Airbnb Zero Reviews Classification

By Ji Noh, Minwoo Sohn, and Tiffany Lee

Introduction

Airbnb, an online marketplace established in 2008, revolutionized lodging by allowing hosts to rent out unused space or entire homes. This platform offered a cost-effective and unique alternative to traditional hotels and B&Bs. One challenge guests face is determining the quality of a listing without any firsthand experience. This issue has given rise to a phenomenon known as social proof, where potential guests rely heavily on reviews from others to assess the desirability of a listing. New listings, however, suffer from a lack of reviews, creating a "cold start" problem where both Airbnb and prospective guests struggle to evaluate the quality of these properties.

Therefore, this project aims to explore the potential of Airbnb listings with zero reviews, which might be overlooked due to the absence of feedback, yet could be promising options for travelers. Our goal is to assess these properties' value and enhance their visibility to potential guests by categorizing each zero-review listing as ‘Poor’, ‘Average’, or ‘Great’. Through this classification, Airbnb can better recommend specific listings to different users, addressing the initial challenge of the cold start problem for new listings on the platform.

Dataset Overview

Our dataset originates from Kaggle's Inside Airbnb USA Dataset, which encompasses 8.94 GB of data across 30 U.S. cities. For our project, we selected 15 U.S. cities, which we have evenly divided into three regions: East, Central, and West. We chose these 15 cities to get a representation of small vs. large cities for each of our regions.

The following is the breakdown of our three regions:

- East:
    - Broward County (FL)
    - Jersey City (NJ)
    - New York City (NY)
    - Cambridge (MA)
    - Washington DC
- Central:
    - Nashville (TN)
    - Denver (CO)
    - Ausin (TX)
    - Chicago (IL)
    - New Orleans (LA)
- West:
    - Seattle (WA)

    - Los Angeles (CA)
    - San Francisco (CA)
    - Portland (OR)
    - Clark County (NV)

For each city, there are seven data files each. Most of our data files are in CSV format while we have one file that is a GeoJSON format. The data files we have are the following in their respective format for each city:

- Calendar - CSV
- Listings - CSV
- Listings_detailed - CSV
- Neighbourhoods - CSV
- Neighbourhoods - GeoJSON
- Reviews - CSV
- Reviews_detailed - CSV

The full data dictionary for each of these data files can be found on our GitHub repository. We ingested the data files using Spark to conduct the large-scale data processing and created the pipeline needed to read in our CSV files into a Spark DataFrame so that we can clean and manipulate the data as needed for these files. When ingesting each of the CSV files into a Spark DataFrame, we either had Spark DataFrame infer the schema or gave an explicit schema to define the respective data types for each of the CSV files. For the GeoJSON file, we used Geopandas to convert the GeoJSON into pandas dataframe to retain the spatial information in the dataset easily to view what is inside the file.

Upon reviewing the data, our team found that 'listing_detailed.csv' offered the most comprehensive details on properties listed in the city, so we believed this table would be most relevant to us for feature engineering.

Data Preprocessing - Missing Data Imputation

In the process of preparing our dataset for exploratory data analysis (EDA), we encountered the issue of missing data, which is common in real-world datasets and can significantly affect the quality of statistical analysis and machine learning models. To address this, we formulated a strategy for imputing missing values for various columns, ensuring the integrity of our dataset without discarding valuable information.

For property-related features such as 'Bathrooms', 'Bedrooms', and 'Beds', we developed a multi-step imputation process. Initially, when encountering missing 'Bathroom' data, we used the number of bedrooms as a proxy, under the assumption that the number of bathrooms is often proportionate to the number of bedrooms in a property. If the 'Bedrooms' data was also missing, we then imputed the missing 'Bathroom' value with half the number of beds, based on typical property layouts. In cases where 'Bedrooms' data was absent, we rounded up the 'Bathroom' count to the nearest integer, inferring that a property should have at least one bedroom for each bathroom listed. For missing 'Beds' data, we defaulted to the number of bedrooms, and if all

three — 'Bathrooms', 'Bedrooms', and 'Beds' — were missing, we considered inputting a zero, although this decision required further discussion about its implications on our analysis.

When examining host-related features, we approached missing 'Host_since' dates by imputing them with the most recent date in our dataset, formatted to match the existing date entries, such as 'March 2023'. If the 'Host_location' was missing or not applicable, we imputed the value as 'unknown', which allowed us to maintain data uniformity without assuming inaccurate host locations, as ‘unknown’ value was already present in the data.

For the boolean fields such as 'Host_is_superhost', 'Host_identity_verified', and 'Host_has_profile_pic', we imputed missing values with 'false', aligning with the conservative assumption that unverified or absent data should not confer statuses typically earned through positive actions or verifications. Similarly, for 'Host_listings_count' and 'Host_total_listings_count', we assumed a single listing in the absence of data, which seemed to be a safe baseline assumption.

The 'Host_verifications' field, when missing, was set to 'None', which was already an existing category within our dataset, thus maintaining consistency. This choice was based on the premise that no data on verifications implies no verification methods were reported. For 'Calculated_host_listings_count', we applied the same logic as with 'Host_listings_count', defaulting to 1 to imply a minimum level of host activity.

Our treatment of missing data within 'Review_scores_value' required a nuanced approach. We noted that the absence of review scores was often justified by a lack of reviews, fitting our project focus on newly listed properties without feedback. However, for the negligible portion of data (0.5%) with missing 'Review_scores_value' despite having reviews, we opted to exclude these rows due to the limited reviews not providing a robust picture of the listing's quality.

