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_r 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_ months_from_last_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

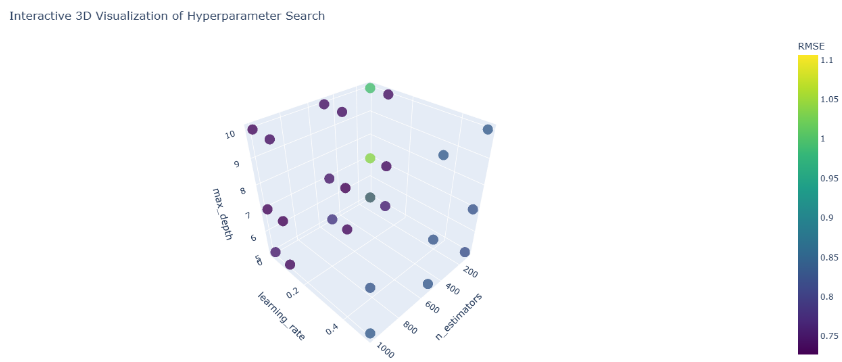


Figure 1: CV for XGBoost

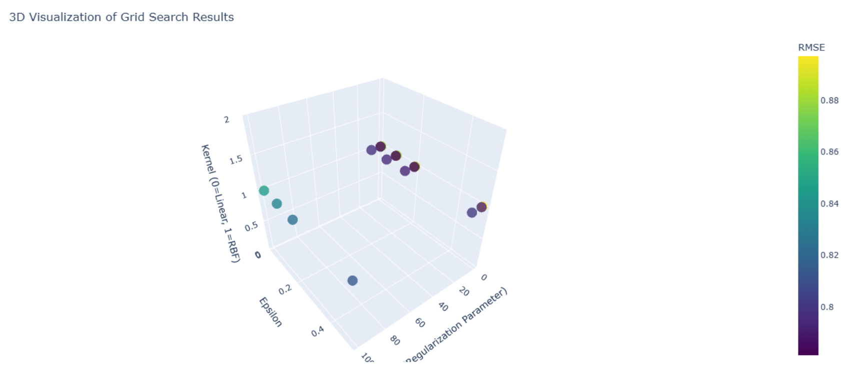


Figure 2: CV for SVM

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).

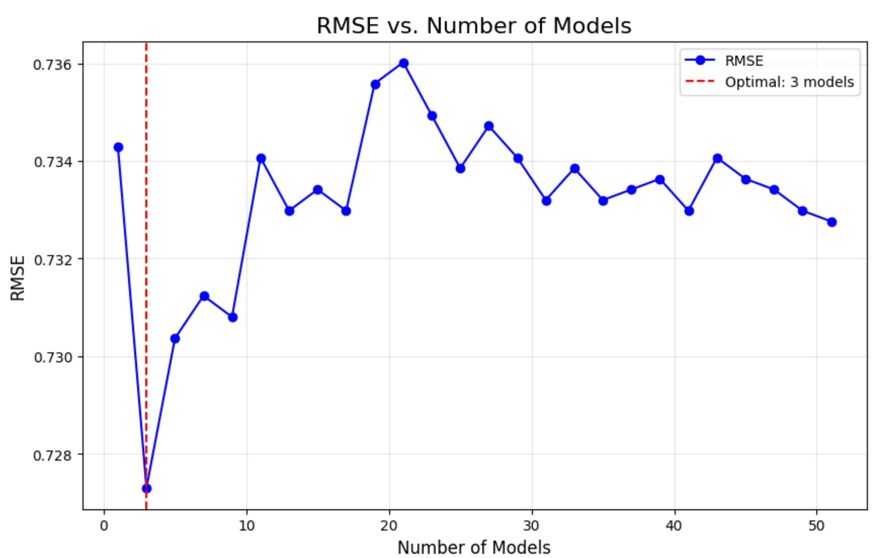


Figure 3: Cross-validation results for selecting the optimal number of models used in majority voting.

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 Figure4,5,6).

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 Figures 7 and 8), the first figure shows the training data, where prices are represented by varying color intensities, and the test data distribution is indicated by red points. It is evident that the test data distribution is uniform. The second figure divides the prices into three distinct intervals. 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 signif-

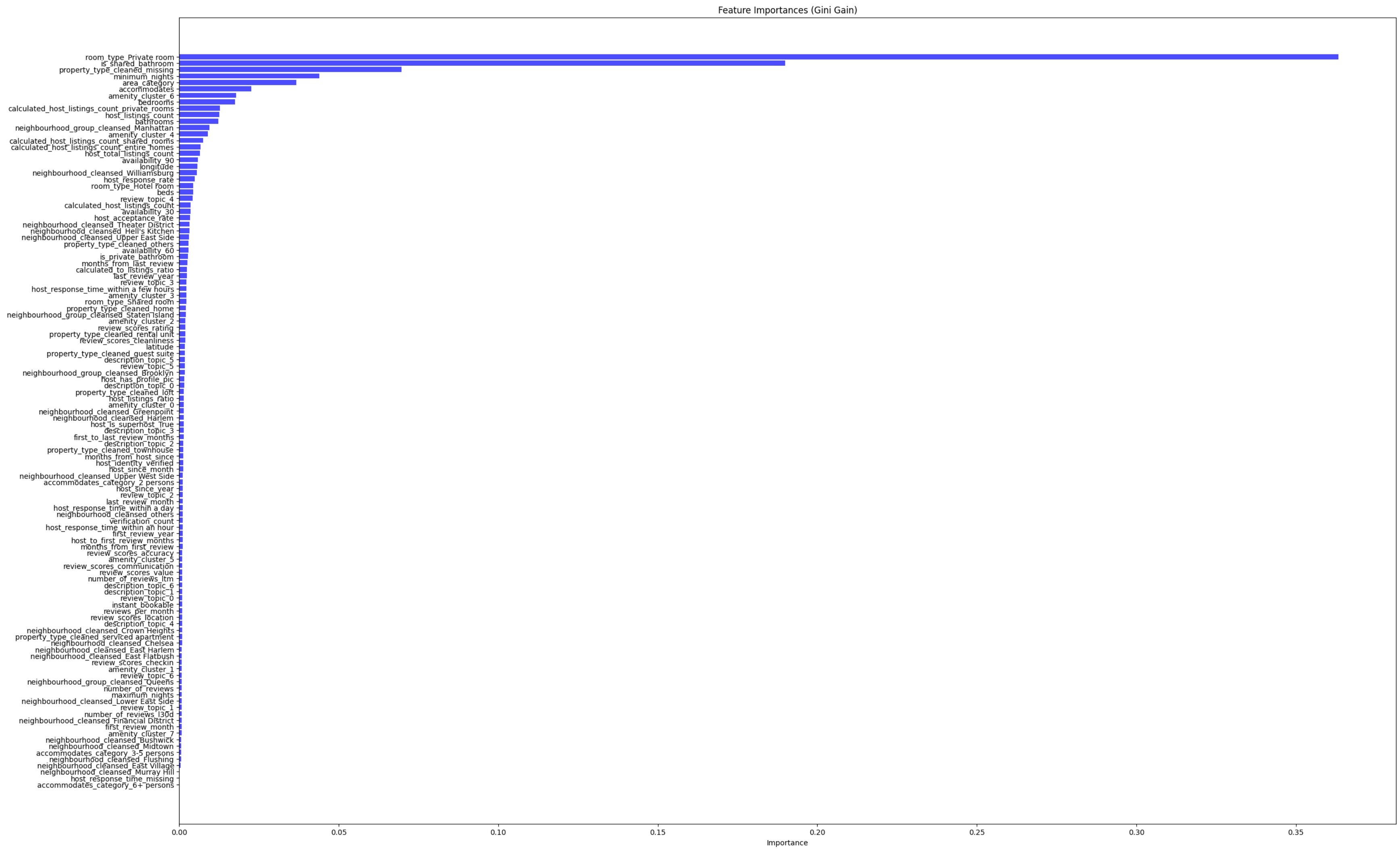


Figure 4: Feature importance based on the Gini index. This highlights the relative importance of features in predicting price. Features with higher Gini importance contribute more to the model's predictive power.

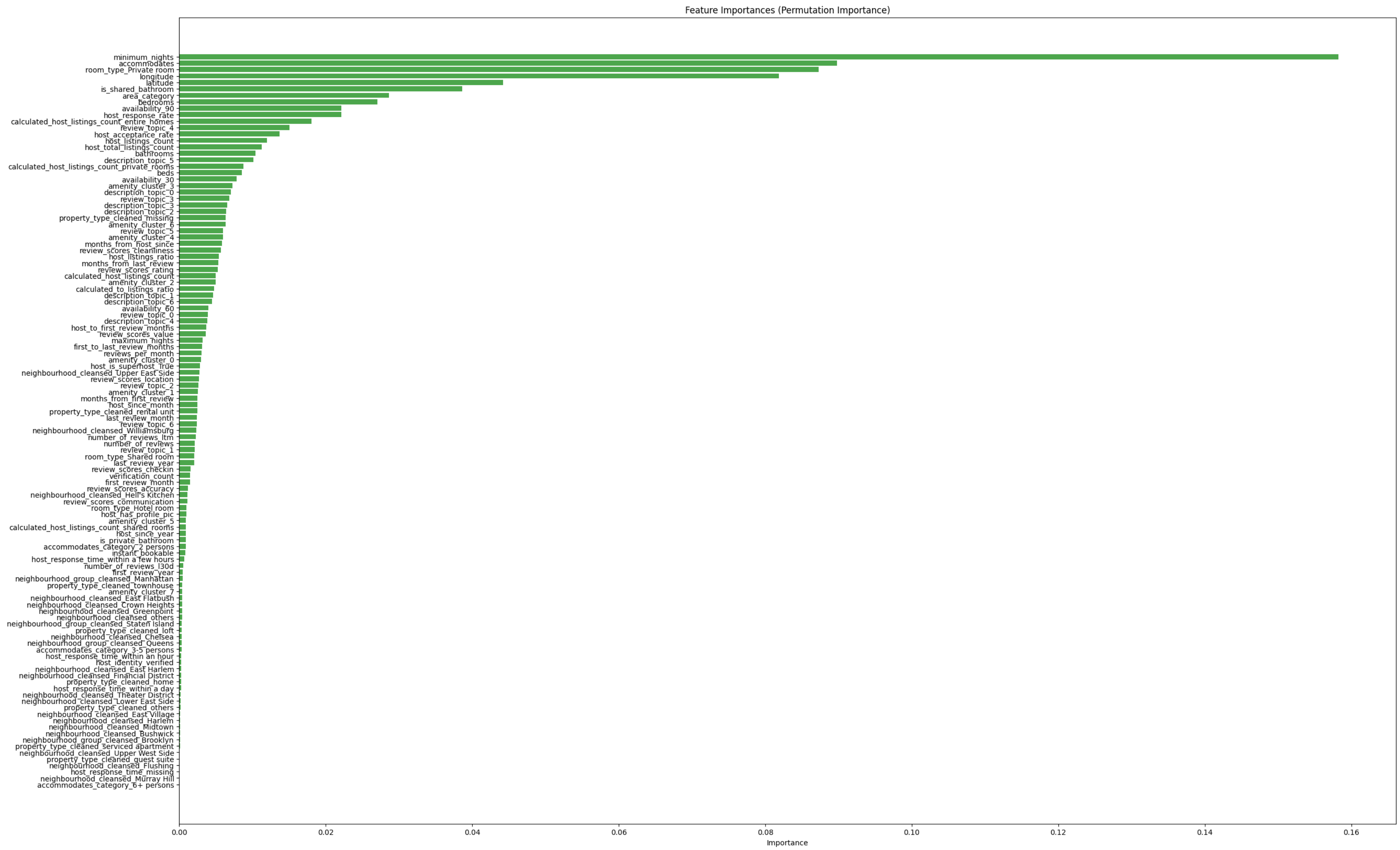


Figure 5: Permutation-based feature importance. This plot shows the impact of permuting each feature on the model's RMSE, providing insight into the robustness and reliability of each feature.

