

In-Class Kaggle Competition Writeup

Yixiao Wang

Nov 26th 2024
Kaggle ID: YixiaoWang0102

1 Exploratory Analysis

For this Kaggle competition, our primary objective is to predict the prices of Airbnbs in New York City using information about their location, amenities, host details, availability, and various other factors.

I began by briefly exploring the data. I examined the correlation matrix and histograms of the numerical features to understand their distributions, and I calculated the number of missing values for each feature. The missing data primarily falls into two categories: those related to reviews and those associated with host responses. Below is a detailed analysis of each feature, along with an explanation of the feature engineering steps applied. It is important to note that this report primarily focuses on the feature engineering process itself. The exploratory data analysis (EDA) is a complex and extensive component of this project. I utilized the `Sweetviz` library to generate an automated EDA report. For completeness, the generated EDA report has been attached in the last few pages of this document for reference.

I should first note that all the thresholds and cluster numbers used in feature engineering were determined based on parameters that performed well in my training experiments. However, this aspect did not undergo rigorous cross-validation due to the excessively long training time. For instance, when centralizing **latitude** and **longitude**, I did not use the mean or median for centering; instead, I subtracted two "magic" constants. This approach unexpectedly improved performance by 1%. While I do not know the exact reason for this improvement, the results consistently demonstrated enhanced performance. Furthermore, the specific thresholds relied heavily on my intuition and iterative experimentation, combined with feature importance evaluations. For example, the truncation threshold for **neighbourhood_cleansed** was guided by feature importance analysis, but the final value of 200 (as

opposed to 250 or 150) was a personal choice. There is no strict criterion or rule supporting the rationale behind this specific decision.

1.1 Feature-by-Feature Analysis

First, I categorized the features into three main types: text features (which refer to lengthy sentences that require semantic understanding or word clustering, excluding categorical variables such as neighborhood), categorical features, and numerical features. I also addressed date variables separately. Below is the feature engineering process organized by feature type.

1.1.1 Text Features

name: This column was removed during preprocessing because it did not offer significant predictive value given the availability of the house description.

description, reviews:

For both reviews and descriptions:

- Text data was preprocessed through tokenization, stemming, and the removal of special characters.
- Translation of non-English text in reviews was conducted using the **Googletrans Translator** module. If the translation fails or the text is already in English, the original text is preserved.
- Latent Dirichlet Allocation (LDA) was utilized to extract topic features from the text data. For each topic, a column was added to represent the probability of that topic within the text.

amenities:

- Amenities were divided into individual components.
- Each amenity was assigned to a cluster using KMeans clustering based on Sentence-BERT embeddings.
- Cluster counts were computed for each property, generating features `amenity_cluster_0`, `amenity_cluster_1`, ..., `amenity_cluster_20`. Then we manually established the mapping rules.

1.1.2 Categorical Features

property_type: Cleaned and grouped rare categories into 'others'. Low-frequency categories were merged based on a threshold of 20 occurrences.

neighbourhood_cleansed: Rare neighborhoods (fewer than 200 listings) were categorized as 'others'.

neighbourhood_group_cleansed: Used directly for analysis without modification.

host_response_time: Missing values have been replaced with the placeholder 'missing'.

host_is_superhost: Converted to binary values (1 for True, 0 for False).

host_verifications: Converted into binary features for specific verifications (phone, email, work_email). The total number of verifications was included as a separate feature.

room_type: One-hot encoded into separate binary columns.

bathrooms_text: Extracted information regarding private or shared bathrooms into is_private_bathroom and is_shared_bathroom.

has_availability: This column was dropped as it showed little variance.

instant_bookable: Converted to binary (1 for True, 0 for False).

1.1.3 Numerical Features

latitude, longitude:

- Adjusted to centralized values for geographical interpretation.
- A **categorical area feature (area_category)** was developed based on geographical clusters. It is important to note that this feature was entirely manually crafted, with specific dividing lines clearly marked in the corresponding figure. The importance of this feature is evident, as it substantially enhances the accuracy of price predictions. This is further illustrated in the feature importance plot, where **area_category** ranks prominently. In terms of direct impact, omitting this manually defined feature would lead to an approximate increase of 0.01 in the root mean square error (RMSE).

host_response_rate, host_acceptance_rate: Missing values were replaced with a placeholder value of 9999.

host_listings_count, host_total_listings_count: Ratios (host_listings_ratio and calculated_to_listings_ratio) were created to capture trends.

calculated_host_listings_count, calculated_host_listings_count_entire_homes, calculated_host_listings_count_private_rooms, calculated_host_listings_count_shared_rooms: Retained for analysis without modification.

accommodates: Binned into categories such as 1 person, 2 persons, 3-5 persons, and 6+ persons.

bathrooms, bedrooms, beds: Missing values were replaced with 0.

availability_30, availability_60, availability_90, availability_365: Normalized by their respective timeframes.

minimum_nights, maximum_nights: 1125 in maximum_nights was replaced with 9999.

number_of_reviews, number_of_reviews_ltm, number_of_reviews_l30d: Retained as-is for further modeling.

review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_scores_location, review_scores_value: Missing values replaced with 9999.

reviews_per_month: Missing values replaced with 0.

1.1.4 Date Variables

host_since, first_review, last_review:

- Split into year and month.
- Intervals between dates were calculated as new features (host_to_first_review_months, first_to_last_review_months).
- Additional features for months from the current date (months_from_host_since, months_from_first_review) were added.

1.1.5 Target

price

1.1.6 Transform

Scale high-variance columns: Columns with a standard deviation greater than 1 were transformed using a log1p transformation ($\log(x+1)$) to reduce skewness and normalize their distributions. This ensures that features with high variance do not dominate the modeling process, improving the overall performance of the model.

1.1.7 Dropped Columns

The following columns were removed from the dataset during preprocessing:

- **name, description, reviews, amenities, property_type, host_since, first_review, last_review, host_verifications, phone, email, work_email, bathrooms_text, availability_365, has_availability.**

Reasons for dropping columns:

- **Processed columns:** Columns such as **name, description, reviews, amenities, property_type, host_since, first_review, last_review** were fully utilized for feature extraction. Once the relevant features were derived, these original columns became redundant and were dropped to reduce noise.
- **Low importance:** Columns like **phone** and **availability_365** had minimal importance when evaluated against their influence on price prediction.
- **Intuitive justification:**
 - Host verification methods (**phone, email, work_email**) primarily provide static information about the host and are unlikely to directly influence the price at a specific time.
 - Yearly availability (**availability_365**) covers too long a period to provide relevant information for immediate pricing. Shorter-term availability, such as **availability_30** or **availability_60**, provides more actionable insights and was retained for modeling.

2 Model Selection: XGBoost and Support Vector Regression (SVR)

In this section, we clarify that our prediction metric is **Root Mean Squared Error (RMSE)**. Although the target variable consists of six categorical price intervals, there is an inherent ordinal relationship among these categories. Therefore, we adopt regression models to complete the task. One specific detail is that, when performing bagging after training the models, we use rounding followed by majority voting (i.e., mode) instead of directly using the average.

2.1 XGBoost Model

XGBoost is an efficient gradient-boosted decision tree algorithm, renowned for its computational efficiency and flexibility. Boosting methods are generally an excellent starting point, and specifically, XGBoost is particularly useful based on my experience. The reasons for selecting XGBoost are as follows:

- Decision Tree-based models and Boosting are my favorite algorithms learned in this course. Additionally, XGBoost is often referred to as the “champion” algorithm in many Kaggle competitions.
- XGBoost offers a wide range of hyperparameters for tuning, providing significant flexibility for fine-tuning in subsequent stages.
- XGBoost is computationally fast. Although it is slower than LightGBM, its speed is still acceptable within the scope of this project. In contrast, CatBoost was observed to be slower for our use case.

2.2 Support Vector Regression (SVR) Model

Support Vector Regression (SVR) is a kernel-based regression method suitable for modeling nonlinear relationships. From a theoretical perspective, radial basis functions (RBFs) are effective tools for mapping data into higher-dimensional spaces, which align with the kernel space concepts we studied in class. Therefore, I decided to experiment with SVR for this project. However, after comprehensive evaluation, XGBoost was ultimately chosen for the final model due to its superior performance. More specific,

- SVR performs well in high-dimensional feature spaces, especially with moderate-sized datasets.
- RBF kernels enable the modeling of nonlinear relationships, making SVR suitable for complex prediction problems.
- Tuned SVR models often exhibit strong generalization ability and are robust to outliers.

3 Training

3.1 XGBoost

XGBoost (eXtreme Gradient Boosting) is an efficient gradient-boosted decision tree algorithm. Its training process iteratively constructs decision trees, where each new tree fits the residuals of the previous iteration to progressively reduce model error. Additionally, XGBoost incorporates second-order derivative information, making loss function optimization more precise compared to GBDT, which only uses first-order Taylor expansion. Furthermore, regularization terms are added to prevent model overfitting.

3.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) finds the maximum-margin hyperplane in the feature space to perform classification. When data is not linearly separable in the original space, SVM employs kernel functions to map the data into a higher-dimensional space, making it linearly separable. For the linearly separable case, SVM transforms the problem into a dual optimization problem, solved using a quadratic solver. Commonly used kernel functions include linear kernel, polynomial kernel, and radial basis function (RBF) kernel. In the higher-dimensional space, SVM identifies the hyperplane that maximizes the margin, effectively classifying the data.

3.3 Time Estimation

Training the XGBoost model for approximately 500 iterations took about one and a half hours on an L4 GPU.

4 Hyperparameter Selection

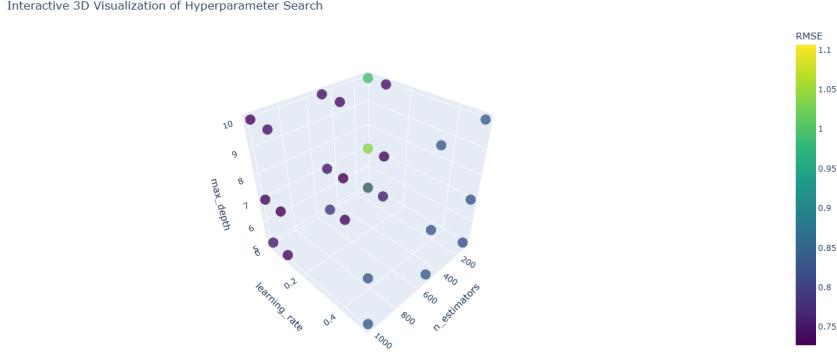


Figure 1: CV for XGBoost

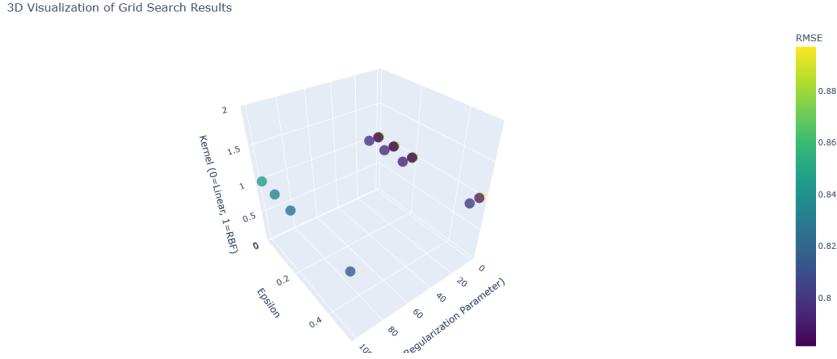


Figure 2: CV for SVM

My overall parameter training strategy involves first performing cross-validation (CV) to determine the basic important parameter ranges,(see Figure1,2). Since my final choice is **XGBoost**, which is based on a tree model, I prioritized tuning the most impactful hyperparameters: *max depth*, *learning rate*, and *number of estimators*. These hyperparameters directly influence the model's capacity to capture complexity, learning efficiency, and overfitting potential. initially setting a wide range to explore potential values. By narrowing the ranges through CV, I was able to set an effective foundation for further automated hyperparameter optimization. Afterward, I use the automated hyperparameter optimization tool, Optuna, to search for better parameters systematically. For the CV portion, I selected 2-3 key parameters for adjustment and plotted results to show the outcomes for both models. In these visualizations, darker colors indicate lower RMSE, and the plots reveal clear trends in parameter changes. Specifically, for my primary method, XGBoost, I experimented with various training approaches. Ultimately, I adopted the simplest approach, setting RMSE as the objective for regression, as it achieved the best results. My specific attempts included:

1. Custom-defined objective: Considering that our target output must be integers, I experimented with defining a custom objective that enforced integer constraints and then calculated RMSE for training.
2. Separate training: Since we observed that the presence or absence of reviews had a significant impact on price trends, I considered training separate models for listings with and without reviews.
3. Mapping predictions to integers: To better map predictions to integers, I explored the following approaches:
 - Rounding directly to the nearest integer.
 - Learning an adaptive shrinkage coefficient to adjust predictions based on the estimated value and the mean (however, this approach proved unstable, as the learned coefficients did not perform well on the test set).
 - Majority vote (major voting): This was the method I ultimately adopted, combining predictions from multiple strong models through voting to achieve a bagging effect. The optimal number of models used for voting was selected based on cross-validation results (see Figure 3).

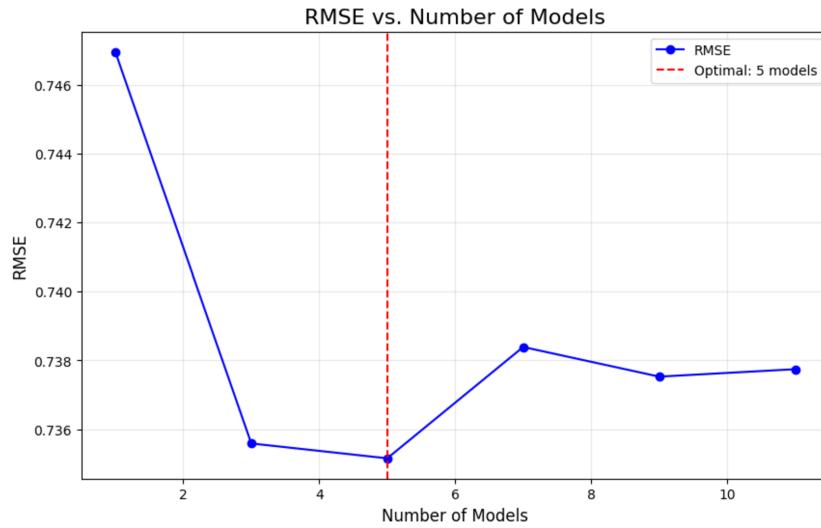


Figure 3: Cross-validation results for selecting the optimal number of models used in majority voting.

5 Data Splits

My data splitting strategy involves dividing the training data into 80% for training and 20% for testing. During training, I utilize 5-fold cross-validation to further validate the model. Throughout the training process, I ensure that the results from the 20% test data remain hidden to avoid data leakage. It is worth noting that the manually added categorical area feature, based on geographical clusters, does not lead to overfitting. This is because the region segmentation was manually defined and derived from the initial 80% training data, ensuring no overlap with the test set and mitigating the risk of overfitting.

6 Reflection on Progress

6.1 Challenges with Natural Language Processing (NLP)

One of the most significant challenges I faced was the failure to effectively process natural language information. I experimented with various approaches, including Latent Dirichlet Allocation (LDA), direct encoding with K-Nearest Neighbors (KNN), and sentiment extraction, but none yielded satisfactory results. For example, in the classification of amenities, I struggled to identify a reliable classification scheme. Ultimately, I manually curated the classification by initially extracting 20 basic categories and then refining the mapping until I was satisfied with the result. Similarly, I believe there is untapped potential in extracting more valuable insights from reviews. However, my limited familiarity with NLP hindered my ability to achieve better outcomes in this area. Although I considered using OpenAI's API for large-scale data processing, issues such as formatting inconsistencies rendered it unstable for datasets with thousands of entries. Overall, this area remains the most significant challenge in my project.

6.2 Handling Missing Values

Dealing with missing values also posed a considerable challenge. From the distribution of `price`, it became evident that missing values could significantly distort predictions. I hypothesized that imputing missing values with a distinct high value, such as 9999, could encode the information carried by their absence. I also experimented with splitting the data into two separate training sets based on whether certain values were missing. However, the results from this approach were suboptimal. For `reviews`, I compared replacing missing

values with a specific placeholder value against imputing the median (a more stable measure than the mean). The latter yielded slightly better numerical results, suggesting that a simpler imputation strategy could be more effective.