Exploratory Data Analysis

Given the seven different datasets for each city along with dividing our data into three distinct regions, we first conducted Exploratory Data Analysis (EDA) for each specific region before conducting EDA on all 15 cities at once. To combine our different data frames, we used pySpark to join the different tables together to do EDA.

For each region’s EDA, we have created three Python notebooks, each housed within the 'milestone1' folder of our GitHub repository. These notebooks facilitate a comprehensive examination of our entire dataset on a regional level, including identifying missing values, analyzing data types, assessing the general range of each column, and compiling essential statistics such as mean, median, and standard deviation.

Plots that we created within our region specific notebooks can be grouped into the following categories:

- Exploring price feature through the following graphics:
    - Adjusted and non-adjusted mean price changes over time
    - Average price based on availability
    - Number of times the listing price and adjusted price differ
    - Average listing price given the number of people a listing is able to accommodate
    - Identify whether there is a relationship between price and review scores through a correlation heatmap
- Analyzing reviews related features through the following graphics:
    - Total number of reviews for each city and for each city's neighborhood
    - Total number of reviews on a per day and yearly basis
    - Number of reviewers who have written a certain number of reviews within a specific region.
- Assessing general listing property and Airbnb information:
    - Daily availability count for each city
    - General common amenities found in each region
    - Number of host sign-ups by year and how do host sign-ups compare relative to each city
    - Comparison of room type count for each city
- Explored the data inside the geojson file, which showed us that the file is able to render each city's geographic area

The EDA on a regional level helped us first identify interesting trends that we wanted to consider pursuing for our aggregate 15 cities EDA. The aggregate 15 cities EDA can be found in our 'milestone2' folder on our Github repository.

Most of our plots that we created for the aggregate 15 cities can be grouped into the following categories:

- Examining individual property features:
    - Distribution of accommodates feature for listings
    - Distribution of bedrooms feature
    - Distribution of num_bath feature
- Exploring amenities related information for all listings:
    - Total number of amenities for all cities
    - Identifying the number of listings that have a certain number of essential amenities as an aggregate and by each city
- Exploring the number of reviews for all cities:
    - Number of listings with no reviews by city
    - Identifying the relationship between number of reviews and when review scores are null.
    - Identifying the relationship between number of reviews and when host response time is empty.
- Assessing other general listings distributions:
    - Number of listings per city
    - Number of listings based on host verification type

- Frequency of host listings count

In this report, we have attached certain plots that we found most meaningful when we conducted our aggregate 15 cities EDA.

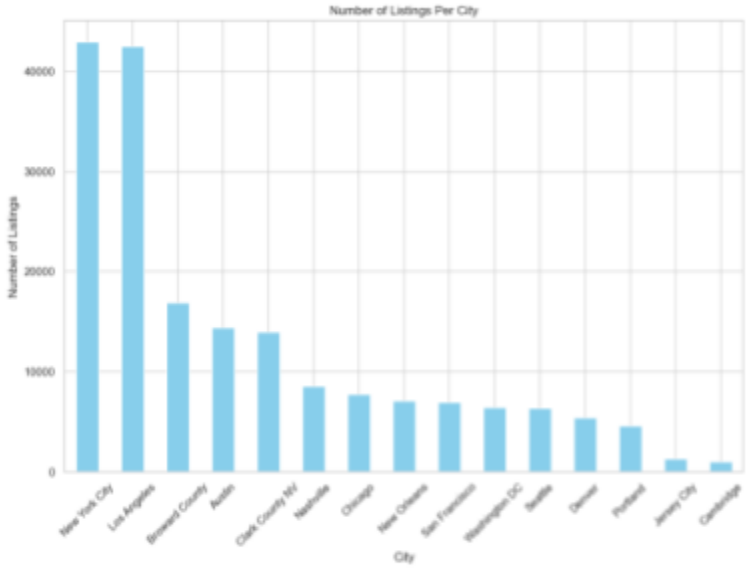

Figure 1: East and West regions generally have large number of listings, unlike the Central region.

Figure 1 demonstrates that even though we purposely try to select cities that would provide us a mix of large and small cities equally in each of our regions, we see that the Central region generally has a smaller number of listings compared to the other two regions.

