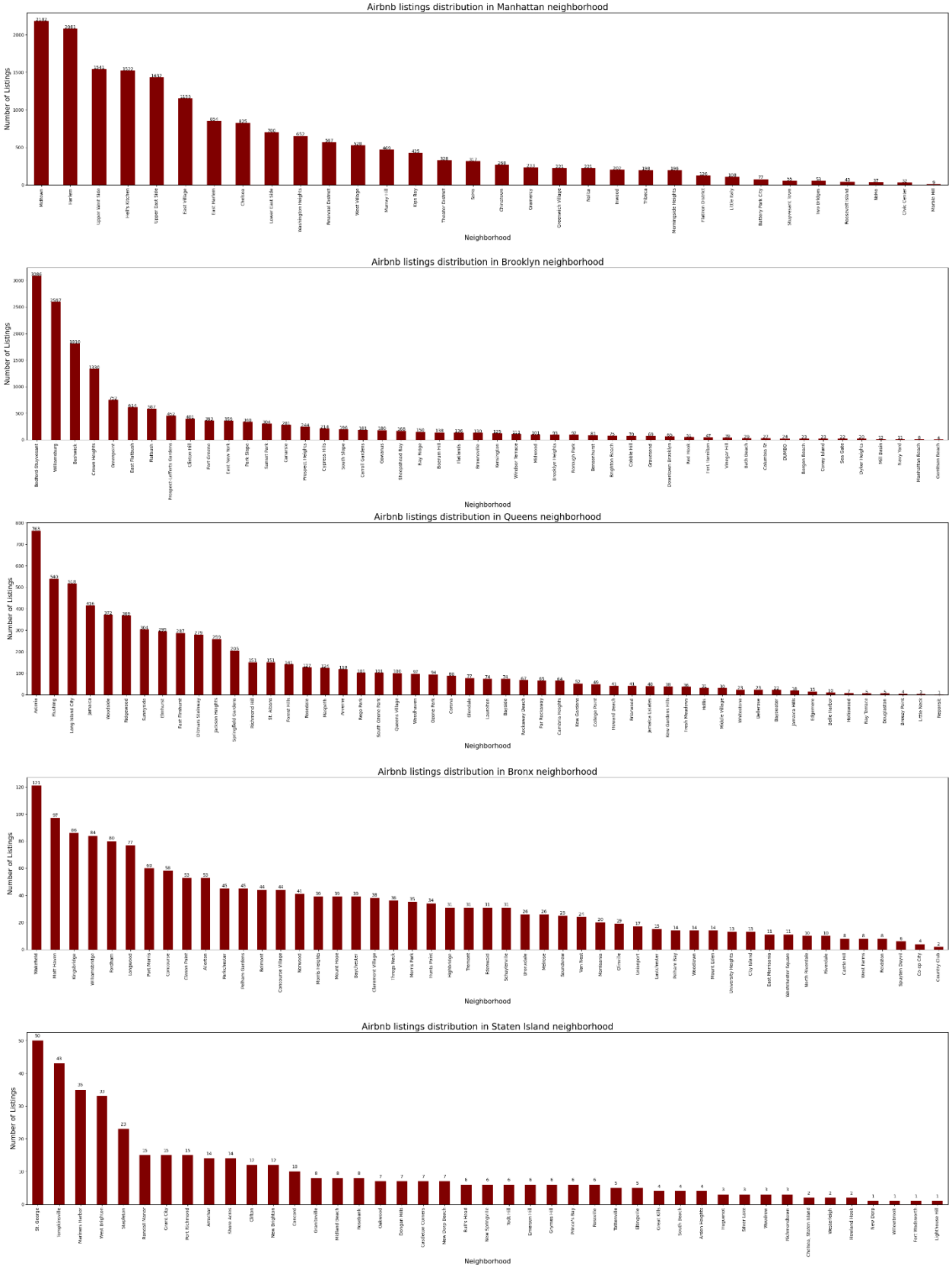

Figure 2. Distributions of Airbnb listings in each neighborhood in corresponding neighborhood group.

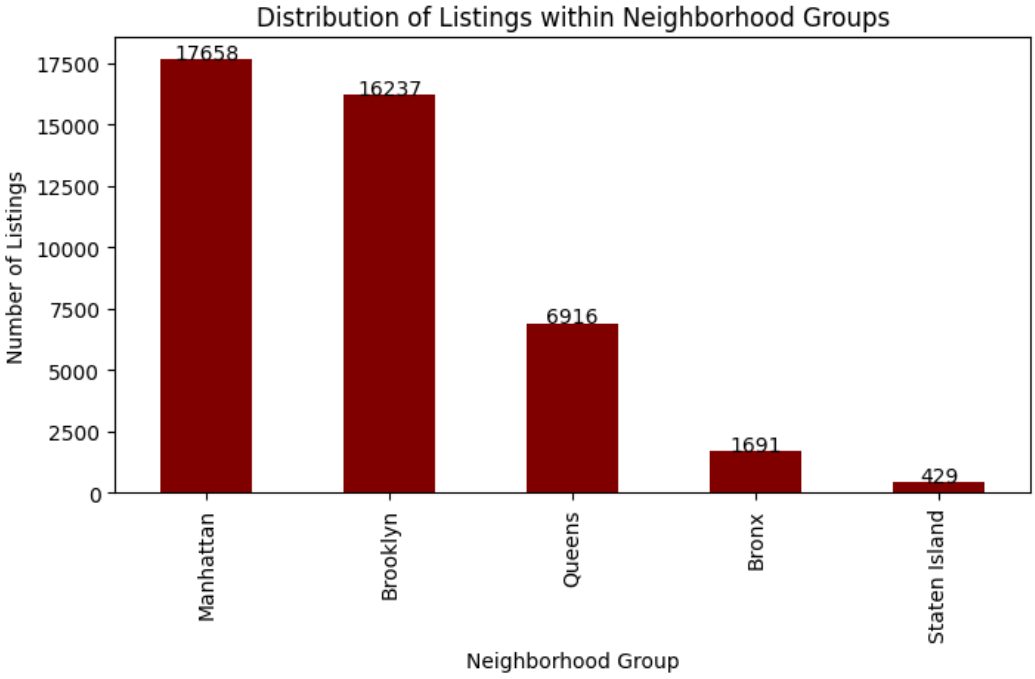

Figure 3. Distributions of Airbnb listings in each neighborhood group.

