Airbnb Price Prediction Analysis for Bozeman, Montana

Executive Summary

The goal of this project is to predict Airbnb listing prices in Bozeman, Montana, by identifying key factors influencing pricing and trends. By leveraging data from Airbnb's listings and reviews datasets, we developed models to extract insights and actionable recommendations for hosts, customers, and Airbnb itself.

Key findings include:

- **Significant Features:** Property size (bedrooms, bathrooms), amenities, and location are the strongest predictors of price.
- **Sentiment Analysis:** Positive reviews increase perceived value and pricing potential.
- **Model Performance:** Neural Network and Random Forest models outperformed others, achieving high accuracy while capturing complex relationships.

The findings provide valuable recommendations:

- **For Hosts:** Focus on enhancing high-impact features like amenities and positive guest experiences to justify higher prices.
- **For Airbnb:** Incorporate advanced pricing tools that factor in guest sentiment and geographic trends.
- **For Customers:** Use sentiment-driven insights to identify premium-value listings.

This analysis highlights the importance of integrating data-driven approaches in pricing strategies to maximize profitability and customer satisfaction.

Summary of the Data Exploration Process and Problem Statement

In the initial stages of the project, we obtained two key datasets—one containing Airbnb listings and another capturing user reviews. Our data exploration aimed to understand the overall structure, completeness, and distribution of the variables within these datasets, as well as identify patterns that could inform our business questions and modeling strategies.

Listings Dataset Exploration:

- The listings dataset originally comprised 554 rows and 75 columns, offering detailed information on host profiles, property characteristics, location data, and listing activity.
- Key variables included pricing, availability, number of reviews, and property attributes (e.g., the number of bedrooms and bathrooms). Some important fields, such as price and review_scores_rating, contained missing values—4.69% and 11.73% respectively—necessitating careful imputation strategies.
- Summary statistics revealed a median property offering around 2 bedrooms and accommodating roughly 4–5 guests, with a moderately high availability of 212 days per year. Reviews per month averaged around 2, indicating a steady stream of guest feedback and bookings.
- The distribution of minimum_nights and the presence of outliers in price suggested the need for data cleaning and potential feature transformation before modeling.

Reviews Dataset Exploration:

- The reviews dataset included 40,050 entries spanning 6 columns. This richer dataset of textual feedback provided valuable qualitative insights into guest satisfaction, property conditions, and areas of improvement.
- Apart from a few missing text entries, the comments column was largely complete. The presence of detailed, user-generated text allowed for advanced text mining techniques, sentiment analysis, and feature extraction.
- Converting the review dates into a proper datetime format enabled temporal analyses of guest feedback trends.

Problem Statement:

Our primary objective is to combine the structured (listings) and unstructured (reviews) data to gain holistic insights into what drives pricing, occupancy, and guest satisfaction. By understanding how amenities, location, host experience, and guest sentiment influence performance metrics such as nightly rates, booking frequency, and review ratings, we aim to:

- Identify Key Revenue Drivers: Determine which listing attributes (e.g., amenities, property type, location) and host factors correlate most strongly with higher prices and occupancy rates.
- Optimize Guest Experience: Use textual sentiment analysis of reviews to uncover common pain points, highlight valued features, and suggest improvements for hosts to enhance their listings.

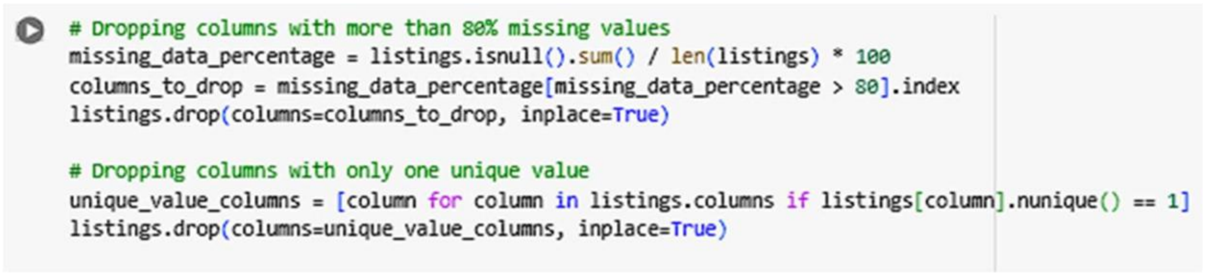
- Inform Business Strategies: Offer actionable recommendations to Airbnb, hosts, and prospective guests, such as ideal pricing strategies, targeted marketing efforts based on geographic clusters, and improvements to property features or services that align with guest preferences.
- In essence, the exploration phase guided us through data cleaning, feature engineering, and initial insights. This foundation sets the stage for subsequent modeling—price prediction, sentiment-driven recommendations, and improved decision-making for Airbnb and its stakeholders.

Data Cleaning and Feature Engineering for listing datasets

Data Cleaning & Standardization:

- Removed trailing spaces and irregular whitespace from text fields.
- Converted string values to title case for consistent categorization and better readability.
- Dropped columns with over 80% missing values and those containing only one unique value to reduce noise.
- Eliminated duplicate entries to ensure unique listings and maintain data integrity.

Sample Code:

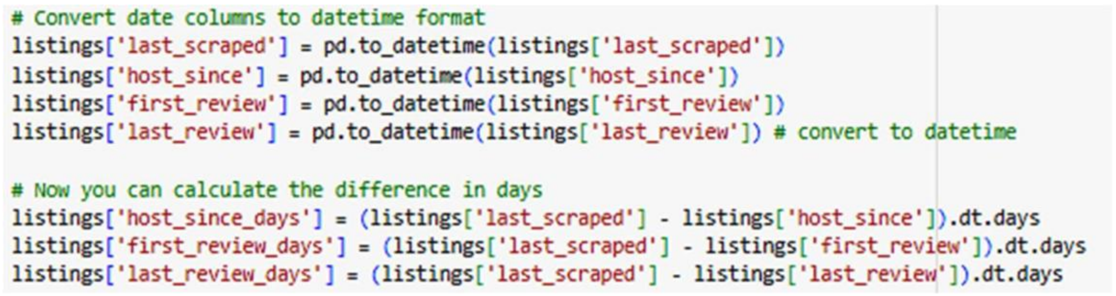


```
# Dropping columns with more than 80% missing values
missing_data_percentage = listings.isnull().sum() / len(listings) * 100
columns_to_drop = missing_data_percentage[missing_data_percentage > 80].index
listings.drop(columns=columns_to_drop, inplace=True)

# Dropping columns with only one unique value
unique_value_columns = [column for column in listings.columns if listings[column].nunique() == 1]
listings.drop(columns=unique_value_columns, inplace=True)
```

Datetime Conversion & Feature Derivation (Feature Engineering):

- Converted date-related fields (e.g., host_since, first_review, last_scraped) into proper datetime format.
- Engineered new time-based features (e.g., host_since_days, last_review_days) to capture temporal dynamics, such as host experience and property recency.

Sample Code:

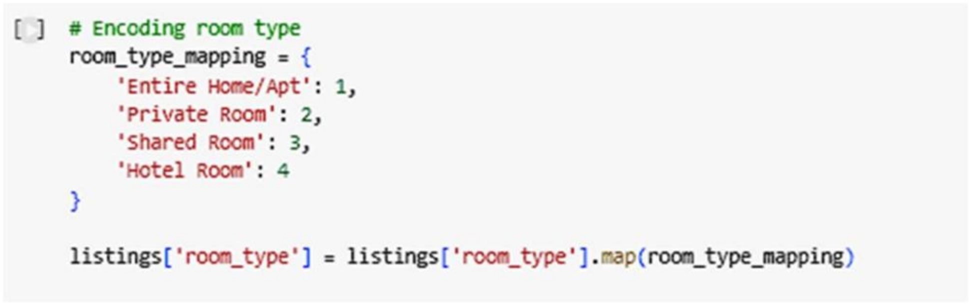


```
# Convert date columns to datetime format
listings['last_scraped'] = pd.to_datetime(listings['last_scraped'])
listings['host_since'] = pd.to_datetime(listings['host_since'])
listings['first_review'] = pd.to_datetime(listings['first_review'])
listings['last_review'] = pd.to_datetime(listings['last_review']) # convert to datetime

# Now you can calculate the difference in days
listings['host_since_days'] = (listings['last_scraped'] - listings['host_since']).dt.days
listings['first_review_days'] = (listings['last_scraped'] - listings['first_review']).dt.days
listings['last_review_days'] = (listings['last_scraped'] - listings['last_review']).dt.days
```

Categorical Encoding (Feature Engineering):

- Mapped room types, host response times, and property types into numeric or categorical codes.
- Simplified textual categories to facilitate modeling and statistical analysis.

Sample Code:

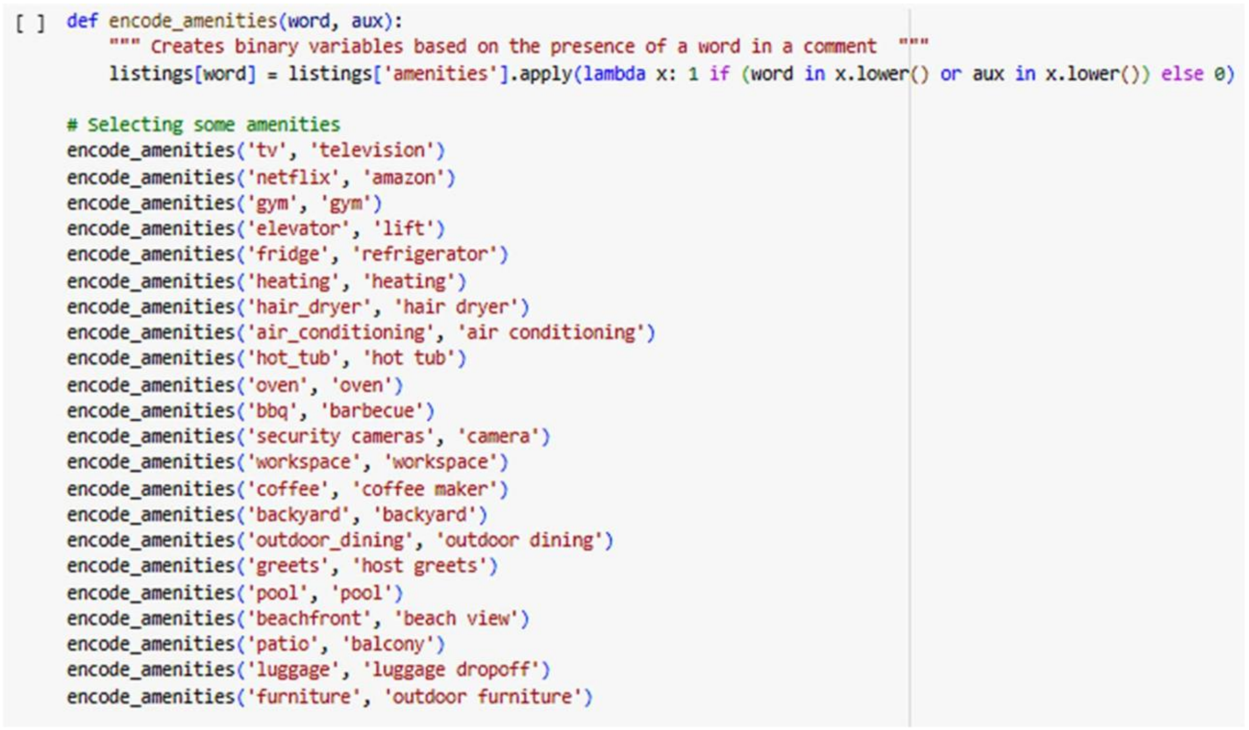


```
[ ]  # Encoding room type
     room_type_mapping = {
         'Entire Home/Apt': 1,
         'Private Room': 2,
         'Shared Room': 3,
         'Hotel Room': 4
     }

     listings['room_type'] = listings['room_type'].map(room_type_mapping)
```

Amenities Extraction & Binary Encoding (Feature Engineering):

- Parsed unstructured text of amenities into structured lists.
- Created binary indicator variables for key amenities (e.g., “Air Conditioning,” “Gym,” “Netflix”) to quantify their impact on pricing, occupancy, and guest satisfaction.