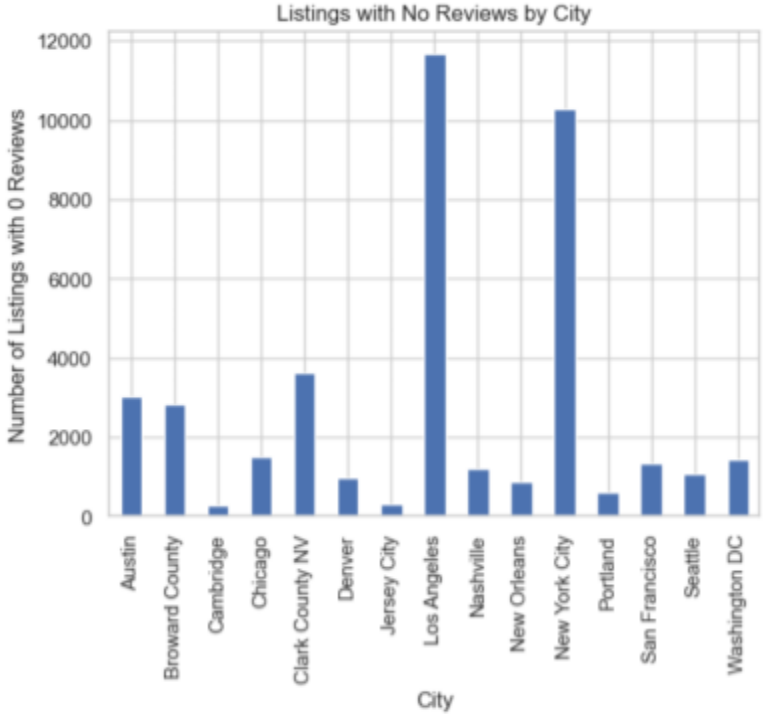

Figure 2: Graphic shows the number of listings with zero reviews.

Figure 2 above shows that generally the cities with the largest number of listings will also have the most number of listings with zero reviews.

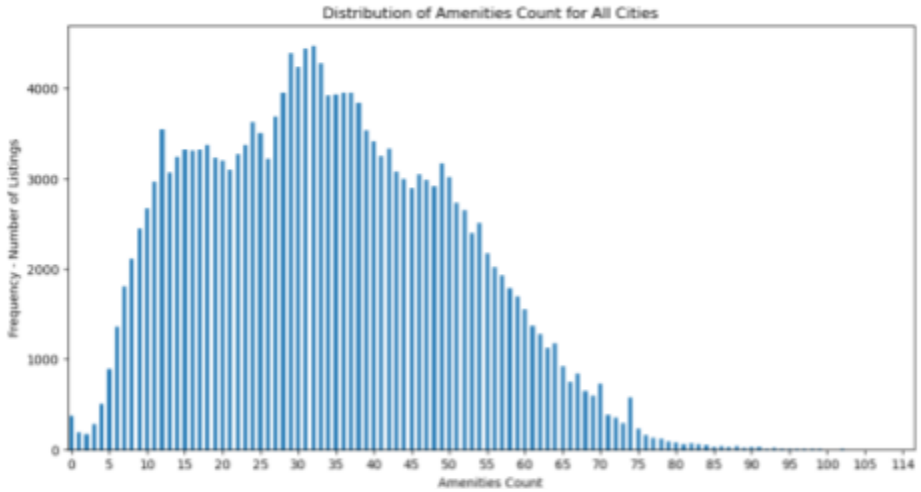

Figure 3: Graphic shows that there is a positive/right skew where the mode is around the mid 30s range.

Figure 3 above shows that over half of the listings in our dataset will generally have up to 40 amenities, but we wanted to see if the number of amenities is influenced by region/city.

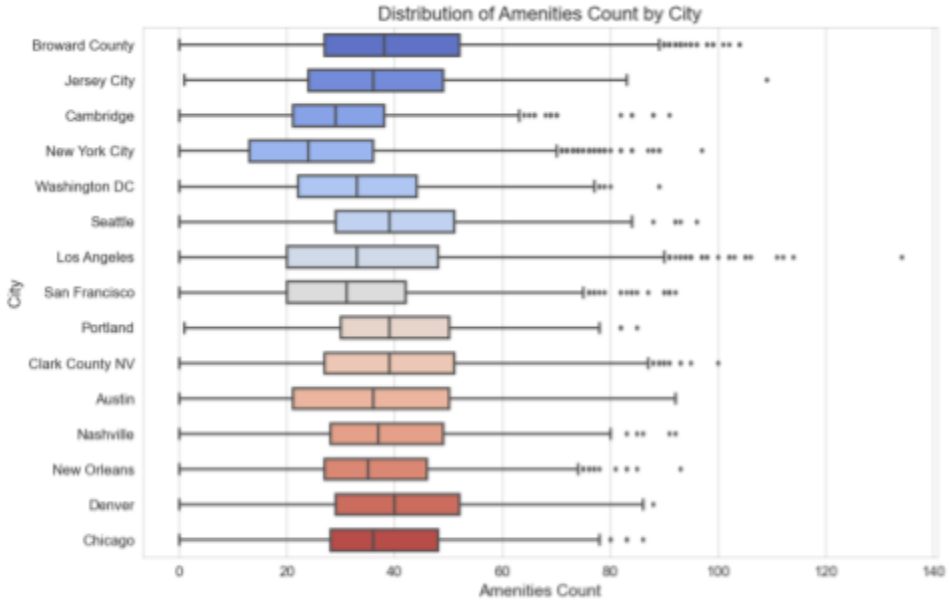

Figure 4: The median number of amenities is similar for all cities.

In figure 4 above, we see that the total count of amenities' median for each city is similar to each other, so we can conclude that there won't be major regional differences in terms of the number of amenities a listing would list on Airbnb.