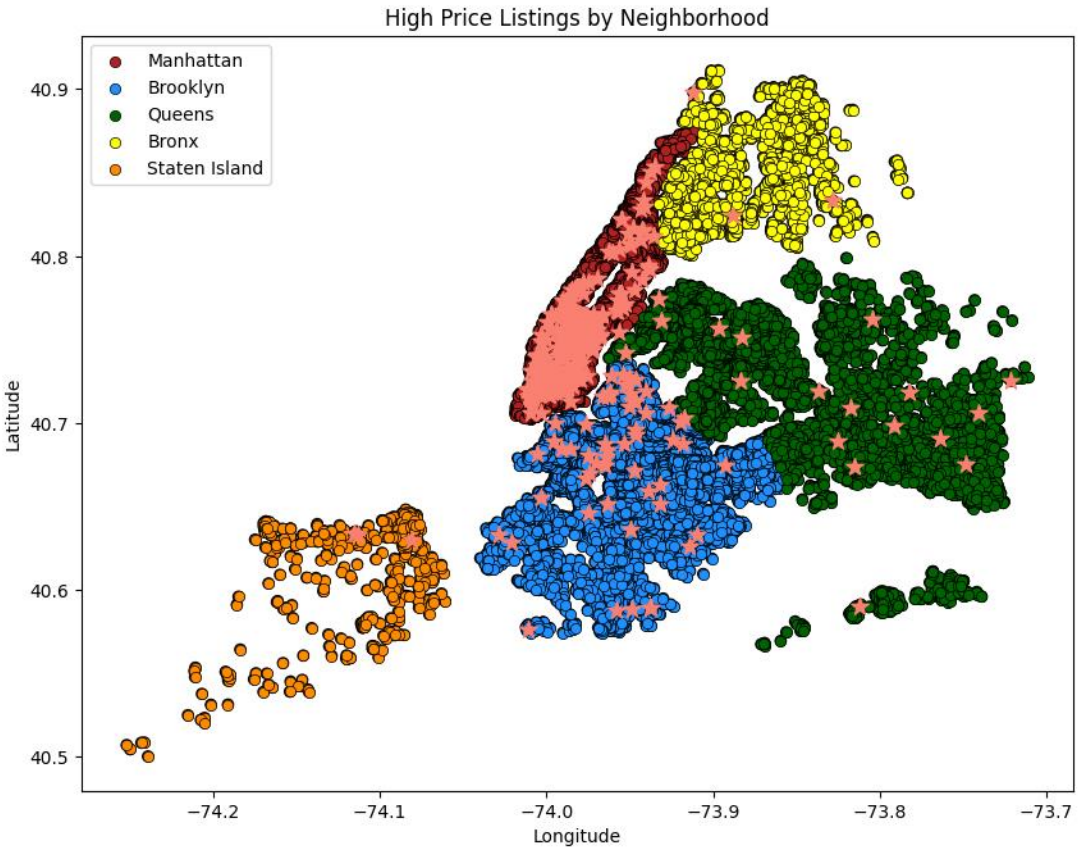

Figure 4. Geospatial map for each neighborhood group and high-priced Airbnb listings.

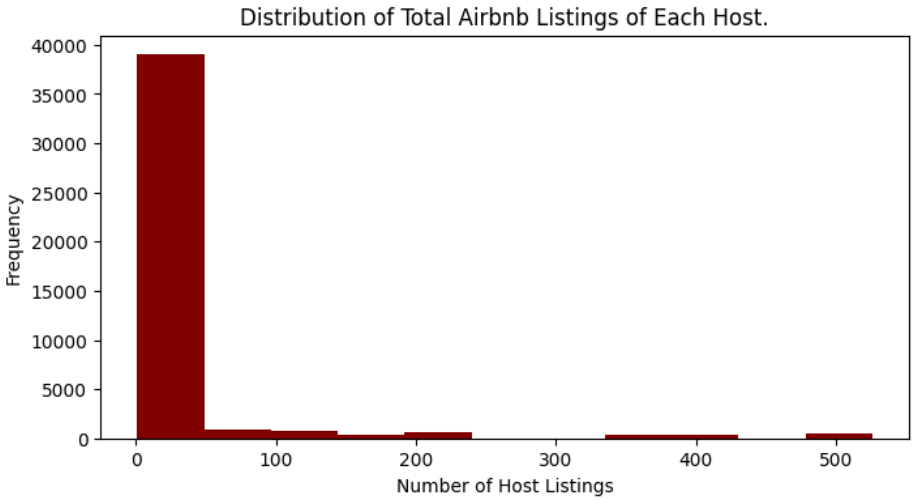

Figure 5. Distribution of total Airbnb listings each host owns.

```
Host: Justin                  Listings Count: 101
Host: RoomPicks By Antony     Listings Count: 107
Host: Ken                     Listings Count: 113
Host: RoomPicks By Victoria   Listings Count: 120
Host: Kaz                     Listings Count: 124
Host: Stay With Vibe          Listings Count: 124
Host: Jeniffer                Listings Count: 131
Host: Urban Furnished         Listings Count: 159
Host: Momoyo                  Listings Count: 178
Host: Shogo                   Listings Count: 192
Host: Hiroki                  Listings Count: 207
Host: June                    Listings Count: 222
Host: RoomPicks               Listings Count: 356
Host: Eugene                  Listings Count: 394
Host: Blueground              Listings Count: 526
```

Figure 6. Host names that have more than 100 Airbnb listings.

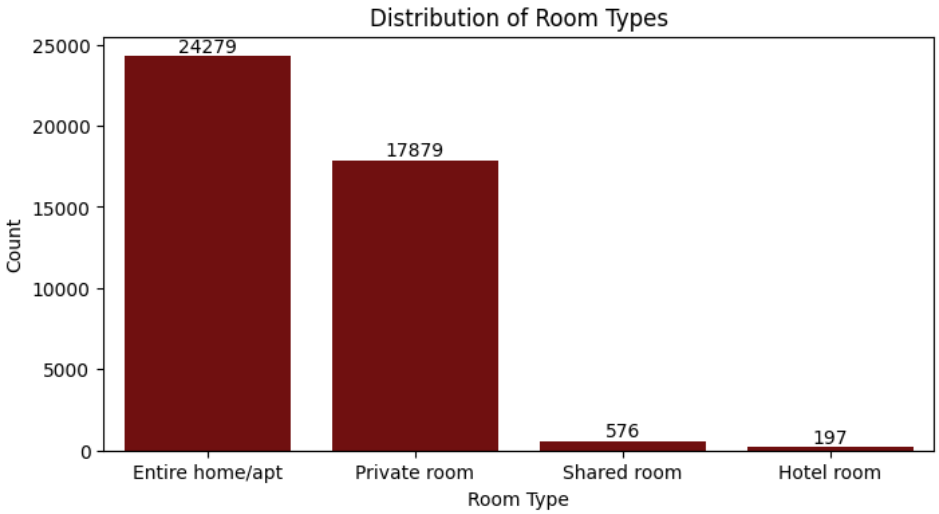

Figure 7. Distribution of Airbnb room types.

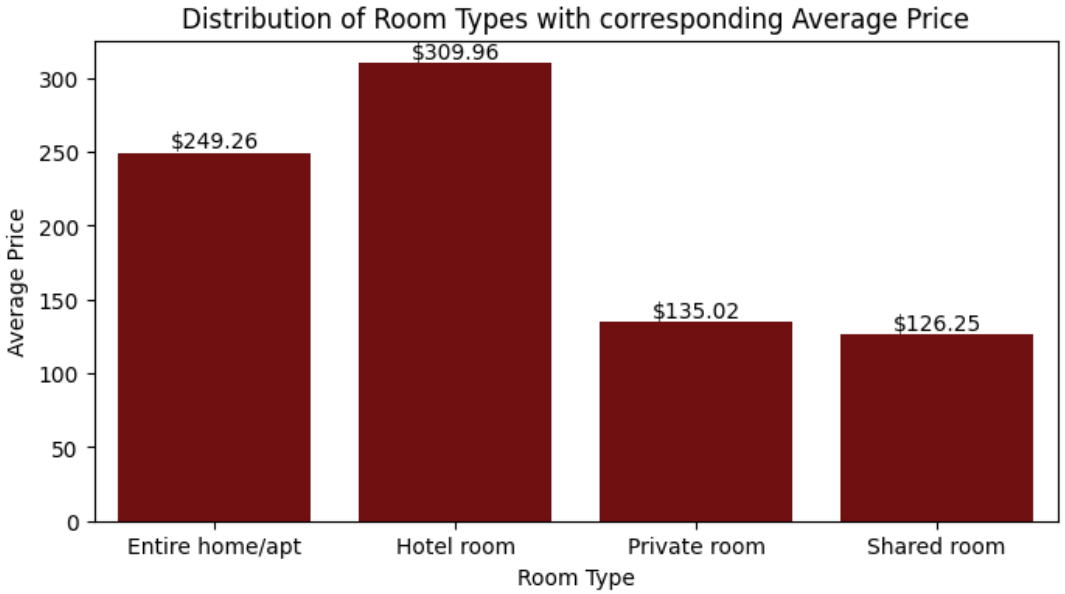

Figure 8. Distribution of room types with average price for each room type.