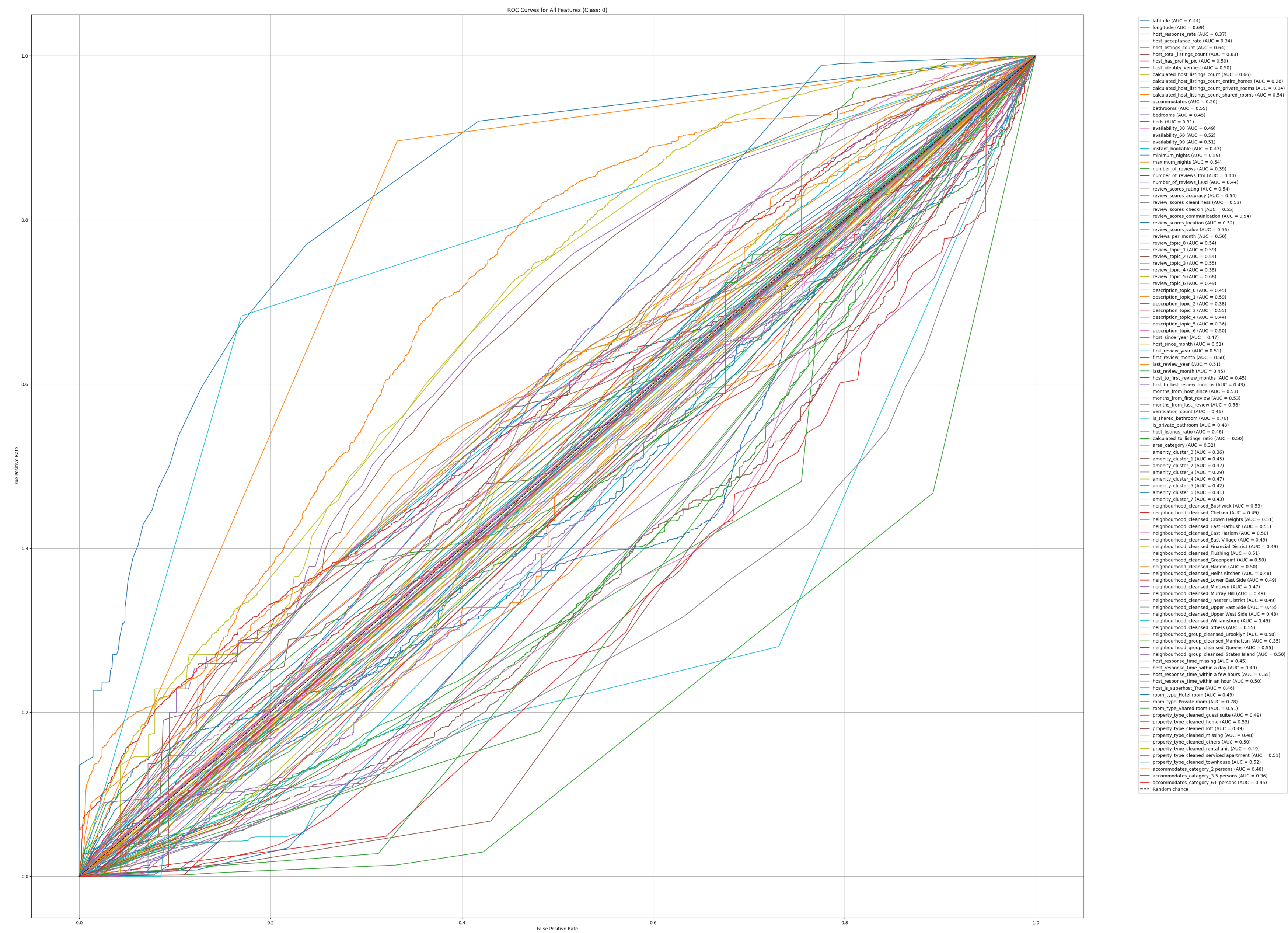


Figure 6: Feature importance based on AUC (Area Under the Curve). This plot emphasizes the discriminative power of individual features in predicting different price categories.

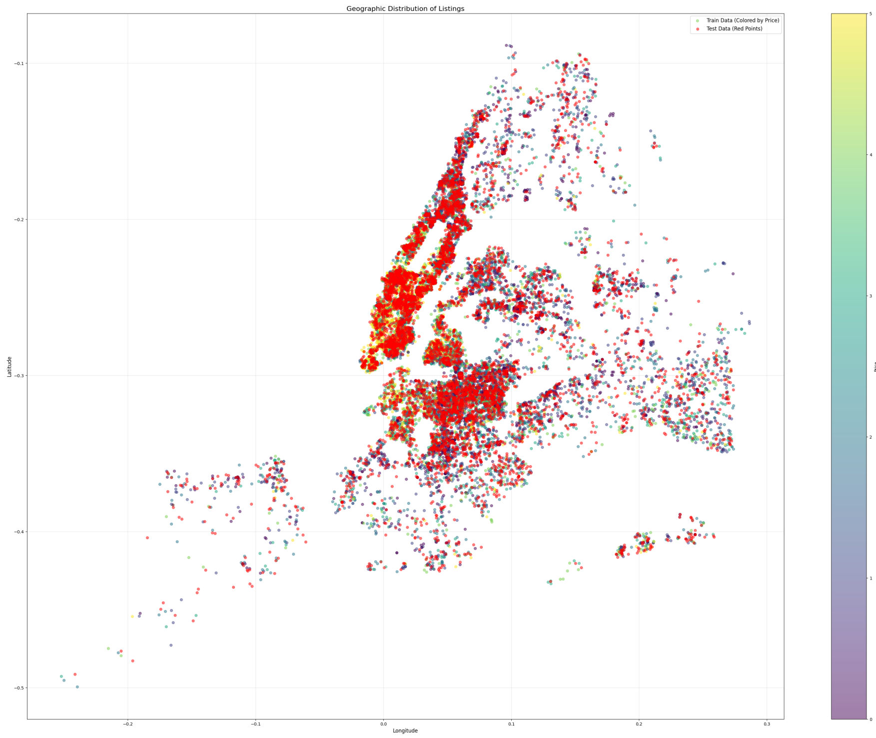


Figure 7: Geographic distribution of listings with prices. Red points indicate test data locations, showing that the test data is uniformly distributed.

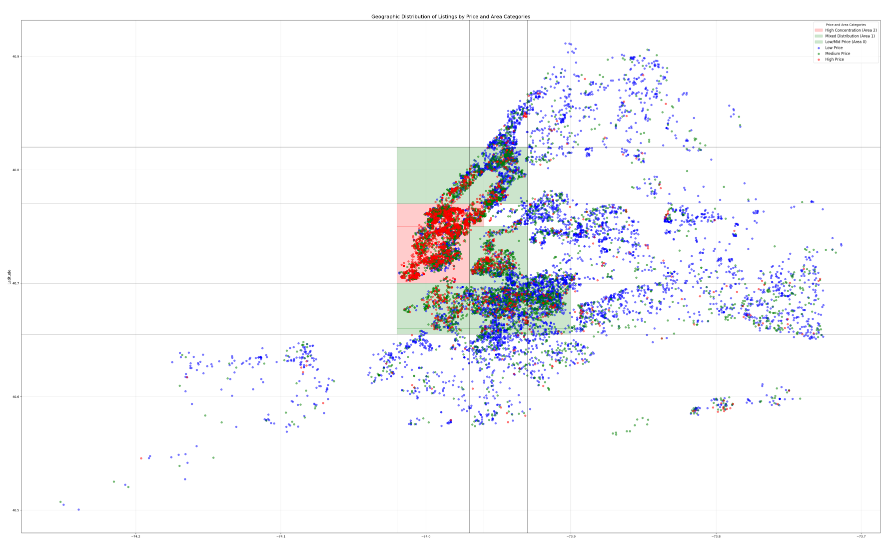


Figure 8: This map categorizes areas into three price levels (low, medium, high). Based on price concentration, areas with the highest concentration are marked with a red overlay (category 2), moderately mixed areas with green (category 1), and the remaining areas are labeled as category 0.

icantly different from one that offers a full set of toiletries, including shampoo, conditioner, face wash, and cosmetics.

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 bathroom 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

```
# Download libraries
!pip install optuna
!pip install googletrans
!pip install langdetect
!pip install sweetviz

# Import libraries
# Import general libraries
import pandas as pd
import numpy as np
from datetime import datetime
from tqdm import tqdm
import re
import matplotlib.pyplot as plt
import seaborn as sns

# Import NLP-related libraries
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from langdetect import detect
from googletrans import Translator

# Sentence embeddings and clustering
from sentence_transformers import SentenceTransformer
from sklearn.cluster import KMeans

# Machine Learning libraries
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import AdaBoostRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import mean_squared_error
from sklearn.inspection import permutation_importance
from sklearn.preprocessing import StandardScaler

```

```
    # Optimization library
    import optuna

    # Statistical utilities
    from scipy.stats import mode

    # Visualization libraries
    import plotly.graph_objects as go
    import sweetviz as sv
    from IPython.display import HTML
```

Listing 2: Data Download

```
# This is the original data, but the LDA process is quite lengthy, so
    we saved the original data with the LDA results included.
# The LDA code is provided in the preprocessing section but has been
    commented out.import pandas as pd
# train_data = pd.read_csv('/content/drive/My Drive/Colab
    Notebooks/final_project/train.csv', parse_dates=['host_since',
    'first_review', 'last_review'])
# test_data = pd.read_csv('/content/drive/My Drive/Colab
    Notebooks/final_project/test.csv', parse_dates=['host_since',
    'first_review', 'last_review'])

train_data = pd.read_csv('/content/drive/My Drive/Colab
    Notebooks/final_project/train_LDA.csv', parse_dates=['host_since',
    'first_review', 'last_review'])
test_data = pd.read_csv('/content/drive/My Drive/Colab
    Notebooks/final_project/test_LDA.csv', parse_dates=['host_since',
    'first_review', 'last_review'])

# This is the processed amenities mapping used for analyzing
    amenities, which is based on train_data
# and includes the parts I manually handled, so it can be directly
    imported.
# The categories for numbering are roughly as follows:
# 0: Entertainment and networking facilities, such as wifi, TV,
    Bluetooth speakers
# 1: Safety-related items, such as locker, alarm
# 2: Kitchen-related items, such as oven, refrigerator
# 3: Daily necessities, such as shampoo, conditioner
# 4: Sports and health facilities, such as gym, pool
# 5: Additional or paid services
# 6: Family- or baby-friendly facilities
```

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

Listing 3: EDA

```
# # Create a report
# report = sv.analyze(train_data)

# # Save the report as an HTML file
# report.show_html("/content/drive/My Drive/Colab
   Notebooks/final_project/train_data_distribution.html")

# Load the HTML file content
with open("/content/drive/My Drive/Colab
   Notebooks/final_project/train_data_distribution.html", "r") as f:
html_content = f.read()

# Display the report directly in Colab
HTML(html_content)

# Select only numerical columns from train_data
numerical_columns = train_data.select_dtypes(include=[np.number])

# Calculate the correlation matrix
correlation_matrix = numerical_columns.corr()

# Plot the correlation matrix using seaborn
plt.figure(figsize=(40, 30))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f",
   cmap="coolwarm", cbar=True, square=True)
plt.title("Correlation Matrix of Numerical Features", fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.show()