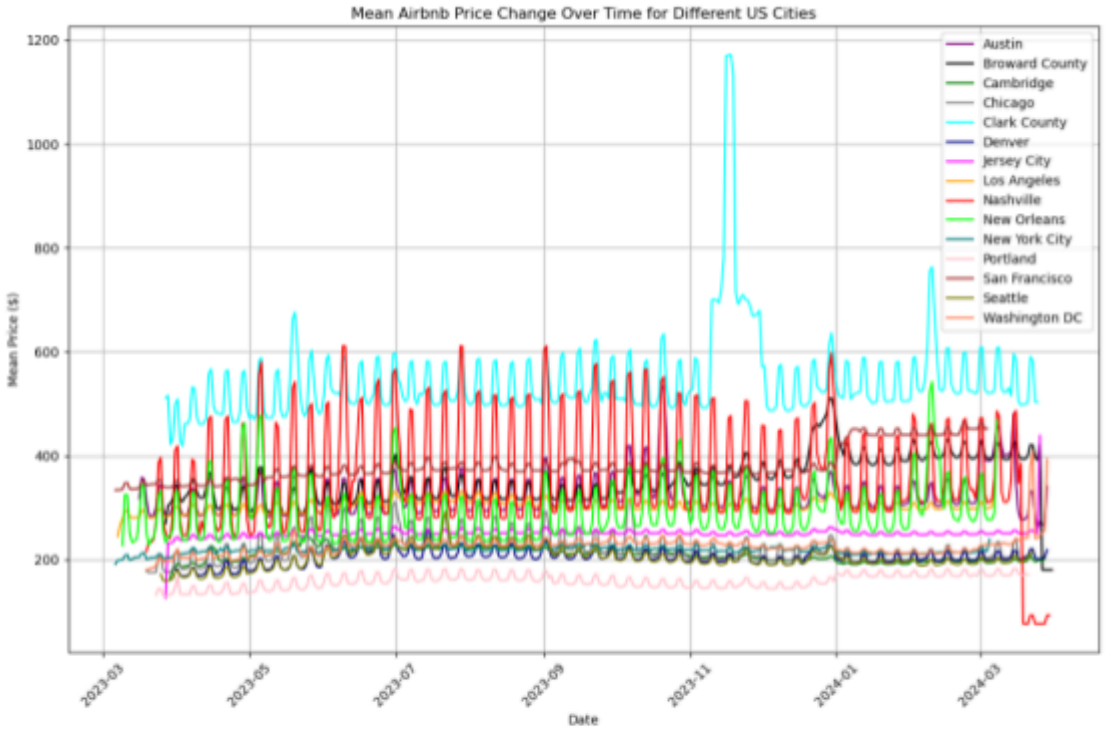

Figure 5: There is a spike in Clark County because of the Formula One Car Race.

In figure 5 above, we see that Airbnb prices go through seasonality, but we saw that Clark County experienced an unusual spike due to the Formula One Car Race happening at that period of time. This shows that points of interest impact Airbnb prices significantly, but we ultimately decided to not use daily prices in our models because there is difficulty in accounting for one-off events and due to computational issues with Google Cloud Services (GCS).

Feature Engineering

In the feature engineering phase of our project, we enhanced our dataset by creating additional variables to provide richer insights into the properties listed.

- Number of Amenities

Understanding that the amenities offered in a listing could significantly influence guest satisfaction, we initially tackled the 'Amenities' field. Originally, this column was formatted as a list object, detailing all amenities offered by the host for each property. This format presented a challenge for utilization in Spark ML, which requires numerical input for features. To adapt the 'Amenities' data for our machine learning model, we transformed this list into a more usable format by counting the number of amenities provided in each listing, thereby creating the '*num_amenities*' feature. This numerical representation allowed us to integrate the amenities information effectively into our predictive modeling process.

- Essential Amenities

To more accurately reflect guest expectations, we identified a set of amenities considered essential for a satisfactory stay; Fridge, AC, Kitchen, Wifi, and basic essentials. These amenities were chosen because they represent the minimum requirements that guests typically expect to ensure a comfortable and satisfactory stay at an Airbnb property.

With this in mind, we developed a new feature titled '*Essential_amenities*' that quantifies how many of these critical amenities are included in each listing, with scores ranging from 0 (none included) to 5 (all included).

Given the variability in how hosts list amenities—due to personal input styles or synonyms—handling these discrepancies was crucial. For instance, the term 'fridge' could appear as 'refrigerator', 'mini fridge', or 'mini-fridge'. Similarly, air conditioning was variously listed as 'air conditioning', 'air conditioner', 'AC', 'A/C', or 'central air'. To ensure consistency and accuracy, we implemented error handling procedures to recognize and count these variations effectively within the 'Essential_amenities' feature, ensuring a robust analysis of each property's appeal based on its amenities.

- Bathroom

To address the description of bathrooms in listings, which often included non-standard terms like 'half bathroom', we extracted only numeric values to standardize this information into a new '*Num_baths*' feature.

- Neighborhood

To enhance the accuracy of location data within our dataset, we addressed ambiguities in the 'Neighborhood' attribute by concatenating each neighborhood name with its corresponding city name. This adjustment was necessary due to the occurrence of identical neighborhood names in different cities, such as 'Hollywood' in both Broward County, FL, and Los Angeles, CA. By merging the neighborhood and city names, we ensured that each location was uniquely identifiable, thereby eliminating confusion and improving the precision of our geographic data.

- Full-time Host

In our analysis, we also looked into the profiles of Airbnb hosts and added new *'Full_time_host'* feature. This attribute assesses the level of a host's engagement with the platform by considering their number of listings. Specifically, if a host has 10 or more listings, we classify them as a 'Full_time_host,' under the assumption that they are likely more dedicated and experienced, possibly treating Airbnb hosting as a full-time occupation. This distinction helps identify hosts who may offer a higher standard of service, owing to their extensive involvement in hosting.