Sample Code:

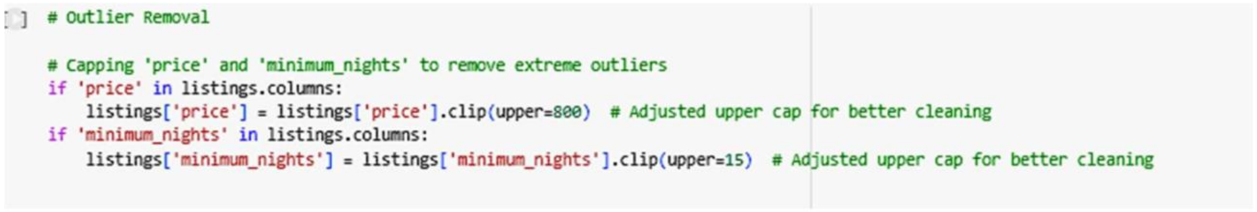


```
[ ]  def encode_amenities(word, aux):
         """ Creates binary variables based on the presence of a word in a comment  """
         listings[word] = listings['amenities'].apply(lambda x: 1 if (word in x.lower() or aux in x.lower()) else 0)

     # Selecting some amenities
     encode_amenities('tv', 'television')
     encode_amenities('netflix', 'amazon')
     encode_amenities('gym', 'gym')
     encode_amenities('elevator', 'lift')
     encode_amenities('fridge', 'refrigerator')
     encode_amenities('heating', 'heating')
     encode_amenities('hair_dryer', 'hair dryer')
     encode_amenities('air_conditioning', 'air conditioning')
     encode_amenities('hot_tub', 'hot tub')
     encode_amenities('oven', 'oven')
     encode_amenities('bbq', 'barbecue')
     encode_amenities('security cameras', 'camera')
     encode_amenities('workspace', 'workspace')
     encode_amenities('coffee', 'coffee maker')
     encode_amenities('backyard', 'backyard')
     encode_amenities('outdoor_dining', 'outdoor dining')
     encode_amenities('greets', 'host greets')
     encode_amenities('pool', 'pool')
     encode_amenities('beachfront', 'beach view')
     encode_amenities('patio', 'balcony')
     encode_amenities('luggage', 'luggage dropoff')
     encode_amenities('furniture', 'outdoor furniture')
```

Outlier & Missing Value Treatment:

- Capped extreme values in pricing and minimum nights to mitigate distortions from outliers.
- Imputed missing values using median or placeholder strategies to maintain dataset consistency.

Sample Code:

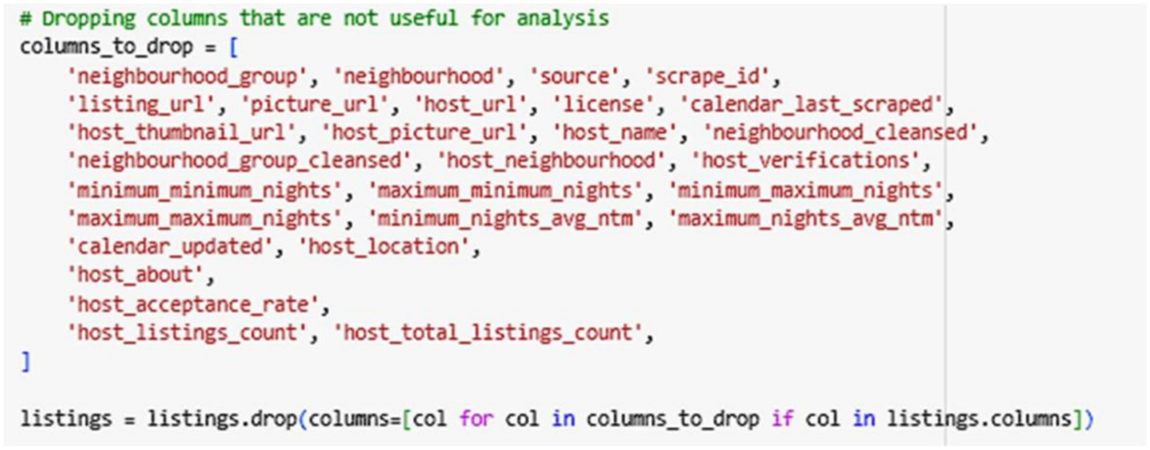


```
[ ]  # Outlier Removal

     # Capping 'price' and 'minimum_nights' to remove extreme outliers
     if 'price' in listings.columns:
         listings['price'] = listings['price'].clip(upper=800)  # Adjusted upper cap for better cleaning
     if 'minimum_nights' in listings.columns:
         listings['minimum_nights'] = listings['minimum_nights'].clip(upper=15)  # Adjusted upper cap for better cleaning
```

Non-Informative Columns Removal:

- Removed non-essential columns (e.g., URLs, IDs) and extraneous metadata that do not contribute to the analytical goals.

Sample Code:



```
# Dropping columns that are not useful for analysis
columns_to_drop = [
    'neighbourhood_group', 'neighbourhood', 'source', 'scrape_id',
    'listing_url', 'picture_url', 'host_url', 'license', 'calendar_last_scraped',
    'host_thumbnail_url', 'host_picture_url', 'host_name', 'neighbourhood_cleansed',
    'neighbourhood_group_cleansed', 'host_neighbourhood', 'host_verifications',
    'minimum_minimum_nights', 'maximum_minimum_nights', 'minimum_maximum_nights',
    'maximum_maximum_nights', 'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm',
    'calendar_updated', 'host_location',
    'host_about',
    'host_acceptance_rate',
    'host_listings_count', 'host_total_listings_count',
]

listings = listings.drop(columns=[col for col in columns_to_drop if col in listings.columns])
```

Geospatial Feature Engineering and Clustering

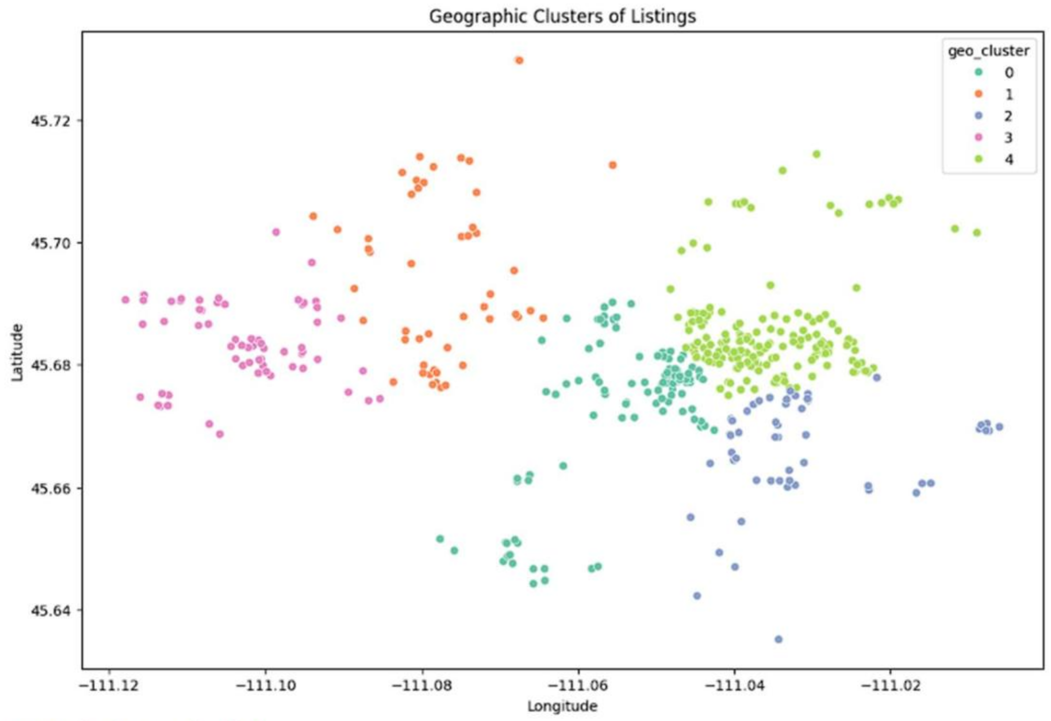
- Where geographic coordinates were available, we leveraged latitude and longitude fields to gain additional insights into the spatial distribution of listings. By extracting these two variables, we performed K-Means clustering to identify distinct geographic clusters within the city. This process grouped listings into several spatially coherent segments, which can help stakeholders understand location-based market dynamics.
- **Clustering Approach:**
    - Applied K-Means clustering on latitude and longitude to segment listings into five geographic clusters.
    - The resulting clusters reveal patterns such as property hotspots, emerging neighborhoods, and areas with potential for new listings.

- **Visualization & Insights:**
    - Generated a scatter plot illustrating each listing's cluster membership overlaid on longitude and latitude coordinates.

Sample Code:

```python
# Perform clustering if latitude and longitude exist
if 'latitude' in listings.columns and 'longitude' in listings.columns:
    geo_features = listings[['latitude', 'longitude']].dropna()

    # Perform K-Means clustering
    kmeans = KMeans(n_clusters=5, random_state=42).fit(geo_features)
    listings['geo_cluster'] = kmeans.labels_

    # Scatter plot with clusters
    plt.figure(figsize=(12, 8))
    sns.scatterplot(x='longitude', y='latitude', hue='geo_cluster', data=listings, palette='Set2')
    plt.title('Geographic Clusters of Listings')
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.show()
else:
    print("Error: 'latitude' or 'longitude' column is missing. Clustering cannot be performed.")
```

Output:



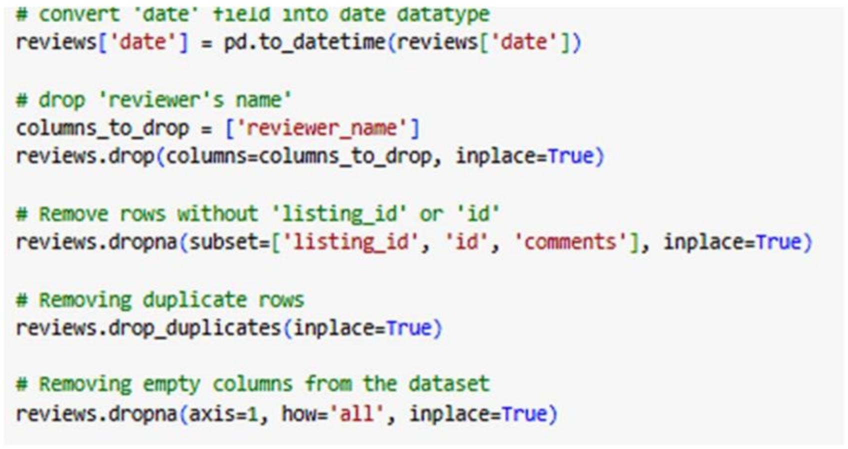
```
Average Price by Geographic Cluster:
 geo_cluster
0    217.793388
1    218.940000
2    226.018182
3    196.093750
4    251.673367
Name: price, dtype: float64
```

Data Cleaning and Sentimental Analysis for Review dataset

Basic Cleaning:

- Ensured proper date formatting by converting the date field into a datetime type.
- Removed unnecessary fields (e.g., reviewer_name) to streamline the dataset.
- Dropped rows without essential identifiers (e.g., missing listing_id or id) to maintain dataset integrity.
- Eliminated duplicate entries and empty columns to improve data quality.

Sample Code:

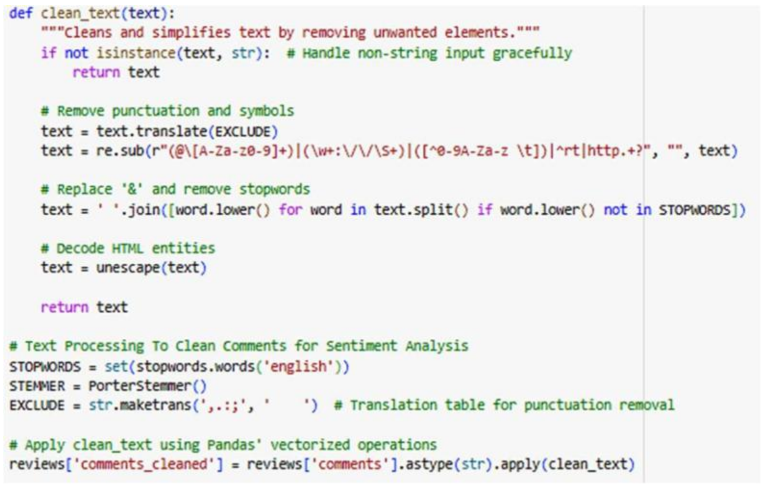


```
# convert 'date' field into date datatype
reviews['date'] = pd.to_datetime(reviews['date'])

# drop 'reviewer's name'
columns_to_drop = ['reviewer_name']
reviews.drop(columns=columns_to_drop, inplace=True)

# Remove rows without 'listing_id' or 'id'
reviews.dropna(subset=['listing_id', 'id', 'comments'], inplace=True)

# Removing duplicate rows
reviews.drop_duplicates(inplace=True)

# Removing empty columns from the dataset
reviews.dropna(axis=1, how='all', inplace=True)
```

Text Preprocessing & Normalization:

- Cleaned and standardized the textual data in the comments field:
- Removed punctuation, special symbols, and HTML entities.
- Converted all text to lower case for uniformity.
- Removed common English stopwords to highlight more meaningful terms.
- Applied stemming or lemmatization (if implemented) to reduce words to their root forms, enhancing text mining effectiveness.