```

```python
import matplotlib.pyplot as plt

# We will mark the `test_data` dataset in red and visualize `train_data` with price as the color scale
# to check if the distribution is even.

# Set the figure size
plt.figure(figsize=(40, 30))

# Plot the scatter plot for `train_data`, with price represented by color
scatter_train = plt.scatter(
train_data['longitude'],   # Longitude data
train_data['latitude'],    # Latitude data
c=train_data['price'],     # Price data for color
cmap='viridis',            # Use `viridis` colormap
alpha=0.5,                 # Set transparency
label='Train Data (Colored by Price)'  # Legend label
)

# Plot the scatter plot for `test_data`, marked in red
plt.scatter(
test_data['longitude'],    # Longitude data
test_data['latitude'],     # Latitude data
color='red',               # Use red color for points
alpha=0.5,                 # Set transparency
label='Test Data (Red Points)'  # Legend label
)

# Add a color bar to show the price scale
plt.colorbar(scatter_train, label='Price')

# Add title and axis labels
plt.title('Geographic Distribution of Listings', fontsize=16)  # Set the title
plt.xlabel('Longitude', fontsize=12)  # Set X-axis label
plt.ylabel('Latitude', fontsize=12)   # Set Y-axis label

# Display grid lines for better readability
plt.grid(alpha=0.3)

# Add legend
plt.legend(fontsize=12)

```

```
# Show the plot
plt.show()

# Categorize `price` into three levels
# Define price range bins
train_data['price_category'] = pd.cut(
train_data['price'],
bins=[-1, 2, 4, 6],  # Define ranges: Low (0-2), Medium (3-4), High (5-6)
labels=['Low', 'Medium', 'High']  # Labels for categories
)

# Set color mapping
color_mapping = {'Low': 'blue', 'Medium': 'green', 'High': 'red'}
train_data['price_color'] = train_data['price_category'].map(color_mapping)

# Plot the data
plt.figure(figsize=(50, 30))
for category, color in color_mapping.items():
subset = train_data[train_data['price_category'] == category]
plt.scatter(
subset['longitude'],
subset['latitude'],
c=color,
alpha=0.5,
label=f'{category} Price'
)

# Add gridlines for reference areas
plt.axhline(y=40.7, color='black', linestyle='-', linewidth=0.5)  # Horizontal line at 40.7
plt.axhline(y=40.77, color='black', linestyle='-', linewidth=0.5)  # Horizontal line at 40.77
plt.axhline(y=40.82, color='black', linestyle='-', linewidth=0.5)  # Horizontal line at 40.82
plt.axhline(y=40.655, color='black', linestyle='-', linewidth=0.5)  # Horizontal line at 40.655
plt.axhline(y=40.75, color='black', linestyle='-', linewidth=0.5)  # Horizontal line at 40.75
plt.axhline(y=40.66, color='black', linestyle='-', linewidth=0.5)  # Horizontal line at 40.66
plt.axvline(x=-73.9, color='black', linestyle='-', linewidth=0.5)  # Vertical line at -73.9
```

```
plt.axvline(x=-73.97, color='black', linestyle='-', linewidth=0.5)  # Vertical line at -73.97
plt.axvline(x=-73.93, color='black', linestyle='-', linewidth=0.5)  # Vertical line at -73.93
plt.axvline(x=-74.02, color='black', linestyle='-', linewidth=0.5)  # Vertical line at -74.02
plt.axvline(x=-73.96, color='black', linestyle='-', linewidth=0.5)  # Vertical line at -73.96




# Add legend
plt.legend(title='Price Category', fontsize=12)

# Add title and axis labels
plt.title('Geographic Distribution of Listings by Price Category', fontsize=16)
plt.xlabel('Longitude', fontsize=12)
plt.ylabel('Latitude', fontsize=12)

# Display grid
plt.grid(alpha=0.3)

# Show the plot
plt.show()
```

Listing 4: Feature Engineering

```
# Initialize PorterStemmer
stemmer = PorterStemmer()

# # Initialize Translator
# translator = Translator()

# def translate_text(text):
#     """
#     Translate non-English text to English.
#     """
#     try:
#         if detect(text) != 'en':  # Check if the text is not in English
#             return translator.translate(text, src='auto', dest='en').text
```

```
#         return text
#     except:
#         return text  # Return original text if translation fails

def preprocess_airbnb_data(dataframe, amenities_cluster_mapping=amenities_cluster_mapping, lda_model_desc=None,
lda_model_reviews=None, vectorizer_desc=None, vectorizer_reviews=None,
num_clusters=8, n_topics_desc=7, n_topics_reviews=7):
"""
Preprocess the Airbnb dataset:
Includes date splitting, feature processing, LDA model generation and application, category alignment, etc.
"""
########################################################################
# 1. Date Processing
########################################################################
# 1.1 Split date columns into year and month
def split_date_column(df, column_name):
"""
Splits a date column into year and month columns.
Note: Day is not extracted, as it is less relevant to the importance of the data.
"""
df[f'{column_name}_year'] = df[column_name].dt.year
df[f'{column_name}_month'] = df[column_name].dt.month
return df

# List of date columns to process
date_columns = ['host_since', 'first_review', 'last_review']
for col in date_columns:
dataframe = split_date_column(dataframe, col)

# 1.2 Calculate date intervals in months
def calculate_date_intervals_months(df, start_col, end_col, new_col_name):
"""
Calculates the interval (in months) between two date columns, handling NaN values.
The calculation is based on full months (e.g., from 2023-01-15 to 2023-03-10 equals 2 months).
"""
def calculate_month_diff(start, end):
if pd.isnull(start) or pd.isnull(end):
```

```
return 0  # Replace NaN with 0
return (end.year - start.year) * 12 + (end.month - start.month)

df[new_col_name] = df.apply(lambda row: calculate_month_diff(row[start_col], row[end_col]), axis=1)
return df

# Calculate intervals in months between relevant columns
dataframe = calculate_date_intervals_months(dataframe, 'host_since', 'first_review', 'host_to_first_review_months')
dataframe = calculate_date_intervals_months(dataframe, 'first_review', 'last_review', 'first_to_last_review_months')
dataframe['host_to_first_review_months'] = dataframe['host_to_first_review_months'].apply(lambda x: max(x, 0))
dataframe['first_to_last_review_months'] = dataframe['first_to_last_review_months'].apply(lambda x: max(x, 0))

# 1.3 Calculate months from current date
current_date = datetime(2024, 11, 1)  # Set the current reference date

def calculate_months_from_now(df, column_name, new_column_name):
"""
Calculates the number of months from a given date column to the current date.
Missing values are filled with 0. For differences within the same month, the result is 0.
"""
def calculate_month_diff(start, end):
if pd.isnull(start) or pd.isnull(end):
return 0  # Missing values are replaced with a placeholder (0)
years_diff = end.year - start.year
months_diff = end.month - start.month
total_months = years_diff * 12 + months_diff
if end.day < start.day:  # Adjust for partial months
total_months += 1
return max(total_months, 0)

df[new_column_name] = df[column_name].apply(lambda x: calculate_month_diff(x, current_date))
return df

# Calculate months from current date for relevant columns
print("Calculating months from current date...")
dataframe = calculate_months_from_now(dataframe, 'host_since',
```

```
        'months_from_host_since')
    dataframe = calculate_months_from_now(dataframe, 'first_review', 'months_from_first_review')
    dataframe = calculate_months_from_now(dataframe, 'last_review', 'months_from_last_review')

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

    ########################################################################
    # 2. Category Features Processing
    ########################################################################

    # 2.1 Creating binary columns for host verifications and counting methods of verification
    verification_types = ['phone', 'email', 'work_email']
    for verification in verification_types:
    dataframe[verification] = dataframe['host_verifications'].apply(lambda x: 1 if verification in x else 0)

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

    # 2.2 Creating columns for shared or private bathrooms
    dataframe['is_shared_bathroom'] = dataframe['bathrooms_text'].apply(
    lambda x: 1 if 'shared' in str(x).lower() else 0
    )
    dataframe['is_private_bathroom'] = dataframe['bathrooms_text'].apply(
    lambda x: 1 if 'private' in str(x).lower() else 0
    )

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

    # Ratios provide trend insights about the host's property data
    dataframe['host_listings_ratio'] = dataframe['host_listings_count'] / dataframe['host_total_listings_count']
    dataframe['calculated_to_listings_ratio'] = dataframe['calculated_host_listings_count'] / dataframe['host_listings_count']
```

```
# 2.4 Cleaning and processing the 'property_type' column
def clean_property_type(column, threshold=20):
"""
Cleans the 'property_type' column by standardizing text and grouping rare categories into 'others'.

Parameters:
column (pd.Series): The property_type column to clean.
threshold (int): Minimum frequency to retain a category.

Returns:
pd.Series: Cleaned property_type column with low-frequency categories grouped as 'others'.
"""
# Standardize and clean property_type values
cleaned_column = (
column
.str.lower()
.str.replace(r"^(private room|shared room|entire|room in)\s+in\s+", "", regex=True)
.str.replace(r"^(private room|shared room|entire|room)", "", regex=True)
.str.replace(r"\sin\s.*$", "", regex=True)
.str.replace(r"\bin\b", "", regex=True)
.str.strip()
)

# Replace missing or empty values with 'missing'
cleaned_column = cleaned_column.replace(["", None, np.nan], "missing")

# Count the frequency of each category
value_counts = cleaned_column.value_counts()

# Group rare categories below the threshold into 'others'
cleaned_column = cleaned_column.apply(
lambda x: x if x == "missing" or value_counts[x] >= threshold else "others"
)

return cleaned_column

# 2.5 Further processing for 'property_type_cleaned'
dataframe['property_type_cleaned'] =
```

```
clean_property_type(dataframe['property_type'], threshold=20)

# Count the occurrences of each value and group rare values into 'others'
value_counts = dataframe['property_type_cleaned'].value_counts()
dataframe['property_type_cleaned'] = dataframe['property_type_cleaned'].apply(
lambda x: x if value_counts[x] >= 100 else 'others'
)

# 2.6 Handling rare values in 'neighbourhood_cleansed'
if 'neighbourhood_cleansed' in dataframe.columns:
neighbourhood_counts = dataframe['neighbourhood_cleansed'].value_counts()
rare_neighbourhoods = neighbourhood_counts[neighbourhood_counts < 200].index
dataframe['neighbourhood_cleansed'] = dataframe['neighbourhood_cleansed'].replace(rare_neighbourhoods, 'others')

#########################################################################
# 3. Numerical Features Processing
#########################################################################

# 3.1 Normalize availability columns
availability_columns = {
    'availability_365': 365,
    'availability_90': 90,
    'availability_60': 60,
    'availability_30': 30
}
for col, total_days in availability_columns.items():
dataframe[col] = dataframe[col] / total_days

# 3.2 Handling missing values
dataframe.loc[dataframe['maximum_nights'] == 1125, 'maximum_nights'] = 9999  # Replace default value with a better estimate
dataframe['bedrooms'] = dataframe['bedrooms'].fillna(0)  # Fill missing values with 0
dataframe['bathrooms'] = dataframe['bathrooms'].fillna(0)

# 3.3 Create a new feature 'accommodates_category'
def categorize_accommodates(x):
if x == 1:
```

```
return '1 person'
elif x == 2:
return '2 persons'
elif 3 <= x <= 5:
return '3-5 persons'
else:
return '6+ persons'

dataframe['accommodates_category'] = dataframe['accommodates'].apply(categorize_accommodates)

# Fill missing values for 'host_response_time'
dataframe['host_response_time'] = dataframe['host_response_time'].fillna('missing')

# 3.4 Assign a geographical category based on latitude and longitude
def assign_area_category(row):
"""
Assign an area category based on latitude and longitude rules.
Categories:
2: High concentration of specific price distributions
1: Mixed price distribution
0: Predominantly low/mid-price areas
"""
if (
(40.75 <= row['latitude'] <= 40.77 and -74.02 <= row['longitude'] < -73.96) or
(40.7 <= row['latitude'] <= 40.75 and -74.02 <= row['longitude'] < -73.97)
):
return 2
elif (
(40.77 <= row['latitude'] <= 40.82 and -74.02 <= row['longitude'] < -73.93) or
(40.7 <= row['latitude'] <= 40.75 and -73.97 <= row['longitude'] < -73.93) or
(40.66 <= row['latitude'] <= 40.7 and -74.02 <= row['longitude'] < -73.9)
):
return 1
else:
return 0

dataframe['area_category'] = dataframe.apply(assign_area_category,
```

```
    axis=1)

# Adjust latitude and longitude to centralized values
dataframe['latitude'] = dataframe['latitude'] - 41
dataframe['longitude'] = dataframe['longitude'] + 74

# 3.5 Fill missing values for review scores and rates with a default
    value
missing_vars = [
'review_scores_rating', 'review_scores_accuracy',
    'review_scores_cleanliness',
'review_scores_checkin', 'review_scores_communication',
    'review_scores_location',
'review_scores_value', 'reviews_per_month', 'host_response_rate',
    'host_acceptance_rate'
]
for var in missing_vars:
dataframe[var] = dataframe[var].fillna(9999)

########################################################################
# 4. Text Data Processing
########################################################################

# # 4.1 LDA Feature Extraction Function
# def lda_processing(text_column, vectorizer=None, lda_model=None,
    n_topics=5):
#     """
#     Extract LDA features from a text column.

#     Parameters:
#         text_column (pd.Series): Column containing text data.
#         vectorizer (CountVectorizer): Optional pre-fitted
    CountVectorizer.
#         lda_model (LatentDirichletAllocation): Optional pre-fitted
    LDA model.
#         n_topics (int): Number of topics for LDA.

#     Returns:
#         tuple: Topic matrix, fitted vectorizer, and fitted LDA
    model.
#     """
#     print(f"Vectorizing {text_column.name}...")
#     if vectorizer is None:
#         vectorizer = CountVectorizer(max_features=5000,
```

```
    stop_words='english')
#         text_matrix = vectorizer.fit_transform(text_column.astype(str))
#     else:
#         text_matrix = vectorizer.transform(text_column.astype(str))

#     print(f"Applying LDA on {text_column.name}...")
#     if lda_model is None:
#         lda_model = LatentDirichletAllocation(n_components=n_topics, random_state=42)
#         topic_matrix = lda_model.fit_transform(text_matrix)
#     else:
#         topic_matrix = lda_model.transform(text_matrix)

#     return topic_matrix, vectorizer, lda_model


# # 4.2 Text Preprocessing Function
# def preprocess_text(text, stemmer):
#     """
#     Custom text preprocessing: tokenization, stemming, removing special characters.

#     Parameters:
#         text (str): Input text.
#         stemmer (PorterStemmer): Stemmer instance for stemming tokens.

#     Returns:
#         str: Processed text.
#     """
#     if pd.isnull(text) or text.strip() == '':
#         return "missing"

#     # Replace escape characters
#     text = text.replace("\\'", "'").replace("\\\"", "\"")

#     # Remove special characters and convert to lowercase
#     text = re.sub(r'\W+', ' ', text).lower()

#     # Tokenize and apply stemming
#     tokens = text.split()
#     stemmed_tokens = [stemmer.stem(token) for token in tokens]
#     return ' '.join(stemmed_tokens)
```

```python


# # 4.3 Preprocessing and LDA for 'description'
# tqdm.pandas()
# stemmer = PorterStemmer()

# print("Preprocessing descriptions...")
# dataframe['description'] = dataframe['description'].astype(str).progress_apply(
#     lambda x: preprocess_text(x, stemmer)
# )

# print("Applying LDA on description...")
# description_topics, vectorizer_desc, lda_model_desc = lda_processing(
#     dataframe['description'], vectorizer=None, lda_model=None, n_topics=5
# )
# for i in range(description_topics.shape[1]):
#     dataframe[f'description_topic_{i}'] = description_topics[:, i]


# # 4.4 Preprocessing and LDA for 'reviews'
# print("Translating reviews...")
# dataframe['reviews'] = dataframe['reviews'].astype(str).progress_apply(
#     lambda x: translate_text(x)  # Assumes `translate_text` is defined elsewhere
# )

# print("Preprocessing reviews...")
# dataframe['reviews'] = dataframe['reviews'].progress_apply(
#     lambda x: preprocess_text(x, stemmer)
# )

# print("Applying LDA on reviews...")
# review_topics, vectorizer_reviews, lda_model_reviews = lda_processing(
#     dataframe['reviews'], vectorizer=None, lda_model=None, n_topics=5
# )
# for i in range(review_topics.shape[1]):
#     dataframe[f'review_topic_{i}'] = review_topics[:, i]