6.3 Feature Selection

In terms of feature selection, aside from eliminating obviously irrelevant features, most features—despite appearing uninformative (as suggested by low correlation coefficients, low importance scores, non-significant AUCs, or large p-values in stepwise regression)—still influenced the prediction results. For instance, I expected that setting a high threshold for filtering `neighbourhood` categories would improve performance, but the results were unsatisfactory. Similarly, when I attempted to train the model using only the top 10, 20, or 50 most important features, the results were consistently worse than using all available features. This indicates that even seemingly minor features contribute meaningfully when combined in the training process.

7 Kaggle Submission and Performance Analysis

My Kaggle username is **YixiaoWang0102**. Currently, my model achieves a score of **0.727** on the public leaderboard. Based on the results from previous submissions, we can draw the following conclusions:

- After processing textual information, the model’s score improved from **0.75** to **0.74**.
- After adding numerous detailed feature engineering steps, the model’s performance decreased slightly to **0.735**.
- Finally, using `mode` for bagging further improved the score from **0.733** to **0.727**.

These results indicate that **feature engineering** remains the most critical step in achieving better model performance.

8 Interpretability

I have done extensive work on the interpretability of feature engineering and feature importance. In this section, I will elaborate on my analytical steps for feature engineering,

which also serves as a supplement to the first section where I primarily focused on implementation details. Firstly, I show the feature importance and feature ROC curves here (see Figure 4, 5, 6).

8.1 Minimum Nights

Although I did not perform any specific feature engineering for this variable, the feature importance analysis revealed that the **minimum nights** variable significantly impacts price prediction. This aligns with our understanding: properties listed for longer-term rentals may have lower price expectations, as hosts often provide discounts to encourage extended stays. Conversely, short-term rentals, particularly those targeting tourists, tend to be priced higher due to higher demand and the premium nature of short-term accommodations.

8.2 Geographical Information Analysis

From the initial feature importance analysis (see Figure 7, 8), it became evident that longitude, latitude, and the Manhattan area significantly influence rental prices. This insight reinforced my belief that rental prices are heavily affected by geographical factors, aligning with common knowledge. To explore this further, I plotted longitude and latitude coordinates, using color to represent rental prices. While raw coordinates were too dispersed to be informative and neighborhood-level granularity was too coarse, careful observation revealed distinct clusters of high-price listings in Lower Manhattan and northern Brooklyn. Other areas in Manhattan and central Brooklyn showed a mix of high and medium-priced listings, while most other areas were dominated by low to medium-priced rentals. Based on these patterns, I manually divided the geographical regions into three distinct categories, capturing price variability effectively.

8.3 Amenity Features

I believe that both the number of amenities and the diversity of amenity types play a critical role in predicting rental prices. Initially, I only counted whether a listing had a certain type of amenity. However, after further consideration, I revised this to count the number of items within each amenity category. For instance, a listing with only one shampoo is significantly different from one that offers a full set of toiletries, including shampoo, conditioner, face wash, and cosmetics.

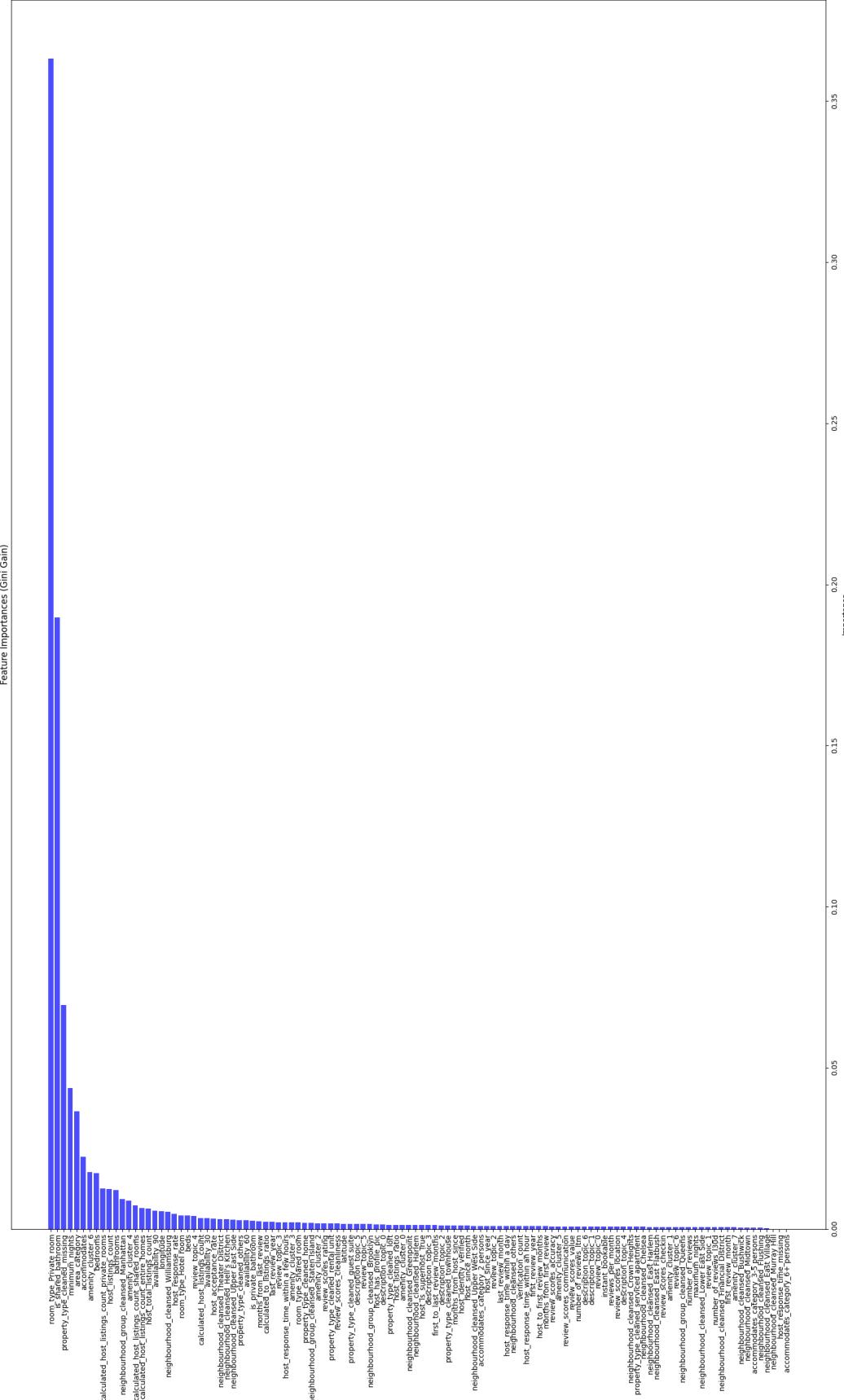


Figure 4: Feature importance based on the Gini index. This highlights the relative importance of features in predicting price. Features with higher Gini importance contribute more to the model's predictive power.

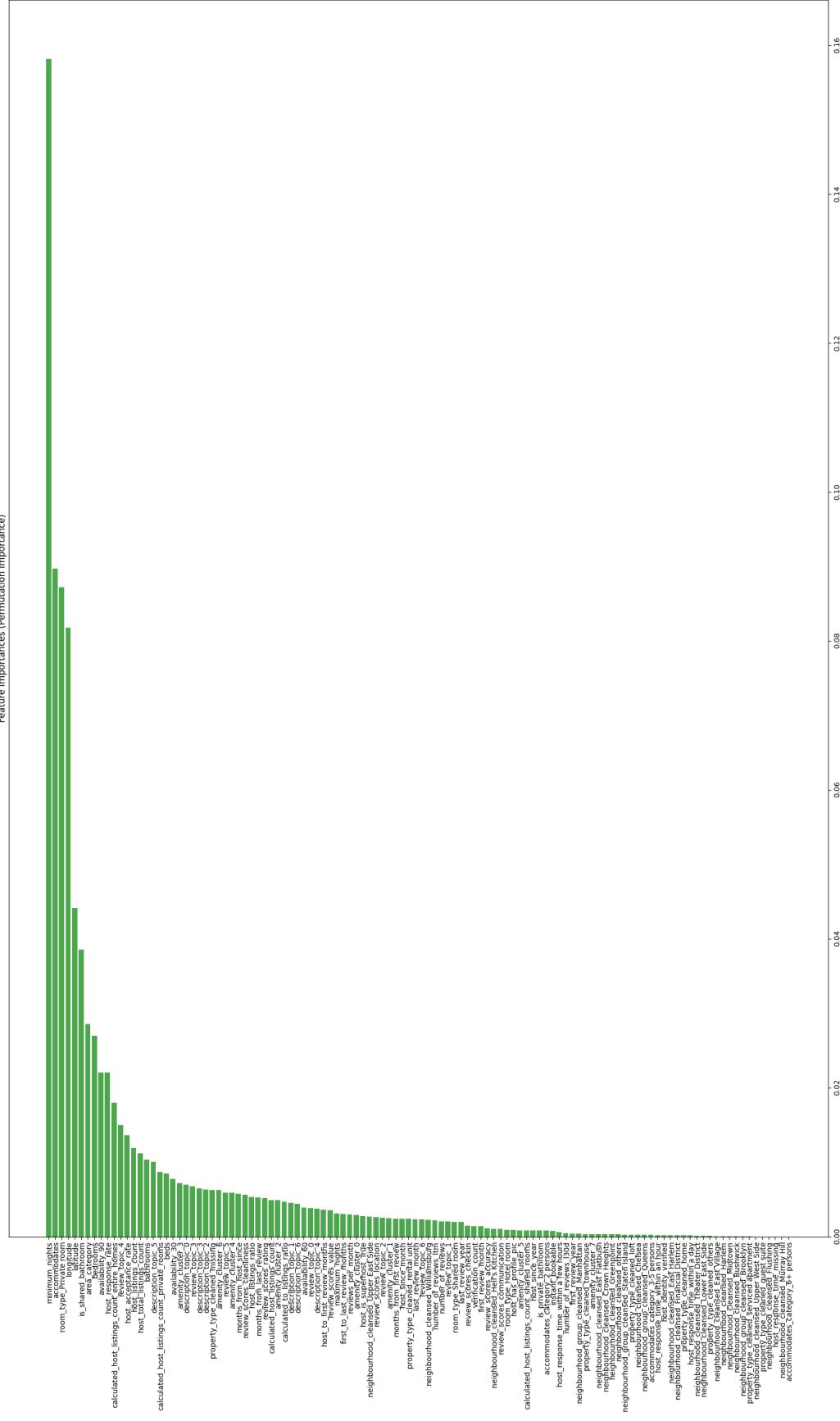


Figure 5: Permutation-based feature importance. This plot shows the impact of permuting each feature on the model's RMSE, providing insight into the robustness and reliability of each feature.

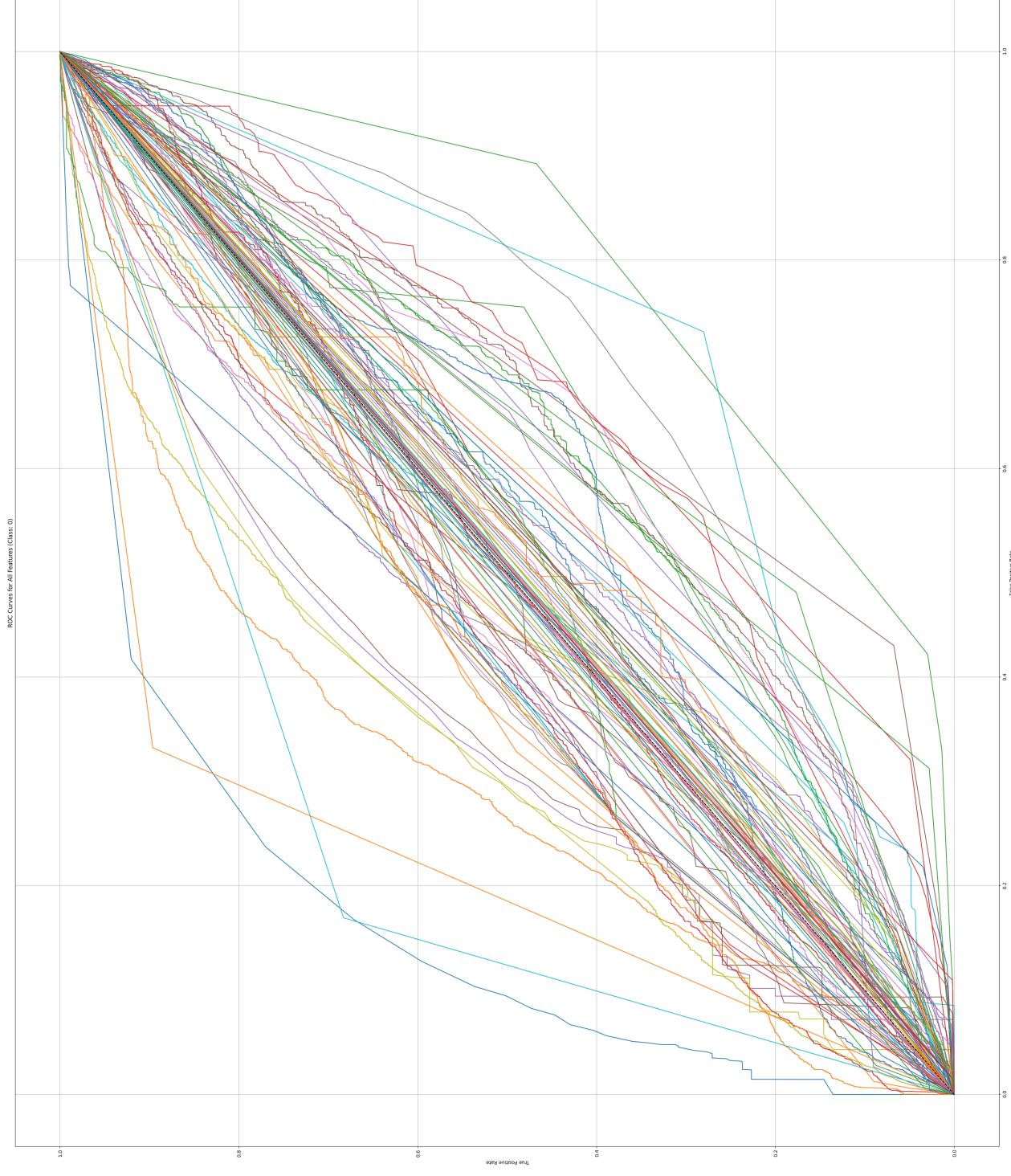


Figure 6: Feature importance based on AUC (Area Under the Curve). This plot emphasizes the discriminative power of individual features in predicting different price categories.

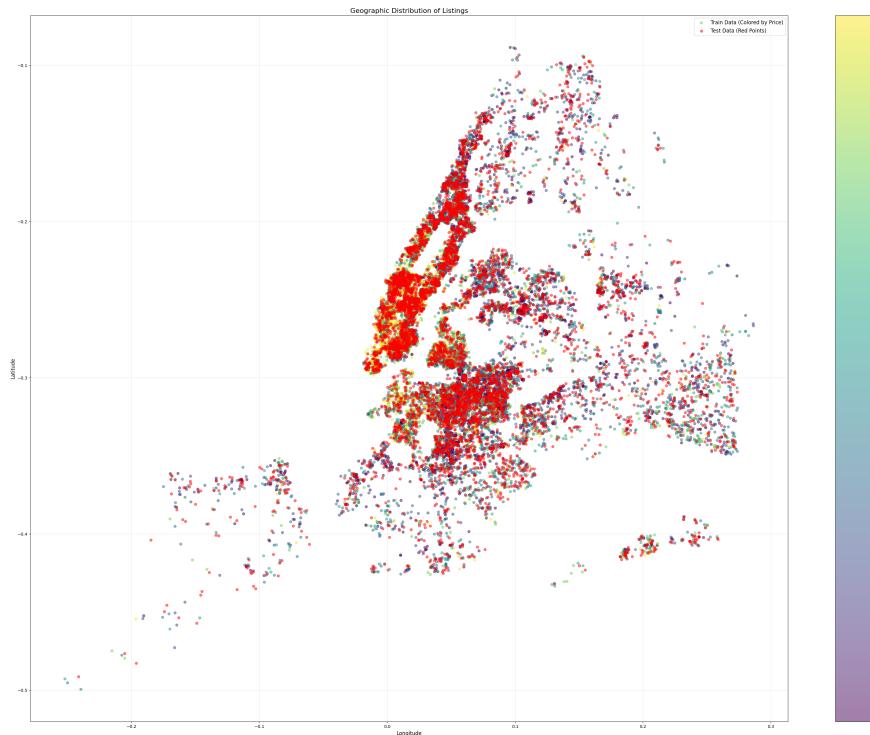


Figure 7: Geographic distribution of listings with prices. Red points indicate test data locations, showing that the test data is uniformly distributed.