Sample Code:



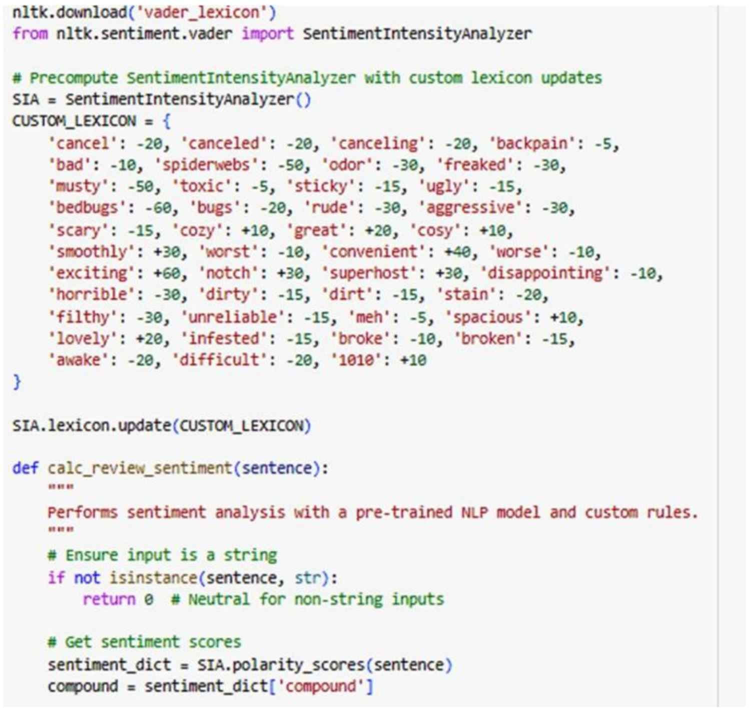
```
def clean_text(text):
    """Cleans and simplifies text by removing unwanted elements."""
    if not isinstance(text, str):  # Handle non-string input gracefully
        return text

    # Remove punctuation and symbols
    text = text.translate(EXCLUDE)
    text = re.sub(r"(@\[A-Za-z0-9]+)|(\w+:\/\/\S+)|([^0-9A-Za-z \t])|^rt|http.+?", "", text)

    # Replace '&' and remove stopwords
    text = ' '.join([word.lower() for word in text.split() if word.lower() not in STOPWORDS])

    # Decode HTML entities
    text = unescape(text)

    return text

# Text Processing To Clean Comments for Sentiment Analysis
STOPWORDS = set(stopwords.words('english'))
STEMMER = PorterStemmer()
EXCLUDE = str.maketrans(',.:;', '    ')  # Translation table for punctuation removal

# Apply clean_text using Pandas' vectorized operations
reviews['comments_cleaned'] = reviews['comments'].astype(str).apply(clean_text)
```

Sentiment Analysis Integration:

- Leveraged the VADER sentiment analysis tool to assess the emotional tone of guest reviews.

- Adjusted the sentiment lexicon with a custom dictionary to refine polarity scores and better reflect the hospitality context.
- Assigned sentiment scores to each review comment, enabling categorization into positive, negative, or neutral feedback.

Sample Code:



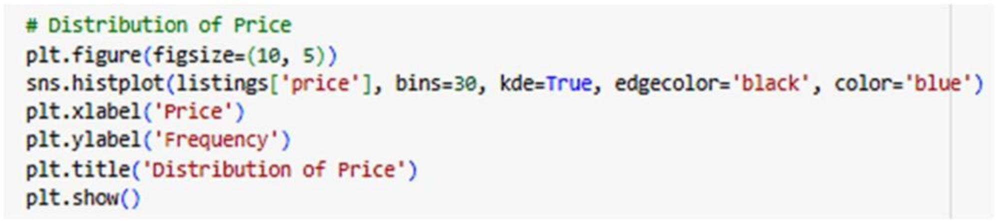
```python
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Precompute SentimentIntensityAnalyzer with custom lexicon updates
SIA = SentimentIntensityAnalyzer()
CUSTOM_LEXICON = {
    'cancel': -20, 'canceled': -20, 'canceling': -20, 'backpain': -5,
    'bad': -10, 'spiderwebs': -50, 'odor': -30, 'freaked': -30,
    'musty': -50, 'toxic': -5, 'sticky': -15, 'ugly': -15,
    'bedbugs': -60, 'bugs': -20, 'rude': -30, 'aggressive': -30,
    'scary': -15, 'cozy': +10, 'great': +20, 'cosy': +10,
    'smoothly': +30, 'worst': -10, 'convenient': +40, 'worse': -10,
    'exciting': +60, 'notch': +30, 'superhost': +30, 'disappointing': -10,
    'horrible': -30, 'dirty': -15, 'dirt': -15, 'stain': -20,
    'filthy': -30, 'unreliable': -15, 'meh': -5, 'spacious': +10,
    'lovely': +20, 'infested': -15, 'broke': -10, 'broken': -15,
    'awake': -20, 'difficult': -20, '1010': +10
}

SIA.lexicon.update(CUSTOM_LEXICON)

def calc_review_sentiment(sentence):
    """
    Performs sentiment analysis with a pre-trained NLP model and custom rules.
    """
    # Ensure input is a string
    if not isinstance(sentence, str):
        return 0  # Neutral for non-string inputs

    # Get sentiment scores
    sentiment_dict = SIA.polarity_scores(sentence)
    compound = sentiment_dict['compound']
```

Exploratory Data Analysis for Listing

Price Distribution and Characteristics

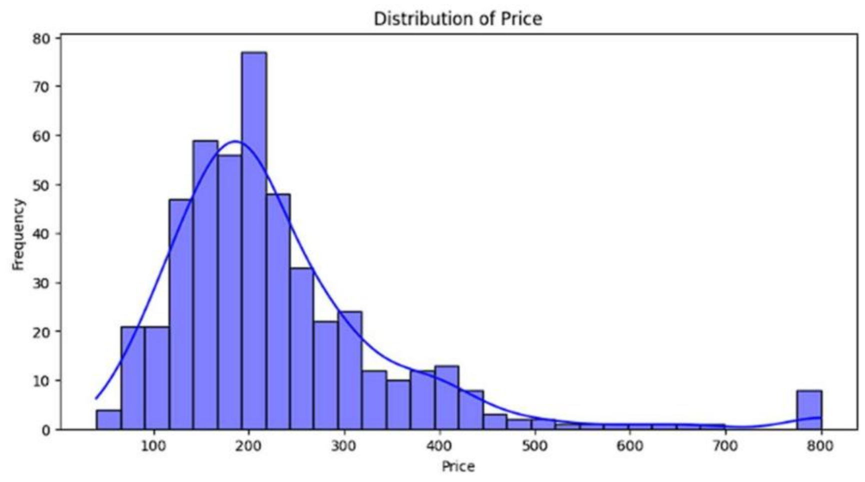
- Created a histogram of the price feature to visualize its frequency distribution and identify skewness or potential outliers.
- Observed a concentration of listings at certain price points, suggesting that many hosts set similar rates.
- Highlighted the need for outlier handling (e.g., capping extreme values) to improve subsequent modeling accuracy.

Sample Code:



```python
# Distribution of Price
plt.figure(figsize=(10, 5))
sns.histplot(listings['price'], bins=30, kde=True, edgecolor='black', color='blue')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Distribution of Price')
plt.show()
```

Output:



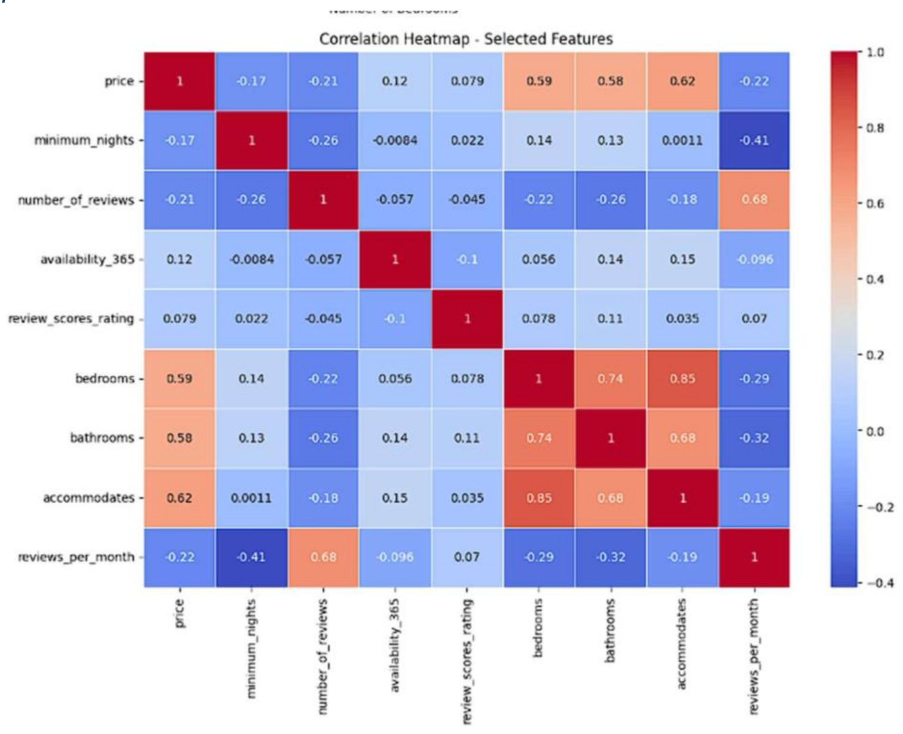
Property Features and Their Distributions

- Examined the distribution of bedrooms via a histogram, finding that most listings offer a modest number of rooms (e.g., 1–2 bedrooms).
- Generated a correlation heatmap of selected features (price, minimum_nights, number_of_reviews, availability_365, review_scores_rating, bedrooms, bathrooms, accommodates, reviews_per_month) to identify which variables have the strongest linear relationships with each other.
- Developed a scatter plot mapping longitude vs. latitude and coloring points by price, revealing geographic patterns in pricing. Higher-priced listings might cluster in certain neighborhoods or city centers.

Sample Code:

```
# Correlation Heatmap with Selected Features
selected_features = ['price', 'minimum_nights', 'number_of_reviews', 'availability_365',
                     'review_scores_rating', 'bedrooms', 'bathrooms', 'accommodates', 'reviews_per_month']
if 'neighbourhood' in listings.columns and listings['neighbourhood'].dtype != 'object': # Check if neighbourhood column exists and is numeric
    selected_features.append('neighbourhood')

corr_matrix = listings[selected_features].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap - Selected Features')
plt.show()
```

Output:



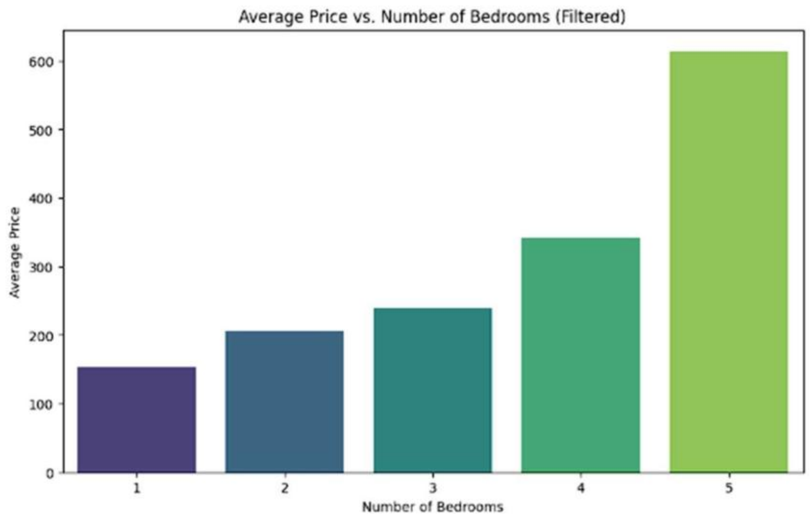
Relationship Between Pricing and Property Attributes

- Filtered listings by removing extreme outliers and plotted a bar chart to compare price vs. number_of_bedrooms, showing how larger accommodations may command higher prices but also noting diminishing returns if price does not proportionally increase with more bedrooms.
- Explored amenities by counting the number of features each listing offers. A histogram of amenities_count provided insight into the distribution and frequency of popular amenities, potentially signaling which extras justify higher prices or improved guest satisfaction.

Sample Code:

```
# Price vs. Number of Bedrooms (Filtered for Common Values and Outliers Removed per Category)
filtered_listings = listings[(listings['bedrooms'] > 0) & (listings['bedrooms'] <= 5)]
filtered_listings = filtered_listings.groupby('bedrooms', group_keys=False).apply(lambda x: x[x['price'] <= x['price'].quantile(0.90)])
plt.figure(figsize=(10, 6))
sns.barplot(x='bedrooms', y='price', data=filtered_listings, estimator=np.mean, errorbar=None, palette='viridis')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Average Price')
plt.title('Average Price vs. Number of Bedrooms (Filtered)')
plt.show()
```

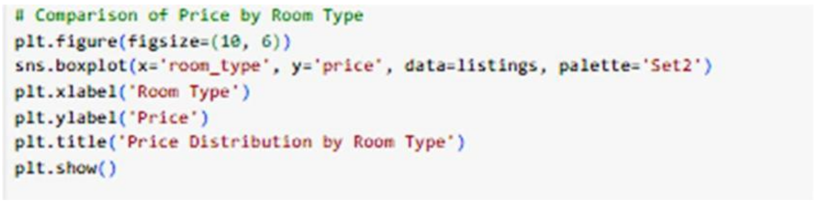
Output:



Categorical Comparisons and Price Segmentation

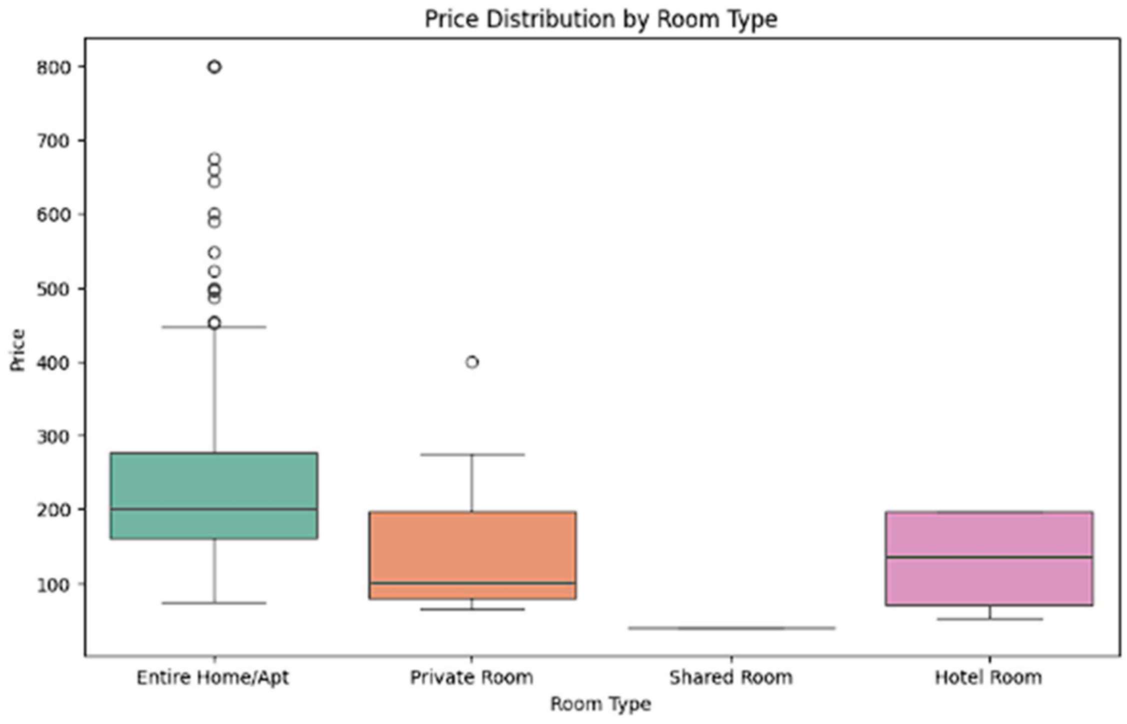
- Used boxplots to compare price distributions across different room_type categories (e.g., entire home, private room, shared room), illustrating how property types influence pricing tiers.
- Performed a similar boxplot analysis of price vs. accommodates (filtered to commonly occurring values). This helped determine whether there's a clear upward trend in pricing as the capacity for guests increases, informing strategies around listing configurations and pricing structures.

Sample Code:



```
# Comparison of Price by Room Type
plt.figure(figsize=(10, 6))
sns.boxplot(x='room_type', y='price', data=listings, palette='Set2')
plt.xlabel('Room Type')
plt.ylabel('Price')
plt.title('Price Distribution by Room Type')
plt.show()
```

Output:



Key Takeaways

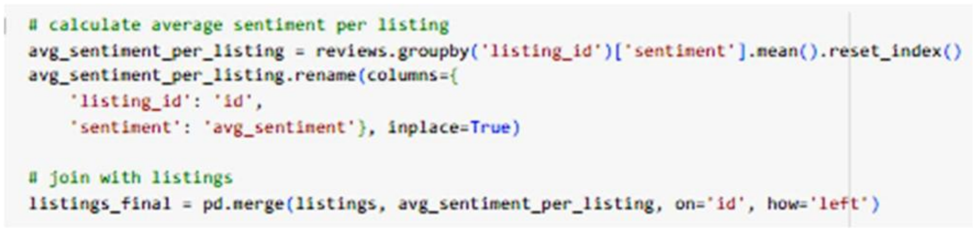
- The visualizations guided understanding of how property characteristics (bedrooms, accommodates), location, and amenities influence pricing and availability patterns.
- Correlations provided clues for feature selection in modeling, while geographic and categorical plots revealed patterns that might inform marketing or property improvement decisions.
- Together, these EDA steps build a foundation for more refined analyses, including pricing optimization, property recommendations, and tailored business strategies for hosts and Airbnb

Dataset Finalization

Integrating Sentiment Insights:

- Calculated the average sentiment score per listing from the reviews dataset.
- Merged this sentiment metric back into the listings dataset, enabling sentiment-informed price analysis and prediction.

Sample Code:

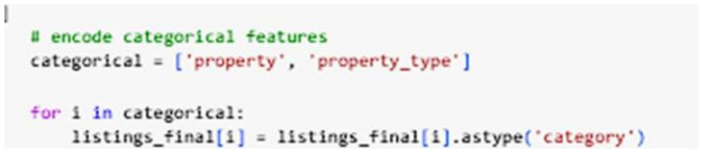


```
# calculate average sentiment per listing
avg_sentiment_per_listing = reviews.groupby('listing_id')['sentiment'].mean().reset_index()
avg_sentiment_per_listing.rename(columns={
    'listing_id': 'id',
    'sentiment': 'avg_sentiment'}, inplace=True)

# join with listings
listings_final = pd.merge(listings, avg_sentiment_per_listing, on='id', how='left')
```

Categorical Encoding & Transformations:

- Converted selected categorical fields (e.g., property, property_type) into categorical data types for more efficient analysis and modeling.
- Transformed boolean flags (e.g., host_has_profile_pic, host_is_superhost, host_identity_verified, instant_bookable) into binary (0/1) variables to facilitate direct inclusion in machine learning models.

Sample Code:

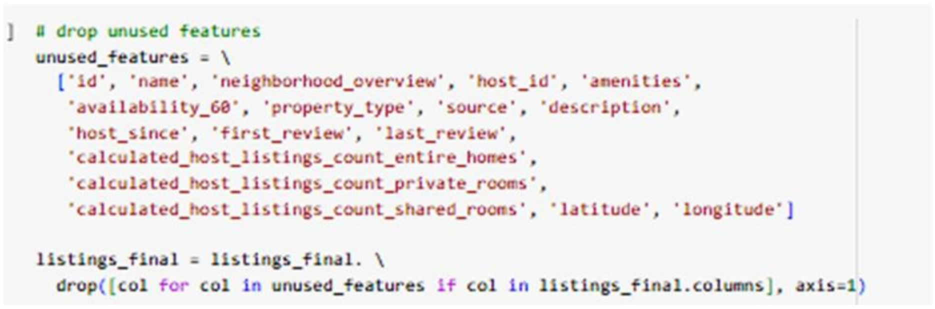


```
# encode categorical features
categorical = ['property', 'property_type']

for i in categorical:
    listings_final[i] = listings_final[i].astype('category')
```

Dropping Unnecessary Features:

- Removed fields that provided little analytical value or were redundant, such as high-level descriptions, metadata, or columns excessively missing data.
- Streamlined the feature set to focus on variables most relevant to price prediction, guest experience, and operational insights.

Sample Code:

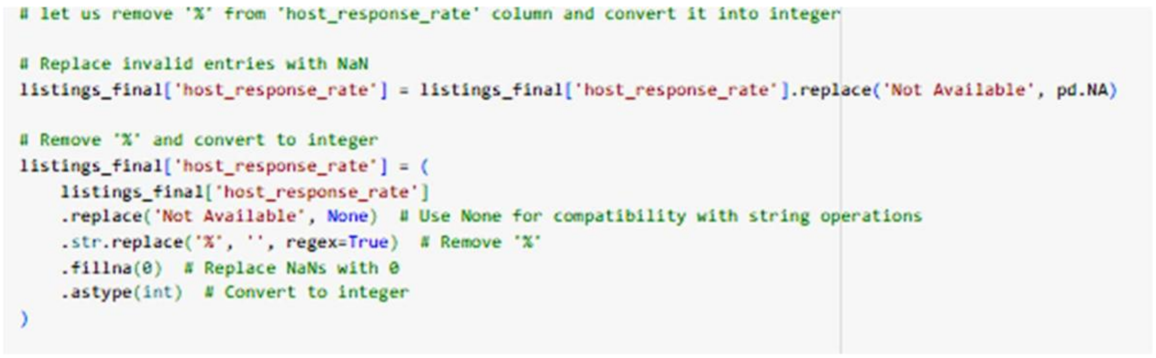


```
# drop unused features
unused_features = \
  ['id', 'name', 'neighborhood_overview', 'host_id', 'amenities',
   'availability_60', 'property_type', 'source', 'description',
   'host_since', 'first_review', 'last_review',
   'calculated_host_listings_count_entire_homes',
   'calculated_host_listings_count_private_rooms',
   'calculated_host_listings_count_shared_rooms', 'latitude', 'longitude']

listings_final = listings_final. \
  drop([col for col in unused_features if col in listings_final.columns], axis=1)
```

Data Cleaning & Type Conversion:

- Cleaned the host_response_rate column by removing extraneous characters (e.g., ‘%’, ‘X’) and replacing invalid values with NaN.
- Converted host_response_rate into a proper integer format, ensuring that host responsiveness can be analyzed numerically.

Sample Code:



```
# let us remove '%' from 'host_response_rate' column and convert it into integer

# Replace invalid entries with NaN
listings_final['host_response_rate'] = listings_final['host_response_rate'].replace('Not Available', pd.NA)

# Remove '%' and convert to integer
listings_final['host_response_rate'] = (
    listings_final['host_response_rate']
    .replace('Not Available', None)  # Use None for compatibility with string operations
    .str.replace('%', '', regex=True)  # Remove '%'
    .fillna(0)  # Replace NaNs with 0
    .astype(int)  # Convert to integer
)
```

Outcome:
After these final steps, the dataset is cleaner, more consistent, and enriched with sentiment data from guest reviews. The processed feature set now offers a balanced mix of quantitative, categorical, and sentiment-driven insights, setting the stage for effective modeling and more actionable recommendations for Airbnb and its stakeholders.

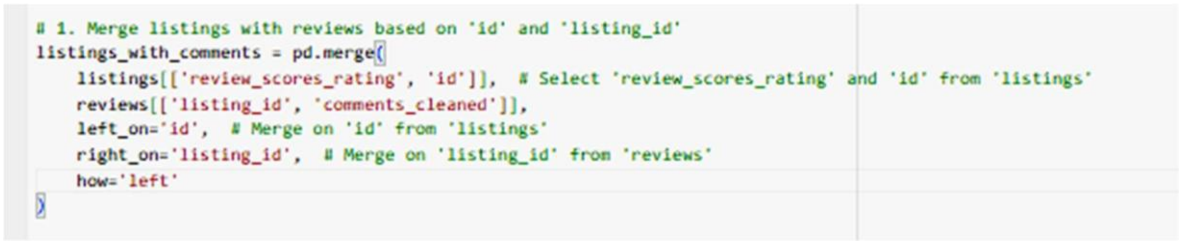
Textual Insights and Sentiment Exploration

After integrating the cleaned and preprocessed review data with the listings information, we conducted a deeper analysis of guest feedback to uncover patterns in what distinguishes higher-rated listings from lower-rated ones.

Merging Listings and Reviews:

- Combined the listings dataset with its corresponding review comments using the listing and review IDs, ensuring that each property's textual feedback could be analyzed in context.

Sample Code:

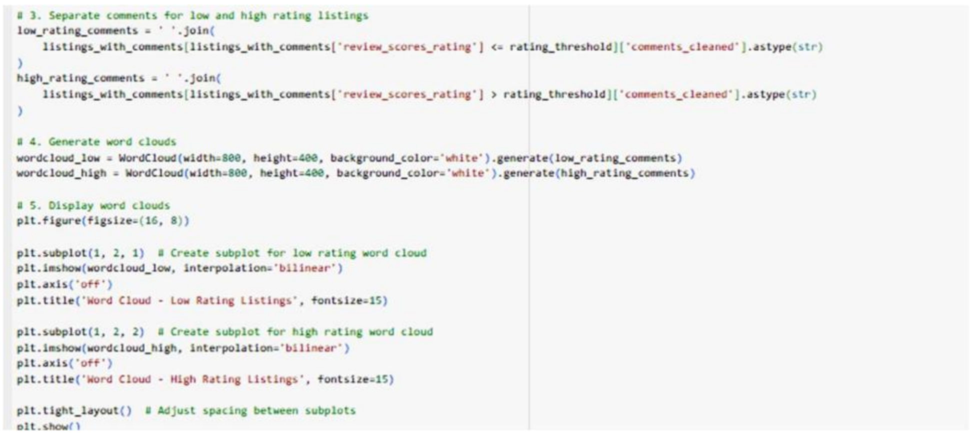


```
# 1. Merge listings with reviews based on 'id' and 'listing_id'
listings_with_comments = pd.merge(
    listings[['review_scores_rating', 'id']],  # Select 'review_scores_rating' and 'id' from 'listings'
    reviews[['listing_id', 'comments_cleaned']],
    left_on='id',  # Merge on 'id' from 'listings'
    right_on='listing_id',  # Merge on 'listing_id' from 'reviews'
    how='left'
)
```

Sentiment Threshold and Segmentation:

- Used the median review scores rating to split the dataset into two groups: “high rating listings” and “low rating listings.”
- This stratification allowed us to isolate the textual differences between listings that consistently delighted guests and those that fell short of expectations.