```

```
# 4.5 Amenities Clustering
print("Processing amenities...")
dataframe['amenities'] = dataframe['amenities'].apply(
lambda x: str(x).replace('[', '').replace(']', '').replace('"', '').split(', ')
)

# Generate cluster mapping if not provided
if amenities_cluster_mapping is None:
print("Generating amenities cluster mapping...")

# Extract all unique amenities
all_amenities = dataframe['amenities'].explode().unique().tolist()
print(f"Unique amenities found: {len(all_amenities)}")

# Generate embeddings using Sentence-BERT
sbert_model = SentenceTransformer('all-distilroberta-v1')
print("Generating embeddings for amenities...")
amenity_embeddings = sbert_model.encode(all_amenities)

# Perform clustering
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
clusters = kmeans.fit_predict(amenity_embeddings)

# Map amenities to clusters
amenities_cluster_mapping = dict(zip(all_amenities, clusters))
print("Amenity cluster mapping created.")

# Initialize cluster count columns
for cluster_id in range(num_clusters):
dataframe[f'amenity_cluster_{cluster_id}'] = 0

# Count amenities per cluster for each row
print("Counting amenities per cluster for each row...")
for index, amenities_list in tqdm(dataframe['amenities'].items(), desc="Processing amenities"):
cluster_counts = {}
for amenity in amenities_list:
if amenity in amenities_cluster_mapping:
cluster_id = amenities_cluster_mapping[amenity]
cluster_counts[cluster_id] = cluster_counts.get(cluster_id, 0) + 1

# Assign counts to respective cluster columns
for cluster_id, count in cluster_counts.items():
```

```python
dataframe.at[index, f'amenity_cluster_{cluster_id}'] = count

#############################################################################
# 5. Transform
#############################################################################

# 5.1 Scale high-variance columns
# Select numeric columns (excluding 'price') for scaling
scale_sensitive_columns = dataframe.select_dtypes(include=['float64', 'int64']).columns.drop('price', errors='ignore')
# Exclude 'amenity_cluster' columns from scaling
scale_sensitive_columns = [col for col in scale_sensitive_columns if not col.startswith('amenity_cluster')]

# Apply logarithmic transformation to high-variance columns
for col in scale_sensitive_columns:
if dataframe[col].std() > 1:  # Log-transform only if the standard deviation is high
dataframe[col] = np.log1p(dataframe[col])  # Use log1p to handle zero values safely

# 5.2 Drop unnecessary columns
columns_to_drop = [
'name', 'description', 'reviews', 'amenities', 'property_type',
'host_since', 'first_review', 'last_review', 'host_verifications',
'phone', 'email', 'work_email', 'bathrooms_text', 'availability_365',
'has_availability'
]

# Drop specified columns, ignoring errors if they don't exist in the dataframe
dataframe.drop(columns=columns_to_drop, inplace=True, errors='ignore')


return dataframe, amenities_cluster_mapping, lda_model_desc, lda_model_reviews, vectorizer_desc, vectorizer_reviews


# Preprocess train and test datasets with shared mappings/models
train_data, amenities_cluster_mapping, lda_model_desc, lda_model_reviews, vectorizer_desc, vectorizer_reviews = preprocess_airbnb_data(train_data)

test_data, _, _, _, _, _ = preprocess_airbnb_data(
```

```
    test_data, amenities_cluster_mapping, lda_model_desc,
        lda_model_reviews, vectorizer_desc, vectorizer_reviews
    )

    # One-hot encode categorical variables in train and test datasets
    train_data_dummies = pd.get_dummies(train_data, drop_first=True)
    test_data_dummies = pd.get_dummies(test_data, drop_first=True)

    # Align test data columns with train data, filling missing columns
        with 0
    test_data_dummies =
        test_data_dummies.reindex(columns=train_data_dummies.columns,
        fill_value=0)

    # Ensure 'price' is not included in the test dataset
    if 'price' in test_data_dummies.columns:
    test_data_dummies = test_data_dummies.drop(columns=['price'])

    # Verify the processed test dataset
    test_data_dummies.head()
```

Listing 5: Data Download

```
    # This is the original data, but the LDA process is quite lengthy, so
        we saved the original data with the LDA results included.
    # The LDA code is provided in the preprocessing section but has been
        commented out.import pandas as pd
    # train_data = pd.read_csv('/content/drive/My Drive/Colab
        Notebooks/final_project/train.csv', parse_dates=['host_since',
        'first_review', 'last_review'])
    # test_data = pd.read_csv('/content/drive/My Drive/Colab
        Notebooks/final_project/test.csv', parse_dates=['host_since',
        'first_review', 'last_review'])

    train_data = pd.read_csv('/content/drive/My Drive/Colab
        Notebooks/final_project/train_LDA.csv', parse_dates=['host_since',
        'first_review', 'last_review'])
    test_data = pd.read_csv('/content/drive/My Drive/Colab
        Notebooks/final_project/test_LDA.csv', parse_dates=['host_since',
        'first_review', 'last_review'])

    # This is the processed amenities mapping used for analyzing
        amenities, which is based on train_data
    # and includes the parts I manually handled, so it can be directly
```

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

Listing 6: Feature Importance

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

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

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

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

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

```

# Sort by Permutation Importance
sorted_idx_perm = perm_importance.importances_mean.argsort()[::-1]
features_sorted_perm = features.columns[sorted_idx_perm]
perm_sorted = perm_importance.importances_mean[sorted_idx_perm]

# Plot Gini Gain Feature Importance
plt.figure(figsize=(30, 20))
plt.barh(features_sorted_gini, gini_sorted, color="blue", alpha=0.7)
plt.gca().invert_yaxis()
plt.title("Feature Importances (Gini Gain)")
plt.xlabel("Importance")
plt.show()

# Plot Permutation Importance
plt.figure(figsize=(30, 20))
plt.barh(features_sorted_perm, perm_sorted, color="green", alpha=0.7)
plt.gca().invert_yaxis()
plt.title("Feature Importances (Permutation Importance)")
plt.xlabel("Importance")
plt.show()

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize, LabelEncoder

# Assuming train_data_dummies is already loaded
# Extract 'price' as target and binarize it
target = train_data_dummies['price']
features = train_data_dummies.drop(columns=['price'])

# Encode target into numeric classes if not already encoded
le = LabelEncoder()
target_encoded = le.fit_transform(target)

# Binarize the target for multiclass OvR
target_binarized = label_binarize(target_encoded,
    classes=range(len(le.classes_)))

# Select a specific class (e.g., Class 0)
class_index = 0  # Change this index for other classes
class_name = le.classes_[class_index]