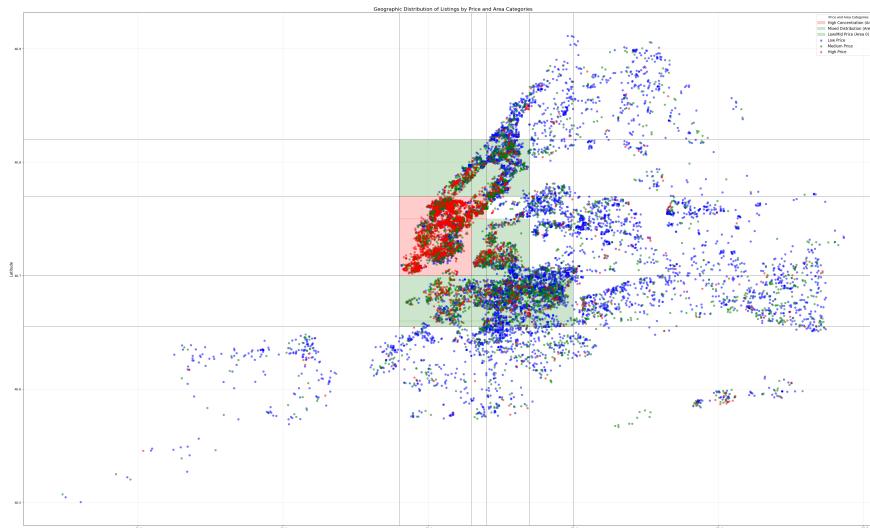


Figure 8: This map categorizes areas into three price levels (low, medium, high). Based on price concentration, areas with the highest concentration are marked with a red overlay (category 2), moderately mixed areas with green (category 1), and the remaining areas are labeled as category 0.

8.4 Reviews

During my analysis, I discovered that some reviews were not written in English but in languages such as Korean and Chinese. To address this, I used a translation tool to convert all reviews into English, ensuring the resulting feature was more robust and meaningful.

8.5 Availability Variables

For availability-related features, I observed that 30, 60, and 90-day availability variables have overlapping information. To address this, I tested two approaches: calculating differences between these variables and normalizing the available days by the total number of days. I chose the latter approach because normalization not only reduced the variability and scale of the data but also provided a direct measure of a property's "demand level" from short-term to long-term availability. Additionally, I removed the 365-day availability feature due to its low importance. Similarly, for host registration dates and review dates, I retained only the year and month information, discarding specific dates due to their low feature importance.

8.6 Time Interval Variables

For date-related variables, I focused on time intervals rather than absolute dates. These intervals fall into two categories. The first category includes the interval between the start of hosting and the first review, as well as the time between the first and most recent review. These intervals reflect how quickly a property becomes active and how long it remains active. The second category measures the time elapsed from these events to the current date, capturing how long the property has been on the market and the recency of user activity. Both categories are directly related to rental pricing.

8.7 Host Listing Ratios

For features related to host listings, such as cumulative listings, total listings, and listings across all platforms, I computed ratios to account for their inherent hierarchical relationships. These ratios reveal the host's preferences for specific platforms and correlate with their pricing strategies.

8.8 Room Privacy and Bathroom Text Analysis

Room privacy is a crucial determinant of pricing. To capture this, I explicitly extracted information on whether a bathroom was *private* or *shared* from the textual data. By parsing the descriptions for keywords such as “private” and “shared,” I created a categorical feature that significantly improved the interpretability and performance of the pricing model. This feature directly aligns with user expectations regarding privacy and amenities, further enhancing the model’s predictive power.

9 Code

Listing 1: Basic Preparation

```
1 # Download libraries
2 !pip install optuna
3 !pip install googletrans
4 !pip install langdetect
5 !pip install sweetviz
6
7 # Import libraries
8 # Import general libraries
9 import pandas as pd
10 import numpy as np
11 from datetime import datetime
12 from tqdm import tqdm
13 import re
14 import matplotlib.pyplot as plt
15 import seaborn as sns
16
17 # Import NLP-related libraries
18 from nltk.stem.porter import PorterStemmer
19 from nltk.tokenize import word_tokenize
20 from sklearn.feature_extraction.text import CountVectorizer
21 from sklearn.decomposition import LatentDirichletAllocation
22 from langdetect import detect
23 from googletrans import Translator
24
25 # Sentence embeddings and clustering
26 from sentence_transformers import SentenceTransformer
27 from sklearn.cluster import KMeans
28
29 # Machine Learning libraries
30 from sklearn.linear_model import LinearRegression
31 from sklearn.ensemble import AdaBoostRegressor
32 from sklearn.svm import SVR
33 from xgboost import XGBRegressor
34 from lightgbm import LGBMRegressor
35 from sklearn.impute import SimpleImputer
36 from sklearn.model_selection import train_test_split, KFold
37 from sklearn.metrics import mean_squared_error
38 from sklearn.inspection import permutation_importance
39 from sklearn.preprocessing import StandardScaler
40
```

```

41 # Optimization library
42 import optuna
43
44 # Statistical utilities
45 from scipy.stats import mode
46
47 # Visualization libraries
48 import plotly.graph_objects as go
49 import sweetviz as sv
50 from IPython.display import HTML

```

Listing 2: Data Download

```

1 # This is the original data, but the LDA process is quite lengthy, so
2     # we saved the original data with the LDA results included.
3 # The LDA code is provided in the preprocessing section but has been
4     # commented out.import pandas as pd
5 # train_data = pd.read_csv('/content/drive/My Drive/Colab
6     # Notebooks/final_project/train.csv', parse_dates=['host_since',
7         'first_review', 'last_review'])
8 # test_data = pd.read_csv('/content/drive/My Drive/Colab
9     # Notebooks/final_project/test.csv', parse_dates=['host_since',
10        'first_review', 'last_review'])
11
12 train_data = pd.read_csv('/content/drive/My Drive/Colab
13     # Notebooks/final_project/train_LDA.csv', parse_dates=['host_since',
14         'first_review', 'last_review'])
15 test_data = pd.read_csv('/content/drive/My Drive/Colab
16     # Notebooks/final_project/test_LDA.csv', parse_dates=['host_since',
17         'first_review', 'last_review'])
18
19 # This is the processed amenities mapping used for analyzing
20     # amenities, which is based on train_data
21 # and includes the parts I manually handled, so it can be directly
22     # imported.
23
24 # The categories for numbering are roughly as follows:
25 # 0: Entertainment and networking facilities, such as wifi, TV,
26     # Bluetooth speakers
27 # 1: Safety-related items, such as locker, alarm
28 # 2: Kitchen-related items, such as oven, refrigerator
29 # 3: Daily necessities, such as shampoo, conditioner
30 # 4: Sports and health facilities, such as gym, pool
31 # 5: Additional or paid services
32 # 6: Family- or baby-friendly facilities

```

```

19 # 7: Unclear classifications, such as information related to days of
20 # the week
21 amenities_df = pd.read_excel("/content/drive/My Drive/Colab
22 Notebooks/final_project/processed_amenities_cluster_mapping.xlsx")
23 amenities_cluster_mapping = dict(zip(amenities_df['amenity'],
24 amenities_df['cluster_id']))
25 print("First 10 items in amenities_cluster_mapping:")
26 for i, (key, value) in enumerate(amenities_cluster_mapping.items()):
27     print(f"{key}: {value}")
28     if i == 10:
29         break

```

Listing 3: EDA

```

1 # # Create a report
2 # report = sv.analyze(train_data)
3
4 # # Save the report as an HTML file
5 # report.show_html("/content/drive/My Drive/Colab
6 Notebooks/final_project/train_data_distribution.html")
7
8 # Load the HTML file content
9 with open("/content/drive/My Drive/Colab
10 Notebooks/final_project/train_data_distribution.html", "r") as f:
11 html_content = f.read()
12
13 # Display the report directly in Colab
14 HTML(html_content)
15
16 # Select only numerical columns from train_data
17 numerical_columns = train_data.select_dtypes(include=[np.number])
18
19 # Calculate the correlation matrix
20 correlation_matrix = numerical_columns.corr()
21
22 # Plot the correlation matrix using seaborn
23 plt.figure(figsize=(40, 30))
24 sns.heatmap(correlation_matrix, annot=True, fmt=".2f",
25 cmap="coolwarm", cbar=True, square=True)
26 plt.title("Correlation Matrix of Numerical Features", fontsize=16)
27 plt.xticks(rotation=45, ha='right')
28 plt.yticks(rotation=0)
29 plt.show()

```

```

28 import matplotlib.pyplot as plt
29
30 # We will mark the 'test_data' dataset in red and visualize
31 # 'train_data' with price as the color scale
32 # to check if the distribution is even.
33
34 # Set the figure size
35 plt.figure(figsize=(40, 30))
36
37 # Plot the scatter plot for 'train_data', with price represented by
38 # color
39 scatter_train = plt.scatter(
40     train_data['longitude'],      # Longitude data
41     train_data['latitude'],       # Latitude data
42     c=train_data['price'],        # Price data for color
43     cmap='viridis',              # Use 'viridis' colormap
44     alpha=0.5,                  # Set transparency
45     label='Train Data (Colored by Price)' # Legend label
46 )
47
48 # Plot the scatter plot for 'test_data', marked in red
49 plt.scatter(
50     test_data['longitude'],      # Longitude data
51     test_data['latitude'],       # Latitude data
52     color='red',                # Use red color for points
53     alpha=0.5,                  # Set transparency
54     label='Test Data (Red Points)' # Legend label
55 )
56
57 # Add a color bar to show the price scale
58 plt.colorbar(scatter_train, label='Price')
59
60 # Add title and axis labels
61 plt.title('Geographic Distribution of Listings', fontsize=16) # Set
62 # the title
63 plt.xlabel('Longitude', fontsize=12) # Set X-axis label
64 plt.ylabel('Latitude', fontsize=12) # Set Y-axis label
65
66 # Display grid lines for better readability
67 plt.grid(alpha=0.3)
68
69 # Add legend
70 plt.legend(fontsize=12)

```

```

69     # Show the plot
70     plt.show()
71
72     # Categorize 'price' into three levels
73     # Define price range bins
74     train_data['price_category'] = pd.cut(
75         train_data['price'],
76         bins=[-1, 2, 4, 6],    # Define ranges: Low (0-2), Medium (3-4), High
77             (5-6)
78         labels=['Low', 'Medium', 'High']    # Labels for categories
79     )
80
80     # Set color mapping
81     color_mapping = {'Low': 'blue', 'Medium': 'green', 'High': 'red'}
82     train_data['price_color'] =
83         train_data['price_category'].map(color_mapping)
84
84     # Plot the data
85     plt.figure(figsize=(50, 30))
86     for category, color in color_mapping.items():
87         subset = train_data[train_data['price_category'] == category]
88         plt.scatter(
89             subset['longitude'],
90             subset['latitude'],
91             c=color,
92             alpha=0.5,
93             label=f'{category} Price'
94         )
95
96     # Add gridlines for reference areas
97     plt.axhline(y=40.7, color='black', linestyle='--', linewidth=0.5)    #
98         Horizontal line at 40.7
99     plt.axhline(y=40.77, color='black', linestyle='--', linewidth=0.5)    #
100        Horizontal line at 40.77
101     plt.axhline(y=40.82, color='black', linestyle='--', linewidth=0.5)    #
102        Horizontal line at 40.82
103     plt.axhline(y=40.655, color='black', linestyle='--', linewidth=0.5)    #
104        Horizontal line at 40.655
105     plt.axhline(y=40.75, color='black', linestyle='--', linewidth=0.5)    #
106        Horizontal line at 40.75
107     plt.axhline(y=40.66, color='black', linestyle='--', linewidth=0.5)    #
108        Horizontal line at 40.66
109     plt.axvline(x=-73.9, color='black', linestyle='--', linewidth=0.5)    #
110        Vertical line at -73.9

```

```

104 plt.axvline(x=-73.97, color='black', linestyle='-', linewidth=0.5) #
    Vertical line at -73.97
105 plt.axvline(x=-73.93, color='black', linestyle='-', linewidth=0.5) #
    Vertical line at -73.93
106 plt.axvline(x=-74.02, color='black', linestyle='-', linewidth=0.5) #
    Vertical line at -74.02
107 plt.axvline(x=-73.96, color='black', linestyle='-', linewidth=0.5) #
    Vertical line at -73.96
108
109
110
111
112 # Add legend
113 plt.legend(title='Price Category', fontsize=12)
114
115 # Add title and axis labels
116 plt.title('Geographic Distribution of Listings by Price Category',
            fontsize=16)
117 plt.xlabel('Longitude', fontsize=12)
118 plt.ylabel('Latitude', fontsize=12)
119
120 # Display grid
121 plt.grid(alpha=0.3)
122
123 # Show the plot
124 plt.show()

```

Listing 4: Feature Engineering

```

1 # Initialize PorterStemmer
2 stemmer = PorterStemmer()
3
4 # # Initialize Translator
5 # translator = Translator()
6
7 # def translate_text(text):
8 #     """
9 #         Translate non-English text to English.
10 #     """
11 #     try:
12 #         if detect(text) != 'en': # Check if the text is not in
13 #             English
14 #                 return translator.translate(text, src='auto',
dest='en').text

```

```

14     #         return text
15     #     except:
16     #         return text # Return original text if translation fails
17
18     def preprocess_airbnb_data(dataframe,
19         amenities_cluster_mapping=amenities_cluster_mapping,
20         lda_model_desc=None,
21         lda_model_reviews=None, vectorizer_desc=None, vectorizer_reviews=None,
22         num_clusters=8, n_topics_desc=7, n_topics_reviews=7):
23         """
24             Preprocess the Airbnb dataset:
25             Includes date splitting, feature processing, LDA model generation and
26                 application, category alignment, etc.
27         """
28
29         ######
30         # 1. Date Processing
31         ######
32         # 1.1 Split date columns into year and month
33         def split_date_column(df, column_name):
34             """
35                 Splits a date column into year and month columns.
36                 Note: Day is not extracted, as it is less relevant to the importance
37                     of the data.
38
39                 df[f'{column_name}_year'] = df[column_name].dt.year
40                 df[f'{column_name}_month'] = df[column_name].dt.month
41                 return df
42
43         # List of date columns to process
44         date_columns = ['host_since', 'first_review', 'last_review']
45         for col in date_columns:
46             dataframe = split_date_column(dataframe, col)
47
48         # 1.2 Calculate date intervals in months
49         def calculate_date_intervals_months(df, start_col, end_col,
50             new_col_name):
51             """
52                 Calculates the interval (in months) between two date columns,
53                     handling NaN values.
54                 The calculation is based on full months (e.g., from 2023-01-15 to
55                     2023-03-10 equals 2 months).
56
57                 df = calculate_month_diff(start, end):
58                 if pd.isnull(start) or pd.isnull(end):

```

```

51     return 0 # Replace NaN with 0
52     return (end.year - start.year) * 12 + (end.month - start.month)
53
54 df[new_col_name] = df.apply(lambda row:
55     calculate_month_diff(row[start_col], row[end_col]), axis=1)
56
57 # Calculate intervals in months between relevant columns
58 dataframe = calculate_date_intervals_months(dataframe, 'host_since',
59     'first_review', 'host_to_first_review_months')
60 dataframe = calculate_date_intervals_months(dataframe,
61     'first_review', 'last_review', 'first_to_last_review_months')
62 dataframe['host_to_first_review_months'] =
63     dataframe['host_to_first_review_months'].apply(lambda x: max(x, 0))
64 dataframe['first_to_last_review_months'] =
65     dataframe['first_to_last_review_months'].apply(lambda x: max(x, 0))
66
67 # 1.3 Calculate months from current date
68 current_date = datetime(2024, 11, 1) # Set the current reference date
69
70 def calculate_months_from_now(df, column_name, new_column_name):
71 """
72 Calculates the number of months from a given date column to the
73 current date.
74 Missing values are filled with 0. For differences within the same
75 month, the result is 0.
76 """
77
78 def calculate_month_diff(start, end):
79     if pd.isnull(start) or pd.isnull(end):
80         return 0 # Missing values are replaced with a placeholder (0)
81     years_diff = end.year - start.year
82     months_diff = end.month - start.month
83     total_months = years_diff * 12 + months_diff
84     if end.day < start.day: # Adjust for partial months
85         total_months += 1
86     return max(total_months, 0)
87
88 df[new_column_name] = df[column_name].apply(lambda x:
89     calculate_month_diff(x, current_date))
90
91
92 # Calculate months from current date for relevant columns
93 print("Calculating months from current date...")
94 dataframe = calculate_months_from_now(dataframe, 'host_since',