Sample Code:

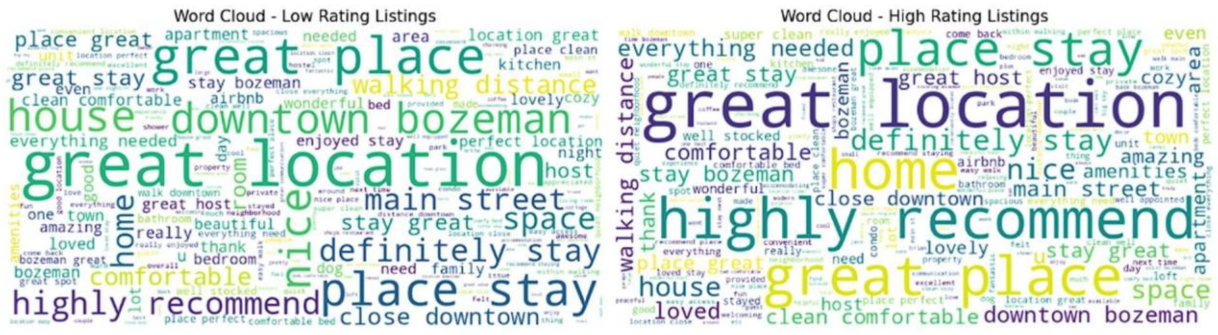


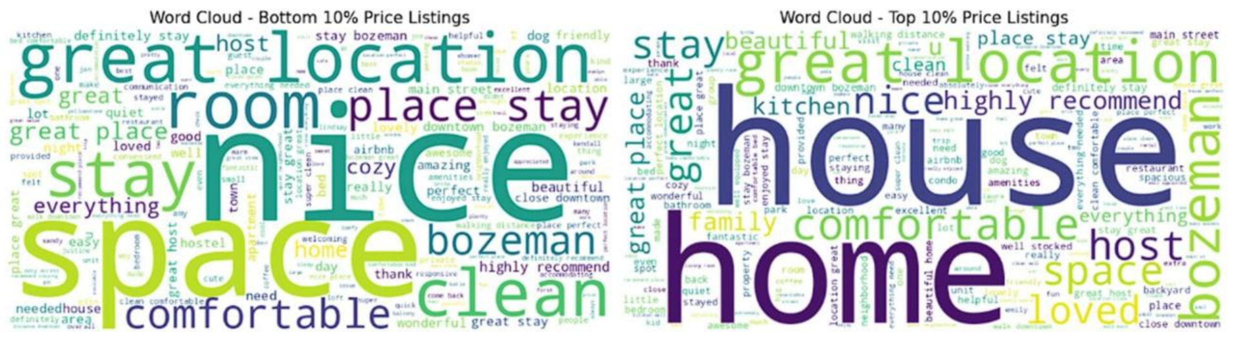
```
# 3. Separate comments for low and high rating listings
low_rating_comments = ' '.join(
    listings_with_comments[listings_with_comments['review_scores_rating'] <= rating_threshold]['comments_cleaned'].astype(str)
)
high_rating_comments = ' '.join(
    listings_with_comments[listings_with_comments['review_scores_rating'] > rating_threshold]['comments_cleaned'].astype(str)
)

# 4. Generate word clouds
wordcloud_low = WordCloud(width=800, height=400, background_color='white').generate(low_rating_comments)
wordcloud_high = WordCloud(width=800, height=400, background_color='white').generate(high_rating_comments)

# 5. Display word clouds
plt.figure(figsize=(16, 8))

plt.subplot(1, 2, 1)  # Create subplot for low rating word cloud
plt.imshow(wordcloud_low, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud - Low Rating Listings', fontsize=15)

plt.subplot(1, 2, 2)  # Create subplot for high rating word cloud
plt.imshow(wordcloud_high, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud - High Rating Listings', fontsize=15)

plt.tight_layout()  # Adjust spacing between subplots
plt.show()
```

Word Cloud Visualization:

- Generated two separate word clouds from the combined text of low-rated and high-rated listing comments.
- The low rating word cloud revealed terms associated with negative experiences, unmet expectations, or missing amenities.
- The high rating word cloud highlighted attributes guests appreciate, such as hospitality, location highlights, cleanliness, and unique amenities.

Wordclouds:





Price Prediction Model

1. Collinearity Mitigation Based on Correlation Matrix

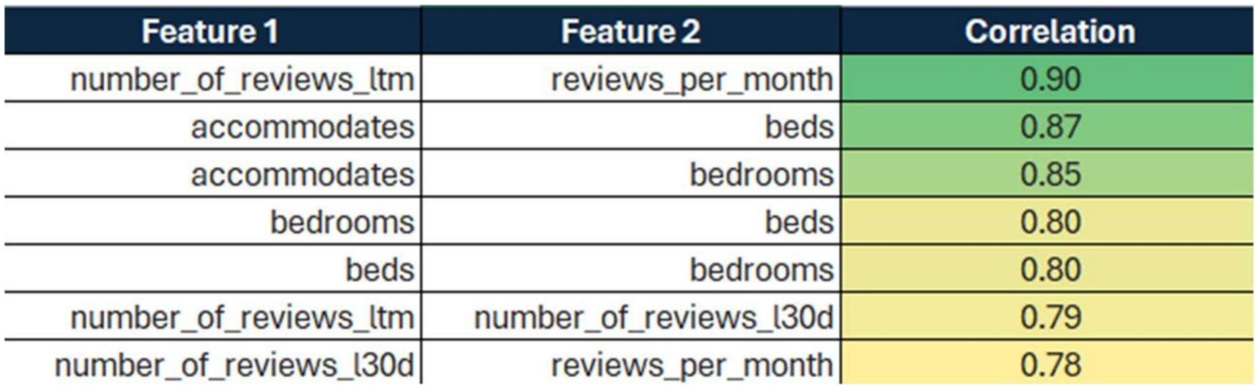
Objective:

Correlated features can lead to:

- **Unstable coefficient estimates** in linear models.
- **Misleading feature importance scores** in tree-based models

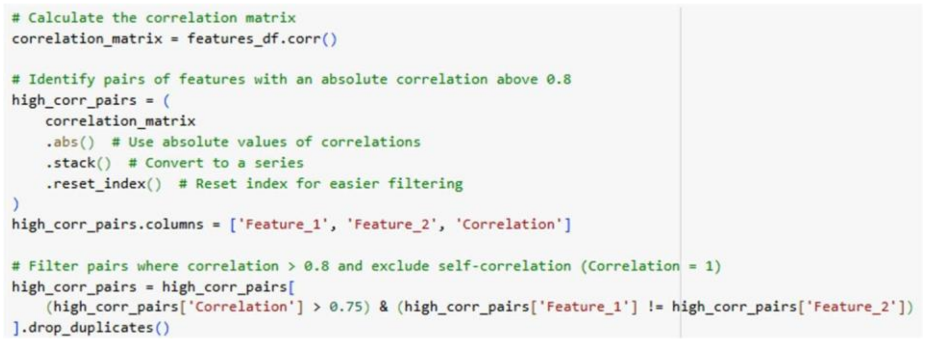
Process:

- Pairs of features with correlation values above the threshold are identified.
- From each pair, one feature is removed to eliminate redundancy.
- An absolute correlation threshold of **0.75** was used to detect and handle multicollinearity



Feature 1	Feature 2	Correlation
number_of_reviews_ltm	reviews_per_month	0.90
accommodates	beds	0.87
accommodates	bedrooms	0.85
bedrooms	beds	0.80
beds	bedrooms	0.80
number_of_reviews_ltm	number_of_reviews_l30d	0.79
number_of_reviews_l30d	reviews_per_month	0.78

Sample Code:



```
# Calculate the correlation matrix
correlation_matrix = features_df.corr()

# Identify pairs of features with an absolute correlation above 0.8
high_corr_pairs = (
    correlation_matrix
    .abs()  # Use absolute values of correlations
    .stack()  # Convert to a series
    .reset_index()  # Reset index for easier filtering
)
high_corr_pairs.columns = ['Feature_1', 'Feature_2', 'Correlation']

# Filter pairs where correlation > 0.8 and exclude self-correlation (Correlation = 1)
high_corr_pairs = high_corr_pairs[
    (high_corr_pairs['Correlation'] > 0.75) & (high_corr_pairs['Feature_1'] != high_corr_pairs['Feature_2'])
].drop_duplicates()
```

2. Collinearity Mitigation Based on VIFs

Objective:

- Features that can be expressed as a **linear combination** of other features introduce redundancy and instability in estimates.
- Variance Inflation Factor (VIF) is used to quantify multicollinearity.

VIF Formula:

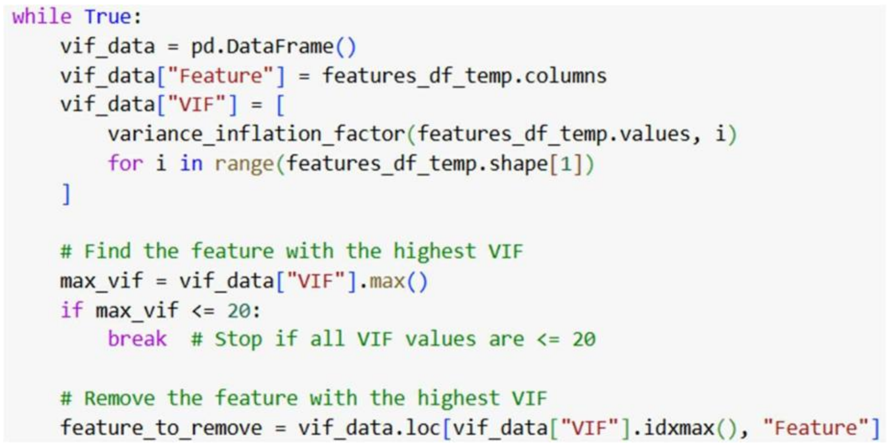
$$VIF(X_i) = \frac{1}{1-R_i^2}$$

- Here, ***R_i^2*** is the R^2 value obtained by regressing feature ***X_i*** on all other features.

Procedure:

1. Calculate the VIF for all features.
2. Identify the feature with the **highest VIF**.
3. If the highest VIF is greater than **20**, remove the corresponding feature.
4. Repeat until all remaining features have VIF values below or equal to 20.

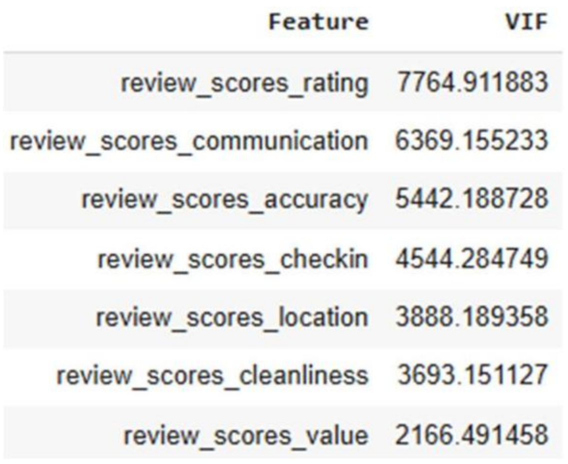
Sample Code:



```python
while True:
    vif_data = pd.DataFrame()
    vif_data["Feature"] = features_df_temp.columns
    vif_data["VIF"] = [
        variance_inflation_factor(features_df_temp.values, i)
        for i in range(features_df_temp.shape[1])
    ]

    # Find the feature with the highest VIF
    max_vif = vif_data["VIF"].max()
    if max_vif <= 20:
        break  # Stop if all VIF values are <= 20

    # Remove the feature with the highest VIF
    feature_to_remove = vif_data.loc[vif_data["VIF"].idxmax(), "Feature"]
```

Sample Features

The following table shows a sample list of features that are severely affected by the collinearity.



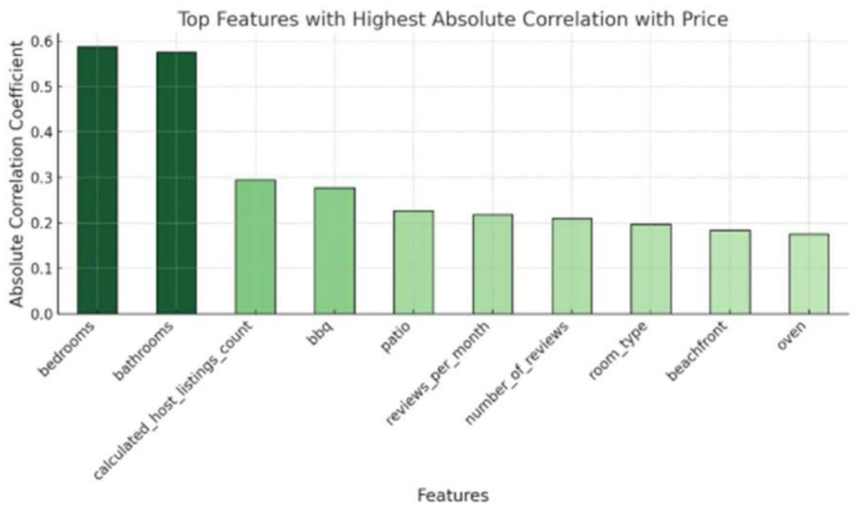
Feature	VIF
review_scores_rating	7764.911883
review_scores_communication	6369.155233
review_scores_accuracy	5442.188728
review_scores_checkin	4544.284749
review_scores_location	3888.189358
review_scores_cleanliness	3693.151127
review_scores_value	2166.491458

Key Takeaways:

- Both correlation matrix and VIF-based methods are critical for reducing multicollinearity, which enhances model stability and interpretability.
- The correlation matrix is useful for pairwise feature analysis, while VIF provides a broader view of linear dependencies across multiple features.

3. Correlation with the Target variable

Before we fit any model, it is a good idea to look at correlation between features and the target variable to assess the strength of linear association.
For simplicity, let's look only at the top correlated variables.