```

```
# Initialize the plot
plt.figure(figsize=(40, 30))

# Loop through each feature and plot ROC curve for the selected class
for feature in features.columns:
feature_values = features[feature]

# Handle missing values by filling with the median or another method
if feature_values.isna().any():
feature_values = feature_values.fillna(feature_values.median())

# Handle non-numeric features by encoding them
if not pd.api.types.is_numeric_dtype(feature_values):
feature_values = pd.factorize(feature_values)[0]

# Compute ROC curve and AUC for the selected class
fpr, tpr, _ = roc_curve(target_binarized[:, class_index],
    feature_values)
roc_auc = auc(fpr, tpr)

# Add the ROC curve to the plot
plt.plot(fpr, tpr, label=f'{feature} (AUC = {roc_auc:.2f})')

# Finalize the plot
plt.plot([0, 1], [0, 1], 'k--', label='Random chance')  # Diagonal
    line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curves for All Features (Class: {class_name})')
plt.legend(loc='best', bbox_to_anchor=(1.05, 1))
plt.grid()
plt.tight_layout()
plt.show()
```

Listing 7: Data Split

```
# Separate features and target from the train dataset
target = train_data_dummies['price']
features = train_data_dummies.drop(columns=['price'])

# Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
features, target, test_size=0.2, random_state=42
)
```

```
print(f"Training set size: {X_train.shape}, Test set size: {X_test.shape}")

# Initialize the imputer for missing values
imputer = SimpleImputer(strategy='median')

# Impute missing values for train and test sets
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Impute missing values for the full training dataset and the true test dataset
X_full_train = imputer.fit_transform(features)  # Full training features
y_full_train = target
X_true_test = imputer.transform(test_data_dummies)  # True test dataset features
```

Listing 8: XGBoost

```
# Ensure that X_train and y_train are NumPy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)

# Parameter search range
n_estimators_range = [100, 500, 1000]
learning_rate_range = [0.01, 0.1, 0.5]
max_depth_range = [5, 7, 10]

# Cross-validation setup
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Store results for visualization
results = []

# Iterate through parameter combinations
for n_estimators in n_estimators_range:
for learning_rate in learning_rate_range:
for max_depth in max_depth_range:
print(f"Processing: n_estimators={n_estimators}, learning_rate={learning_rate}, max_depth={max_depth}")

rmse_scores = []

```

```
# 5-fold cross-validation
for train_index, valid_index in kf.split(X_train):
# Split training and validation sets
X_train_cv, X_valid_cv = X_train[train_index], X_train[valid_index]
y_train_cv, y_valid_cv = y_train[train_index], y_train[valid_index]

# Build the model
model_xgb = XGBRegressor(
n_estimators=n_estimators,
learning_rate=learning_rate,
max_depth=max_depth,
tree_method="hist",  # Use histogram-based method for faster training
random_state=42
)

# Train the model
model_xgb.fit(X_train_cv, y_train_cv, eval_set=[(X_valid_cv,
    y_valid_cv)], verbose=False)

# Predict on the validation set
y_pred_cv = model_xgb.predict(X_valid_cv)
rmse_cv = np.sqrt(mean_squared_error(y_valid_cv, y_pred_cv))
rmse_scores.append(rmse_cv)

# Calculate the average RMSE for the current parameter combination
mean_rmse = np.mean(rmse_scores)
print(f"  Average RMSE: {mean_rmse:.4f}")

# Append the result for visualization
results.append((n_estimators, learning_rate, max_depth, mean_rmse))

# Convert results to a structured format
results = np.array(results)
n_estimators_vals = results[:, 0]
learning_rate_vals = results[:, 1]
max_depth_vals = results[:, 2]
rmse_vals = results[:, 3]

# Find the best parameters and RMSE
best_index = np.argmin(rmse_vals)
best_params = {
    "n_estimators": int(n_estimators_vals[best_index]),
    "learning_rate": learning_rate_vals[best_index],
    "max_depth": int(max_depth_vals[best_index])
```

```
}
best_rmse = rmse_vals[best_index]

print("\nOptimization complete.")
print("Best Parameters:")
print(best_params)
print(f"Best RMSE: {best_rmse:.4f}")

# Create an interactive 3D scatter plot with Plotly
fig = go.Figure()

# Add scatter points
fig.add_trace(go.Scatter3d(
x=n_estimators_vals,
y=learning_rate_vals,
z=max_depth_vals,
mode='markers',
marker=dict(
size=8,
color=rmse_vals,  # Color by RMSE
colorscale='Viridis',  # Color scale
colorbar=dict(title="RMSE"),
opacity=0.8
)
))

# Set axis labels and title
fig.update_layout(
scene=dict(
xaxis_title="n_estimators",
yaxis_title="learning_rate",
zaxis_title="max_depth"
),
title="Interactive 3D Visualization of Hyperparameter Search",
margin=dict(l=0, r=0, b=0, t=40)
)

# Show the plot
fig.show()


# Define the Optuna objective function
def objective(trial):
# Define hyperparameters to optimize
```

```python
    param = {
        "n_estimators": trial.suggest_int("n_estimators", 500, 1500),
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.2),
        "max_depth": trial.suggest_int("max_depth", 7, 15),
        "subsample": trial.suggest_float("subsample", 0.8, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.6, 1.0),
        "gamma": trial.suggest_float("gamma", 0.07, 0.1),
        "reg_alpha": trial.suggest_float("reg_alpha", 1, 20),
        "reg_lambda": trial.suggest_float("reg_lambda", 0, 10),
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 10),
        "scale_pos_weight": trial.suggest_float("scale_pos_weight", 1, 2),
        "max_delta_step": trial.suggest_int("max_delta_step", 0, 10),
        "grow_policy": "depthwise",
        "random_state": 42
    }

    # Create XGBoost model
    model_xgb = XGBRegressor(**param)

    # Train the model
    model_xgb.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)

    # Predict on the validation set and calculate RMSE
    y_pred_xgb = model_xgb.predict(X_test)
    rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
    return rmse_xgb

    # Use Optuna to optimize hyperparameters
    study_xgb = optuna.create_study(direction="minimize")
    study_xgb.optimize(objective, n_trials=100)  # Run 500 trials for the final submission

    # Print the best parameters and RMSE
    print("Best parameters (XGBoost):", study_xgb.best_params)
    print("Best RMSE (XGBoost):", study_xgb.best_value)
    # Retrieve the top 51 trial parameters (51 for final submission)
    top_trials_xgb = study_xgb.trials_dataframe().sort_values(by="value").head(11)
    counter = 0
    models_xgb = []

    # Train models using the top 51 trial parameters
```

```python
for _, trial_row in top_trials_xgb.iterrows():
counter += 1
print(f"Training model {counter}...")

# Extract parameters
trial_number = int(trial_row["number"])
best_params_xgb = study_xgb.trials[trial_number].params

# Create and train the model
model_xgb = XGBRegressor(**best_params_xgb)
model_xgb.fit(X_train, y_train, verbose=False)
models_xgb.append(model_xgb)

# Get predictions from all models
predictions_xgb = np.array([model.predict(X_test) for model in models_xgb])

# Clip predictions to the range [0, 5] and round them
predictions_xgb_clipped = np.round(np.clip(predictions_xgb, 0, 5))

# Initialize the RMSE results list
rmse_results = []

# Compute RMSE for 1, 3, 5, ..., 51 models
for num_models in range(1, 12, 2):  # 1, 3, 5, ..., 51
# Compute the mode of predictions from the first num_models models
y_pred_xgb_mode = mode(predictions_xgb_clipped[:num_models], axis=0).mode.squeeze()

# Ensure y_pred_xgb_mode is a 1D array with the same shape as y_test
y_pred_xgb_mode = y_pred_xgb_mode.flatten()

# Compute RMSE and store the result
rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb_mode))
rmse_results.append((num_models, rmse_xgb))

# Convert results to an array for further processing
rmse_results = np.array(rmse_results)

# Find the number of models that give the minimum RMSE
optimal_index = np.argmin(rmse_results[:, 1])
optimal_models = int(rmse_results[optimal_index, 0])
optimal_rmse = rmse_results[optimal_index, 1]
```

```python
    print(f"Optimal number of models: {optimal_models}")
    print(f"Minimum RMSE: {optimal_rmse:.4f}")

    # Plot RMSE vs. number of models
    plt.figure(figsize=(10, 6))
    plt.plot(rmse_results[:, 0], rmse_results[:, 1], marker='o', linestyle='-', color='blue', label='RMSE')
    plt.axvline(optimal_models, color='red', linestyle='--', label=f'Optimal: {optimal_models} models')
    plt.title('RMSE vs. Number of Models', fontsize=16)
    plt.xlabel('Number of Models', fontsize=12)
    plt.ylabel('RMSE', fontsize=12)
    plt.grid(alpha=0.3)
    plt.legend()
    plt.show()
```

Listing 9: SVM

```python
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_scaled = np.array(X_train_scaled)
y_train = np.array(y_train)
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'C': [0.1, 1, 10, 100],             # Regularization parameter
    'epsilon': [0.01, 0.1, 0.2, 0.5],  # Epsilon in the epsilon-SVR model
    'kernel': ['rbf']         # Kernel types to consider
}

# Set up cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Initialize variables to store the best result and all results for visualization
best_rmse = float('inf')
best_params = None
results = []  # Store all combinations for 3D visualization

# Manual Grid Search
print("Starting manual grid search...")
for kernel in param_grid['kernel']:
```

```python
for C in param_grid['C']:
for epsilon in param_grid['epsilon']:
print(f"Training with kernel={kernel}, C={C}, epsilon={epsilon}")

rmse_scores = []
for train_index, val_index in kf.split(X_train_scaled):
# Split data into training and validation sets
X_train_cv, X_val_cv = X_train_scaled[train_index], X_train_scaled[val_index]
y_train_cv, y_val_cv = y_train[train_index], y_train[val_index]

# Train the model
svr = SVR(kernel=kernel, C=C, epsilon=epsilon)
svr.fit(X_train_cv, y_train_cv)

# Predict and calculate RMSE
y_val_pred = svr.predict(X_val_cv)
rmse = np.sqrt(mean_squared_error(y_val_cv, y_val_pred))
rmse_scores.append(rmse)

# Calculate mean RMSE across folds
mean_rmse = np.mean(rmse_scores)
print(f"Mean RMSE: {mean_rmse:.4f}")

# Store the results for 3D visualization
results.append((kernel, C, epsilon, mean_rmse))

# Update best parameters if current RMSE is better
if mean_rmse < best_rmse:
best_rmse = mean_rmse
best_params = {'kernel': kernel, 'C': C, 'epsilon': epsilon}

# Train the final model with the best parameters on the full training set
print("\nTraining final model with best parameters...")
print(f"Best Parameters: {best_params}")
svr = SVR(**best_params)
svr.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = svr.predict(X_test_scaled)

# Calculate RMSE on the test set
final_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

```
print(f"Final Test RMSE: {final_rmse:.4f}")

# 3D Plot of RMSE values
results = np.array(results)
kernels = results[:, 0]
Cs = results[:, 1].astype(float)
epsilons = results[:, 2].astype(float)
rmses = results[:, 3].astype(float)

# Convert kernel types to numeric values for 3D plotting
kernel_map = {'linear': 0, 'rbf': 1}
kernel_numeric = np.array([kernel_map[k] for k in kernels])

fig = go.Figure()