- Host Verification

In response to data processing challenges with Spark, we adapted the *'Host_verification'* field, which initially consisted of a list detailing the methods hosts used to verify their identities, such as combinations of 'email', 'phone', ‘work_email’ and others. This field is crucial as it conveys the level of identity verification a host has completed, imparting a sense of trustworthiness to potential guests.To accommodate Spark's data handling requirements, we transformed these lists into a concise encoded string format. For instance, 'phone' became 'p', 'email' was encoded as 'e', and combinations like 'email' and 'phone' were represented as 'ep', or ‘email’, ‘phone’ and ‘work_email’ being encoded as ‘epw’. This encoding covers various combinations, from single methods to more complex arrays of verification, ensuring we retain essential information in a format that seamlessly integrates with our analytical tools.

- Target

Lastly, we refined the *'review_scores_value'* field, which initially featured continuous numeric values ranging from 0 to 5. To better facilitate analysis and enhance predictive accuracy regarding listing quality, we converted these scores into categorical variables with defined thresholds. Listings with scores from 0 to 4 were categorized as 'Poor', those scoring between 4 and 4.8 were deemed 'Average', and scores from 4.8 to 5 were classified as 'Great'. We chose the 4.8 threshold based on Airbnb's criteria for awarding the 'Super Host' badge, which is given to hosts whose properties achieve overall ratings of 4.8 or higher and other factors like low cancellation rate, high response rate, etc. This distinction is significant as it signals exceptional host performance and reliability, factors that greatly influence guest satisfaction and trust. By categorizing the review scores, we aimed to simplify the data output for clearer interpretation and more effective application, particularly in enhancing the visibility and attractiveness of listings, especially those new or without prior reviews.

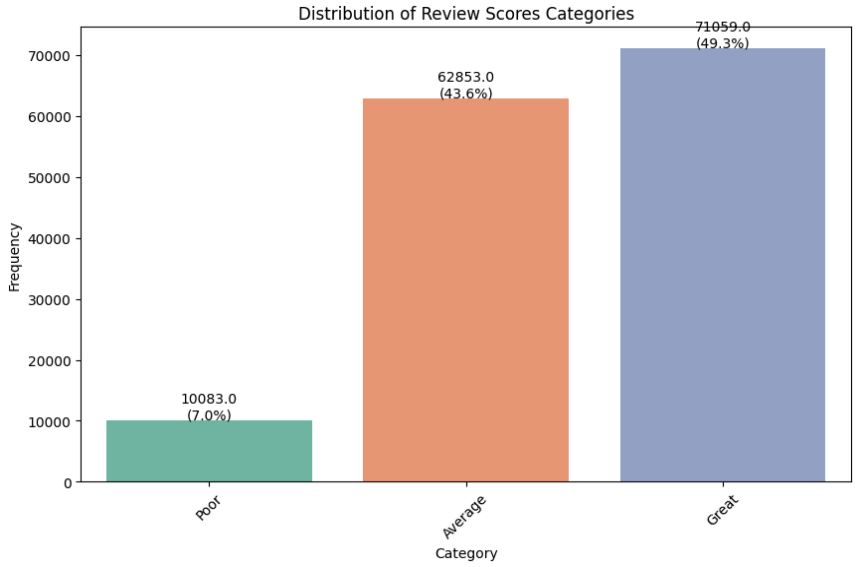

Figure 6: Distribution of Review Scores Categories

This categorization resulted in 7% of the data being classified as 'Poor', 43.6% as 'Average', and 49.3% as 'Great'. A significant challenge we faced with this approach was the unbalanced distribution of classes, which could potentially skew our model's accuracy. Although converting to a binary classification system might simplify the analysis, we chose to maintain the three categories. We believed that merging vastly different scores, such as 0 and 4.5, into a single category would not accurately reflect the diverse quality of properties, making it essential to retain the three-tier system despite the skewed data distribution.

After this process, given the three categories in our target column, we structured our analysis around a multi-class classification model with three distinct classes.

- Encoding Categorical Columns to Numeric for Spark ML

To construct our pipeline, which orchestrates a sequence of transformations to prepare our training and test datasets for machine learning, we utilized VectorAssembler. This tool is essential for merging both numeric and indexed categorical features into a singular feature vector column named "features". Before leveraging VectorAssembler, it was imperative to convert all string or categorical features into a numeric format, enabling their integration into the pipeline.

Once established, the pipeline facilitated the transformation of string features into numeric representations and the amalgamation of these with existing numeric features into a unified vector format. This setup not only streamlined the entire transformation process but also allowed for the seamless application of our Spark ML models.

In total, our modeling pipeline employed 18 features:

- Eight columns were categorical string-related features, each processed through a StringIndexer to convert them into numerical indices, which are the following:
    - host_location
    - Host_is_superhost
    - Host_has_profile_pic
    - Host_identity_verified
    - City
    - Room_type
    - Full_time_host
    - host_verifications_clean
- Ten columns were numeric features, directly incorporated into the model. The columns that were numerical are the following:
    - Host_listings_count
    - Host_total_listings_count
    - Accommodates
    - Num_bath

- Bedrooms
- Beds
- Price
- Calculated_host_listings_count
- Amenities_count
- essential_amenities

These features were collectively combined into a single vector via VectorAssembler, a crucial step for modeling in Spark ML. The target outcome for our models was to classify each Airbnb listing as ‘Poor’, ‘Average’, or ‘Great’, which was also encoded as numeric values, 2, 1, 0 respectively, based on the transformed review scores. This comprehensive pipeline enabled efficient transformations and was integral to applying our three distinct machine learning models. Please note that due to computational issues with Google Cloud, we ended up changing our pipeline to use 7, 10, and 13 features in our modeling.