```

```

    'months_from_host_since')

87  dataframe = calculate_months_from_now(dataframe, 'first_review',
88      'months_from_first_review')
89  dataframe = calculate_months_from_now(dataframe, 'last_review',
90      'months_from_last_review')

91  # 1.4 Drop original date columns
92  print("Dropping original date columns...")
93  dataframe.drop(columns=date_columns, inplace=True)

94  ######
95  # 2. Category Features Processing
96  #####
97
98  # 2.1 Creating binary columns for host verifications and counting
99  # methods of verification
100 verification_types = ['phone', 'email', 'work_email']
101 for verification in verification_types:
102     dataframe[verification] =
103         dataframe['host_verifications'].apply(lambda x: 1 if verification
104             in x else 0)

105
106  # Count the total number of verification methods
107  dataframe['verification_count'] =
108      dataframe[verification_types].sum(axis=1)

109
110  # 2.2 Creating columns for shared or private bathrooms
111  dataframe['is_shared_bathroom'] = dataframe['bathrooms_text'].apply(
112      lambda x: 1 if 'shared' in str(x).lower() else 0
113  )
114  dataframe['is_private_bathroom'] = dataframe['bathrooms_text'].apply(
115      lambda x: 1 if 'private' in str(x).lower() else 0
116  )

117  # 2.3 Creating ratio features for host listings
118  required_columns = ['host_listings_count',
119      'host_total_listings_count', 'calculated_host_listings_count']

120
121  # Ratios provide trend insights about the host's property data
122  dataframe['host_listings_ratio'] = dataframe['host_listings_count'] /
123      dataframe['host_total_listings_count']
124  dataframe['calculated_to_listings_ratio'] =
125      dataframe['calculated_host_listings_count'] /
126      dataframe['host_listings_count']

```

```

120
121 # 2.4 Cleaning and processing the 'property_type' column
122 def clean_property_type(column, threshold=20):
123 """
124 Cleans the 'property_type' column by standardizing text and grouping
125 rare categories into 'others'.
126
127 Parameters:
128 column (pd.Series): The property_type column to clean.
129 threshold (int): Minimum frequency to retain a category.
130
131 Returns:
132 pd.Series: Cleaned property_type column with low-frequency categories
133 grouped as 'others'.
134 """
135
136 # Standardize and clean property_type values
137 cleaned_column = (
138     column
139     .str.lower()
140     .str.replace(r"^(private room|shared room|entire|room in)\s+in\s+", "",
141                 regex=True)
142     .str.replace(r"^(private room|shared room|entire|room)", "", ,
143                 regex=True)
144     .str.replace(r"\sin\s.*$", "", , regex=True)
145     .str.replace(r"\bin\b", "", , regex=True)
146     .str.strip()
147 )
148
149 # Replace missing or empty values with 'missing'
150 cleaned_column = cleaned_column.replace(["", None, np.nan], "missing")
151
152 # Count the frequency of each category
153 value_counts = cleaned_column.value_counts()
154
155 # Group rare categories below the threshold into 'others'
156 cleaned_column = cleaned_column.apply(
157     lambda x: x if x == "missing" or value_counts[x] >= threshold else
158     "others"
159 )
160
161 return cleaned_column
162
163
164 # 2.5 Further processing for 'property_type_cleaned'
165 dataframe['property_type_cleaned'] =

```

```

    clean_property_type(dataframe['property_type'], threshold=20)
159
160    # Count the occurrences of each value and group rare values into
161    # 'others'
162    value_counts = dataframe['property_type_cleaned'].value_counts()
163    dataframe['property_type_cleaned'] =
164        dataframe['property_type_cleaned'].apply(
165            lambda x: x if value_counts[x] >= 100 else 'others'
166        )
167
168    # 2.6 Handling rare values in 'neighbourhood_cleansed'
169    if 'neighbourhood_cleansed' in dataframe.columns:
170        neighbourhood_counts =
171            dataframe['neighbourhood_cleansed'].value_counts()
172        rare_neighbourhoods = neighbourhood_counts[neighbourhood_counts <
173            200].index
174        dataframe['neighbourhood_cleansed'] =
175            dataframe['neighbourhood_cleansed'].replace(rare_neighbourhoods,
176            'others')
177
178    ######
179    # 3. Numerical Features Processing
180    #####
181
182    # 3.1 Normalize availability columns
183    availability_columns = {
184        'availability_365': 365,
185        'availability_90': 90,
186        'availability_60': 60,
187        'availability_30': 30
188    }
189    for col, total_days in availability_columns.items():
190        dataframe[col] = dataframe[col] / total_days
191
192    # 3.2 Handling missing values
193    dataframe.loc[dataframe['maximum_nights'] == 1125, 'maximum_nights'] =
194        9999 # Replace default value with a better estimate
195    dataframe['bedrooms'] = dataframe['bedrooms'].fillna(0) # Fill
196        missing values with 0
197    dataframe['bathrooms'] = dataframe['bathrooms'].fillna(0)
198
199    # 3.3 Create a new feature 'accommodates_category'
200    def categorize_accommodates(x):
201        if x == 1:

```

```

194     return '1 person'
195 elif x == 2:
196     return '2 persons'
197 elif 3 <= x <= 5:
198     return '3-5 persons'
199 else:
200     return '6+ persons'
201
202 dataframe['accommodates_category'] =
203     dataframe['accommodates'].apply(categorize_accommodates)
204
205 # Fill missing values for 'host_response_time',
206 dataframe['host_response_time'] =
207     dataframe['host_response_time'].fillna('missing')
208
209 # 3.4 Assign a geographical category based on latitude and longitude
210 def assign_area_category(row):
211 """
212 Assign an area category based on latitude and longitude rules.
213 Categories:
214 2: High concentration of specific price distributions
215 1: Mixed price distribution
216 0: Predominantly low/mid-price areas
217 """
218 if (
219     (40.75 <= row['latitude'] <= 40.77 and -74.02 <= row['longitude'] <
220         -73.96) or
221     (40.7 <= row['latitude'] <= 40.75 and -74.02 <= row['longitude'] <
222         -73.97)
223 ):
224     return 2
225 elif (
226     (40.77 <= row['latitude'] <= 40.82 and -74.02 <= row['longitude'] <
227         -73.93) or
228     (40.7 <= row['latitude'] <= 40.75 and -73.97 <= row['longitude'] <
229         -73.93) or
230     (40.66 <= row['latitude'] <= 40.7 and -74.02 <= row['longitude'] <
231         -73.9)
232 ):
233     return 1
234 else:
235     return 0
236
237 dataframe['area_category'] = dataframe.apply(assign_area_category,

```

```

        axis=1)

231
232     # Adjust latitude and longitude to centralized values
233     dataframe['latitude'] = dataframe['latitude'] - 41
234     dataframe['longitude'] = dataframe['longitude'] + 74
235
236     # 3.5 Fill missing values for review scores and rates with a default
237     # value
238     missing_vars = [
239         'review_scores_rating', 'review_scores_accuracy',
240         'review_scores_cleanliness',
241         'review_scores_checkin', 'review_scores_communication',
242         'review_scores_location',
243         'review_scores_value', 'reviews_per_month', 'host_response_rate',
244         'host_acceptance_rate'
245     ]
246     for var in missing_vars:
247         dataframe[var] = dataframe[var].fillna(9999)
248
249 ######
250 # 4. Text Data Processing
251 ######
252
253 # # 4.1 LDA Feature Extraction Function
254 # def lda_processing(text_column, vectorizer=None, lda_model=None,
255 #     n_topics=5):
256 #     """
257 #         Extract LDA features from a text column.
258 #
259 #         Parameters:
260 #             text_column (pd.Series): Column containing text data.
261 #             vectorizer (CountVectorizer): Optional pre-fitted
262 #                 CountVectorizer.
263 #             lda_model (LatentDirichletAllocation): Optional pre-fitted
264 #                 LDA model.
265 #             n_topics (int): Number of topics for LDA.
266 #
267 #         Returns:
268 #             tuple: Topic matrix, fitted vectorizer, and fitted LDA
269 #                 model.
270 #             """
271 #             print(f"Vectorizing {text_column.name}...")
272 #             if vectorizer is None:
273 #                 vectorizer = CountVectorizer(max_features=5000,

```

```

        stop_words='english')
266     #         text_matrix =
267     #         vectorizer.fit_transform(text_column.astype(str))
268     #     else:
269     #         text_matrix = vectorizer.transform(text_column.astype(str))
270
271     #     print(f"Applying LDA on {text_column.name}...")
272     #     if lda_model is None:
273     #         lda_model =
274     #             LatentDirichletAllocation(n_components=n_topics, random_state=42)
275     #             topic_matrix = lda_model.fit_transform(text_matrix)
276     #     else:
277     #         topic_matrix = lda_model.transform(text_matrix)
278
279
280     # # 4.2 Text Preprocessing Function
281     # def preprocess_text(text, stemmer):
282     #     """
283     #         Custom text preprocessing: tokenization, stemming, removing
284     #             special characters.
285
286     #         Parameters:
287     #             text (str): Input text.
288     #             stemmer (PorterStemmer): Stemmer instance for stemming
289     #                 tokens.
290
291     #         Returns:
292     #             str: Processed text.
293     #             """
294
295     #         if pd.isnull(text) or text.strip() == '':
296     #             return "missing"
297
298     #         # Replace escape characters
299     #         text = text.replace("\\\\'", "'").replace("\\\\\"", "\"")
300
301     #         # Remove special characters and convert to lowercase
302     #         text = re.sub(r'\W+', ' ', text).lower()
303
304     #         # Tokenize and apply stemming
305     #         tokens = text.split()
306     #         stemmed_tokens = [stemmer.stem(token) for token in tokens]
307     #         return ' '.join(stemmed_tokens)

```

```

305
306
307     # # 4.3 Preprocessing and LDA for 'description'
308     # tqdm.pandas()
309     # stemmer = PorterStemmer()
310
311     # print("Preprocessing descriptions...")
312     # dataframe['description'] =
313         #     dataframe['description'].astype(str).progress_apply(
314             #         lambda x: preprocess_text(x, stemmer)
315             #     )
316
317     # print("Applying LDA on description...")
318     # description_topics, vectorizer_desc, lda_model_desc =
319         #     lda_processing(
320             #         dataframe['description'], vectorizer=None, lda_model=None,
321             #         n_topics=5
322             #     )
323
324     # for i in range(description_topics.shape[1]):
325     #     dataframe[f'description_topic_{i}'] = description_topics[:, i]
326
327
328     # # 4.4 Preprocessing and LDA for 'reviews'
329     # print("Translating reviews...")
330     # dataframe['reviews'] =
331         #     dataframe['reviews'].astype(str).progress_apply(
332             #         lambda x: translate_text(x) # Assumes 'translate_text' is
333                 # defined elsewhere
334             #     )
335
336     # print("Preprocessing reviews...")
337     # dataframe['reviews'] = dataframe['reviews'].progress_apply(
338         #         lambda x: preprocess_text(x, stemmer)
339         #     )
340
341     # print("Applying LDA on reviews...")
342     # review_topics, vectorizer_reviews, lda_model_reviews =
343         #     lda_processing(
344             #         dataframe['reviews'], vectorizer=None, lda_model=None,
345             #         n_topics=5
346             #     )
347
348     # for i in range(review_topics.shape[1]):
349     #     dataframe[f'review_topic_{i}'] = review_topics[:, i]
350

```

```

342 # 4.5 Amenities Clustering
343 print("Processing amenities...")
344 dataframe['amenities'] = dataframe['amenities'].apply(
345     lambda x: str(x).replace('[', '').replace(']', '').replace('"',
346     '')).split(',')
347 )
348
349 # Generate cluster mapping if not provided
350 if amenities_cluster_mapping is None:
351     print("Generating amenities cluster mapping...")
352
353 # Extract all unique amenities
354 all_amenities = dataframe['amenities'].explode().unique().tolist()
355 print(f"Unique amenities found: {len(all_amenities)}")
356
357 # Generate embeddings using Sentence-BERT
358 sbert_model = SentenceTransformer('all-distilroberta-v1')
359 print("Generating embeddings for amenities...")
360 amenity_embeddings = sbert_model.encode(all_amenities)
361
362 # Perform clustering
363 kmeans = KMeans(n_clusters=num_clusters, random_state=42)
364 clusters = kmeans.fit_predict(amenity_embeddings)
365
366 # Map amenities to clusters
367 amenities_cluster_mapping = dict(zip(all_amenities, clusters))
368 print("Amenity cluster mapping created.")
369
370 # Initialize cluster count columns
371 for cluster_id in range(num_clusters):
372     dataframe[f'amenity_cluster_{cluster_id}'] = 0
373
374 # Count amenities per cluster for each row
375 print("Counting amenities per cluster for each row...")
376 for index, amenities_list in tqdm(dataframe['amenities'].items(),
377                                     desc="Processing amenities"):
378     cluster_counts = {}
379     for amenity in amenities_list:
380         if amenity in amenities_cluster_mapping:
381             cluster_id = amenities_cluster_mapping[amenity]
382             cluster_counts[cluster_id] = cluster_counts.get(cluster_id, 0) + 1
383
384 # Assign counts to respective cluster columns
385 for cluster_id, count in cluster_counts.items():

```

```

384     dataframe.at[index, f'amenity_cluster_{cluster_id}'] = count
385
386     ######
387     # 5. Transform
388     #####
389
390     # 5.1 Scale high-variance columns
391     # Select numeric columns (excluding 'price') for scaling
392     scale_sensitive_columns = dataframe.select_dtypes(include=['float64',
393             'int64']).columns.drop('price', errors='ignore')
394     # Exclude 'amenity_cluster' columns from scaling
395     scale_sensitive_columns = [col for col in scale_sensitive_columns if
396         not col.startswith('amenity_cluster')]
397
398     # Apply logarithmic transformation to high-variance columns
399     for col in scale_sensitive_columns:
400         if dataframe[col].std() > 1:  # Log-transform only if the standard
401             deviation is high
402             dataframe[col] = np.log1p(dataframe[col])  # Use log1p to handle zero
403             values safely
404
405     # 5.2 Drop unnecessary columns
406     columns_to_drop = [
407         'name', 'description', 'reviews', 'amenities', 'property_type',
408         'host_since', 'first_review', 'last_review', 'host_verifications',
409         'phone', 'email', 'work_email', 'bathrooms_text', 'availability_365',
410         'has_availability'
411     ]
412
413     # Drop specified columns, ignoring errors if they don't exist in the
414     # dataframe
415     dataframe.drop(columns=columns_to_drop, inplace=True, errors='ignore')
416
417     return dataframe, amenities_cluster_mapping, lda_model_desc,
418         lda_model_reviews, vectorizer_desc, vectorizer_reviews
419
420
421     # Preprocess train and test datasets with shared mappings/models
422     train_data, amenities_cluster_mapping, lda_model_desc,
423         lda_model_reviews, vectorizer_desc, vectorizer_reviews =
424             preprocess_airbnb_data(train_data)
425
426     test_data, _, _, _, _, _ = preprocess_airbnb_data(

```

```

420     test_data, amenities_cluster_mapping, lda_model_desc,
421         lda_model_reviews, vectorizer_desc, vectorizer_reviews
422     )
423
424     # One-hot encode categorical variables in train and test datasets
425     train_data_dummies = pd.get_dummies(train_data, drop_first=True)
426     test_data_dummies = pd.get_dummies(test_data, drop_first=True)
427
428     # Align test data columns with train data, filling missing columns
429         with 0
430     test_data_dummies =
431         test_data_dummies.reindex(columns=train_data_dummies.columns,
432             fill_value=0)
433
434     # Ensure 'price' is not included in the test dataset
435     if 'price' in test_data_dummies.columns:
436         test_data_dummies = test_data_dummies.drop(columns=['price'])
437
438     # Verify the processed test dataset
439     test_data_dummies.head()