# Add scatter points
fig.add_trace(go.Scatter3d(
x=Cs,
y=epsilons,
z=kernel_numeric,
mode='markers',
marker=dict(
size=8,
color=rmses,  # Color by RMSE
colorscale='Viridis',  # Color scale
colorbar=dict(title="RMSE"),
opacity=0.8
)
))

# Set axis labels and title
fig.update_layout(
scene=dict(
xaxis_title="C (Regularization Parameter)",
yaxis_title="Epsilon",
zaxis_title="Kernel (0=Linear, 1=RBF)"
),
title="3D Visualization of Grid Search Results",
margin=dict(l=0, r=0, b=0, t=40)
)

# Show the plot
fig.show()
```

Listing 10: Test

```
# Retrieve the top 10 trials for XGBoost
top_trials_xgb = study_xgb.trials_dataframe().sort_values(by="value").head(39)

# Use a bagging approach to retrain XGBoost models
models_xgb_test = []
for _, trial_row in top_trials_xgb.iterrows():
# Extract parameters
trial_number = int(trial_row["number"])
best_params_xgb = study_xgb.trials[trial_number].params

# Create and train the model
model_xgb = XGBRegressor(**best_params_xgb)
model_xgb.fit(X_full_train, y_full_train, verbose=False)
models_xgb_test.append(model_xgb)

# Make predictions on the test set
predictions_xgb_test = np.array([model.predict(X_true_test) for model in models_xgb_test])
# Clip predictions to the range [0,5] and round to the nearest integer
predictions_xgb_clipped = np.round(np.clip(predictions_xgb_test, 0, 5))  # (10, n_samples)
# Take the mode (most common value) across models for each sample
y_pred_xgb_mode = mode(predictions_xgb_clipped, axis=0).mode.squeeze()  # Remove unnecessary dimensions
# Ensure `y_pred_xgb_mode` is a 1D array and matches the shape of `y_test`
y_pred_xgb_mode = y_pred_xgb_mode.flatten()
# Final predictions for the test set
y_pred_xgb_test = y_pred_xgb_mode
# Load the test dataset
test_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/final_project/test_LDA.csv', parse_dates=['host_since', 'first_review', 'last_review'])
# Assuming `test_data['id']` is loaded, prepare the submission file
submission = pd.DataFrame({
    'id': test_data['id'],  # Unique identifier for the test set
    'price': y_pred_xgb_test  # Predictions from the model
})