Modeling

In the modeling phase of our project, we focused on addressing a classification problem by evaluating various models. Our group selected the following three classification models for this study:

1. Different Models

Random Forest: A versatile and robust ensemble learning method that builds multiple decision trees during training and outputs the class that is the mode of the classes (classification) of the individual trees. Random forests perform well for a large range of data items without overfitting, handling large feature spaces and interactions between features effectively.

Decision Tree: A decision support tool that uses a tree-like model of decisions and their possible consequences. It's simple to understand and interpret, and is particularly useful for handling non-linear data patterns that involve complex interactions between features.

Multinomial Logistic Regression: A linear model for classification rather than regression. It is used when the response variable is categorical. The model is good for situations where you can linearly separate classes.

These models were selected because they are well-suited for handling the complexity and variety of the data involved in this project. Random Forests and Decision Trees are particularly beneficial for their robustness and ease of use in interpretability, which is crucial for our extensive feature set. Multinomial Logistic Regression was chosen for its efficacy in providing probabilistic outputs for classifications, which can be valuable for threshold tuning.

2. Modeling Pipeline

Data Preparation

Our analysis was conducted within a Spark session, requiring specific steps to manipulate and prepare the data appropriately. We initiated by starting the Spark session and loading our dataset in CSV format. After loading the data, we performed the following transformations:

Data Type Conversion: The data contains several features which needed to be converted from string type to numeric types to facilitate analysis. This included casting counts of listings, accommodations, number of baths, bedrooms, beds, and amenities into integer or double formats as some fields like price and number of baths could contain decimals.

StringIndexer: For categorical features, such as host type and room type, we applied StringIndexer to convert these string values into numeric indices.

VectorAssembler: A VectorAssembler was used to combine all feature columns into a single vector column, simplifying the input structure for machine learning models.
Model Optimization and Feature Selection

We constructed a pipeline that includes the indexer, assembler, and label indexer as stages before fitting it to the training data. This allowed for efficient preprocessing and model training within a unified framework.

Following the data preparation, we utilized a train-test split to evaluate the models effectively, setting aside 30% of the data for testing to ensure that our model generalizes well to new data.

In the initial stages of our model development, we started by including 18 features. However, during hyperparameter tuning, we encountered issues with node breakdowns in Google Cloud Storage, prompting a reassessment of our strategy. To address this, we streamlined our feature set to 13, establishing this as our baseline for subsequent analyses.

The features retained for further modeling were carefully chosen to balance model complexity with performance:

- Categorical Features: Host type (superhost status), city, room type, hosting frequency (full-time host), host verification methods.
- Numerical Features: Total listings by the host, number of accommodations, bathrooms, bedrooms, beds, price, amenities count, essential amenities.

With the reduced feature set, we trained our models and proceeded to optimize the hyperparameters. Using a grid search approach combined with cross-validation, we

systematically tested various combinations of parameters to identify the configurations that yielded the best performance.

To further refine our model, we conducted a feature importance analysis. This analysis helped identify the most influential features, providing insights into potential reductions in feature count that could simplify the model without compromising its predictive power. Following this, we engaged in a second round of hyperparameter tuning to enhance the model based on the reduced feature set.

Throughout this process, we developed four distinct model variants for each type of classifier:

1. Baseline Model (13 features)
2. Hyperparameter-Tuned Model (13 features)
3. Feature Importance-Based Model
4. Hyperparameter-Tuned Model with Feature Importance

Each model was evaluated based on its accuracy, computational efficiency, and ability to generalize across unseen data. This iterative refinement helped in understanding the trade-offs between model complexity and performance.

The final step in our analysis was to deploy the best-performing model to predict categories for listings that had not yet received any reviews. This application was crucial for assessing the model's practical utility in a real-world scenario, allowing us to understand how different properties might be classified based on their features alone, potentially guiding decisions in listing management and marketing strategies.

3. Evaluation Metrics

A key tool in classification is the confusion matrix, a table that allows visualization of the performance of an algorithm. It shows the actual versus predicted classifications, with correct predictions on the diagonal and errors off-diagonal. This matrix forms the basis for calculating additional performance metrics:

Accuracy: The ratio of correctly predicted observations to the total observations.

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{Total Observations}}$$

Precision: The ratio of correctly predicted positive observations to the total predicted positives.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

Recall (Sensitivity): The ratio of correctly predicted positive observations to all observations in the actual class.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

F1 Score: The weighted average of Precision and Recall.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

In our classification model evaluation for Airbnb listings, we opted to prioritize accuracy over the F1 score, which is generally favored in scenarios with unbalanced data. Our decision was influenced by the nature of the Airbnb platform, where listings with poor ratings are typically improved by hosts or removed over time. This dynamic suggests that even with the addition of new data, the distribution of class ratings is unlikely to become balanced, as poorly rated listings tend not to persist. While accuracy provides a clear measure of overall performance, it's important to note that it might not capture the model's effectiveness in detecting rarer, poor ratings—a consideration that could be addressed in future analyses if detecting such outcomes becomes a priority.