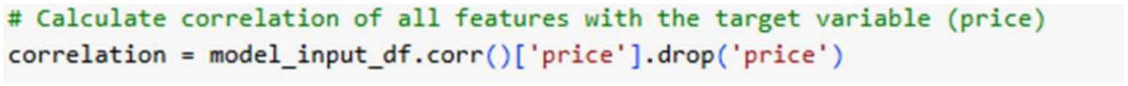


Key Takeaways:

1. **Bedrooms:**
    - More bedrooms typically result in higher prices due to increased capacity and value.
2. **Bathrooms:**
    - Properties with more bathrooms are often priced higher, reflecting their convenience and luxury.

2. **Calculated Host Listings Count:**
    - Professional hosts with more listings might adopt pricing strategies that maximize revenue.
2. **Amenities:**
    - Having BBQs, patio, beachfront are positively associated with the price

Sample Code:



```
# Calculate correlation of all features with the target variable (price)
correlation = model_input_df.corr()['price'].drop('price')
```

Model Build

Our approach is to build multiple models starting with simple base linear regression models to complex neural networks to extract strength from each model and to evaluate performance across them.
Building multiple models allows for a balance between **interpretability**, **predictive power**, and **complexity**. By starting with simpler models and progressing to more complex ones, you can:

- Identify the best-performing model for your dataset.
- Gain a deeper understanding of feature impacts.
- Ensure robust, accurate, and actionable predictions tailored to different business needs.

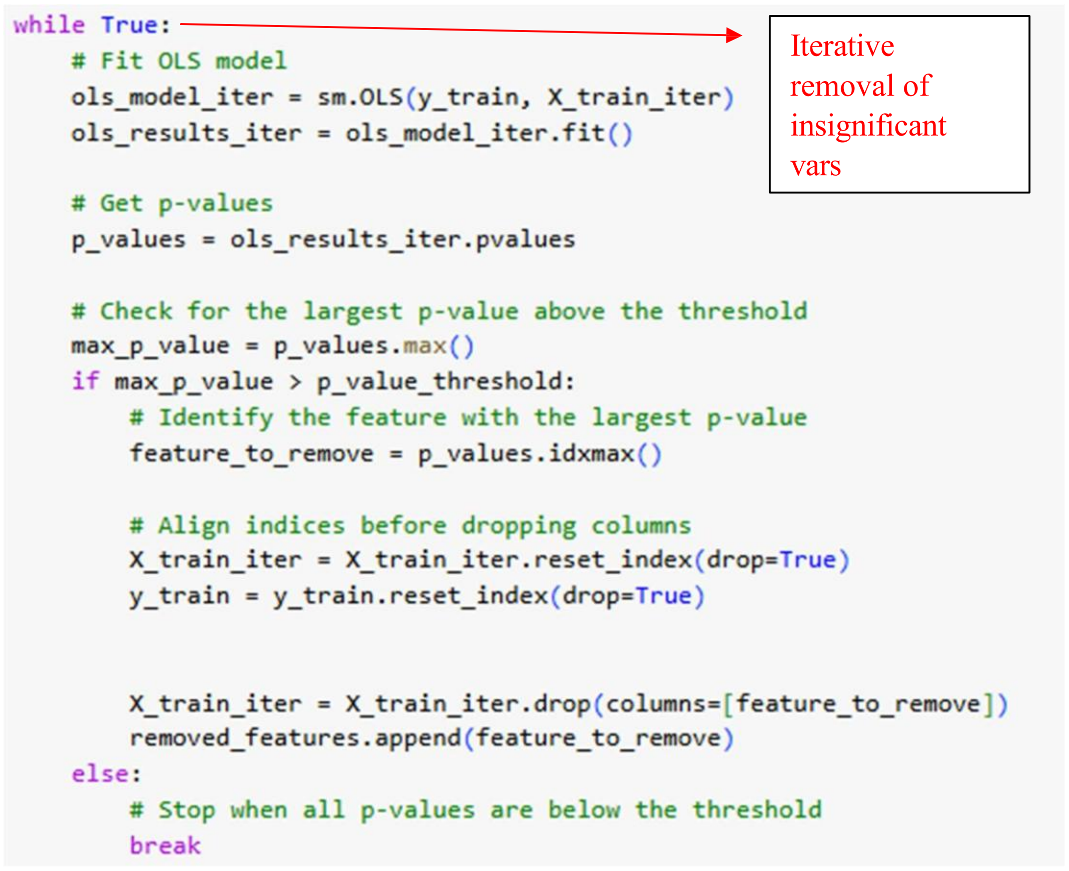
Model 1 – Linear Regression

Objective

- Assume a linear relationship between features and the target.
- Suitable as a baseline model to provide simple, interpretable insights.
- Can quickly reveal key predictors and the direction of their impact.

Variable Selection Approach:

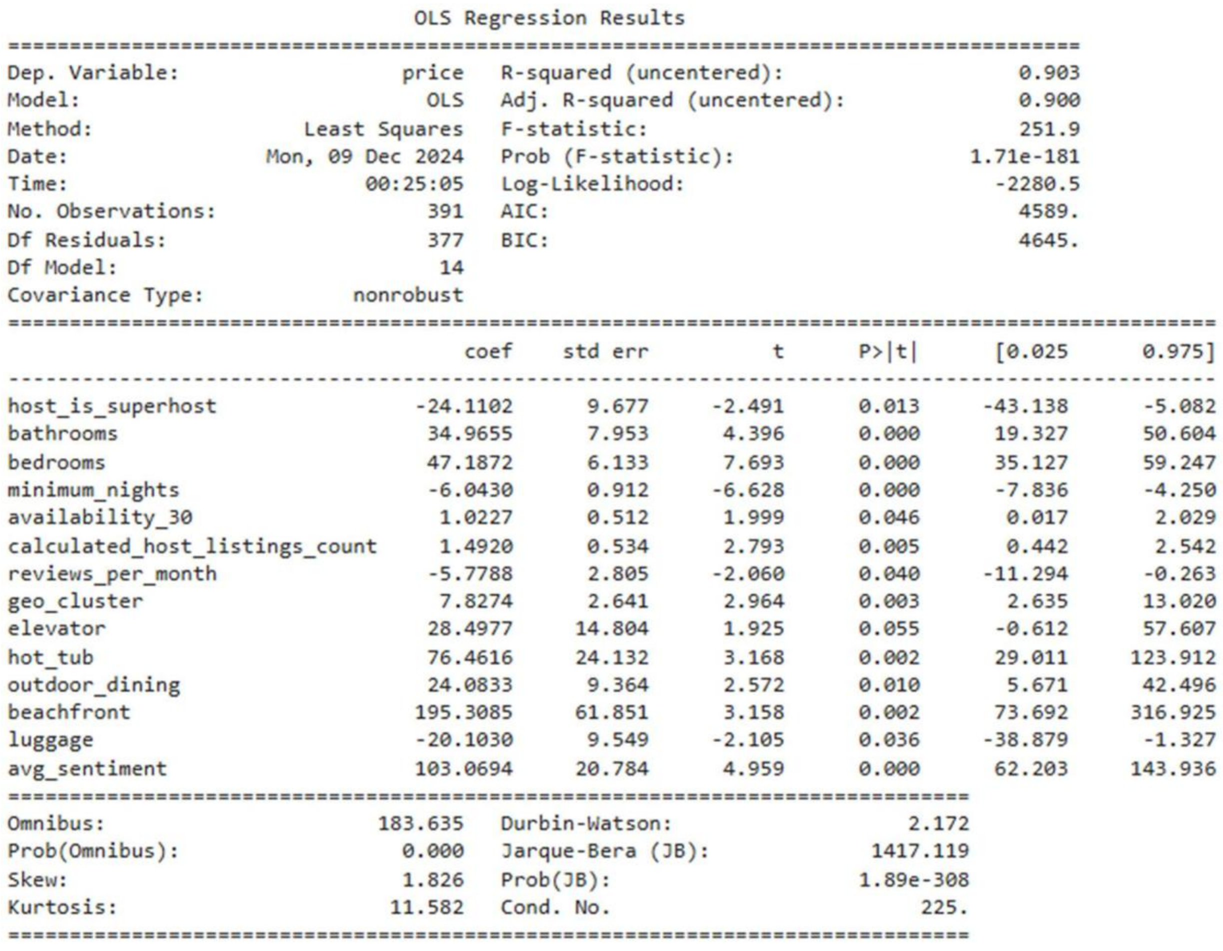
Features were selected iteratively based on statistical significance (p-value < 0.1). Insignificant variables were removed step-by-step.



```
while True:
    # Fit OLS model
    ols_model_iter = sm.OLS(y_train, X_train_iter)
    ols_results_iter = ols_model_iter.fit()

    # Get p-values
    p_values = ols_results_iter.pvalues

    # Check for the largest p-value above the threshold
    max_p_value = p_values.max()
    if max_p_value > p_value_threshold:
        # Identify the feature with the largest p-value
        feature_to_remove = p_values.idxmax()

        # Align indices before dropping columns
        X_train_iter = X_train_iter.reset_index(drop=True)
        y_train = y_train.reset_index(drop=True)


        X_train_iter = X_train_iter.drop(columns=[feature_to_remove])
        removed_features.append(feature_to_remove)
    else:
        # Stop when all p-values are below the threshold
        break
```

Iterative removal of insignificant vars

Snapshot of the model results



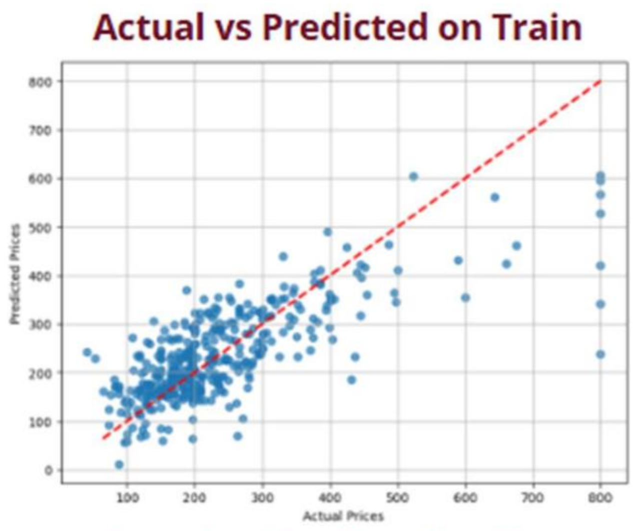
```
                            OLS Regression Results
=======================================================================================
Dep. Variable:                  price   R-squared (uncentered):                   0.903
Model:                            OLS   Adj. R-squared (uncentered):              0.900
Method:                 Least Squares   F-statistic:                              251.9
Date:                Mon, 09 Dec 2024   Prob (F-statistic):                   1.71e-181
Time:                        00:25:05   Log-Likelihood:                         -2280.5
No. Observations:                 391   AIC:                                      4589.
Df Residuals:                     377   BIC:                                      4645.
Df Model:                          14
Covariance Type:            nonrobust
==================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------
host_is_superhost              -24.1102      9.677     -2.491      0.013     -43.138      -5.082
bathrooms                       34.9655      7.953      4.396      0.000      19.327      50.604
bedrooms                        47.1872      6.133      7.693      0.000      35.127      59.247
minimum_nights                  -6.0430      0.912     -6.628      0.000      -7.836      -4.250
availability_30                  1.0227      0.512      1.999      0.046       0.017       2.029
calculated_host_listings_count   1.4920      0.534      2.793      0.005       0.442       2.542
reviews_per_month               -5.7788      2.805     -2.060      0.040     -11.294      -0.263
geo_cluster                      7.8274      2.641      2.964      0.003       2.635      13.020
elevator                        28.4977     14.804      1.925      0.055      -0.612      57.607
hot_tub                         76.4616     24.132      3.168      0.002      29.011     123.912
outdoor_dining                  24.0833      9.364      2.572      0.010       5.671      42.496
beachfront                     195.3085     61.851      3.158      0.002      73.692     316.925
luggage                        -20.1030      9.549     -2.105      0.036     -38.879      -1.327
avg_sentiment                  103.0694     20.784      4.959      0.000      62.203     143.936
==============================================================================
Omnibus:                      183.635   Durbin-Watson:                   2.172
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1417.119
Skew:                           1.826   Prob(JB):                    1.89e-308
Kurtosis:                      11.582   Cond. No.                         225.
==============================================================================
```

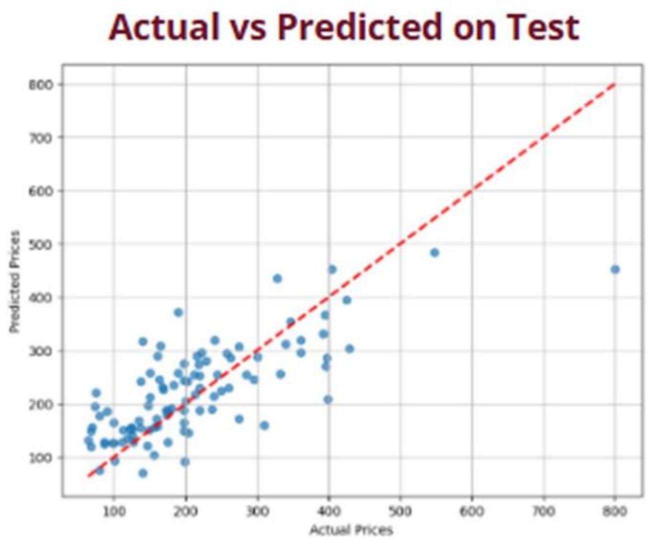
Model Performance:

- Adjusted R2 on the **training set**: **0.90**.
- Adjusted R2 on the **test set**: **0.55**.
- The model performs well on the training data but has reduced generalization capacity on unseen (test) data, likely due to overfitting.