# Save the results as an Excel file
submission.to_excel('/content/drive/My Drive/Colab
```

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

DataFrame | **NO COMPARISON TARGET**

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

ASSOCIATIONS

DataFrame

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)

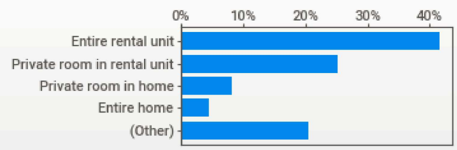
2 description

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 step from the door of this newly refur…
39	<1%	Relax with the whole family at this peaceful place to stay.
14,962	98%	(Other)

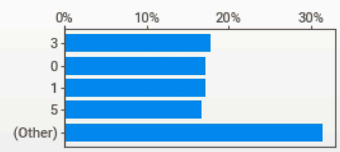
3 property_type

VALUES: 15,696 (100%)
MISSING: ---
DISTINCT: 59 (<1%)



4 price

VALUES: 15,696 (100%)
MISSING: ---
DISTINCT: 6 (<1%)



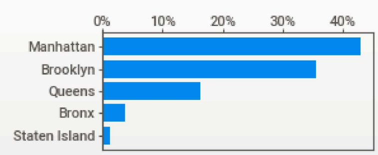
5 neighbourhood_cleansed

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

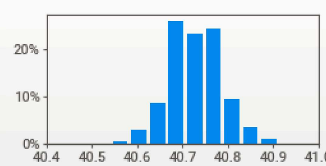
VALUES: 15,696 (100%)
MISSING: ---
DISTINCT: 5 (<1%)



7 latitude

VALUES: 15,696 (100%)
MISSING: ---
DISTINCT: 12,648 (81%)
ZEROES: ---

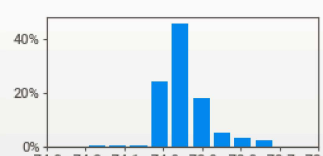
MAX	40.911	RANGE	0.411
95%	40.824	IQR	0.077
Q3	40.762	STD	0.058
AVG	40.727	VAR	0.003
MEDIAN	40.725		
Q1	40.686	KURT.	0.084
5%	40.635	SKEW	0.120
MIN	40.500	SUM	639k



8 longitude

VALUES: 15,696 (100%)
MISSING: ---
DISTINCT: 12,175 (78%)
ZEROES: ---

MAX	-73.714	RANGE	0.538
95%	-73.813	IQR	0.062
Q3	-73.921	STD	0.060
AVG	-73.943	VAR	0.004
MEDIAN	-73.952		
Q1	-73.983	KURT.	2.77
5%	-74.006	SKEW	1.06
MIN	-74.252	SUM	-1.2M



9 host_since

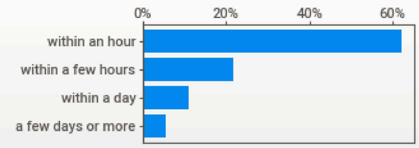
VALUES: 15,696 (100%)
MISSING: ---

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

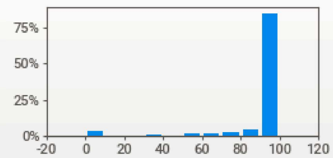
VALUES:	13,493	(86%)
MISSING:	2,203	(14%)
DISTINCT:	4	(<1%)



11 host_response_rate

VALUES:	13,493	(86%)
MISSING:	2,203	(14%)
DISTINCT:	65	(<1%)
ZEROES:	500	(3%)

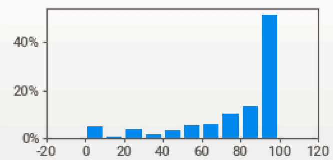
MAX	100	RANGE	100
95%	100	IQR	3.00
Q3	100	STD	22.3
MEDIAN	100	VAR	498
AVG	91		
Q1	97	KURT.	9.29
5%	33	SKEW	-3.16
MIN	0	SUM	1.2M



12 host_acceptance_rate

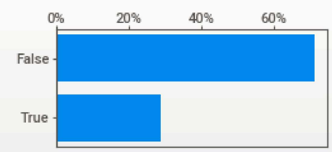
VALUES:	13,643	(87%)
MISSING:	2,053	(13%)
DISTINCT:	97	(<1%)
ZEROES:	642	(4%)

MAX	100	RANGE	100
95%	100	IQR	31.0
Q3	100	STD	27.9
MEDIAN	91	VAR	778
AVG	79		
Q1	69	KURT.	1.26
5%	9	SKEW	-1.48
MIN	0	SUM	1.1M



13 host_is_superhost

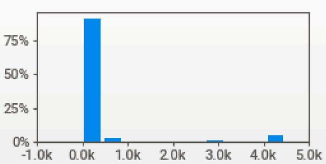
VALUES:	15,445	(98%)
MISSING:	251	(2%)
DISTINCT:	2	(<1%)



14 host_listings_count

VALUES:	15,696	(100%)
MISSING:	---	
DISTINCT:	97	(<1%)
ZEROES:	---	

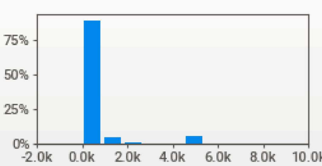
MAX	4,494	RANGE	4,493
95%	2,950	IQR	20.0
Q3	21	STD	984
AVG	288	VAR	969k
MEDIAN	3		
Q1	1	KURT.	13.0
5%	1	SKEW	3.81
MIN	1	SUM	4.5M



15 host_total_listings_count

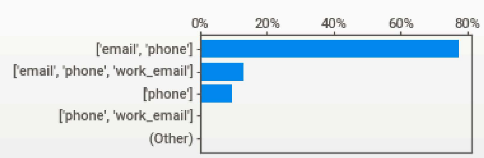
VALUES:	15,696	(100%)
MISSING:	---	
DISTINCT:	132	(<1%)
ZEROES:	---	

MAX	9,019	RANGE	9,018
95%	4,784	IQR	29.0
Q3	31	STD	1,205
AVG	393	VAR	1.5M
MEDIAN	5		
Q1	2	KURT.	12.7
5%	1	SKEW	3.60
MIN	1	SUM	6.2M



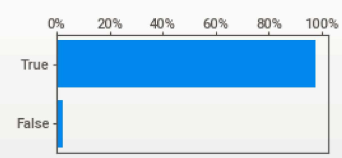
16 host_verifications

VALUES:	15,696	(100%)
MISSING:	---	
DISTINCT:	6	(<1%)



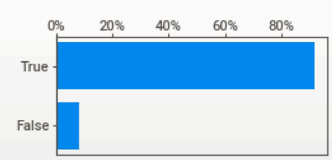
17 host_has_profile_pic

VALUES:	15,696	(100%)
MISSING:	---	
DISTINCT:	2	(<1%)



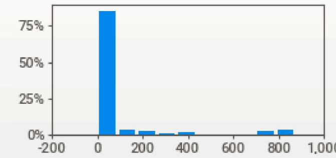
18 host_identity_verified

VALUES:	15,696	(100%)
MISSING:	---	
DISTINCT:	2	(<1%)



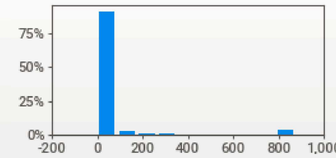
19 calculated_host_listings_count

VALUES:	15,696 (100%)	MAX	876	RANGE	875
MISSING:	---	95%	719	IQR	16.0
DISTINCT:	71 (<1%)	Q3	17	STD	198
ZEROES:	---	AVG	75	VAR	39,221
		MEDIAN	3	KURT.	9.26
		Q1	1	SKEW	3.20
		5%	1	SUM	1.2M
		MIN	1		



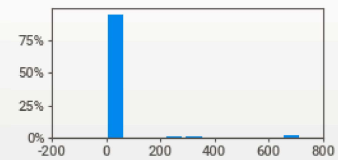
20 calculated_host_listings_count_entire_homes

VALUES:	15,696 (100%)	MAX	876	RANGE	876
MISSING:	---	95%	213	IQR	4.00
DISTINCT:	53 (<1%)	Q3	4	STD	166
ZEROES:	5,433 (35%)	AVG	45	VAR	27,700
		MEDIAN	1	KURT.	19.2
		Q1	0	SKEW	4.49
		5%	0	SUM	711k
		MIN	0		



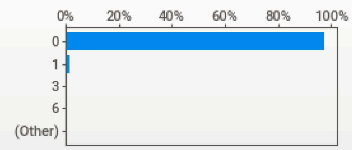
21 calculated_host_listings_count_private_rooms

VALUES:	15,696 (100%)	MAX	719	RANGE	719
MISSING:	---	95%	213	IQR	3.00
DISTINCT:	40 (<1%)	Q3	3	STD	118
ZEROES:	7,788 (50%)	AVG	28	VAR	13,807
		MEDIAN	1	KURT.	25.8
		Q1	0	SKEW	5.09
		5%	0	SUM	435k
		MIN	0		



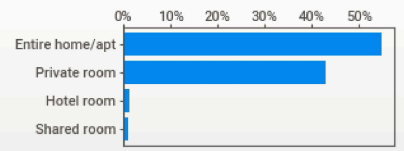
22 calculated_host_listings_count_shared_rooms

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



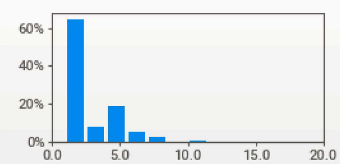
23 room_type

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



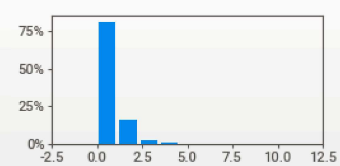
24 accommodates

VALUES:	15,696 (100%)	MAX	16.0	RANGE	15.0
MISSING:	---	95%	6.0	IQR	2.00
DISTINCT:	16 (<1%)	Q3	4.0	STD	1.89
ZEROES:	---	AVG	2.8	VAR	3.58
		MEDIAN	2.0	KURT.	8.92
		Q1	2.0	SKEW	2.37
		5%	1.0	SUM	43,785
		MIN	1.0		



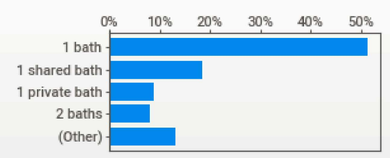
25 bathrooms

VALUES:	15,693 (>99%)	MAX	11.5	RANGE	11.5
MISSING:	3 (<1%)	95%	2.0	IQR	0.00
DISTINCT:	16 (<1%)	Q3	1.0	STD	0.511
ZEROES:	353 (2%)	AVG	1.2	VAR	0.262
		MEDIAN	1.0	KURT.	26.9
		Q1	1.0	SKEW	3.09
		5%	1.0	SUM	18,390
		MIN	0.0		



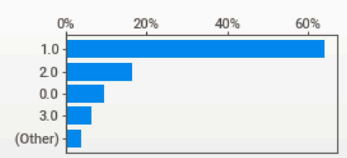
26 bathrooms_text

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



27 bedrooms

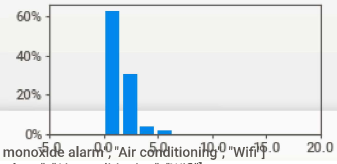
VALUES:	15,662 (>99%)
MISSING:	34 (<1%)
DISTINCT:	10 (<1%)



28 beds

VALUES:	15,612 (>99%)	MAX	16.0	RANGE	16.0
MISSING:	84 (<1%)	95%	4.0	IQR	1.00
		Q3	2.0	STD	1.11

DISTINCT:	15 (<1%)	AVG	1.6	VAR	1.24	
ZEROES:	415 (3%)	MEDIAN	1.0	KURT.	11.5	
		Q1	1.0	SKEW	2.56	
		5%	1.0	SUM	25,037	
		MIN	0.0			



29 amenities

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

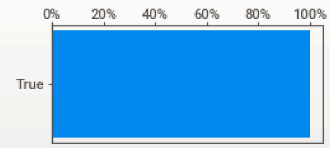
DISTINCT: 13,314 (85%)

132	<1%	["Kitchen", "TV", "Smoke alarm", "Washer", "Carbon monoxide alarm", "Air conditioning", "Wifi"]
79	<1%	["Kitchen", "TV", "Smoke alarm", "Carbon monoxide alarm", "Air conditioning", "Wifi"]
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", "L...
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

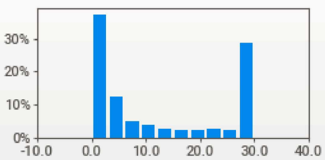
VALUES: 15,578 (>99%)
MISSING: 118 (<1%)

DISTINCT: 1 (<1%)



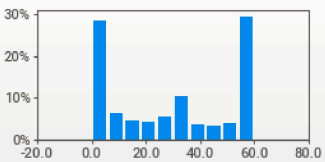
31 availability_30

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	
ZEROES:	4,591 (29%)	AVG	12.1	VAR	156	
		MEDIAN	6.0	KURT.	-1.58	
		Q1	0.0	SKEW	0.437	
		5%	0.0	SUM	191k	
		MIN	0.0			



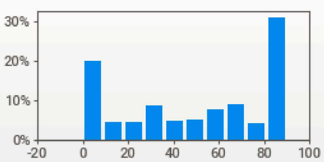
32 availability_60

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	
ZEROES:	2,890 (18%)	MEDIAN	30.0	VAR	564	
		AVG	29.5	KURT.	-1.60	
		Q1	3.0	SKEW	0.055	
		5%	0.0	SUM	463k	
		MIN	0.0			



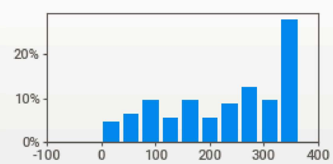
33 availability_90

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	
ZEROES:	2,209 (14%)	MEDIAN	57.0	VAR	1,133	
		AVG	50.4	KURT.	-1.43	
		Q1	19.0	SKEW	-0.246	
		5%	0.0	SUM	790k	
		MIN	0.0			



34 availability_365

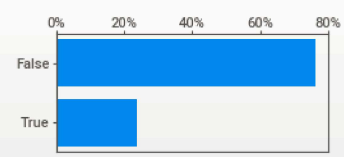
VALUES:	15,696 (100%)	MAX	365	RANGE	365	
MISSING:	---	95%	365	IQR	198	
DISTINCT:	366 (2%)	Q3	335	STD	110	
ZEROES:	121 (<1%)	MEDIAN	256	VAR	12,165	
		AVG	231	KURT.	-1.13	
		Q1	137	SKEW	-0.410	
		5%	39	SUM	3.6M	
		MIN	0			



35 instant_bookable

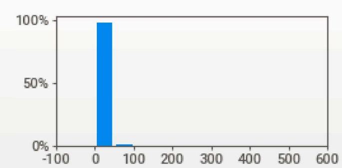
VALUES: 15,696 (100%)
MISSING: ---

DISTINCT: 2 (<1%)



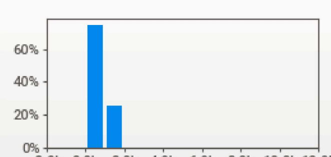
36 minimum_nights

VALUES:	15,696 (100%)	MAX	500	RANGE	499	
MISSING:	---	95%	31	IQR	0.00	
DISTINCT:	60 (<1%)	Q3	30	STD	22.2	
ZEROES:	---	MEDIAN	30	VAR	493	
		AVG	27	KURT.	135	
		Q1	30	SKEW	9.11	
		5%	1	SUM	420k	
		MIN	1			



37 maximum_nights

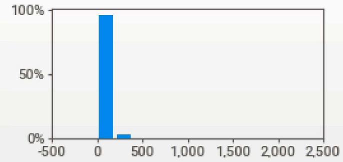
VALUES:	15,696 (100%)	MAX	10,000	RANGE	9,999	
MISSING:	---	95%	1,125	IQR	1,005	
DISTINCT:	175 (1%)	Q3	1,125	STD	412	
ZEROES:	---	AVG	476	VAR	170k	
		MEDIAN	365	KURT.	17.2	
		Q1	120	SKEW	1.51	
		5%	29	SUM	7.5M	
		MIN	1			



number_of_reviews