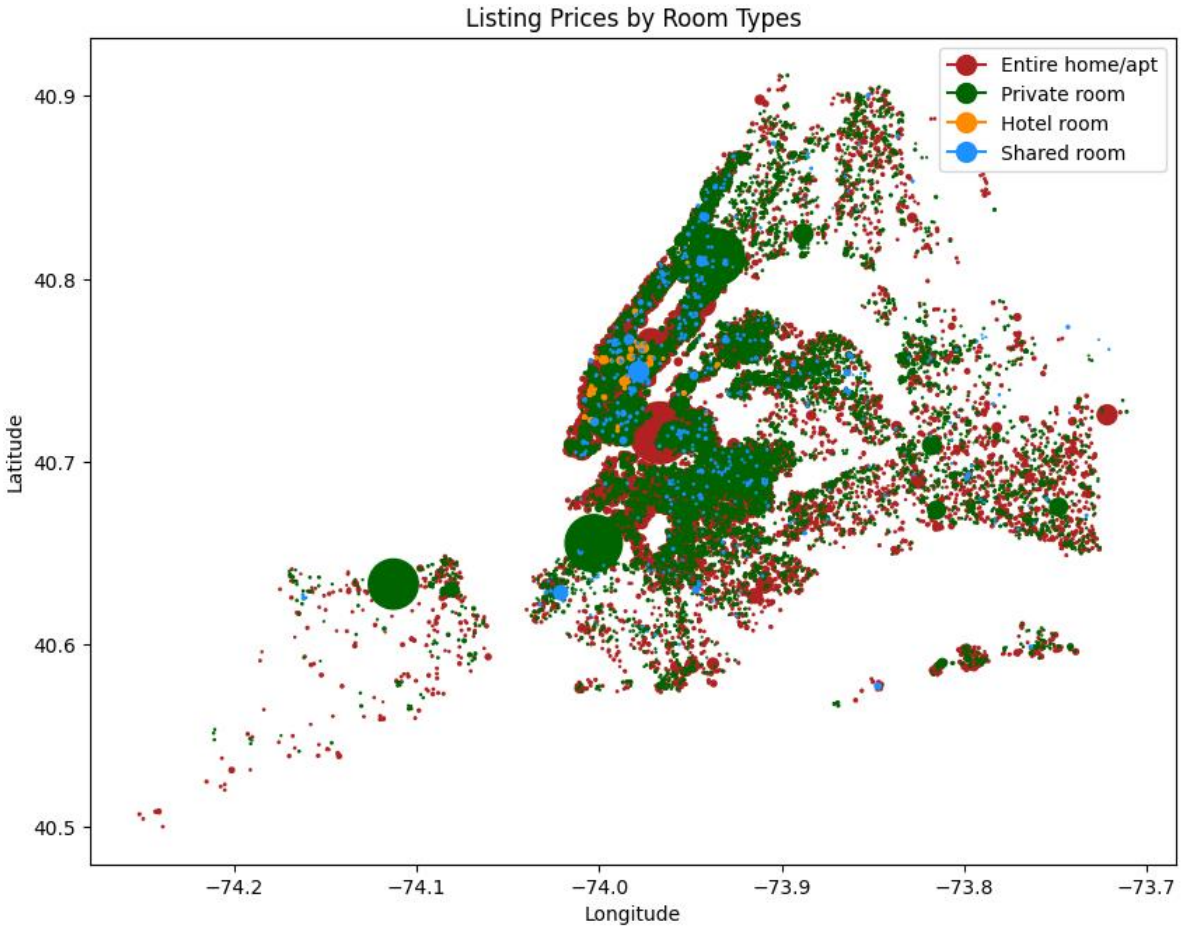

Figure 9. Geospatial map for Airbnb room types with corresponding price.

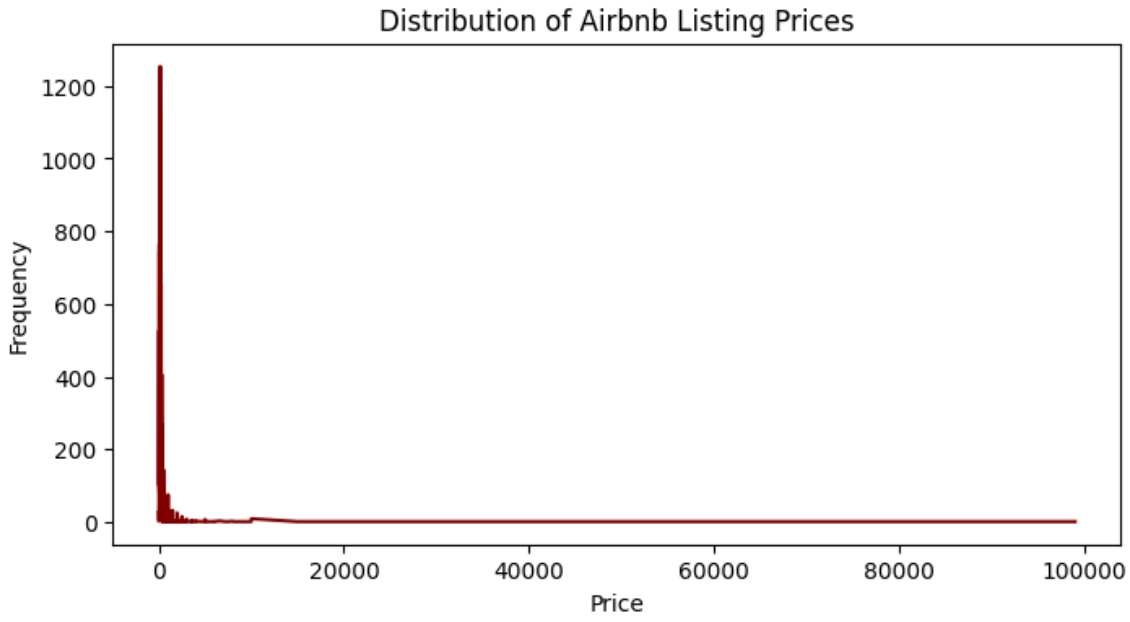

Figure 10. Distribution of Price attribute.

```
Percentile 75: 200.0
Percentile 76: 208.0
Percentile 77: 215.0
Percentile 78: 221.0
Percentile 79: 227.0
Percentile 80: 236.0
Percentile 81: 245.0
Percentile 82: 250.0
Percentile 83: 250.0
Percentile 84: 263.0
Percentile 85: 275.0
Percentile 86: 285.0
Percentile 87: 298.0
Percentile 88: 300.0
Percentile 89: 318.0
Percentile 90: 339.0
Percentile 91: 350.0
Percentile 92: 376.59999999999854
Percentile 93: 400.0
Percentile 94: 444.0
Percentile 95: 499.0
Percentile 96: 550.0
Percentile 97: 686.0999999999985
Percentile 98: 878.4000000000015
Percentile 99: 1262.0
Percentile 100: 99000.0
```

Figure 11. Percentile distribution for price attribute.