Actual vs Predicted Plot:

Let’s look at how well the model has predicted the price on both training and the test datasets.





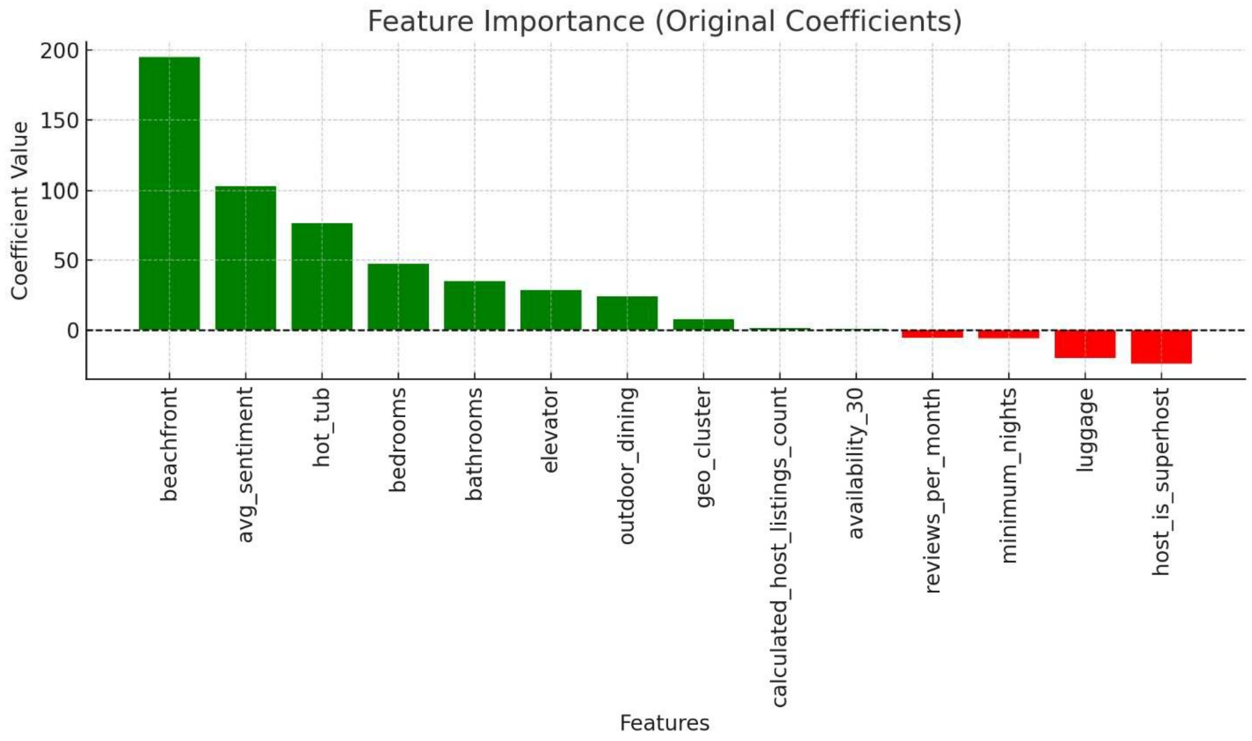
The **"Actual vs Predicted on Train"** graph displays a strong correlation between actual prices (x-axis) and predicted prices (y-axis):

- Most points align closely with the red diagonal line, which represents the ideal scenario where predictions match actual values perfectly.
- Indicates that the model performs well on the training data, with minimal error or deviation.

The **"Actual vs Predicted on Test"** graph shows a weaker alignment between actual and predicted values:

- While there is some correlation, a significant number of points deviate from the red diagonal line.
- There is greater dispersion of points, particularly at higher price ranges, suggesting the model struggles to predict accurately for extreme values.

Model Interpretations



1. **Top Features (Positive Coefficients)**:
    - **Beachfront**:
        - The most impactful feature, with a coefficient value close to 200.
        - Properties with beachfront locations significantly boost prices due to their premium appeal.
    - **Average Sentiment**:
        - Reflects customer sentiment (e.g., reviews), indicating higher prices for properties with positive customer feedback.
    - **Hot Tub**:
        - A popular amenity that adds luxury, positively influencing pricing.
    - **Bedrooms and Bathrooms**:
        - Indicate property size and capacity. These are critical for pricing, as larger properties accommodate more guests and offer greater comfort.
2. **Mid-Tier Features**:
    - **Elevator, Outdoor Dining, and Geo Cluster**:
        - Elevator and outdoor dining enhance convenience and luxury, appealing to higher-paying guests.
        - Geo cluster captures regional variations in pricing, reflecting the influence of location.
3. **Negatively Impactful Features**:
    - **Host is Superhost**:
        - Associated with a slight decrease in price. This could indicate that superhosts often price competitively to maintain high occupancy rates.
    - **Luggage**:
        - May reflect properties offering less luxurious or budget-friendly amenities.

- **Minimum Nights**:
  - Higher minimum night requirements negatively impact pricing, as it may deter short-term guests.
- **Reviews per Month**:
  - Frequent reviews may not always correspond to higher prices, possibly reflecting smaller or budget properties.

---

3. Interpretations

- **High-Impact Features**:
  - Amenities like **beachfront**, **hot tub**, and **bedrooms** strongly influence price and should be emphasized in pricing models.
- **Negative Influences**:
  - Features like **minimum nights** and **superhost status** highlight the nuanced trade-offs in pricing strategies.

Business Recommendations

1. **For Hosts**:
   - Focus on upgrading high-impact amenities like adding a hot tub or emphasizing beachfront appeal.
   - Adjust pricing to reflect positive customer feedback and sentiment.
   - Reassess the competitive pricing strategy for superhost properties to maximize value.
2. **For Airbnb**:
   - Incorporate features like sentiment analysis into pricing tools to refine recommendations.
   - Highlight regional price variations (geo clusters) to support dynamic pricing strategies.
3. **For Customers**:
   - Use filters for high-impact amenities (e.g., beachfront or hot tub) to identify properties offering premium experiences.

Model 2 – Random Forest Regressor

Objective

The Random Forest Regressor was chosen due to its ability to handle:

- **Nonlinear relationships**: Captures complex interactions between features and the target variable without requiring explicit modeling.
- **Feature interactions**: Automatically identifies and incorporates interactions without manual intervention.
- **Multicollinearity**: Less sensitive to correlated features, making it robust for datasets where multicollinearity is a concern.

Parameter Selection – Hyperparameter Optimization

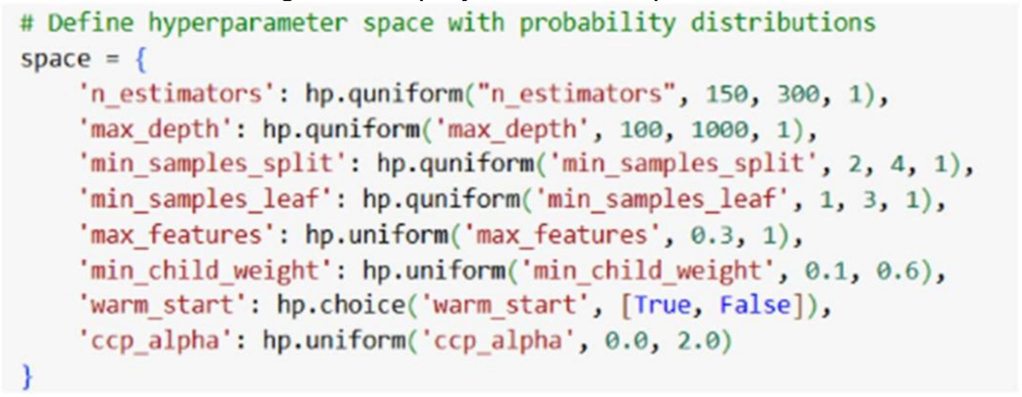
Hyperparameters Optimized:

- n_estimators: Number of decision trees in the forest, ranging from 150 to 300.
- max_depth: Maximum depth of each tree, ranging from 100 to 1000, to control complexity.

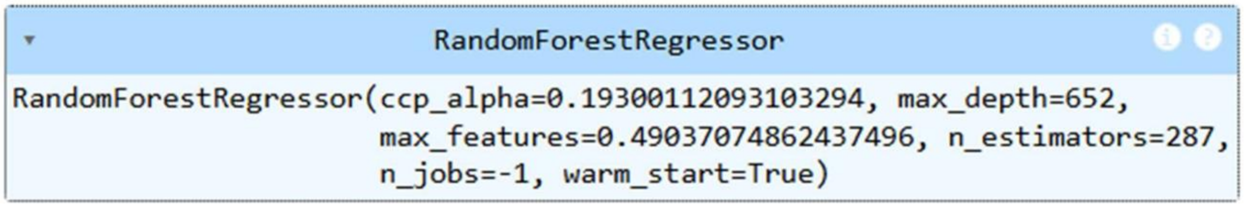
- min_samples_split: Minimum number of samples required to split a node (values between 2 and 4).
- min_samples_leaf: Minimum number of samples required at leaf nodes, ranging from 1 to 3.
- max_features: Number of features to consider at each split, ranging between 0.3 and 1.
- ccp_alpha: Complexity parameter for minimal cost complexity pruning (values between 0 and 2).

Implementation:

A probabilistic hyperparameter search was conducted to identify optimal configurations. The final model settings are displayed in the output block, such as:



```
# Define hyperparameter space with probability distributions
space = {
    'n_estimators': hp.quniform("n_estimators", 150, 300, 1),
    'max_depth': hp.quniform('max_depth', 100, 1000, 1),
    'min_samples_split': hp.quniform('min_samples_split', 2, 4, 1),
    'min_samples_leaf': hp.quniform('min_samples_leaf', 1, 3, 1),
    'max_features': hp.uniform('max_features', 0.3, 1),
    'min_child_weight': hp.uniform('min_child_weight', 0.1, 0.6),
    'warm_start': hp.choice('warm_start', [True, False]),
    'ccp_alpha': hp.uniform('ccp_alpha', 0.0, 2.0)
}
```



```
RandomForestRegressor

RandomForestRegressor(ccp_alpha=0.19300112093103294, max_depth=652,
                      max_features=0.49037074862437496, n_estimators=287,
                      n_jobs=-1, warm_start=True)
```

- max_depth = 652
- n_estimators = 287
- ccp_alpha = 0.19

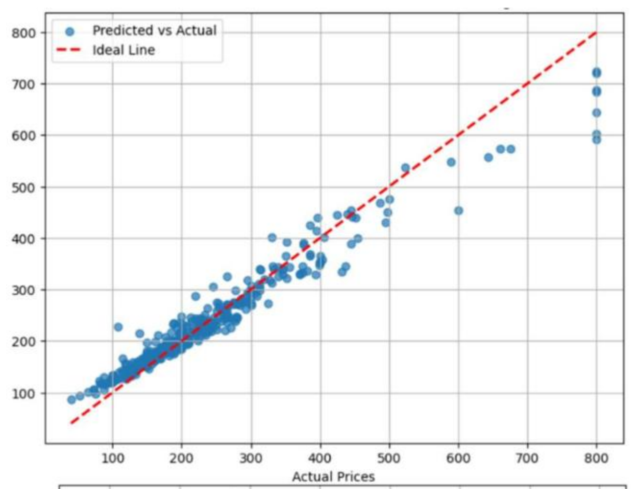
Model Performance

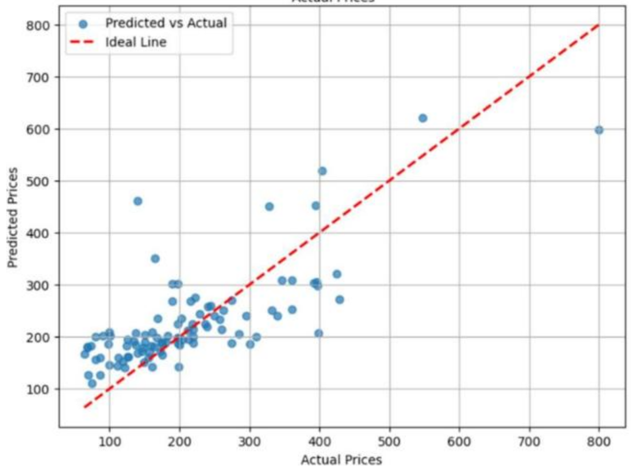
- **Training Data**:
  - R2: **0.93**
  - The model performs exceptionally well on the training data, indicating it captures most patterns and variance.
- **Test Data**:
  - R2: **0.54**
  - A noticeable drop in performance on unseen data suggests potential overfitting.