```

Listing 5: Data Download

```

1  # This is the original data, but the LDA process is quite lengthy, so
2      we saved the original data with the LDA results included.
3
4  # The LDA code is provided in the preprocessing section but has been
5      commented out.
6  import pandas as pd
7
8  # train_data = pd.read_csv('/content/drive/My Drive/Colab
9      Notebooks/final_project/train.csv', parse_dates=['host_since',
10         'first_review', 'last_review'])
11
12 # test_data = pd.read_csv('/content/drive/My Drive/Colab
13     Notebooks/final_project/test.csv', parse_dates=['host_since',
14         'first_review', 'last_review'])
15
16
17 train_data = pd.read_csv('/content/drive/My Drive/Colab
18     Notebooks/final_project/train_LDA.csv', parse_dates=['host_since',
19         'first_review', 'last_review'])
20
21 test_data = pd.read_csv('/content/drive/My Drive/Colab
22     Notebooks/final_project/test_LDA.csv', parse_dates=['host_since',
23         'first_review', 'last_review'])
24
25
26 # This is the processed amenities mapping used for analyzing
27     amenities, which is based on train_data
28
29 # and includes the parts I manually handled, so it can be directly

```

```

    imported.

11 # The categories for numbering are roughly as follows:
12 # 0: Entertainment and networking facilities, such as wifi, TV,
13     Bluetooth speakers
14 # 1: Safety-related items, such as locker, alarm
15 # 2: Kitchen-related items, such as oven, refrigerator
16 # 3: Daily necessities, such as shampoo, conditioner
17 # 4: Sports and health facilities, such as gym, pool
18 # 5: Additional or paid services
19 # 6: Family- or baby-friendly facilities
20 # 7: Unclear classifications, such as information related to days of
21     the week
22
23 amenities_df = pd.read_excel("/content/drive/My Drive/Colab
24 Notebooks/final_project/processed_amenities_cluster_mapping.xlsx")
25 amenities_cluster_mapping = dict(zip(amenities_df['amenity'],
26                                     amenities_df['cluster_id']))
27 print("First 10 items in amenities_cluster_mapping:")
28 for i, (key, value) in enumerate(amenities_cluster_mapping.items()):
29     print(f"{key}: {value}")
30     if i == 10:
31         break

```

Listing 6: Feature Importance

```

1 # Separate features and target
2 target = train_data_dummies['price']
3 features = train_data_dummies.drop(columns=['price'])

4
5 # Train an XGBoost model
6 model = XGBRegressor(random_state=42, n_estimators=500,
7     learning_rate=0.05)
8 model.fit(features, target)

9 # Calculate Gini Gain-based feature importance
10 gini_importance = model.feature_importances_

11
12 # Calculate Permutation Importance
13 perm_importance = permutation_importance(model, features, target,
14     n_repeats=10, random_state=42)

15 # Sort by Gini gain importance
16 sorted_idx_gini = np.argsort(gini_importance)[::-1]
17 features_sorted_gini = features.columns[sorted_idx_gini]
18 gini_sorted = gini_importance[sorted_idx_gini]

```

```

19
20 # Sort by Permutation Importance
21 sorted_idx_perm = perm_importance.importances_mean.argsort()[:-1]
22 features_sorted_perm = features.columns[sorted_idx_perm]
23 perm_sorted = perm_importance.importances_mean[sorted_idx_perm]
24
25 # Plot Gini Gain Feature Importance
26 plt.figure(figsize=(30, 20))
27 plt.barh(features_sorted_gini, gini_sorted, color="blue", alpha=0.7)
28 plt.gca().invert_yaxis()
29 plt.title("Feature Importances (Gini Gain)")
30 plt.xlabel("Importance")
31 plt.show()
32
33 # Plot Permutation Importance
34 plt.figure(figsize=(30, 20))
35 plt.barh(features_sorted_perm, perm_sorted, color="green", alpha=0.7)
36 plt.gca().invert_yaxis()
37 plt.title("Feature Importances (Permutation Importance)")
38 plt.xlabel("Importance")
39 plt.show()
40
41 import pandas as pd
42 import matplotlib.pyplot as plt
43 from sklearn.metrics import roc_curve, auc
44 from sklearn.preprocessing import label_binarize, LabelEncoder
45
46 # Assuming train_data_dummies is already loaded
47 # Extract 'price' as target and binarize it
48 target = train_data_dummies['price']
49 features = train_data_dummies.drop(columns=['price'])
50
51 # Encode target into numeric classes if not already encoded
52 le = LabelEncoder()
53 target_encoded = le.fit_transform(target)
54
55 # Binarize the target for multiclass OvR
56 target_binarized = label_binarize(target_encoded,
57                                 classes=range(len(le.classes_)))
58
59 # Select a specific class (e.g., Class 0)
60 class_index = 0 # Change this index for other classes
61 class_name = le.classes_[class_index]

```

```

62     # Initialize the plot
63     plt.figure(figsize=(40, 30))
64
65     # Loop through each feature and plot ROC curve for the selected class
66     for feature in features.columns:
67         feature_values = features[feature]
68
69         # Handle missing values by filling with the median or another method
70         if feature_values.isna().any():
71             feature_values = feature_values.fillna(feature_values.median())
72
73         # Handle non-numeric features by encoding them
74         if not pd.api.types.is_numeric_dtype(feature_values):
75             feature_values = pd.factorize(feature_values)[0]
76
77         # Compute ROC curve and AUC for the selected class
78         fpr, tpr, _ = roc_curve(target_binarized[:, class_index],
79                                 feature_values)
80         roc_auc = auc(fpr, tpr)
81
82         # Add the ROC curve to the plot
83         plt.plot(fpr, tpr, label=f'{feature} (AUC = {roc_auc:.2f})')
84
85         # Finalize the plot
86         plt.plot([0, 1], [0, 1], 'k--', label='Random chance') # Diagonal
87         line
88         plt.xlabel('False Positive Rate')
89         plt.ylabel('True Positive Rate')
90         plt.title(f'ROC Curves for All Features (Class: {class_name})')
91         plt.legend(loc='best', bbox_to_anchor=(1.05, 1))
92         plt.grid()
93         plt.tight_layout()
94         plt.show()

```

Listing 7: Data Split

```

1  # Separate features and target from the train dataset
2  target = train_data_dummies['price']
3  features = train_data_dummies.drop(columns=['price'])
4
5  # Split into training and testing sets (80% train, 20% test)
6  X_train, X_test, y_train, y_test = train_test_split(
7      features, target, test_size=0.2, random_state=42
8  )

```

```

9      print(f"Training set size: {X_train.shape}, Test set size:
10         {X_test.shape}")
11
12     # Initialize the imputer for missing values
13     imputer = SimpleImputer(strategy='median')
14
15     # Impute missing values for train and test sets
16     X_train = imputer.fit_transform(X_train)
17     X_test = imputer.transform(X_test)
18
19     # Impute missing values for the full training dataset and the true
20       test dataset
21     X_full_train = imputer.fit_transform(features) # Full training
22       features
23     y_full_train = target
24     X_true_test = imputer.transform(test_data_dummies) # True test
25       dataset features

```

Listing 8: XGBoost

```

1     # Ensure that X_train and y_train are NumPy arrays
2     X_train = np.array(X_train)
3     y_train = np.array(y_train)
4
5     # Parameter search range
6     n_estimators_range = [100, 500, 1000]
7     learning_rate_range = [0.01, 0.1, 0.5]
8     max_depth_range = [5, 7, 10]
9
10    # Cross-validation setup
11    kf = KFold(n_splits=5, shuffle=True, random_state=42)
12
13    # Store results for visualization
14    results = []
15
16    # Iterate through parameter combinations
17    for n_estimators in n_estimators_range:
18        for learning_rate in learning_rate_range:
19            for max_depth in max_depth_range:
20                print(f"Processing: n_estimators={n_estimators},
21                  learning_rate={learning_rate}, max_depth={max_depth}")
22
23    rmse_scores = []

```

```

24     # 5-fold cross-validation
25     for train_index, valid_index in kf.split(X_train):
26         # Split training and validation sets
27         X_train_cv, X_valid_cv = X_train[train_index], X_train[valid_index]
28         y_train_cv, y_valid_cv = y_train[train_index], y_train[valid_index]
29
30     # Build the model
31     model_xgb = XGBRegressor(
32         n_estimators=n_estimators,
33         learning_rate=learning_rate,
34         max_depth=max_depth,
35         tree_method="hist", # Use histogram-based method for faster training
36         random_state=42
37     )
38
39     # Train the model
40     model_xgb.fit(X_train_cv, y_train_cv, eval_set=[(X_valid_cv,
41             y_valid_cv)], verbose=False)
42
43     # Predict on the validation set
44     y_pred_cv = model_xgb.predict(X_valid_cv)
45     rmse_cv = np.sqrt(mean_squared_error(y_valid_cv, y_pred_cv))
46     rmse_scores.append(rmse_cv)
47
48     # Calculate the average RMSE for the current parameter combination
49     mean_rmse = np.mean(rmse_scores)
50     print(f"  Average RMSE: {mean_rmse:.4f}")
51
52     # Append the result for visualization
53     results.append((n_estimators, learning_rate, max_depth, mean_rmse))
54
55     # Convert results to a structured format
56     results = np.array(results)
57     n_estimators_vals = results[:, 0]
58     learning_rate_vals = results[:, 1]
59     max_depth_vals = results[:, 2]
60     rmse_vals = results[:, 3]
61
62     # Find the best parameters and RMSE
63     best_index = np.argmin(rmse_vals)
64     best_params = {
65         "n_estimators": int(n_estimators_vals[best_index]),
66         "learning_rate": learning_rate_vals[best_index],
67         "max_depth": int(max_depth_vals[best_index])

```

```

67     }
68     best_rmse = rmse_vals[best_index]
69
70     print("\nOptimization complete.")
71     print("Best Parameters:")
72     print(best_params)
73     print(f"Best RMSE: {best_rmse:.4f}")
74
75     # Create an interactive 3D scatter plot with Plotly
76     fig = go.Figure()
77
78     # Add scatter points
79     fig.add_trace(go.Scatter3d(
80         x=n_estimators_vals,
81         y=learning_rate_vals,
82         z=max_depth_vals,
83         mode='markers',
84         marker=dict(
85             size=8,
86             color=rmse_vals, # Color by RMSE
87             colorscale='Viridis', # Color scale
88             colorbar=dict(title="RMSE"),
89             opacity=0.8
90         )
91     ))
92
93     # Set axis labels and title
94     fig.update_layout(
95         scene=dict(
96             xaxis_title="n_estimators",
97             yaxis_title="learning_rate",
98             zaxis_title="max_depth"
99         ),
100        title="Interactive 3D Visualization of Hyperparameter Search",
101        margin=dict(l=0, r=0, b=0, t=40)
102    )
103
104    # Show the plot
105    fig.show()
106
107
108    # Define the Optuna objective function
109    def objective(trial):
110        # Define hyperparameters to optimize

```

```

111 param = {
112     "n_estimators": trial.suggest_int("n_estimators", 500, 1500),
113     "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.2),
114     "max_depth": trial.suggest_int("max_depth", 7, 15),
115     "subsample": trial.suggest_float("subsample", 0.8, 1.0),
116     "colsample_bytree": trial.suggest_float("colsample_bytree", 0.6,
117         1.0),
118     "gamma": trial.suggest_float("gamma", 0.07, 0.1),
119     "reg_alpha": trial.suggest_float("reg_alpha", 1, 20),
120     "reg_lambda": trial.suggest_float("reg_lambda", 0, 10),
121     "min_child_weight": trial.suggest_int("min_child_weight", 1, 10),
122     "scale_pos_weight": trial.suggest_float("scale_pos_weight", 1, 2),
123     "max_delta_step": trial.suggest_int("max_delta_step", 0, 10),
124     "grow_policy": "depthwise",
125     "random_state": 42
126 }
127
128 # Create XGBoost model
129 model_xgb = XGBRegressor(**param)
130
131 # Train the model
132 model_xgb.fit(X_train, y_train, eval_set=[(X_test, y_test)],
133                 verbose=False)
134
135 # Predict on the validation set and calculate RMSE
136 y_pred_xgb = model_xgb.predict(X_test)
137 rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
138 return rmse_xgb
139
140 # Use Optuna to optimize hyperparameters
141 study_xgb = optuna.create_study(direction="minimize")
142 study_xgb.optimize(objective, n_trials=100) # Run 500 trials for the
143                                         final submission
144
145 # Print the best parameters and RMSE
146 print("Best parameters (XGBoost):", study_xgb.best_params)
147 print("Best RMSE (XGBoost):", study_xgb.best_value)
148 # Retrieve the top 51 trial parameters (51 for final submission)
149 top_trials_xgb =
150     study_xgb.trials_dataframe().sort_values(by="value").head(11)
151 counter = 0
152 models_xgb = []
153
154 # Train models using the top 51 trial parameters

```

```

151     for _, trial_row in top_trials_xgb.iterrows():
152         counter += 1
153         print(f"Training model {counter}...")
154
155         # Extract parameters
156         trial_number = int(trial_row["number"])
157         best_params_xgb = study_xgb.trials[trial_number].params
158
159         # Create and train the model
160         model_xgb = XGBRegressor(**best_params_xgb)
161         model_xgb.fit(X_train, y_train, verbose=False)
162         models_xgb.append(model_xgb)
163
164         # Get predictions from all models
165         predictions_xgb = np.array([model.predict(X_test) for model in
166                                     models_xgb])
167
168         # Clip predictions to the range [0, 5] and round them
169         predictions_xgb_clipped = np.round(np.clip(predictions_xgb, 0, 5))
170
171         # Initialize the RMSE results list
172         rmse_results = []
173
174         # Compute RMSE for 1, 3, 5, ..., 51 models
175         for num_models in range(1, 12, 2): # 1, 3, 5, ..., 51
176             # Compute the mode of predictions from the first num_models models
177             y_pred_xgb_mode = mode(predictions_xgb_clipped[:num_models],
178                                    axis=0).mode.squeeze()
179
180             # Ensure y_pred_xgb_mode is a 1D array with the same shape as y_test
181             y_pred_xgb_mode = y_pred_xgb_mode.flatten()
182
183             # Compute RMSE and store the result
184             rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb_mode))
185             rmse_results.append((num_models, rmse_xgb))
186
187             # Convert results to an array for further processing
188             rmse_results = np.array(rmse_results)
189
190             # Find the number of models that give the minimum RMSE
191             optimal_index = np.argmin(rmse_results[:, 1])
192             optimal_models = int(rmse_results[optimal_index, 0])
193             optimal_rmse = rmse_results[optimal_index, 1]

```

```

193     print(f"Optimal number of models: {optimal_models}")
194     print(f"Minimum RMSE: {optimal_rmse:.4f}")
195
196     # Plot RMSE vs. number of models
197     plt.figure(figsize=(10, 6))
198     plt.plot(rmse_results[:, 0], rmse_results[:, 1], marker='o',
199               linestyle='-', color='blue', label='RMSE')
200     plt.axvline(optimal_models, color='red', linestyle='--',
201                  label=f'Optimal: {optimal_models} models')
202     plt.title('RMSE vs. Number of Models', fontsize=16)
203     plt.xlabel('Number of Models', fontsize=12)
204     plt.ylabel('RMSE', fontsize=12)
205     plt.grid(alpha=0.3)
206     plt.legend()
207     plt.show()

```

Listing 9: SVM

```

1      # Standardize the features
2      scaler = StandardScaler()
3      X_train_scaled = scaler.fit_transform(X_train)
4      X_test_scaled = scaler.transform(X_test)
5      X_train_scaled = np.array(X_train_scaled)
6      y_train = np.array(y_train)
7      # Define the parameter grid for hyperparameter tuning
8      param_grid = {
9          'C': [0.1, 1, 10, 100],           # Regularization parameter
10         'epsilon': [0.01, 0.1, 0.2, 0.5],   # Epsilon in the epsilon-SVR
11             model
12             'kernel': ['rbf']           # Kernel types to consider
13         }
14
15      # Set up cross-validation
16      kf = KFold(n_splits=5, shuffle=True, random_state=42)
17
18      # Initialize variables to store the best result and all results for
19      # visualization
20      best_rmse = float('inf')
21      best_params = None
22      results = [] # Store all combinations for 3D visualization
23
24      # Manual Grid Search
25      print("Starting manual grid search...")
26      for kernel in param_grid['kernel']:

```

```

25     for C in param_grid['C']:
26         for epsilon in param_grid['epsilon']:
27             print(f"Training with kernel={kernel}, C={C}, epsilon={epsilon}")
28
29             rmse_scores = []
30             for train_index, val_index in kf.split(X_train_scaled):
31                 # Split data into training and validation sets
32                 X_train_cv, X_val_cv = X_train_scaled[train_index],
33                               X_train_scaled[val_index]
34                 y_train_cv, y_val_cv = y_train[train_index], y_train[val_index]
35
36                 # Train the model
37                 svr = SVR(kernel=kernel, C=C, epsilon=epsilon)
38                 svr.fit(X_train_cv, y_train_cv)
39
40                 # Predict and calculate RMSE
41                 y_val_pred = svr.predict(X_val_cv)
42                 rmse = np.sqrt(mean_squared_error(y_val_cv, y_val_pred))
43                 rmse_scores.append(rmse)
44
45                 # Calculate mean RMSE across folds
46                 mean_rmse = np.mean(rmse_scores)
47                 print(f"Mean RMSE: {mean_rmse:.4f}")
48
49                 # Store the results for 3D visualization
50                 results.append((kernel, C, epsilon, mean_rmse))
51
52                 # Update best parameters if current RMSE is better
53                 if mean_rmse < best_rmse:
54                     best_rmse = mean_rmse
55                     best_params = {'kernel': kernel, 'C': C, 'epsilon': epsilon}
56
57                 # Train the final model with the best parameters on the full training
58                 # set
59                 print("\nTraining final model with best parameters...")
60                 print(f"Best Parameters: {best_params}")
61                 svr = SVR(**best_params)
62                 svr.fit(X_train_scaled, y_train)
63
64                 # Predict on the test set
65                 y_pred = svr.predict(X_test_scaled)
66
67                 # Calculate RMSE on the test set
68                 final_rmse = np.sqrt(mean_squared_error(y_test, y_pred))

```

```

67     print(f"Final Test RMSE: {final_rmse:.4f}")
68
69     # 3D Plot of RMSE values
70     results = np.array(results)
71     kernels = results[:, 0]
72     Cs = results[:, 1].astype(float)
73     epsilon = results[:, 2].astype(float)
74     rmse = results[:, 3].astype(float)
75
76     # Convert kernel types to numeric values for 3D plotting
77     kernel_map = {'linear': 0, 'rbf': 1}
78     kernel_numeric = np.array([kernel_map[k] for k in kernels])
79
80     fig = go.Figure()
81
82     # Add scatter points
83     fig.add_trace(go.Scatter3d(
84         x=Cs,
85         y=epsilon,
86         z=kernel_numeric,
87         mode='markers',
88         marker=dict(
89             size=8,
90             color=rmse, # Color by RMSE
91             colorscale='Viridis', # Color scale
92             colorbar=dict(title="RMSE"),
93             opacity=0.8
94         )
95     ))
96
97     # Set axis labels and title
98     fig.update_layout(
99         scene=dict(
100             xaxis_title="C (Regularization Parameter)",
101             yaxis_title="Epsilon",
102             zaxis_title="Kernel (0=Linear, 1=RBF)"
103         ),
104             title="3D Visualization of Grid Search Results",
105             margin=dict(l=0, r=0, b=0, t=40)
106     )
107
108     # Show the plot
109     fig.show()

```

Listing 10: Test

```
1 # Retrieve the top 10 trials for XGBoost
2 top_trials_xgb =
3     study_xgb.trials_dataframe().sort_values(by="value").head(39)
4
5 # Use a bagging approach to retrain XGBoost models
6 models_xgb_test = []
7 for _, trial_row in top_trials_xgb.iterrows():
8     # Extract parameters
9     trial_number = int(trial_row["number"])
10    best_params_xgb = study_xgb.trials[trial_number].params
11
12    # Create and train the model
13    model_xgb = XGBRegressor(**best_params_xgb)
14    model_xgb.fit(X_full_train, y_full_train, verbose=False)
15    models_xgb_test.append(model_xgb)
16
17    # Make predictions on the test set
18    predictions_xgb_test = np.array([model.predict(X_true_test) for model
19        in models_xgb_test])
20
21    # Clip predictions to the range [0,5] and round to the nearest integer
22    predictions_xgb_clipped = np.round(np.clip(predictions_xgb_test, 0,
23        5)) # (10, n_samples)
24
25    # Take the mode (most common value) across models for each sample
26    y_pred_xgb_mode = mode(predictions_xgb_clipped,
27        axis=0).mode.squeeze() # Remove unnecessary dimensions
28
29    # Ensure 'y_pred_xgb_mode' is a 1D array and matches the shape of
30    # 'y_test'
31    y_pred_xgb_mode = y_pred_xgb_mode.flatten()
32
33    # Final predictions for the test set
34    y_pred_xgb_test = y_pred_xgb_mode
35
36    # Load the test dataset
37    test_data = pd.read_csv('/content/drive/My Drive/Colab
38        Notebooks/final_project/test_LDA.csv', parse_dates=['host_since',
39        'first_review', 'last_review'])
40
41    # Assuming 'test_data['id']' is loaded, prepare the submission file
42    submission = pd.DataFrame({
43        'id': test_data['id'], # Unique identifier for the test set
44        'price': y_pred_xgb_test # Predictions from the model
45    })
46
47
48    # Save the results as an Excel file
49    submission.to_excel('/content/drive/My Drive/Colab
```

```
36     Notebooks/final_project/submission.xlsx', index=False)
      print("The submission file has been saved as submission.xlsx")
```



Get updates, docs & report issues here
 Created & maintained by Francois Bertrand
 Graphic design by Jean-Francois Hains

DataFrame

15696	ROWS
9	DUPPLICATES
555.4 MB	RAM
65	FEATURES
14	CATEGORICAL
43	NUMERICAL
8	TEXT

ASSOCIATIONS

NO COMPARISON TARGET**1** name

VALUES: 15,696 (100%)

MISSING: ---

DISTINCT: 15,189 (97%)

16	<1%	Private bedroom -3 mins to shopping center
11	<1%	Wyndham Midtown 45 2BR/2BA King Bed Suite
10	<1%	Wyndham Midtown 45 1BR/1BA King Bed Suite
10	<1%	Wyndham Midtown 45 Resort King Bed Hotel Room
9	<1%	Blueground FiDi, gym, nr Freedom Tower
9	<1%	138 Bowery-Modern Queen Studio
8	<1%	King Room - 1 bed
15,623	>99%	(Other)

2 description

VALUES: 15,309 (98%)

MISSING: 387 (2%)

DISTINCT: 12,687 (81%)

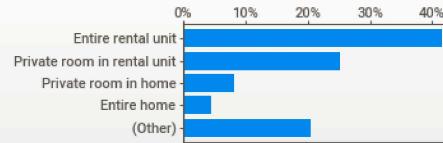
55	<1%	Kick back and relax in this calm, stylish space.
55	<1%	Keep it simple at this peaceful and centrally-located place.
52	<1%	Forget your worries in this spacious and serene space.
49	<1%	The whole group will enjoy easy access to everything from this centrally located place.
49	<1%	Enjoy a stylish experience at this centrally-located place.
48	<1%	Discover the essence of Brooklyn's creative scene, with just a few steps from the door of this newly refur...
39	<1%	Relax with the whole family at this peaceful place to stay.
14,962	98%	(Other)

3 property_type

VALUES: 15,696 (100%)

MISSING: ---

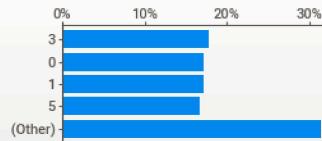
DISTINCT: 59 (<1%)

**4** price

VALUES: 15,696 (100%)

MISSING: ---

DISTINCT: 6 (<1%)

**5** neighbourhood_cleansed

VALUES: 15,696 (100%)

MISSING: ---

DISTINCT: 217 (1%)

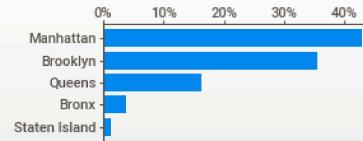
1,236	8%	Bedford-Stuyvesant
889	6%	Midtown
730	5%	Harlem
698	4%	Upper East Side
698	4%	Hell's Kitchen
646	4%	Williamsburg
597	4%	Bushwick
10,202	65%	(Other)

6 neighbourhood_group_cleansed

VALUES: 15,696 (100%)

MISSING: ---

DISTINCT: 5 (<1%)

**7** latitude

VALUES: 15,696 (100%)

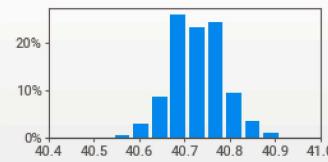
MISSING: ---

DISTINCT: 12,648 (81%)

ZEROES: ---

MAX	40.911
95%	40.824
Q3	40.762
Avg	40.727
Median	40.725
Q1	40.686
5%	40.635
MIN	40.500

RANGE	0.411
IQR	0.077
STD	0.058
VAR	0.003

**8** longitude

VALUES: 15,696 (100%)

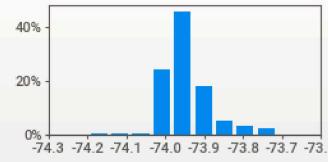
MISSING: ---

DISTINCT: 12,175 (78%)

ZEROES: ---

MAX	-73.714
95%	-73.813
Q3	-73.921
Avg	-73.943
Median	-73.952
Q1	-73.983
5%	-74.006
MIN	-74.252

RANGE	0.538
IQR	0.062
STD	0.060
VAR	0.004

**9** host_since

VALUES: 15,696 (100%)

MISSING: ---

560 4% 2016-12-16 00:00:00

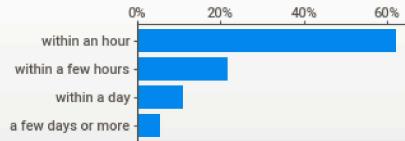
366 2% 2012-08-11 00:00:00

DISTINCT:	4,037 (26%)	228 1% 2019-10-29 00:00:00
		198 1% 2022-02-25 00:00:00
		155 <1% 2017-12-11 00:00:00
		146 <1% 2015-12-16 00:00:00
		99 <1% 2014-10-14 00:00:00
		13,944 89% (Other)

10 host_response_time

VALUES: 13,493 (86%)
MISSING: 2,203 (14%)

DISTINCT: 4 (<1%)



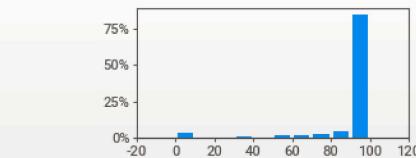
11 host_response_rate

VALUES: 13,493 (86%)
MISSING: 2,203 (14%)

DISTINCT: 65 (<1%)

ZEROES: 500 (3%)
MAX: 100
95%: 100
Q3: 100
MEDIAN: 100
AVG: 91
Q1: 97
5%: 33
MIN: 0

RANGE: 100
IQR: 3.00
STD: 22.3
VAR: 498
KURT.: 9.29
SKEW: -3.16
SUM: 1.2M



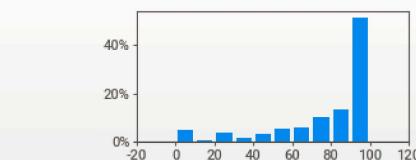
12 host_acceptance_rate

VALUES: 13,643 (87%)
MISSING: 2,053 (13%)

DISTINCT: 97 (<1%)

ZEROES: 642 (4%)
MAX: 100
95%: 100
Q3: 100
MEDIAN: 91
AVG: 79
Q1: 69
5%: 9
MIN: 0

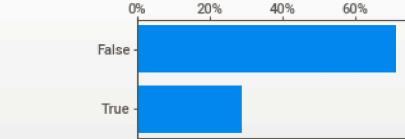
RANGE: 100
IQR: 31.0
STD: 27.9
VAR: 778
KURT.: 1.26
SKEW: -1.48
SUM: 1.1M



13 host_is_superhost

VALUES: 15,445 (98%)
MISSING: 251 (2%)

DISTINCT: 2 (<1%)



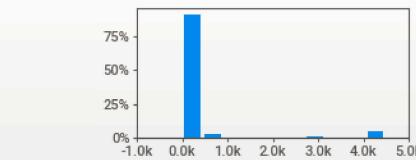
14 host_listings_count

VALUES: 15,696 (100%)
MISSING: ---

DISTINCT: 97 (<1%)

ZEROES: ---
MAX: 4,494
95%: 2,950
Q3: 21
AVG: 288
MEDIAN: 3
Q1: 1
5%: 1
MIN: 1

RANGE: 4,493
IQR: 20.0
STD: 984
VAR: 969k
KURT.: 13.0
SKEW: 3.81
SUM: 4.5M



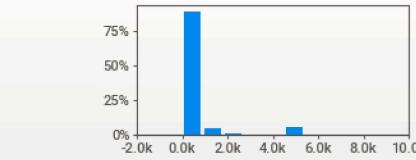
15 host_total_listings_count

VALUES: 15,696 (100%)
MISSING: ---

DISTINCT: 132 (<1%)

ZEROES: ---
MAX: 9,019
95%: 4,784
Q3: 31
AVG: 393
MEDIAN: 5
Q1: 2
5%: 1
MIN: 1

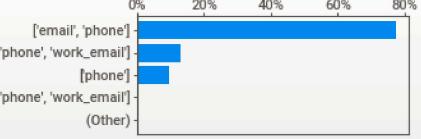
RANGE: 9,018
IQR: 29.0
STD: 1,205
VAR: 1.5M
KURT.: 12.7
SKEW: 3.60
SUM: 6.2M



16 host_verifications

VALUES: 15,696 (100%)
MISSING: ---

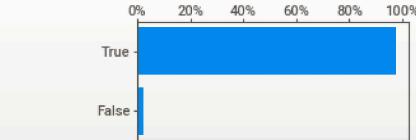
DISTINCT: 6 (<1%)



17 host_has_profile_pic

VALUES: 15,696 (100%)
MISSING: ---

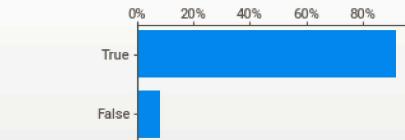
DISTINCT: 2 (<1%)



18 host_identity_verified

VALUES: 15,696 (100%)
MISSING: ---

DISTINCT: 2 (<1%)

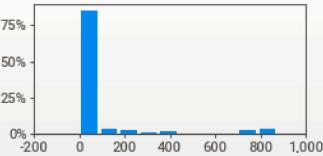


19

↳ calculated_host_listings_count

VALUES:	15,696 (100%)	MAX	876
MISSING:	---	95%	719
DISTINCT:	71 (<1%)	Q3	17
ZEROES:	---	AVG	75
		MEDIAN	3
		Q1	1
		5%	1
		MIN	1

RANGE	875
IQR	16.0
STD	198
VAR	39,221
KURT.	9.26
SKEW	3.20
SUM	1.2M

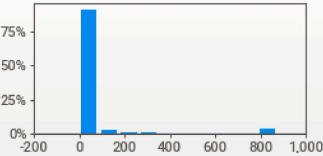


20

↳ calculated_host_listings_count_entire_homes

VALUES:	15,696 (100%)	MAX	876
MISSING:	---	95%	213
DISTINCT:	53 (<1%)	Q3	4
ZEROES:	5,433 (35%)	AVG	45
		MEDIAN	1
		Q1	0
		5%	0
		MIN	0

RANGE	876
IQR	4.00
STD	166
VAR	27,700
KURT.	19.2
SKEW	4.49
SUM	711k

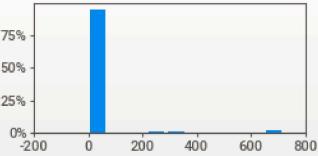


21

↳ calculated_host_listings_count_private_rooms

VALUES:	15,696 (100%)	MAX	719
MISSING:	---	95%	213
DISTINCT:	40 (<1%)	Q3	3
ZEROES:	7,788 (50%)	AVG	28
		MEDIAN	1
		Q1	0
		5%	0
		MIN	0

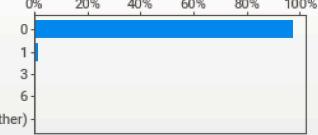
RANGE	719
IQR	3.00
STD	118
VAR	13,807
KURT.	25.8
SKEW	5.09
SUM	435k



22

☒ calculated_host_listings_count_shared_rooms

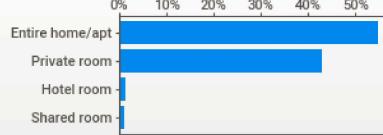
VALUES:	15,696 (100%)
MISSING:	---
DISTINCT:	8 (<1%)



23

☒ room_type

VALUES:	15,696 (100%)
MISSING:	---
DISTINCT:	4 (<1%)