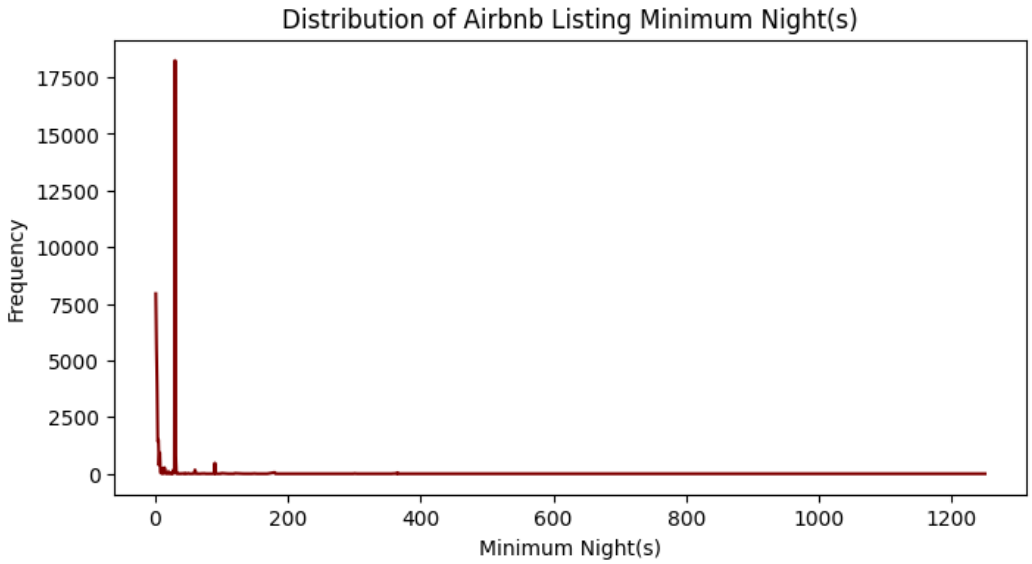

Figure 12. Percentile distribution for minimum nights attribute.

```
Night count frequency:
280        1
88         1
48         1
273        1
36         1
         ...
5       1499
3       3814
2       5524
1       7946
30     18235
Name: minimum_nights, Length: 128, dtype: int64
```

Figure 13. Minimum nights count frequency.

```
Percentile 85: 30.0
Percentile 86: 30.0
Percentile 87: 30.0
Percentile 88: 30.0
Percentile 89: 30.0
Percentile 90: 30.0
Percentile 91: 30.0
Percentile 92: 30.0
Percentile 93: 30.0
Percentile 94: 30.0
Percentile 95: 30.0
Percentile 96: 31.0
Percentile 97: 31.0
Percentile 98: 60.0
Percentile 99: 90.0
Percentile 100: 1250.0
```

Figure 14. Percentile distribution for minimum nights attribute.

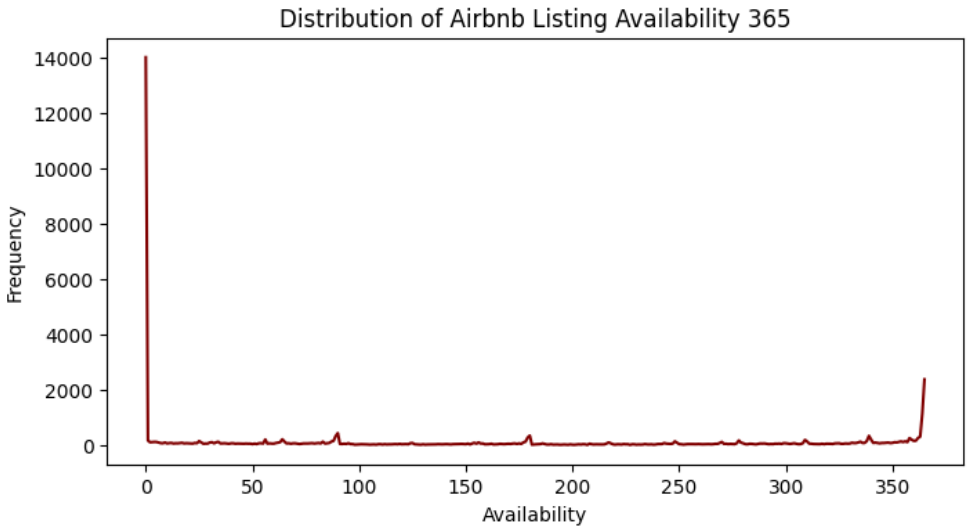

Figure 15. Distribution of availability 365.

```
Night count frequency:
207       18
227       18
198       22
201       22
194       23
         ...
180      352
90       442
364     1152
365     2377
0      13990
Name: availability_365, Length: 366, dtype: int64
```

Figure 16. Availability 365 count frequency.

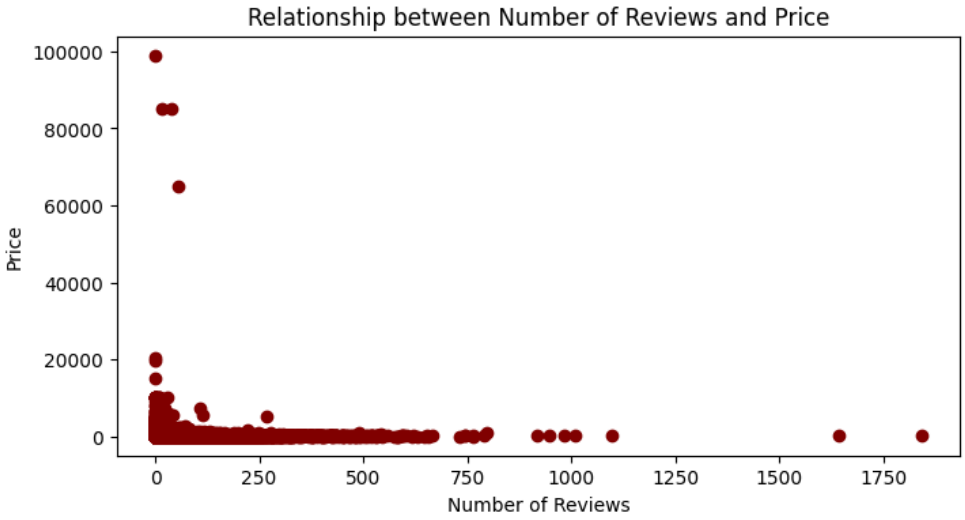

Figure 17. Scatterplot full range Price versus Number of Reviews.

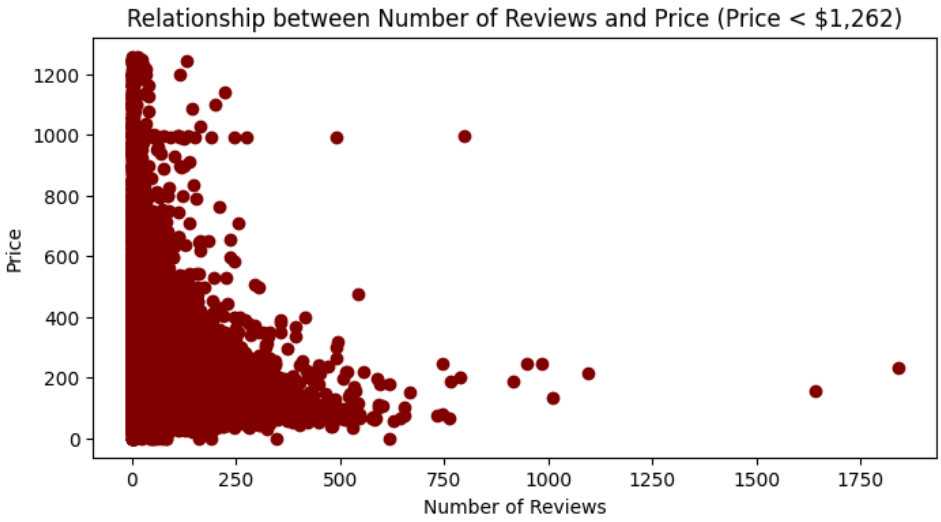

Figure 18. Scatterplot Price less than $1,262 versus corresponding Number of Reviews.