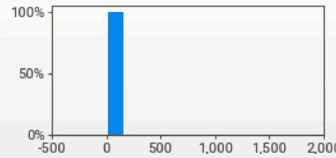
38

VALUES:	15,696 (100%)	MAX	1,941	RANGE	1,941		
MISSING:	---	95%	166	IQR	34.0		
DISTINCT:	434 (3%)	Q3	34	STD	74.0		
ZEROES:	4,474 (29%)	AVG	34	VAR	5,473		
		MEDIAN	6	KURT.	109		
		Q1	0	SKEW	7.06		
		5%	0	SUM	530k		
		MIN	0				



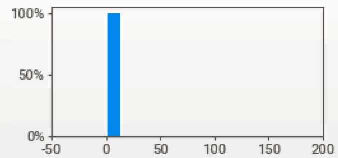
39 number_of_reviews_ltm

VALUES:	15,696 (100%)	MAX	1,772	RANGE	1,772
MISSING:	---	95%	30	IQR	4.00
DISTINCT:	142 (<1%)	Q3	4	STD	23.6
ZEROES:	7,336 (47%)	AVG	6	VAR	557
		MEDIAN	1	KURT.	2,331
		Q1	0	SKEW	37.5
		5%	0	SUM	89,364
		MIN	0		



40 number_of_reviews_l30d

VALUES:	15,696 (100%)	MAX	147	RANGE	147
MISSING:	---	95%	3	IQR	0.00
DISTINCT:	33 (<1%)	Q3	0	STD	2.21
ZEROES:	12,802 (82%)	AVG	0	VAR	4.89
		MEDIAN	0	KURT.	1,689
		Q1	0	SKEW	31.2
		5%	0	SUM	7,442
		MIN	0		



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)

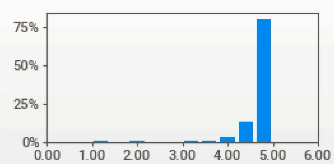
42 last_review

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)

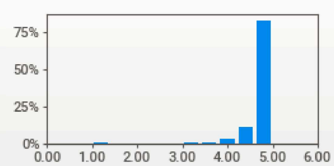
43 review_scores_rating

VALUES:	11,222 (71%)	MAX	5.00	RANGE	4.00
MISSING:	4,474 (29%)	95%	5.00	IQR	0.340
DISTINCT:	149 (<1%)	Q3	5.00	STD	0.463
ZEROES:	---	MEDIAN	4.85	VAR	0.214
		AVG	4.72	KURT.	27.1
		Q1	4.66	SKEW	-4.47
		5%	4.00	SUM	52,961
		MIN	1.00		



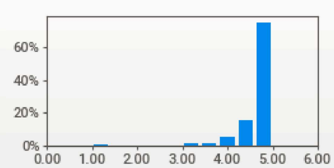
44 review_scores_accuracy

VALUES:	11,222 (71%)	MAX	5.00	RANGE	4.00
MISSING:	4,474 (29%)	95%	5.00	IQR	0.310
DISTINCT:	150 (<1%)	Q3	5.00	STD	0.460
ZEROES:	---	MEDIAN	4.88	VAR	0.212
		AVG	4.74	KURT.	30.4
		Q1	4.69	SKEW	-4.78
		5%	4.00	SUM	53,224
		MIN	1.00		



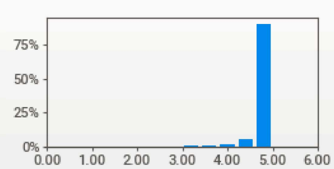
45 review_scores_cleanliness

VALUES:	11,222 (71%)	MAX	5.00	RANGE	4.00
MISSING:	4,474 (29%)	95%	5.00	IQR	0.390
DISTINCT:	156 (<1%)	Q3	4.98	STD	0.483
ZEROES:	---	MEDIAN	4.82	VAR	0.234
		AVG	4.68	KURT.	21.6
		Q1	4.59	SKEW	-3.89
		5%	4.00	SUM	52,515
		MIN	1.00		



46 review_scores_checkin

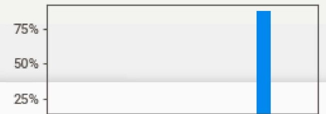
VALUES:	11,222 (71%)	MAX	5.00	RANGE	4.00
MISSING:	4,474 (29%)	95%	5.00	IQR	0.190
DISTINCT:	119 (<1%)	Q3	5.00	STD	0.377
ZEROES:	---	MEDIAN	4.94	VAR	0.142
		AVG	4.83	KURT.	47.7
		Q1	4.81	SKEW	-5.96
		5%	4.33	SUM	54,161
		MIN	1.00		



47 review_scores_communication

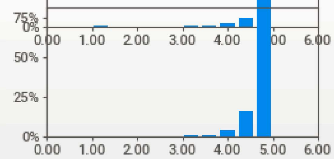
VALUES:	11,222 (71%)	MAX	5.00	RANGE	4.00
MISSING:	4,474 (29%)	95%	5.00	IQR	0.200
DISTINCT:	132 (<1%)	Q3	5.00	STD	0.433
		MEDIAN	4.94	VAR	0.188
		AVG	4.81		

ZEROES:	---	Q1	4.80	KURT.	37.8
		5%	4.19	SKEW	-5.45
		MIN	1.00	SUM	53,958



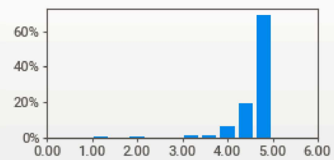
48 review_scores_location

VALUES:	11,221 (71%)	MAX	5.00	RANGE	4.00
MISSING:	4,475 (29%)	95%	5.00	IQR	0.370
DISTINCT:	146 (<1%)	Q3	5.00	STD	0.400
ZEROES:	---	MEDIAN	4.82	VAR	0.160
		AVG	4.72		
		Q1	4.63	KURT.	29.1
		5%	4.00	SKEW	-4.30
		MIN	1.00	SUM	52,984



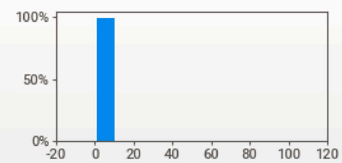
49 review_scores_value

VALUES:	11,222 (71%)	MAX	5.00	RANGE	4.00
MISSING:	4,474 (29%)	95%	5.00	IQR	0.380
DISTINCT:	153 (<1%)	Q3	4.89	STD	0.513
ZEROES:	---	MEDIAN	4.75	VAR	0.263
		AVG	4.61		
		Q1	4.51	KURT.	18.9
		5%	3.86	SKEW	-3.68
		MIN	1.00	SUM	51,728



50 reviews_per_month

VALUES:	11,222 (71%)	MAX	110	RANGE	110
MISSING:	4,474 (29%)	95%	4	IQR	1.44
DISTINCT:	718 (5%)	Q3	2	STD	2.27
ZEROES:	---	AVG	1	VAR	5.15
		MEDIAN	1		
		Q1	0	KURT.	581
		5%	0	SKEW	16.6
		MIN	0	SUM	13,980



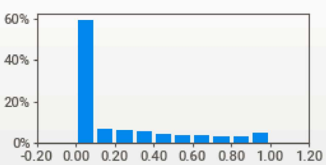
51 reviews

VALUES:	11,222 (71%)
MISSING:	4,474 (29%)
DISTINCT:	11,215 (71%)

3	<1%	Great!
2	<1%	Great stay and great host!
2	<1%	Great place!
2	<1%	Good stay
2	<1%	Good!
2	<1%	Great Stay
1	<1%	Perfect location, good for a short stay -------------------------------- Wonderful, safe, spotless, stylish, and c...
11,208	>99%	(Other)

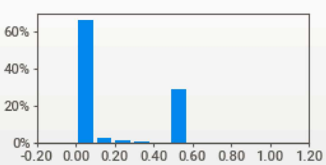
52 review_topic_0

VALUES:	15,696 (100%)	MAX	1.000	RANGE	1.000
MISSING:	---	95%	0.888	IQR	0.330
DISTINCT:	11,189 (71%)	Q3	0.359	STD	0.283
ZEROES:	---	AVG	0.227	VAR	0.080
		MEDIAN	0.071		
		Q1	0.029	KURT.	0.566
		5%	0.000	SKEW	1.35
		MIN	0.000	SUM	3,571



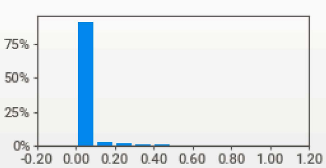
53 review_topic_1

VALUES:	15,696 (100%)	MAX	0.964	RANGE	0.964
MISSING:	---	95%	0.571	IQR	0.571
DISTINCT:	11,189 (71%)	Q3	0.571	STD	0.255
ZEROES:	---	AVG	0.179	VAR	0.065
		MEDIAN	0.003		
		Q1	0.000	KURT.	-1.18
		5%	0.000	SKEW	0.859
		MIN	0.000	SUM	2,807



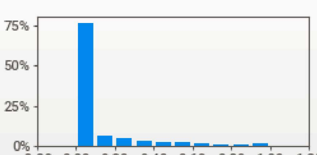
54 review_topic_2

VALUES:	15,696 (100%)	MAX	0.985	RANGE	0.985
MISSING:	---	95%	0.205	IQR	0.071
DISTINCT:	11,189 (71%)	Q3	0.071	STD	0.104
ZEROES:	---	AVG	0.050	VAR	0.011
		MEDIAN	0.005		
		Q1	0.000	KURT.	30.2
		5%	0.000	SKEW	4.86
		MIN	0.000	SUM	789



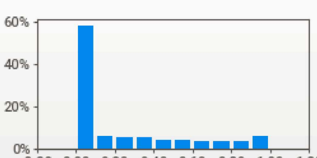
55 review_topic_3

VALUES:	15,696 (100%)	MAX	0.999	RANGE	0.999
MISSING:	---	95%	0.583	IQR	0.079
DISTINCT:	11,189 (71%)	Q3	0.081	STD	0.195
ZEROES:	---	AVG	0.118	VAR	0.038
		MEDIAN	0.071		
		Q1	0.002	KURT.	6.36
		5%	0.000	SKEW	2.54
		MIN	0.000	SUM	1,847



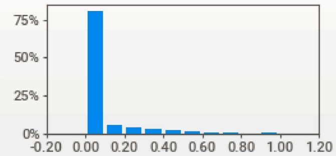
56 review_topic_4

VALUES:	15,696 (100%)	MAX	0.999	RANGE	0.999
MISSING:	---	95%	0.929	IQR	0.358
DISTINCT:	11,189 (71%)	Q3	0.394	STD	0.299
ZEROES:	---	AVG	0.242	VAR	0.090
		MEDIAN	0.071		
		Q1	0.036	KURT.	0.256
		5%	0.000	SKEW	1.26
		MIN	0.000	SUM	3,804



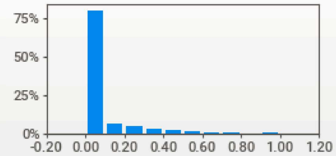
57 review_topic_5

VALUES:	15,696 (100%)	MAX	0.996	RANGE	0.996
MISSING:	---	95%	0.427	IQR	0.071
		Q3	0.071	STD	0.150
DISTINCT:	11,189 (71%)	AVG	0.089	VAR	0.022
		MEDIAN	0.071		
ZEROES:	---	Q1	0.001	KURT.	9.64
		5%	0.000	SKEW	2.90
		MIN	0.000	SUM	1,403



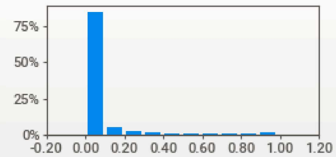
58 review_topic_6

VALUES:	15,696 (100%)	MAX	0.999	RANGE	0.999
MISSING:	---	95%	0.447	IQR	0.070
		Q3	0.071	STD	0.157
DISTINCT:	11,189 (71%)	AVG	0.094	VAR	0.025
		MEDIAN	0.071		
ZEROES:	---	Q1	0.001	KURT.	9.35
		5%	0.000	SKEW	2.89
		MIN	0.000	SUM	1,476



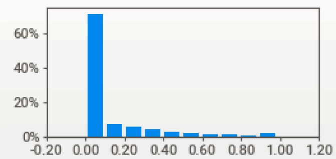
59 description_topic_0

VALUES:	15,696 (100%)	MAX	0.985	RANGE	0.984
MISSING:	---	95%	0.439	IQR	0.017
		Q3	0.020	STD	0.173
DISTINCT:	12,376 (79%)	AVG	0.068	VAR	0.030
		MEDIAN	0.004		
ZEROES:	---	Q1	0.003	KURT.	13.5
		5%	0.002	SKEW	3.62
		MIN	0.001	SUM	1,068



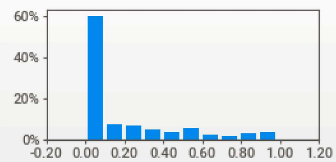
60 description_topic_1

VALUES:	15,696 (100%)	MAX	0.987	RANGE	0.986
MISSING:	---	95%	0.684	IQR	0.147
		Q3	0.151	STD	0.226
DISTINCT:	12,376 (79%)	AVG	0.125	VAR	0.051
		MEDIAN	0.006		
ZEROES:	---	Q1	0.003	KURT.	4.52
		5%	0.003	SKEW	2.25
		MIN	0.001	SUM	1,959



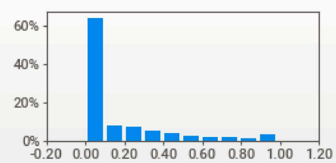
61 description_topic_2

VALUES:	15,696 (100%)	MAX	0.984	RANGE	0.982
MISSING:	---	95%	0.876	IQR	0.301
		Q3	0.304	STD	0.277
DISTINCT:	12,376 (79%)	AVG	0.189	VAR	0.077
		MEDIAN	0.008		
ZEROES:	---	Q1	0.003	KURT.	0.907
		5%	0.002	SKEW	1.44
		MIN	0.001	SUM	2,965



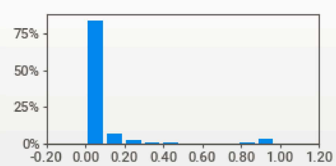
62 description_topic_3

VALUES:	15,696 (100%)	MAX	0.986	RANGE	0.985
MISSING:	---	95%	0.765	IQR	0.235
		Q3	0.239	STD	0.248
DISTINCT:	12,376 (79%)	AVG	0.158	VAR	0.062
		MEDIAN	0.009		
ZEROES:	---	Q1	0.003	KURT.	2.41
		5%	0.003	SKEW	1.79
		MIN	0.002	SUM	2,479



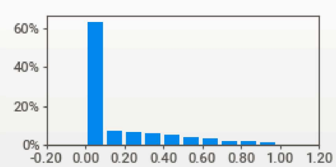
63 description_topic_4

VALUES:	15,696 (100%)	MAX	0.976	RANGE	0.975
MISSING:	---	95%	0.521	IQR	0.047
		Q3	0.050	STD	0.204
DISTINCT:	12,376 (79%)	AVG	0.078	VAR	0.042
		MEDIAN	0.004		
ZEROES:	---	Q1	0.003	KURT.	12.3
		5%	0.002	SKEW	3.61
		MIN	0.001	SUM	1,232



64 description_topic_5

VALUES:	15,696 (100%)	MAX	0.986	RANGE	0.984
MISSING:	---	95%	0.701	IQR	0.258
		Q3	0.262	STD	0.240
DISTINCT:	12,376 (79%)	AVG	0.160	VAR	0.057
		MEDIAN	0.010		
ZEROES:	---	Q1	0.003	KURT.	1.49
		5%	0.002	SKEW	1.56
		MIN	0.001	SUM	2,509



65 description_topic_6

VALUES:	15,696 (100%)	MAX	0.987	RANGE	0.985
MISSING:	---	95%	0.857	IQR	0.406
		Q3	0.410	STD	0.293
DISTINCT:	12,376 (79%)	AVG	0.222	VAR	0.086
		MEDIAN	0.020		
ZEROES:	---	Q1	0.004	KURT.	-0.021
		5%	0.003	SKEW	1.13
		MIN	0.001	SUM	3,484