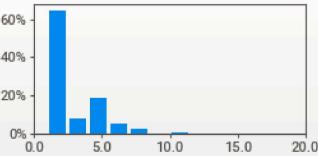


24

↳ accommodates

VALUES:	15,696 (100%)	MAX	16.0
MISSING:	---	95%	6.0
DISTINCT:	16 (<1%)	Q3	4.0
ZEROES:	---	AVG	2.8
		MEDIAN	2.0
		Q1	2.0
		5%	1.0
		MIN	1.0

RANGE	15.0
IQR	2.00
STD	1.89
VAR	3.58
KURT.	8.92
SKEW	2.37
SUM	43,785

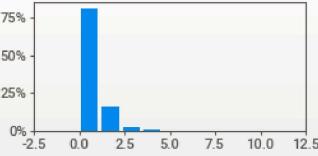


25

↳ bathrooms

VALUES:	15,693 (>99%)	MAX	11.5
MISSING:	3 (<1%)	95%	2.0
DISTINCT:	16 (<1%)	Q3	1.0
ZEROES:	353 (2%)	AVG	1.2
		MEDIAN	1.0
		Q1	1.0
		5%	1.0
		MIN	0.0

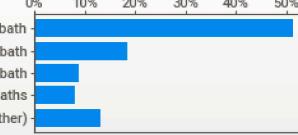
RANGE	11.5
IQR	0.00
STD	0.511
VAR	0.262
KURT.	26.9
SKEW	3.09
SUM	18,390



26

☒ bathrooms_text

VALUES:	15,686 (>99%)
MISSING:	10 (<1%)
DISTINCT:	30 (<1%)

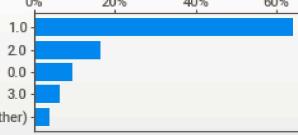


27

☒ bedrooms

VALUES:	15,662 (>99%)
MISSING:	34 (<1%)
DISTINCT:	10 (<1%)

RANGE	1.0
IQR	1.00
STD	1.00
VAR	1.11
KURT.	4.00
SKEW	2.00
SUM	1.2M

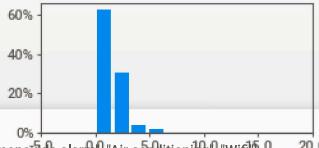
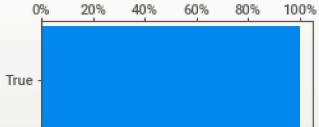
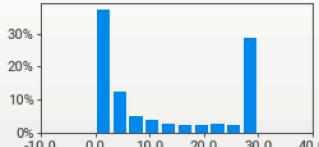
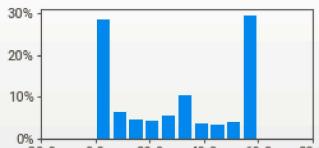
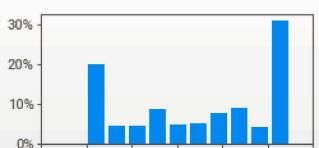
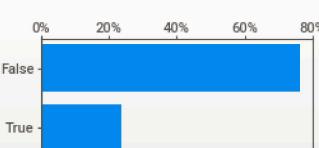
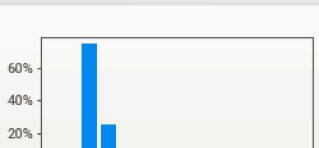


28

↳ beds

VALUES:	15,612 (>99%)	MAX	16.0
MISSING:	84 (<1%)	95%	4.0
DISTINCT:	10 (<1%)	Q3	2.0

RANGE	16.0
IQR	1.00
STD	1.11
VAR	1.21
KURT.	4.00
SKEW	2.00
SUM	1.2M

	DISTINCT: 15 (1%)	AVG 1.6	VAR 1.24	
	ZEROES: 415 (3%)	Q1 1.0	KURT. 11.5	
		5% 1.0	SKW 2.56	
		MIN 0.0	SUM 25,037	
29	amenities			
	VALUES: 15,696 (100%)	132 <1%	["Kitchen", "TV", "Smoke alarm", "Washer", "Carbon monoxide alarm", "Air conditioning", "Wifi"]	
	MISSING: ---	79 <1%	["Kitchen", "TV", "Smoke alarm", "Carbon monoxide alarm", "Air conditioning", "Wifi"]	
	DISTINCT: 13,314 (85%)	52 <1%	["Kitchen", "Hot water", "Dedicated workspace", "Wifi", "Heating", "Smoke alarm", "Carbon monoxide alar...	
		38 <1%	["Dishwasher", "Crib", "Elevator", "TV", "Smoke alarm", "Cooking basics", "Heating", "Refrigerator", "Long t...	
		36 <1%	["Dishwasher", "Dishes and silverware", "Hot water", "Dedicated workspace", "Cooking basics", "Oven", "...	
		35 <1%	["Dishwasher", "Crib", "Elevator", "TV", "Smoke alarm", "Cooking basics", "Heating", "Dryer \u2013 In build...	
		35 <1%	["Dishwasher", "Crib", "Elevator", "TV", "Smoke alarm", "Cooking basics", "Heating", "Dryer \u2013 In build...	
		15,289 97%	(Other)	
30	has_availability			
	VALUES: 15,578 (>99%)			
	MISSING: 118 (<1%)			
	DISTINCT: 1 (<1%)			
31	availability_30			
	VALUES: 15,696 (100%)	MAX 30.0	RANGE 30.0	
	MISSING: ---	95% 30.0	IQR 28.0	
	DISTINCT: 31 (<1%)	Q3 28.0	STD 12.5	
		AVG 12.1	VAR 156	
	ZEROES: 4,591 (29%)	MEDIAN 6.0	KURT. -1.58	
		Q1 0.0	SKW 0.437	
		5% 0.0	SUM 191k	
		MIN 0.0		
32	availability_60			
	VALUES: 15,696 (100%)	MAX 60.0	RANGE 60.0	
	MISSING: ---	95% 60.0	IQR 55.0	
	DISTINCT: 61 (<1%)	Q3 58.0	STD 23.8	
		MEDIAN 30.0	VAR 564	
	ZEROES: 2,890 (18%)	AVG 29.5	KURT. -1.60	
		Q1 3.0	SKW 0.055	
		5% 0.0	SUM 463k	
		MIN 0.0		
33	availability_90			
	VALUES: 15,696 (100%)	MAX 90.0	RANGE 90.0	
	MISSING: ---	95% 90.0	IQR 69.0	
	DISTINCT: 91 (<1%)	Q3 88.0	STD 33.7	
		MEDIAN 57.0	VAR 1,133	
	ZEROES: 2,209 (14%)	AVG 50.4	KURT. -1.43	
		Q1 19.0	SKW -0.246	
		5% 0.0	SUM 790k	
		MIN 0.0		
34	availability_365			
	VALUES: 15,696 (100%)	MAX 365	RANGE 365	
	MISSING: ---	95% 365	IQR 198	
	DISTINCT: 366 (2%)	Q3 335	STD 110	
		MEDIAN 256	VAR 12,165	
	ZEROES: 121 (<1%)	AVG 231	KURT. -1.13	
		Q1 137	SKW -0.410	
		5% 39	SUM 3.6M	
		MIN 0		
35	instant_bookable			
	VALUES: 15,696 (100%)			
	MISSING: ---			
	DISTINCT: 2 (<1%)			
36	minimum_nights			
	VALUES: 15,696 (100%)	MAX 500	RANGE 499	
	MISSING: ---	95% 31	IQR 0.00	
	DISTINCT: 60 (<1%)	Q3 30	STD 22.2	
		MEDIAN 30	VAR 493	
	ZEROES: ---	AVG 27	KURT. 135	
		Q1 30	SKW 9.11	
		5% 1	SUM 420k	
		MIN 1		
37	maximum_nights			
	VALUES: 15,696 (100%)	MAX 10,000	RANGE 9,999	
	MISSING: ---	95% 1,125	IQR 1,005	
	DISTINCT: 175 (1%)	Q3 1,125	STD 412	
		Avg 476	VAR 170k	
	ZEROES: ---	MEDIAN 365	KURT. 17.2	
		Q1 120	SKW 1.51	
		5% 29	SUM 7.5M	
		MIN 1		

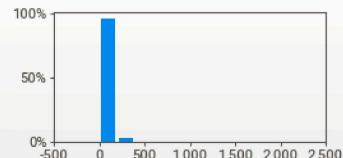
|_ number_of_reviews

38

VALUES: **15,696** (100%)
MISSING: ---
DISTINCT: **434** (<3%)
ZEROES: **4,474** (29%)

MAX **1,941**
95% **166**
Q3 **34**
AVG **34**
MEDIAN **6**
Q1 **0**
5% **0**
MIN **0**

RANGE **1,941**
IQR **34.0**
STD **74.0**
VAR **5,473**
KURT. **109**
SKEW **7.06**
SUM **530k**



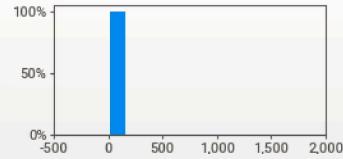
39

↳ number_of_reviews_ltm

VALUES: **15,696** (100%)
MISSING: ---
DISTINCT: **142** (<1%)
ZEROES: **7,336** (47%)

MAX **1,772**
95% **30**
Q3 **4**
AVG **6**
MEDIAN **1**
Q1 **0**
5% **0**
MIN **0**

RANGE **1,772**
IQR **4.00**
STD **23.6**
VAR **557**
KURT. **2,331**
SKEW **37.5**
SUM **89,364**



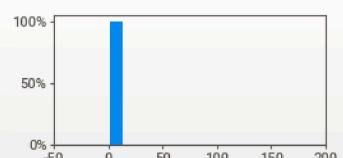
40

↳ number_of_reviews_l30d

VALUES: **15,696** (100%)
MISSING: ---
DISTINCT: **33** (<1%)
ZEROES: **12,802** (82%)

MAX **147**
95% **3**
Q3 **0**
AVG **0**
MEDIAN **0**
Q1 **0**
5% **0**
MIN **0**

RANGE **147**
IQR **0.00**
STD **2.21**
VAR **4.89**
KURT. **1,689**
SKEW **31.2**
SUM **7,442**



41

↳ first_review

VALUES: **11,222** (71%)
MISSING: **4,474** (29%)

DISTINCT: **3,261** (21%)

40 <1% 2023-01-01 00:00:00
32 <1% 2022-07-31 00:00:00
27 <1% 2023-04-30 00:00:00
24 <1% 2024-03-31 00:00:00
24 <1% 2024-06-01 00:00:00
24 <1% 2024-07-01 00:00:00
23 <1% 2023-01-02 00:00:00
11,028 98% (Other)

42

↳ last_review

VALUES: **11,222** (71%)
MISSING: **4,474** (29%)

DISTINCT: **1,390** (9%)

186 2% 2024-08-18 00:00:00
181 2% 2024-08-31 00:00:00
172 2% 2024-09-01 00:00:00
169 2% 2024-07-31 00:00:00
164 1% 2024-09-02 00:00:00
144 1% 2024-08-10 00:00:00
132 1% 2024-08-11 00:00:00
10,074 90% (Other)

43

↳ review_scores_rating

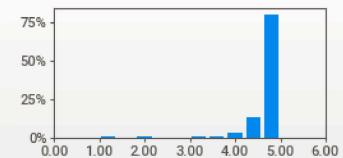
VALUES: **11,222** (71%)
MISSING: **4,474** (29%)

DISTINCT: **149** (<1%)

ZEROES: ---

MAX **5.00**
95% **5.00**
Q3 **5.00**
AVG **4.85**
MEDIAN **4.72**
Q1 **4.66**
5% **4.00**
MIN **1.00**

RANGE **4.00**
IQR **0.340**
STD **0.463**
VAR **0.214**
KURT. **27.1**
SKEW **-4.47**
SUM **52,961**



44

↳ review_scores_accuracy

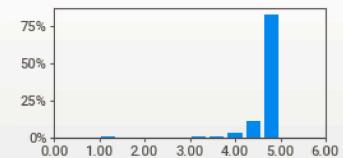
VALUES: **11,222** (71%)
MISSING: **4,474** (29%)

DISTINCT: **150** (<1%)

ZEROES: ---

MAX **5.00**
95% **5.00**
Q3 **5.00**
AVG **4.88**
MEDIAN **4.74**
Q1 **4.69**
5% **4.00**
MIN **1.00**

RANGE **4.00**
IQR **0.310**
STD **0.460**
VAR **0.212**
KURT. **30.4**
SKEW **-4.78**
SUM **53,224**



45

↳ review_scores_cleanliness

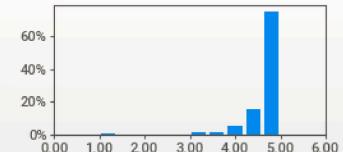
VALUES: **11,222** (71%)
MISSING: **4,474** (29%)

DISTINCT: **156** (<1%)

ZEROES: ---

MAX **5.00**
95% **5.00**
Q3 **4.98**
AVG **4.82**
MEDIAN **4.68**
Q1 **4.59**
5% **4.00**
MIN **1.00**

RANGE **4.00**
IQR **0.390**
STD **0.483**
VAR **0.234**
KURT. **21.6**
SKEW **-3.89**
SUM **52,515**



46

↳ review_scores_checkin

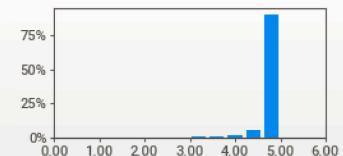
VALUES: **11,222** (71%)
MISSING: **4,474** (29%)

DISTINCT: **119** (<1%)

ZEROES: ---

MAX **5.00**
95% **5.00**
Q3 **5.00**
AVG **4.94**
MEDIAN **4.83**
Q1 **4.81**
5% **4.33**
MIN **1.00**

RANGE **4.00**
IQR **0.190**
STD **0.377**
VAR **0.142**
KURT. **47.7**
SKEW **-5.96**
SUM **54,161**



47

↳ review_scores_communication

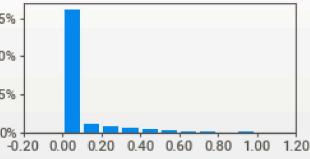
VALUES: **11,222** (71%)
MISSING: **4,474** (29%)

DISTINCT: **132** (<1%)