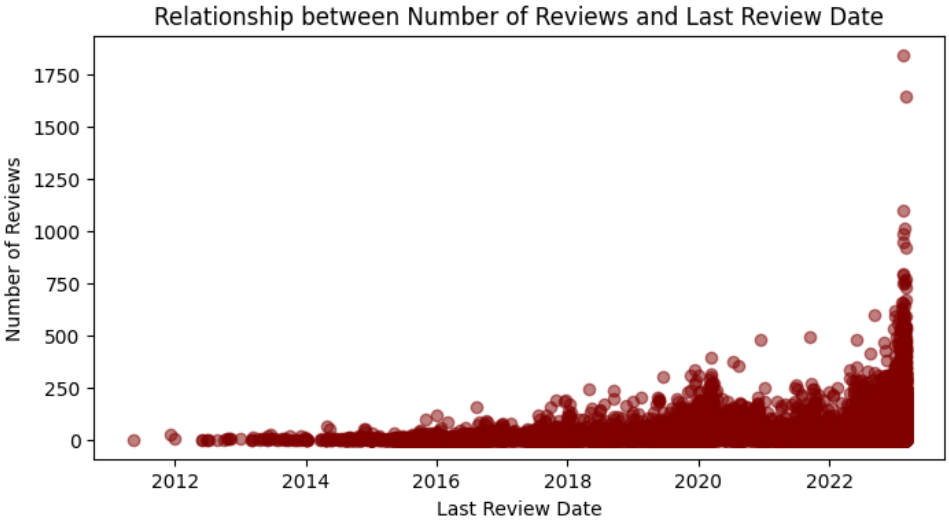

Figure 19. Scatterplot Last Review Date versus Number of Reviews.

```
Percentile 85: 49.0
Percentile 86: 53.0
Percentile 87: 57.0
Percentile 88: 62.0
Percentile 89: 68.0
Percentile 90: 75.0
Percentile 91: 82.0
Percentile 92: 92.0
Percentile 93: 102.0
Percentile 94: 115.0
Percentile 95: 129.0
Percentile 96: 147.0
Percentile 97: 171.0
Percentile 98: 206.0
Percentile 99: 270.0
Percentile 100: 1842.0
```

Figure 20. Percentile distribution for number of reviews attribute.

id	44799007
name	Sonder Battery Park | Studio Apartment
host_id	219517861
host_name	Sonder (NYC)
neighbourhood_group	Manhattan
neighbourhood	Financial District
latitude	40.70617
longitude	-74.01486
room_type	Entire home/apt
price	234
minimum_nights	2
number_of_reviews	1842
last_review	2023-02-19 00:00:00
reviews_per_month	61.26
calculated_host_listings_count	76
availability_365	279
number_of_reviews_ltm	770
license	NaN

Name: 22047, dtype: object

Figure 21. Listing with highest Number of Reviews.

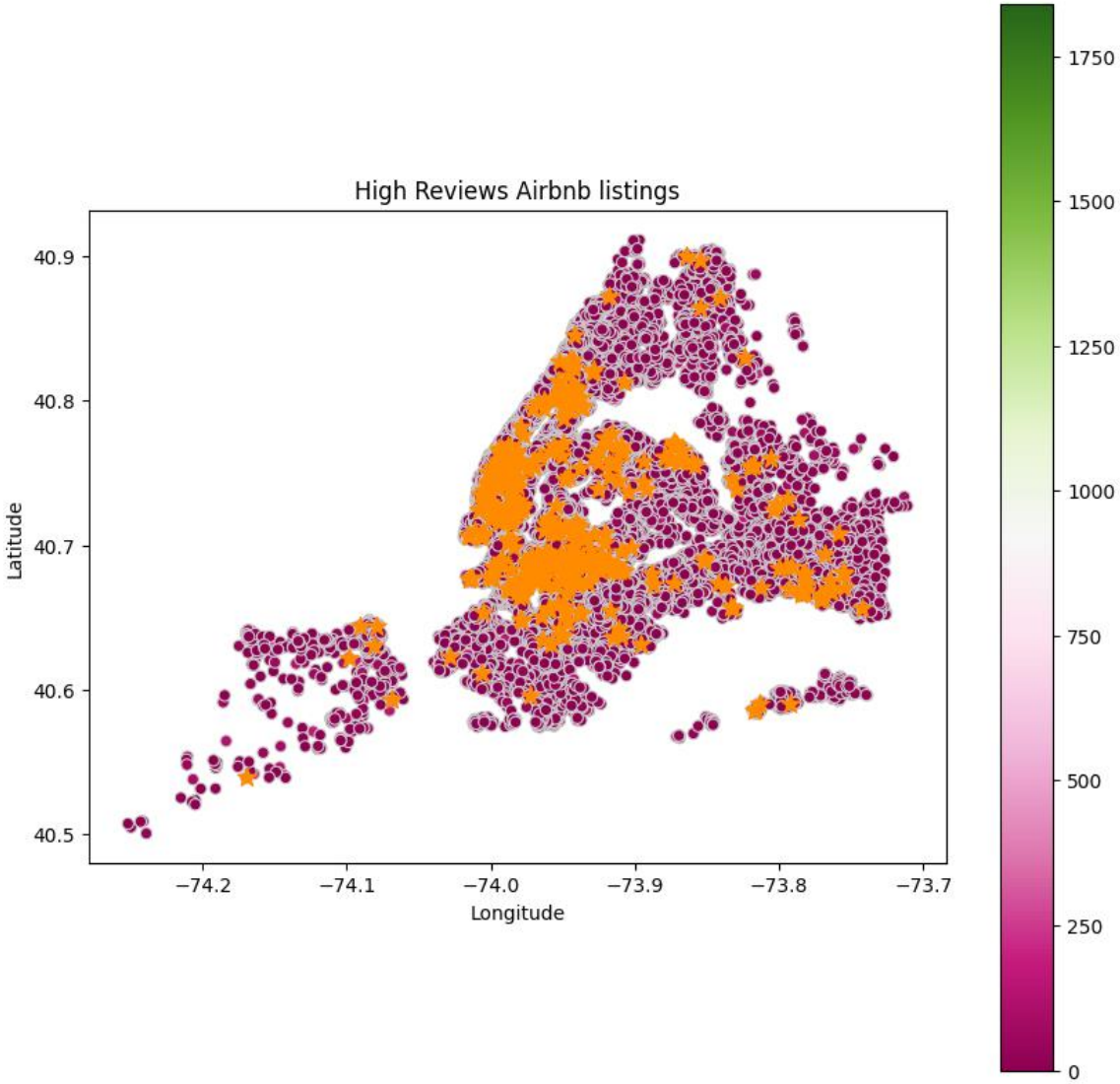

Figure 22. Geospatial map for Airbnb total number of reviews.

Appendix B. Reviews Score Notebook

```
reviews = pd.read_csv("reviews.csv")
print (f" Reviews dataset shape: {reviews.shape}")
reviews.head(10)
#Read Reviews datsaset.
```

```
Reviews dataset shape: (1110024, 6)
```

	listing_id	id	date	reviewer_id	reviewer_name	comments
0	2595	17857	2009-11-21	50679	Jean	Notre séjour de trois nuits.\r Nous avons ...
1	2595	19176	2009-12-05	53267	Cate	Great experience.
2	2595	19760	2009-12-10	38960	Anita	I've stayed with my friend at the Midtown Cast...
3	2595	34320	2010-04-09	71130	Kai-Uwe	We've been staying here for about 9 nights, en...
4	2595	46312	2010-05-25	117113	Alicia	We had a wonderful stay at Jennifer's charming...
5	2595	1238204	2012-05-07	1783688	Sergey	Hi to everyone!\r Would say our greatest c...
6	2595	1293632	2012-05-17	1870771	Loïc	Jennifer was very friendly and helpful, and he...
7	2595	2022498	2012-08-18	2124102	Melanie	This apartment is like a real castle old and u...
8	2595	4682989	2013-05-20	496053	Eric	Jennifer's place was in a great midtown locati...
9	2595	13193832	2014-05-21	13685934	Gerald	Jennifer is a very nice host. Everything is cl...