4. Results

Decision Tree Model

- Baseline Model: Achieved an accuracy of 57.54%. The confusion matrix showed significant misclassifications between classes 0 and 1.
- After Hyperparameter Tuning: Improved accuracy slightly to 58.46%. The best parameters were maxDepth=10, maxBins=1024, and minInstancesPerNode=4. This tuning redistributed some of the misclassifications, particularly increasing correct predictions in class 2.
- Feature Importance: Focused on seven features which were host total listings count, host is superhost indexed, city indexed, amenities count, accommodates, price, and essential amenities, leading to a minor drop in accuracy to 57.20%.
- Feature Importance hyperparameter tuning with these features, accuracy slightly improved to 58.05%.

Multinomial Logistic Regression Model

- Baseline with 13 Features: Started with an accuracy of 55.88%.
- After Hyperparameter Tuning with 13 Features: Marginally improved to 56.00%.
- Feature Importance with 10 Features: Used features like host is superhost indexed, full time host indexed, essential amenities, number of baths, bedrooms, room type indexed, accommodates, beds, host verifications clean indexed, and city indexed. The baseline accuracy decreased to 55.51%. After tuning, it was practically unchanged at 55.53%.

Random Forest Model

- Baseline with 13 Features: Began at a higher baseline accuracy of 57.95% compared to the logistic regression.
- After Hyperparameter Tuning with 13 Features: Significantly increased to 60.01%, showing notable improvement from tuning.
- Feature Importance with 7 Features: Reduced the feature set to host is superhost, city, full time host, host total listings count, accommodates, price, and amenities count. The baseline accuracy slightly improved from the initial full feature set to 58.02%. After tuning, accuracy further increased to 59.65%, demonstrating the effectiveness of feature selection and tuning.

Conclusion

1. Best Model

After evaluating various machine learning models, including Random Forest, Multinomial (Softmax) Linear Regression, and Decision Tree, we found that the Random Forest model equipped with 13 features achieved the highest accuracy at 60.01%. This model reported an average precision score of 59.18%, a weighted recall of 60.01%, and an average F1-Score of 58.22%. For the hyperparameter tuning, Random Forest chose, maxBins = 2200, maxDepth = 12, and had 20 trees.From the confusion matrix, it's evident that the model struggles significantly with the 'Poor' listing category, correctly classifying such listings only 5% of the time. This poor performance is largely attributed to the unbalanced class distribution previously discussed, where 'Poor' categories constituted only 7% of the data, while 'Average' and 'Great' categories comprised over 40% each.

```
+------------+-----+-----+---+
|target_label|    0|    1|  2|
+------------+-----+-----+---+
|         0.0|15115| 6010|117|
|         1.0| 8252|10652| 56|
|         2.0| 1133| 1711|166|
+------------+-----+-----+---+
```

Figure 7: Confusion Matrix of Fine-Tuned Random Forest with 13 Features

Further analysis of our test data results reveals a critical issue with the model's conservative approach in predicting 'Poor' listings. It predicted only 339 listings as 'Poor', compared to the actual 3010, resulting in an 88.7% discrepancy. This significant underprediction underscores the model's difficulty in accurately identifying listings that belong to the 'Poor' category, likely exacerbated by the limited presence of 'Poor' listing reviews in the dataset.

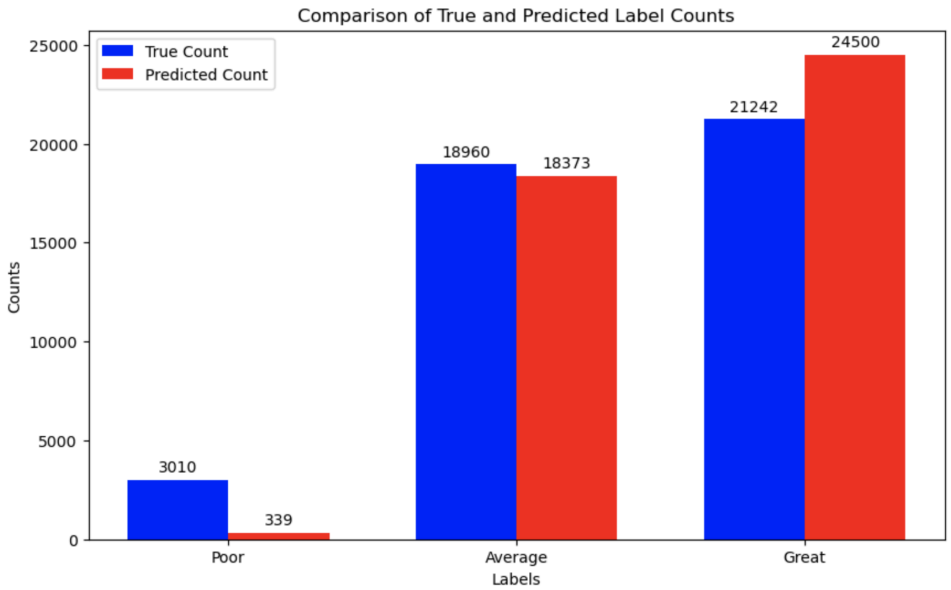

Figure 8: Comparison of True and Predicted Label Counts using Random Forest Model