MAX **5.00**
95% **5.00**
Q3 **5.00**
AVG **4.94**
MEDIAN **4.81**

RANGE **4.00**
IQR **0.200**
STD **0.433**
VAR **0.188**

48	↳ review_scores_location	<p>VALUES: 11,221 (71%) MISSING: 4,475 (29%) DISTINCT: 146 (<1%) ZEROES: ---</p> <table border="1"> <thead> <tr><th></th><th>MAX</th><th>RANGE</th></tr> <tr><th>5%</th><td>4.80</td><td>37.8</td></tr> <tr><th>MIN</th><td>4.19</td><td>5.45</td></tr> <tr><th>Q1</th><td>4.19</td><td>53,958</td></tr> <tr><th>Q3</th><td>5.00</td><td></td></tr> <tr><th>AVG</th><td>4.82</td><td></td></tr> <tr><th>5%</th><td>4.00</td><td></td></tr> <tr><th>MIN</th><td>1.00</td><td></td></tr> </thead> </table>		MAX	RANGE	5%	4.80	37.8	MIN	4.19	5.45	Q1	4.19	53,958	Q3	5.00		AVG	4.82		5%	4.00		MIN	1.00																																
	MAX	RANGE																																																							
5%	4.80	37.8																																																							
MIN	4.19	5.45																																																							
Q1	4.19	53,958																																																							
Q3	5.00																																																								
AVG	4.82																																																								
5%	4.00																																																								
MIN	1.00																																																								
49	↳ review_scores_value	<p>VALUES: 11,222 (71%) MISSING: 4,474 (29%) DISTINCT: 153 (<1%) ZEROES: ---</p> <table border="1"> <thead> <tr><th></th><th>MAX</th><th>RANGE</th></tr> <tr><th>95%</th><td>5.00</td><td>4.00</td></tr> <tr><th>Q3</th><td>5.00</td><td>IQR 0.370</td></tr> <tr><th>MEDIAN</th><td>4.82</td><td>STD 0.400</td></tr> <tr><th>AVG</th><td>4.72</td><td>VAR 0.160</td></tr> <tr><th>5%</th><td>4.00</td><td>KURT. 29.1</td></tr> <tr><th>MIN</th><td>1.00</td><td>SKEW -4.30</td></tr> <tr><th></th><td></td><td>SUM 52,984</td></tr> </thead> </table>		MAX	RANGE	95%	5.00	4.00	Q3	5.00	IQR 0.370	MEDIAN	4.82	STD 0.400	AVG	4.72	VAR 0.160	5%	4.00	KURT. 29.1	MIN	1.00	SKEW -4.30			SUM 52,984																															
	MAX	RANGE																																																							
95%	5.00	4.00																																																							
Q3	5.00	IQR 0.370																																																							
MEDIAN	4.82	STD 0.400																																																							
AVG	4.72	VAR 0.160																																																							
5%	4.00	KURT. 29.1																																																							
MIN	1.00	SKEW -4.30																																																							
		SUM 52,984																																																							
50	↳ reviews_per_month	<p>VALUES: 11,222 (71%) MISSING: 4,474 (29%) DISTINCT: 718 (5%) ZEROES: ---</p> <table border="1"> <thead> <tr><th></th><th>MAX</th><th>RANGE</th></tr> <tr><th>95%</th><td>110</td><td>110</td></tr> <tr><th>Q3</th><td>4</td><td>IQR 1.44</td></tr> <tr><th>MEDIAN</th><td>2</td><td>STD 2.27</td></tr> <tr><th>AVG</th><td>1</td><td>VAR 5.15</td></tr> <tr><th>5%</th><td>0</td><td>KURT. 581</td></tr> <tr><th>MIN</th><td>0</td><td>SKEW 16.6</td></tr> <tr><th></th><td></td><td>SUM 13,980</td></tr> </thead> </table>		MAX	RANGE	95%	110	110	Q3	4	IQR 1.44	MEDIAN	2	STD 2.27	AVG	1	VAR 5.15	5%	0	KURT. 581	MIN	0	SKEW 16.6			SUM 13,980																															
	MAX	RANGE																																																							
95%	110	110																																																							
Q3	4	IQR 1.44																																																							
MEDIAN	2	STD 2.27																																																							
AVG	1	VAR 5.15																																																							
5%	0	KURT. 581																																																							
MIN	0	SKEW 16.6																																																							
		SUM 13,980																																																							
51	↳ reviews	<p>VALUES: 11,222 (71%) MISSING: 4,474 (29%) DISTINCT: 11,215 (71%)</p> <table border="1"> <thead> <tr><th></th><th>3</th><th>2</th><th>1</th><th>0</th><th>>99%</th></tr> <tr><th><1%</th><td>Great!</td><td></td><td></td><td></td><td></td></tr> <tr><th><1%</th><td>Great stay and great host!</td><td></td><td></td><td></td><td></td></tr> <tr><th><1%</th><td>Great place!</td><td></td><td></td><td></td><td></td></tr> <tr><th><1%</th><td>Good stay</td><td></td><td></td><td></td><td></td></tr> <tr><th><1%</th><td>Good!</td><td></td><td></td><td></td><td></td></tr> <tr><th><1%</th><td>Great Stay</td><td></td><td></td><td></td><td></td></tr> <tr><th><1%</th><td>Perfect location, good for a short stay</td><td></td><td></td><td></td><td></td></tr> <tr><th>>99%</th><td>Wonderful, safe, spotless, stylish, and c...</td><td></td><td></td><td></td><td>(Other)</td></tr> </thead> </table>		3	2	1	0	>99%	<1%	Great!					<1%	Great stay and great host!					<1%	Great place!					<1%	Good stay					<1%	Good!					<1%	Great Stay					<1%	Perfect location, good for a short stay					>99%	Wonderful, safe, spotless, stylish, and c...				(Other)	
	3	2	1	0	>99%																																																				
<1%	Great!																																																								
<1%	Great stay and great host!																																																								
<1%	Great place!																																																								
<1%	Good stay																																																								
<1%	Good!																																																								
<1%	Great Stay																																																								
<1%	Perfect location, good for a short stay																																																								
>99%	Wonderful, safe, spotless, stylish, and c...				(Other)																																																				
52	↳ review_topic_0	<p>VALUES: 15,696 (100%) MISSING: --- DISTINCT: 11,189 (71%) ZEROES: ---</p> <table border="1"> <thead> <tr><th></th><th>MAX</th><th>RANGE</th></tr> <tr><th>95%</th><td>1.000</td><td>1.000</td></tr> <tr><th>Q3</th><td>0.888</td><td>IQR 0.330</td></tr> <tr><th>AVG</th><td>0.359</td><td>STD 0.283</td></tr> <tr><th>MEDIAN</th><td>0.227</td><td>VAR 0.080</td></tr> <tr><th>5%</th><td>0.000</td><td>KURT. 0.566</td></tr> <tr><th>MIN</th><td>0.000</td><td>SKEW 1.35</td></tr> <tr><th></th><td></td><td>SUM 3,571</td></tr> </thead> </table>		MAX	RANGE	95%	1.000	1.000	Q3	0.888	IQR 0.330	AVG	0.359	STD 0.283	MEDIAN	0.227	VAR 0.080	5%	0.000	KURT. 0.566	MIN	0.000	SKEW 1.35			SUM 3,571																															
	MAX	RANGE																																																							
95%	1.000	1.000																																																							
Q3	0.888	IQR 0.330																																																							
AVG	0.359	STD 0.283																																																							
MEDIAN	0.227	VAR 0.080																																																							
5%	0.000	KURT. 0.566																																																							
MIN	0.000	SKEW 1.35																																																							
		SUM 3,571																																																							
53	↳ review_topic_1	<p>VALUES: 15,696 (100%) MISSING: --- DISTINCT: 11,189 (71%) ZEROES: ---</p> <table border="1"> <thead> <tr><th></th><th>MAX</th><th>RANGE</th></tr> <tr><th>95%</th><td>0.964</td><td>0.964</td></tr> <tr><th>Q3</th><td>0.571</td><td>IQR 0.571</td></tr> <tr><th>AVG</th><td>0.571</td><td>STD 0.255</td></tr> <tr><th>MEDIAN</th><td>0.003</td><td>VAR 0.065</td></tr> <tr><th>5%</th><td>0.000</td><td>KURT. -1.18</td></tr> <tr><th>MIN</th><td>0.000</td><td>SKEW 0.859</td></tr> <tr><th></th><td></td><td>SUM 2,807</td></tr> </thead> </table>		MAX	RANGE	95%	0.964	0.964	Q3	0.571	IQR 0.571	AVG	0.571	STD 0.255	MEDIAN	0.003	VAR 0.065	5%	0.000	KURT. -1.18	MIN	0.000	SKEW 0.859			SUM 2,807																															
	MAX	RANGE																																																							
95%	0.964	0.964																																																							
Q3	0.571	IQR 0.571																																																							
AVG	0.571	STD 0.255																																																							
MEDIAN	0.003	VAR 0.065																																																							
5%	0.000	KURT. -1.18																																																							
MIN	0.000	SKEW 0.859																																																							
		SUM 2,807																																																							
54	↳ review_topic_2	<p>VALUES: 15,696 (100%) MISSING: --- DISTINCT: 11,189 (71%) ZEROES: ---</p> <table border="1"> <thead> <tr><th></th><th>MAX</th><th>RANGE</th></tr> <tr><th>95%</th><td>0.985</td><td>0.985</td></tr> <tr><th>Q3</th><td>0.205</td><td>IQR 0.071</td></tr> <tr><th>AVG</th><td>0.071</td><td>STD 0.104</td></tr> <tr><th>MEDIAN</th><td>0.050</td><td>VAR 0.011</td></tr> <tr><th>5%</th><td>0.000</td><td>KURT. 30.2</td></tr> <tr><th>MIN</th><td>0.000</td><td>SKEW 4.86</td></tr> <tr><th></th><td></td><td>SUM 789</td></tr> </thead> </table>		MAX	RANGE	95%	0.985	0.985	Q3	0.205	IQR 0.071	AVG	0.071	STD 0.104	MEDIAN	0.050	VAR 0.011	5%	0.000	KURT. 30.2	MIN	0.000	SKEW 4.86			SUM 789																															
	MAX	RANGE																																																							
95%	0.985	0.985																																																							
Q3	0.205	IQR 0.071																																																							
AVG	0.071	STD 0.104																																																							
MEDIAN	0.050	VAR 0.011																																																							
5%	0.000	KURT. 30.2																																																							
MIN	0.000	SKEW 4.86																																																							
		SUM 789																																																							
55	↳ review_topic_3	<p>VALUES: 15,696 (100%) MISSING: --- DISTINCT: 11,189 (71%) ZEROES: ---</p> <table border="1"> <thead> <tr><th></th><th>MAX</th><th>RANGE</th></tr> <tr><th>95%</th><td>0.999</td><td>0.999</td></tr> <tr><th>Q3</th><td>0.583</td><td>IQR 0.079</td></tr> <tr><th>AVG</th><td>0.081</td><td>STD 0.195</td></tr> <tr><th>MEDIAN</th><td>0.118</td><td>VAR 0.038</td></tr> <tr><th>5%</th><td>0.002</td><td>KURT. 6.36</td></tr> <tr><th>MIN</th><td>0.000</td><td>SKEW 2.54</td></tr> <tr><th></th><td></td><td>SUM 1,847</td></tr> </thead> </table>		MAX	RANGE	95%	0.999	0.999	Q3	0.583	IQR 0.079	AVG	0.081	STD 0.195	MEDIAN	0.118	VAR 0.038	5%	0.002	KURT. 6.36	MIN	0.000	SKEW 2.54			SUM 1,847																															
	MAX	RANGE																																																							
95%	0.999	0.999																																																							
Q3	0.583	IQR 0.079																																																							
AVG	0.081	STD 0.195																																																							
MEDIAN	0.118	VAR 0.038																																																							
5%	0.002	KURT. 6.36																																																							
MIN	0.000	SKEW 2.54																																																							
		SUM 1,847																																																							
56	↳ review_topic_4	<p>VALUES: 15,696 (100%) MISSING: --- DISTINCT: 11,189 (71%) ZEROES: ---</p> <table border="1"> <thead> <tr><th></th><th>MAX</th><th>RANGE</th></tr> <tr><th>95%</th><td>0.999</td><td>0.999</td></tr> <tr><th>Q3</th><td>0.929</td><td>IQR 0.358</td></tr> <tr><th>AVG</th><td>0.394</td><td>STD 0.299</td></tr> <tr><th>MEDIAN</th><td>0.242</td><td>VAR 0.090</td></tr> <tr><th>5%</th><td>0.036</td><td>KURT. 0.256</td></tr> <tr><th>MIN</th><td>0.000</td><td>SKEW 1.26</td></tr> <tr><th></th><td></td><td>SUM 3,804</td></tr> </thead> </table>		MAX	RANGE	95%	0.999	0.999	Q3	0.929	IQR 0.358	AVG	0.394	STD 0.299	MEDIAN	0.242	VAR 0.090	5%	0.036	KURT. 0.256	MIN	0.000	SKEW 1.26			SUM 3,804																															
	MAX	RANGE																																																							
95%	0.999	0.999																																																							
Q3	0.929	IQR 0.358																																																							
AVG	0.394	STD 0.299																																																							
MEDIAN	0.242	VAR 0.090																																																							
5%	0.036	KURT. 0.256																																																							
MIN	0.000	SKEW 1.26																																																							
		SUM 3,804																																																							
57	↳ review_topic_5	<p>VALUES: 15,696 (100%) MISSING: --- DISTINCT: 11,189 (71%) ZEROES: ---</p> <table border="1"> <thead> <tr><th></th><th>MAX</th><th>RANGE</th></tr> <tr><th>95%</th><td>0.999</td><td>0.999</td></tr> <tr><th>Q3</th><td>0.929</td><td>IQR 0.358</td></tr> <tr><th>AVG</th><td>0.394</td><td>STD 0.299</td></tr> <tr><th>MEDIAN</th><td>0.071</td><td>VAR 0.090</td></tr> <tr><th>5%</th><td>0.000</td><td>KURT. 0.256</td></tr> <tr><th>MIN</th><td>0.000</td><td>SKEW 1.26</td></tr> <tr><th></th><td></td><td>SUM 3,804</td></tr> </thead> </table>		MAX	RANGE	95%	0.999	0.999	Q3	0.929	IQR 0.358	AVG	0.394	STD 0.299	MEDIAN	0.071	VAR 0.090	5%	0.000	KURT. 0.256	MIN	0.000	SKEW 1.26			SUM 3,804																															
	MAX	RANGE																																																							
95%	0.999	0.999																																																							
Q3	0.929	IQR 0.358																																																							
AVG	0.394	STD 0.299																																																							
MEDIAN	0.071	VAR 0.090																																																							
5%	0.000	KURT. 0.256																																																							
MIN	0.000	SKEW 1.26																																																							
		SUM 3,804																																																							

	VALUES: 15,696 (100%) MISSING: --- DISTINCT: 11,189 (71%) ZEROES: ---	MAX 0.996 95% 0.427 Q3 0.071 AVG 0.089 MEDIAN 0.071 Q1 0.001 5% 0.000 MIN 0.000	RANGE 0.996 IQR 0.071 STD 0.150 VAR 0.022 KURT. 9.64 SKEW 2.90 SUM 1,403	
58	 review_topic_6	VALUES: 15,696 (100%) MISSING: --- DISTINCT: 11,189 (71%) ZEROES: ---	MAX 0.999 95% 0.447 Q3 0.071 AVG 0.094 MEDIAN 0.071 Q1 0.001 5% 0.000 MIN 0.000	RANGE 0.999 IQR 0.070 STD 0.157 VAR 0.025 KURT. 9.35 SKEW 2.89 SUM 1,476
59	 description_topic_0	VALUES: 15,696 (100%) MISSING: --- DISTINCT: 12,376 (79%) ZEROES: ---	MAX 0.985 95% 0.439 Q3 0.020 AVG 0.068 MEDIAN 0.004 Q1 0.003 5% 0.002 MIN 0.001	RANGE 0.984 IQR 0.017 STD 0.173 VAR 0.030 KURT. 13.5 SKEW 3.62 SUM 1,068
60	 description_topic_1	VALUES: 15,696 (100%) MISSING: --- DISTINCT: 12,376 (79%) ZEROES: ---	MAX 0.987 95% 0.684 Q3 0.151 AVG 0.125 MEDIAN 0.006 Q1 0.003 5% 0.003 MIN 0.001	RANGE 0.986 IQR 0.147 STD 0.226 VAR 0.051 KURT. 4.52 SKEW 2.25 SUM 1,959
61	 description_topic_2	VALUES: 15,696 (100%) MISSING: --- DISTINCT: 12,376 (79%) ZEROES: ---	MAX 0.984 95% 0.876 Q3 0.304 AVG 0.189 MEDIAN 0.008 Q1 0.003 5% 0.002 MIN 0.001	RANGE 0.982 IQR 0.301 STD 0.277 VAR 0.077 KURT. 0.907 SKEW 1.44 SUM 2,965
62	 description_topic_3	VALUES: 15,696 (100%) MISSING: --- DISTINCT: 12,376 (79%) ZEROES: ---	MAX 0.986 95% 0.765 Q3 0.239 AVG 0.158 MEDIAN 0.009 Q1 0.003 5% 0.003 MIN 0.002	RANGE 0.985 IQR 0.235 STD 0.248 VAR 0.062 KURT. 2.41 SKEW 1.79 SUM 2,479
63	 description_topic_4	VALUES: 15,696 (100%) MISSING: --- DISTINCT: 12,376 (79%) ZEROES: ---	MAX 0.976 95% 0.521 Q3 0.050 AVG 0.078 MEDIAN 0.004 Q1 0.003 5% 0.002 MIN 0.001	RANGE 0.975 IQR 0.047 STD 0.204 VAR 0.042 KURT. 12.3 SKEW 3.61 SUM 1,232
64	 description_topic_5	VALUES: 15,696 (100%) MISSING: --- DISTINCT: 12,376 (79%) ZEROES: ---	MAX 0.986 95% 0.701 Q3 0.262 AVG 0.160 MEDIAN 0.010 Q1 0.003 5% 0.002 MIN 0.001	RANGE 0.984 IQR 0.258 STD 0.240 VAR 0.057 KURT. 1.49 SKEW 1.56 SUM 2,509
65	 description_topic_6	VALUES: 15,696 (100%) MISSING: --- DISTINCT: 12,376 (79%) ZEROES: ---	MAX 0.987 95% 0.857 Q3 0.410 AVG 0.222 MEDIAN 0.020 Q1 0.004 5% 0.003 MIN 0.001	RANGE 0.985 IQR 0.406 STD 0.293 VAR 0.086 KURT. -0.021 SKEW 1.13 SUM 3,484