Figure 1. Reviews dataset.

```
[ ] from nltk.stem.porter import PorterStemmer
    from tqdm.notebook import tqdm
    ps = PorterStemmer()
    #Text normalization - stemming algorithm.

    from nltk.sentiment import SentimentIntensityAnalyzer
    #Import library to perform sentiment analysis.
    nltk.downloader.download('vader_lexicon')
    sia = SentimentIntensityAnalyzer()
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
[ ] #Example
    sia.polarity_scores("This product is awesome")
```

```
{'neg': 0.0, 'neu': 0.423, 'pos': 0.577, 'compound': 0.6249}
```

```
[ ] def stem(text):
      '''Takes input text and stem it to simple version of each word.
      Ex: 'running' --> 'run' '''
        ps = PorterStemmer()
        y = []
        if isinstance(text, str):  #Check if text is a non-null string.
            for i in text.split():
                y.append(ps.stem(i))
        return " ".join(y)
```

```
[ ] reviewsData["comments"] = tqdm(reviewsData["comments"].apply(lambda x: ps.stem(x) if isinstance(x, str) else x))
    #Apply def stem to each "comments" instance in reviews dataset.
```

```
100% 1110024/1110024 [00:00<00:00, 1378370.77it/s]
```

Figure 2. Stemming text data.

Run polarity score on all reviews. Performs sentiment analysis directly on the text Reviews.

```
[ ]  orig_resPolarity = {}

     #Apply sentiment analysis to the "comments" column with tqdm progress bar.
     for index, row in tqdm(reviewsData.iterrows(), total=reviewsData.shape[0]):
         if isinstance(row["comments"], str):
             text = row["comments"]
             id = row["listing_id"]
             orig_resPolarity[id] = sia.polarity_scores(text)
```

```
100% 1110024/1110024 [16:41<00:00, 1636.21it/s]
```

```
[ ]  orig_resPolarity=pd.DataFrame(orig_resPolarity).T
     print(orig_resPolarity.shape)
     orig_resPolarity=orig_resPolarity.reset_index().rename(columns={'index':"id"})
     orig_resPolarity.head()
```

```
(32625, 4)
```

	id	neg	neu	pos	compound
0	2595	0.035	0.810	0.155	0.9883
1	5121	0.000	0.546	0.454	0.9668
2	5136	0.000	0.607	0.393	0.9682
3	5586	0.000	0.617	0.383	0.9659
4	5178	0.557	0.114	0.330	-0.2500

Figure 3. Run polarity score and performs sentiment analysis directly on all text Reviews.

```
[ ]  def polarity_scoreRoberta(example):
       '''Take input text, tokenize using RoBERTa and returns Polarity scores as dictionary form.'''

       encodedText = tokenizer (example, return_tensors='pt')
       #Tokenize the input text using RoBERTa tokenizer and encode it as PyTorch framework format tensors.
       output = model(**encodedText)
       #Pass encodedText to pre-trained RoBERTa model to perform sequence classification.
       scores = output[0][0].detach().numpy()
       scores=softmax(scores)
       #Retrieve the scores from the model's output and apply softmax normalization to ensure scores are transformed into a probability distribution.
       scoresDict= {"roberta_neg": scores[0],
                    "roberta_neu": scores[1],
                    "roberta_pos": scores[2]}
       return scoresDict
       #Return Polarity score as dictionary form for each 'comments'.
```

roberta_resPolarity result took 21 hours.

```
[ ]  roberta_resPolarity = []

     #Apply sentiment analysis to the "comments" column with tqdm progress bar.
     for i, row in tqdm(reviewsData.iterrows(), total=len(reviewsData)):
         try:
           text = row["comments"]
           idValue = row["listing_id"]
           roberta_scorePolarity = polarity_scoreRoberta(text)
           #Obtain polarity score from RoBERTa pre-trained model.
           roberta_resPolarity.append((idValue, roberta_scorePolarity))
         except:
           print(f"Listing ID {idValue} caused error")
```

```
0% 33/1110024 [00:17<135:59:21, 2.27it/s]
Listing ID 2595 caused error
Listing ID 2595 caused error
Listing ID 2595 caused error
Listing ID 2595 caused error
```

Figure 4. Run polarity score and perform sentiment analysis on the text Reviews by RoBERTa.

```
[ ] import csv
    filename = "reviewsScore.csv"
    header = ["id_value", "polarity_score"]

    with open(filename, "a", newline="") as csvfile:
        writer = csv.writer(csvfile)
        #Check if the file is empty and write the header if needed.
        if csvfile.tell() == 0:
            writer.writerow(header)
        writer.writerows(roberta_resPolarity)

    print(f"Data appended to {filename} successfully.")
```

```
[ ] reviewsScore = pd.read_csv("reviewsScore.xls")
    print (f" Reviews score dataset shape: {reviewsScore.shape}")
    uniqueCount = reviewsScore["id_value"].nunique()
    print(uniqueCount)
    #Read Reviews datsaset.
```

```
 Reviews score dataset shape: (1106421, 2)
32607
```

```
[ ] reviewsScore.head(10)
```

	id_value	polarity_score
0	2595	{'roberta_neg': 0.13033417, 'roberta_neu': 0.8...
1	2595	{'roberta_neg': 0.0071270172, 'roberta_neu': 0...
2	2595	{'roberta_neg': 0.0017230441, 'roberta_neu': 0...
3	2595	{'roberta_neg': 0.0014082947, 'roberta_neu': 0...
4	2595	{'roberta_neg': 0.0012992539, 'roberta_neu': 0...
5	2595	{'roberta_neg': 0.0011575663, 'roberta_neu': 0...
6	2595	{'roberta_neg': 0.0016218657, 'roberta_neu': 0...
7	2595	{'roberta_neg': 0.0829413, 'roberta_neu': 0.31...

Figure 5. Polarity score from RoBERTa model.

Appendix C. Cluster Listing Name Notebook

```
[ ] subway = pd.read_csv("Subway.csv")
    print (f" Subway dataset shape: {subway.shape}")
    subway.head()
    #There are a total of 473 subway stations in NYC.
```

Subway dataset shape: (473, 6)

	URL	OBJECTID	NAME	the_geom	LINE	NOTES
0	http://web.mta.info/nyct/service/	1	Astor Pl	POINT (-73.99106999861966 40.73005400028978)	4-6-6 Express	4 nights, 6-all times, 6 Express-weekdays AM s...
1	http://web.mta.info/nyct/service/	2	Canal St	POINT (-74.00019299927328 40.71880300107709)	4-6-6 Express	4 nights, 6-all times, 6 Express-weekdays AM s...
2	http://web.mta.info/nyct/service/	3	50th St	POINT (-73.98384899986625 40.76172799961419)	1-2	1-all times, 2-nights
3	http://web.mta.info/nyct/service/	4	Bergen St	POINT (-73.97499915116808 40.68086213682956)	2-3-4	4-nights, 3-all other times, 2-all times
4	http://web.mta.info/nyct/service/	5	Pennsylvania Ave	POINT (-73.89488591154061 40.66471445143568)	3-4	4-nights, 3-all other times

Figure 1. Subway dataset.