2. Feature Importance

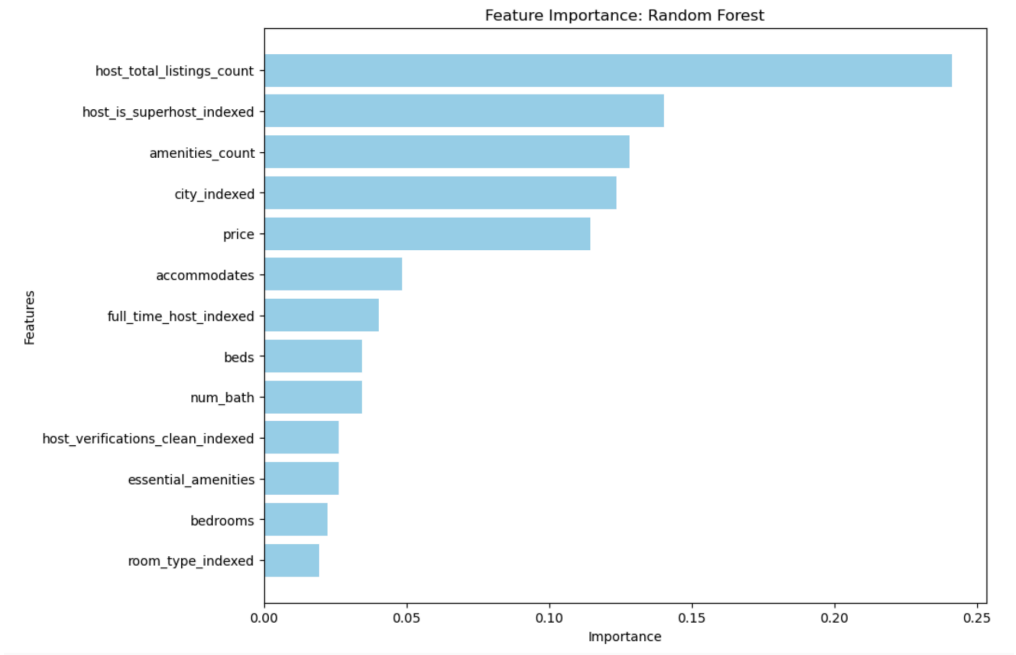

Figure 9: Feature Importance of Fine Tuned Random Forest Using 13 Features

In analyzing the feature importance of our best-performing model, the Random Forest, we focused on understanding which factors most significantly influenced classification outcomes. We observed that 'host_total_listings_count', representing the total number of listings a host manages, and 'host_is_superhost_indexed', indicating whether a host has achieved superhost status on Airbnb, were the most influential features. This finding aligns with our expectations, as these features are indicative of a host's experience and reliability, which can significantly impact guest experiences.

Interestingly, features directly related to the property itself, such as 'beds', 'num_bath', and 'bedrooms', were less influential in our model's classification process. This was unexpected, as one might assume these aspects would be critical in determining the quality of a listing. This discrepancy highlights the unique insights that machine learning can provide into factors that influence guest satisfaction and listing quality.

3. Conclusion

In the competitive landscape of online accommodation booking, Airbnb has consistently sought innovative ways to enhance user experience and streamline property evaluation. Our model involves the deployment of a fine-tuned Random Forest (RF) model designed to classify properties into three categories—'Poor', 'Average', and 'Great'. With an accuracy rate of 60%, this model provides Airbnb with a tool, especially for assessing properties that lack user reviews.

3-1. Real World Application

The implementation of this RF model stands to benefit various stakeholders within the Airbnb ecosystem. Firstly, for the company itself, this model serves as an invaluable internal tool. It assists in gauging the potential popularity of newly listed properties that have not yet accumulated reviews. This predictive capability enables Airbnb to provide hosts with actionable insights and guidelines on how to adjust pricing or improve property features to boost their appeal.

For Airbnb hosts, particularly those new to the platform, the model's insights can help them optimize their listings from the outset, aligning features and pricing with what successfully attracts guests. This proactive approach in property management could lead to better initial reviews, which are crucial for sustaining long-term interest in their listings.

Airbnb customers, on the other hand, could gain significantly from the model's ability to evaluate the value of properties lacking reviews. The absence of guest feedback often makes potential customers hesitant. However, with the model’s classification, customers can gain increased confidence in their booking decisions, even for listings without prior reviews, knowing that the property has been assessed through a reliable, data-driven framework.

3-2. Challenge and Limitation

Despite its utility, the hyperparameter-tuning and operationalization of the RF model are not without challenges. One significant limitation is the model's computational demands. During the fine-tuning phase, a lack of sufficient computational resources with Google Cloud Services necessitated the removal of five potentially informative columns. This reduction could have omitted subtle yet critical nuances that might affect the accuracy and robustness of the model.

Additionally, the representativeness of the data poses another challenge. The model was trained with a limited number of cities from each region, which may not fully capture the diverse range of property types and host qualities across different areas. This limitation could skew the model's predictions, making it less applicable to regions outside its training dataset.

The data used for training the model also suffered from an unbalanced 'Poor' rating distribution, despite a high maximum star rating threshold of 4.0. This imbalance makes it difficult for the model to accurately classify properties into the 'Poor' category, as evidenced by its significant underprediction of such listings. Furthermore, unaccounted regional differences, such as economic factors or recent growth spurts, could influence property popularity independently of the features assessed by the model.

Finally, the inability to validate predictions against a real 'ground truth' for properties with zero reviews means that the model's effectiveness in a real-world scenario remains somewhat theoretical. Without actual guest feedback, the accuracy and utility of the model's classifications cannot be empirically confirmed, leading to potential discrepancies between predicted and actual outcomes.