Find the latitude and longitude of the nearest subway station. Run time 1.15 hour

```
[ ] def nearestStation(row):
      '''Returns the longitude and latitude of the nearest subway station.'''

      listingLocation = (row['latitude'], row['longitude'])
      stationLocations = list(zip(subway['Subway_latitude'], subway['Subway_longitude']))
      distances = [geodesic(listingLocation, station).meters for station in stationLocations]
      nearest_stationIndex = distances.index(min(distances))
      nearest_station = subway.iloc[nearest_stationIndex]
      return pd.Series([nearest_station['Subway_latitude'], nearest_station['Subway_longitude']])

    #Apply the nearestStation function to each row in Listings.
    #Two additional columns of longitude and latitude of nearest subway station.
    listings[['NearestStationLatitude', 'NearestStationLongitude']] = listings.apply(nearestStation, axis=1)
```

```
[ ] listings.head()
```

latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	availability_365	number_of_reviews_ltm	license	NearestStationLatitude	NearestStationLongitude
40.75356	-73.98559	Entire home/apt	150	30	49	2022-06-21	0.30	3	314	1	NaN	40.75418	-73.98459
40.68535	-73.95512	Private room	60	30	50	2019-12-02	0.30	2	365	0	NaN	40.68138	-73.95685
40.80380	-73.96751	Private room	75	2	118	2017-07-21	0.72	1	0	0	NaN	40.80397	-73.96685
40.76457	-73.98317	Private room	68	2	575	2023-02-19	3.41	1	106	52	NaN	40.76457	-73.98073
40.66265	-73.99454	Entire home/apt	275	60	3	2022-08-10	0.03	1	181	1	NaN	40.66541	-73.99287

Figure 2. Results of latitude and longitude of the nearest subway station.

Calculate the distance (miles) between the listing and its nearest subway station.

```
[ ] def nearestDistance(row):
      '''Calculate the distance (miles) between the listings and its nearest subway station using geodesic distance.
      Using logitude and latitude of each point.'''

      propertyLocation = (row['latitude'], row['longitude'])
      stationLocation = (row['NearestStationLatitude'], row['NearestStationLongitude'])
      distance = geodesic(propertyLocation, stationLocation).miles
      return distance

    #Apply the cnearestDistance function to each row in Listings.
    listings['Distance'] = listings.apply(nearestDistance, axis=1)
```

```
[ ] listings.head()
```

latitude	longitude	room_type	price	...	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	availability_365	number_of_reviews_ltm	license	NearestStationLatitude	NearestStationLongitude	Distance
40.75356	-73.98559	Entire home/apt	150	...	49	2022-06-21	0.30	3	314	1	NaN	40.75418	-73.98459	0.067703
40.68535	-73.95512	Private room	60	...	50	2019-12-02	0.30	2	365	0	NaN	40.68138	-73.95685	0.288617
40.80380	-73.96751	Private room	75	...	118	2017-07-21	0.72	1	0	0	NaN	40.80397	-73.96685	0.036540
40.76457	-73.98317	Private room	68	...	575	2023-02-19	3.41	1	106	52	NaN	40.76457	-73.98073	0.128014
40.66265	-73.99454	Entire home/apt	275	...	3	2022-08-10	0.03	1	181	1	NaN	40.66541	-73.99287	0.209688

Figure 3. Results of distances (miles) between the listing and its nearest subway station.

Clustering Listings 'name'

```
[ ] from sklearn.feature_extraction.text import  TfidfVectorizer
    from sklearn.cluster import KMeans
    import re
    #Import modules for text feature extraction and KMeans clustering.

    import nltk
    from nltk.stem.porter import PorterStemmer
    ps = PorterStemmer()
    #Import module for text processing and stemming.
```

```
[ ] description = listings[["id","name"]]
    description.head()
```

	id	name
0	2595	Skylit Midtown Castle
1	5121	BlissArtsSpace!
2	5203	Cozy Clean Guest Room - Family Apt
3	5178	Large Furnished Room Near B'way
4	5136	Large Sunny Brooklyn Duplex, Patio + Garden

Figure 4. Description dataset.

Run time 30 mins.

```
[ ] #Iterate over the columns to remove stop words.
    for stopWord in removingPattern:
        if stopWord in featuresData.columns:
            featuresData.drop(stopWord, axis=1, inplace=True)
        else:
            print(f"Stop Word '{stopWord}' not found. Skipping...")
```

```
Stop Word 'k10mins' not found. Skipping...
```

```
[ ] featuresData.head()
```

	abducens	abnb	abod	abode	abov	absolut	abstract	abt	abund	abundantli	...	zimmer	zoller	Charmer	Chelsea	The	East	Magic	Room	Side	Upper
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 6245 columns

Figure 5. TF-IDF results for Listings 'name'.

Within-Cluster Sum of Squares (WCSS) measures the tightness of a cluster using Euclidean distance. The smaller WCSS value, the closer data points surround cluster centroid.

```
fig, ax = plt.subplots(figsize=(10, 6))
plt.plot(range(1, 15), wcss, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('Elbow Method')
plt.show()
```

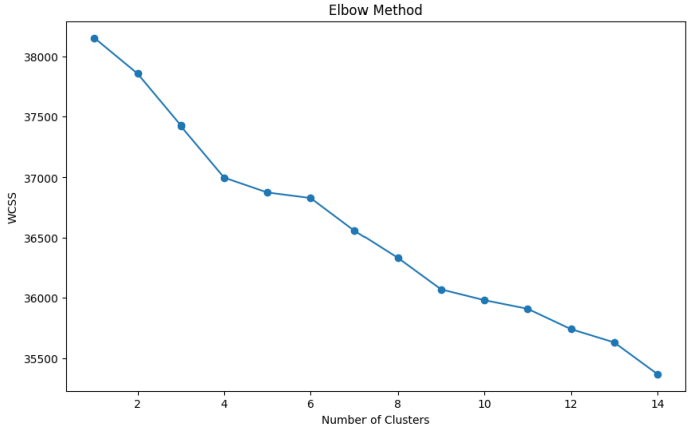

Figure 6. Elbow method.

Choosing total number of clusters k=9, threshold with clear Elbow curve.

```
[ ] k=9
    knnModel = KMeans (n_clusters=k, init= "k-means++", max_iter=100, n_init=1)
    knnModel.fit(featuresData)
    #Create K-means clustering model for 9 clusters.
```

```
KMeans(max_iter=100, n_clusters=9, n_init=1)
```

```
[ ] description["cluster"] = knnModel.labels_
    #Assign the cluster labels generated by the K-means model to Description subset.
```

```
<ipython-input-47-d6a397c4e65c>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  description["cluster"] = knnModel.labels_
```

```
[ ] description.head()
```

	id	name	cluster
0	2595	skylit midtown castl	6
1	5121	blissartsspace!	6
2	5203	cozi clean guest room - famili apt	3
3	5178	larg furnish room near b'way	3
4	5136	larg sunni brooklyn duplex, patio + garden	6

Figure 7. K-means clustering with nine clusters.

```
[ ] uniqueClusters = description["cluster"].unique()

    for clusterLabel in uniqueClusters:
        print(f"Cluster {clusterLabel + 1}:")

        #Get data points (feature names) belonging to each cluster.
        clusterSamples = description[description["cluster"] == clusterLabel]

        print(f"Number of samples: {len(clusterSamples)}")
        print("Sample names:")
        print(clusterSamples["name"].values[:10])
        #Print the first 10 sample names.
        print("\n")
```

```
Cluster 7:
Number of samples: 23194
Sample names:
['skylit midtown castl' 'blissartsspace!'
 'larg sunni brooklyn duplex, patio + garden'
 'rooftop deck/citi views. great apt'
 'lovely, cozy, room 1, best area; legal rental' 'most central location!'
 'uptown sanctuari w/ privat bath (month to month)'
 'central park 1br sunni condo' 'sanctuari in east flatbush'
 'amaz location! wburg. large, bright & tranquil']


Cluster 4:
Number of samples: 6101
Sample names:
['cozi clean guest room - famili apt' "larg furnish room near b'way"
 'comfortable, sunni room' 'room in the heart of astoria'
 'ue beauti blue room' 'room with en suit bathroom & deck'
 'larg b&b style room' 'new york room with a view'
 'larg room - deck access in clinton hill best area'
 'cozi room in east villag with ac']
```

Figure 8. K-means cluster results.

```
[ ] import ast
    reviewsScore["polarity_score"] = reviewsScore["polarity_score"].apply(ast.literal_eval)
    reviewsScore["neg"] = reviewsScore["polarity_score"].apply(lambda x: x["roberta_neg"])
    reviewsScore["neu"] = reviewsScore["polarity_score"].apply(lambda x: x["roberta_neu"])
    reviewsScore["pos"] = reviewsScore["polarity_score"].apply(lambda x: x["roberta_pos"])
    #Separate original polarity_score into three scores.
```

```
[ ] reviewsScore = reviewsScore.drop("polarity_score", axis=1)
    reviewsScore.rename(columns={'id_value': 'id'}, inplace=True)
    reviewsScore['id'] = reviewsScore['id'].astype(int)
    reviewsScore.head()
```

	id	neg	neu	pos
0	2595	0.130334	0.808012	0.061654
1	2595	0.007127	0.046378	0.946495
2	2595	0.001723	0.005813	0.992464
3	2595	0.001408	0.015731	0.982861
4	2595	0.001299	0.006564	0.992137

Figure 9. Roberta polarity score after parse.

```
[ ] finalData.head()
```

ws_per_month	calculated_host_listings_count	availability_365	number_of_reviews_ltm	license	neg	neu	pos	NearestStationLatitude	NearestStationLongitude	Distance	Subway_longitude	Subway_latitude	NAME	cluster
0.30	3	314	1	NaN	0.032255	0.107489	0.860256	40.75418	-73.98459	0.067703	-73.98459	40.75418	42nd St - Bryant Pk	6
0.30	2	365	0	NaN	0.017825	0.042350	0.939825	40.68138	-73.95685	0.288617	-73.95685	40.68138	Franklin Ave	6
0.72	1	0	0	NaN	0.008947	0.014396	0.976657	40.80397	-73.96685	0.036540	-73.96685	40.80397	Cathedral Pkwy (110th St)	3
3.41	1	106	52	NaN	0.064989	0.195013	0.739998	40.76457	-73.98073	0.128014	-73.98073	40.76457	57th St	3
0.03	1	181	1	NaN	0.001492	0.008472	0.990037	40.66541	-73.99287	0.209688	-73.99287	40.66541	Prospect Ave	6

```
[ ] finalData.to_csv('finalDataset.csv', index=False)
```

Figure 10. Final dataset.

Appendix D. Recommendation System Notebook

```
[7] final_df = final_df.rename(columns={'NAME': 'Station_name', 'cluster': 'Airbnb_type', 'neg': 'Rating Negative', 'neu': 'Rating Neutral', 'pos': 'Rating Positive', 'Distance': 'Distance_from_station'})
```

```
[8] final_df.head()
```

	id	neighbourhood_group	price	Rating Negative	Rating Neutral	Rating Positive	Distance_from_station	Station_name	Airbnb_type
0	2595	Manhattan	150	0.032255	0.107489	0.860256	0.067703	42nd St - Bryant Pk	6
1	5121	Brooklyn	60	0.017825	0.042350	0.939825	0.288617	Franklin Ave	6
2	5203	Manhattan	75	0.008947	0.014396	0.976657	0.036540	Cathedral Pkwy (110th St)	3
3	5178	Manhattan	68	0.064989	0.195013	0.739998	0.128014	57th St	3
4	5136	Brooklyn	275	0.001492	0.008472	0.990037	0.209688	Prospect Ave	6

Figure 1. Final dataset with relevant attributes.

Calculate compound ratig score based on Ratings Negative-Neutral-Postive. First option: Scale the original compound scores to a normalized range of 1-5.

```
[13] scoreWeights = {'Rating Negative': 1, 'Rating Neutral': 3, 'Rating Positive': 5}
     compoundScore = ratings[['Rating Negative', 'Rating Neutral', 'Rating Positive']].mul(pd.Series(scoreWeights)).sum(axis=1)
     #Multiply each ratings with corresponding weights and sum up.

     #Scale the compound scores to a normalized range of 1-5.
     rating1 = compoundScore.apply(lambda x: round((x - compoundScore.min()) / (compoundScore.max() - compoundScore.min()) * 4 + 1))

     #Calculate the frequency of each ranking value.
     ratingFrequency1 = rating1.value_counts().sort_index()
     print(ratingFrequency1)
```

```
1      152
2      599
3     1094
4     6959
5    23803
dtype: int64
```

Figure 2. Compound rating score based on scaling to a normalized range of 1-5.

Calculate compound ratig score based on Ratings Negative-Neutral-Postive. Second option: Rounding the original compound score to the nearest integer value.

```
[16] scoreWeights = {'Rating Negative': 1, 'Rating Neutral': 3, 'Rating Positive': 5}

     compoundScore = ratings[['Rating Negative', 'Rating Neutral', 'Rating Positive']].mul(pd.Series(scoreWeights)).sum(axis=1)
     #Multiply each ratings with corresponding weights and sum up.

     #Rounding the compound score to the nearest integer value.
     rating2 = compoundScore.apply(lambda x: round(x))

     #Calculate the frequency of each ranking value.
     ratingFrequency2 = rating2.value_counts().sort_index()
     print(ratingFrequency2)
```

```
1      149
2      599
3     1092
4     7179
5    23588
dtype: int64
```

Figure 3. Compound rating score based on rounding to the nearest integer value.

```
[31] #Display the top recommended properties with station name and distance.
     topRecs = 10
     #Number of recommendations to show.
     recommendedIDs = recommended_properties['id'].head(topRecs).tolist()
     print(f"Top {topRecs} recommended properties:")
     for listingID in recommendedIDs:
       stationDistance = recommendData.loc[recommendData['id'] == listingID, 'Distance_from_station'].values[0]
       stationName = recommendData.loc[recommendData['id'] == listingID, 'Station_name'].values[0]
       print(f"- Property ID: {listingID}, Distance to Nearest Subway Station: {stationDistance:.3f} miles, Nearest Station: {stationName}")
```

```
Top 5 recommended properties:
- Property ID: 838548659003355204, Distance to Nearest Subway Station: 0.144 miles, Nearest Station: Wilson Ave
- Property ID: 21110408, Distance to Nearest Subway Station: 0.435 miles, Nearest Station: Kingston - Throop Aves
- Property ID: 20914711, Distance to Nearest Subway Station: 0.200 miles, Nearest Station: Clinton - Washington Aves
- Property ID: 20914425, Distance to Nearest Subway Station: 0.145 miles, Nearest Station: Gates Ave
- Property ID: 21120183, Distance to Nearest Subway Station: 0.207 miles, Nearest Station: Eastern Pkwy - Bklyn Museum
```

```
[32] recommendData[recommendData["id"]==838548659003355204]
```

	id	neighbourhood_group	price	Distance_from_station	Station_name	Airbnb_type	Rating
32605	838548659003355204	Brooklyn	144	0.144093	Wilson Ave	6	5

Figure 4. Recommendation system result.