Actual vs Predicted Plots

On Training Data

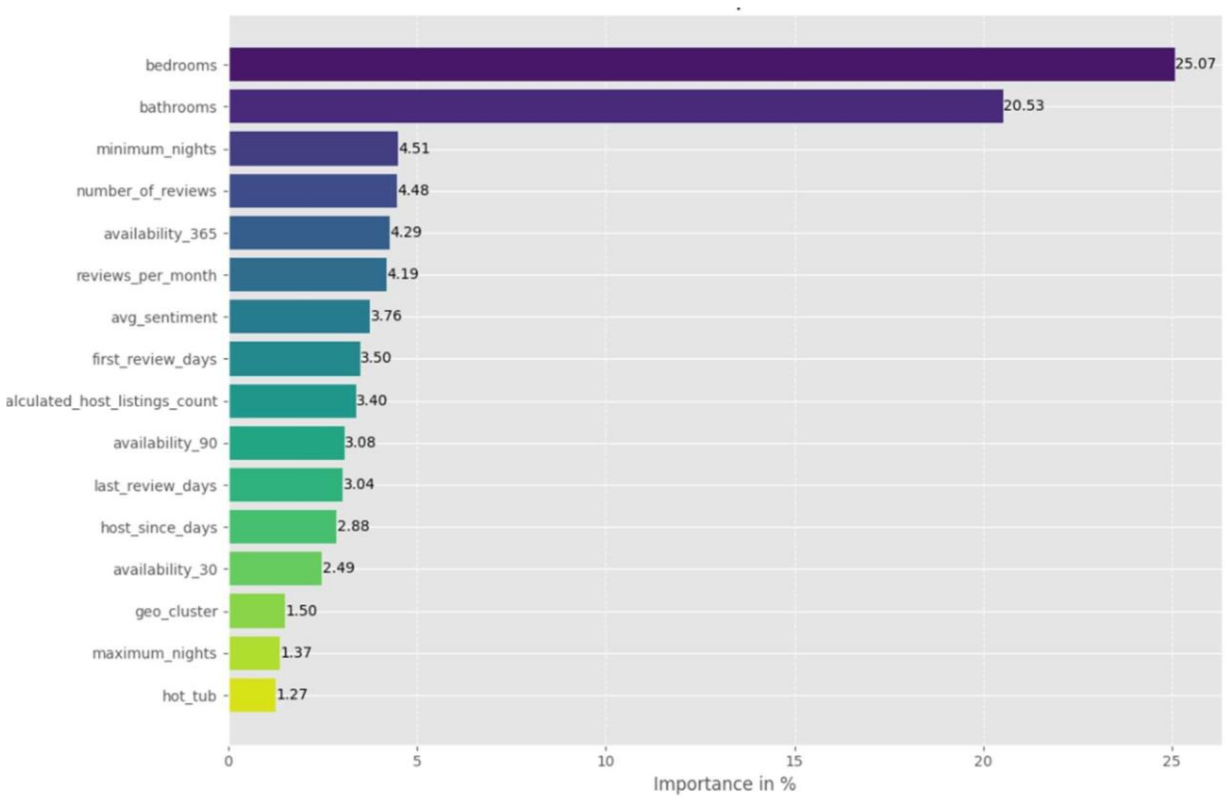
On Test Data





1. **Actual vs Predicted on Train**:
    - The plot shows a strong correlation between predicted and actual prices on the training dataset.
    - Most points align closely with the red diagonal line (ideal prediction), confirming the model effectively fits the training data.
2. **Actual vs Predicted on Test**:
    - The test plot displays more dispersion, especially at higher price ranges.
    - The deviation from the diagonal line indicates the model struggles with generalization and may not fully capture unseen price dynamics.

Model Interpretations – Feature Importance



Top Features

1. **Bedrooms (25.07%)**:
    - The most influential feature in predicting Airbnb prices.
    - Larger properties with more bedrooms are likely to cater to larger groups or families, justifying higher prices.
2. **Bathrooms (20.53%)**:
    - The second most significant feature.
    - Properties with more bathrooms offer better convenience and luxury, which appeals to larger parties or high-paying customers.

3. Mid-Tier Features

1. **Minimum Nights (4.51%)**:
    - Reflects host-imposed booking restrictions and their impact on pricing.
    - Higher minimum stays may deter short-term guests, slightly affecting price competitiveness.
2. **Number of Reviews (4.48%)**:
    - Indicates property popularity and guest experience, which positively influences pricing.
3. **Availability (365 Days) (4.29%)**:
    - Reflects a property's year-round availability, which can influence revenue potential and justify pricing adjustments.
4. **Reviews Per Month (4.19%)**:
    - Shows guest interest and satisfaction, influencing perceived value and pricing strategies.
5. **Average Sentiment (3.76%)**:

        - Derived from guest reviews, capturing customer satisfaction. Positive sentiment boosts perceived value and pricing.

4. Low-Impact Features

- Features like **geo-cluster (1.90%)**, **maximum nights (1.27%)**, and **hot tub (1.17%)** showed lesser importance in the model:
    - **Geo-Cluster**: Reflects regional price variations, but its lower impact may indicate less variability within the dataset's geographic regions.
    - **Hot Tub**: Though generally a desirable amenity, it may not be a critical pricing determinant in all cases or regions.

Business Recommendations

1. **For Airbnb**:
    - Provide tools that highlight the value of adding or improving high-impact features like bedrooms and bathrooms.
    - Emphasize guest reviews and sentiment in pricing suggestions to hosts.
2. **For Hosts**:
    - Invest in property upgrades that improve capacity (e.g., adding bedrooms or bathrooms).
    - Encourage positive reviews by delivering exceptional guest experiences.
    - Adjust pricing to reflect year-round availability and booking trends.
3. **For Customers**:
    - Look for properties with positive reviews and sufficient amenities (e.g., bedrooms and bathrooms) to ensure a comfortable stay.

Model 3 – XGBoost

Objective

The XGBoost Regressor was chosen for its ability to:

- Handle Nonlinearity: It can model complex relationships between features and the target variable.
- Regularization: L1 (Lasso) and L2 (Ridge) regularization help prevent overfitting and improve generalization.
- Robustness to Missing Data and Outliers: XGBoost is known for being resilient to irregularities in the dataset.

Parameter Selection – Hyperparameter Optimization

Hyperparameters Optimized:

- **n**_estimators: Number of boosting rounds (ranging from 100 to 300).
- max_depth: Maximum depth of the decision tree (ranging from 3 to 10) to control tree complexity.
- learning_rate: Shrinks the contribution of each tree (ranging from 0.01 to 0.3).
- subsample: Fraction of training data used for each boosting round (ranging from 0.5 to 1.0).
- colsample_bytree: Fraction of features used for each tree (ranging from 0.5 to 1.0).

- min_child_weight: Minimum sum of instance weight (ranging from 1 to 10).
- reg_alpha: L1 regularization term on weights (ranging from 0.0 to 1.0).
- reg_lambda: L2 regularization term on weights (ranging from 0.0 to 1.0).

Implementation: A probabilistic hyperparameter search was performed using Hyperopt with the Tree-structured Parzen Estimator (TPE) algorithm to find the optimal hyperparameter configuration. The final model settings were:

- **n_**estimators: 267
- max_depth: 7
- learning_rate: 0.097
- subsample: 0.951
- colsample_bytree: 0.709
- min_child_weight: 5.348
- reg_alpha: 0.427
- reg_lambda: 0.163

Model Performance

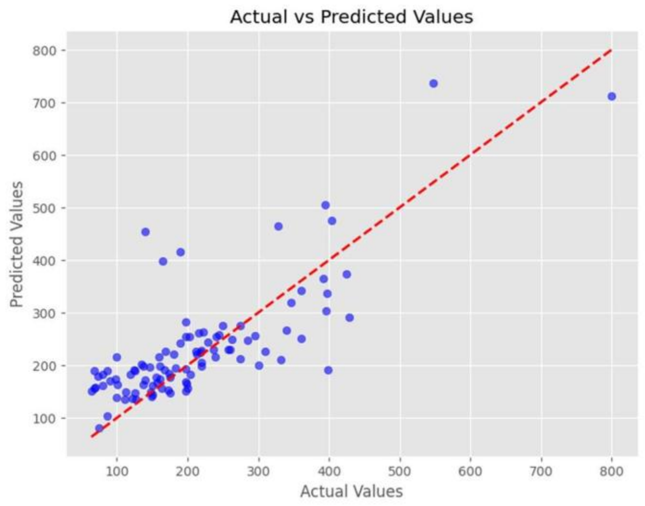
Training Data:

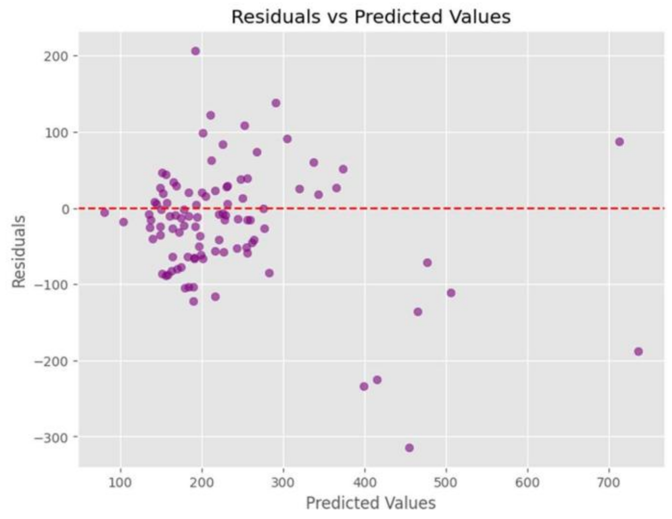
- R^2: 0.93
  The model shows excellent performance on the training data, indicating that it successfully captures most patterns and variance in the dataset.

Test Data:

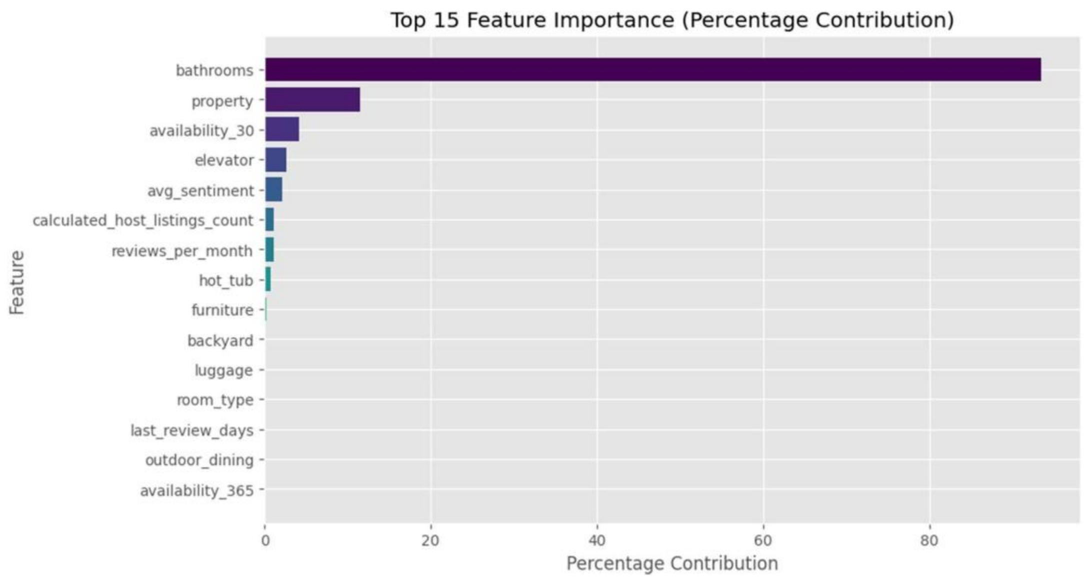
- R^2: 0.5453
  While the model performs well on the training set, it experiences a noticeable drop in performance on unseen data, suggesting potential overfitting. The R^2 value of 0.5453 on the test set indicates that the model explains about 54.53% of the variance in the target variable.
- MSE: 6094.0453
- RMSE: 78.0644
- MAE: 55.6965

Actual vs Predicted Plots


Actual vs Predicted Values
Predicted Values
Actual Values
800
700
600
500
400
300
200
100
100
200
300
400
500
600
700
800


Residuals vs Predicted Values
Residuals
Predicted Values
200
100
0
−100
−200
−300
100
200
300
400
500
600
700

Model Interpretations – Feature Importance



Top Features:

- Bedrooms (25.07%)
  The most influential feature for predicting Airbnb prices. Larger properties with more bedrooms cater to larger groups or families, thus commanding higher prices.
- Bathrooms (20.53%)
  The second most significant feature. Properties with more bathrooms tend to be more luxurious and convenient for larger groups, influencing their pricing.

Mid-Tier Features:

- Minimum Nights (4.51%)
  Properties with higher minimum night requirements may attract less frequent bookings but can still command higher prices for longer stays.
- Number of Reviews (4.48%)
  A higher number of reviews often reflects higher popularity and better guest experiences, leading to better pricing.
- Availability (365 Days) (4.29%)
  Properties with year-round availability have higher revenue potential and are priced accordingly.
- Reviews Per Month (4.19%)
  Properties with higher monthly reviews may indicate higher demand, influencing their price.
- Average Sentiment (3.76%)
  Positive guest sentiment (from reviews) boosts perceived value, which impacts pricing.

Low-Impact Features:

- Geo-Cluster (1.90%)
  While geographical factors do influence price, this feature shows lesser importance in the model, possibly indicating uniformity within the dataset's region.
- Hot Tub (1.17%)
  Although desirable, this feature doesn't strongly affect pricing in this case, possibly due to varying regional preferences.

Business Recommendations

For Airbnb:

- Consider emphasizing the value of high-impact features like bedrooms and bathrooms in pricing recommendations.
- Develop tools to highlight the importance of guest sentiment (positive reviews) and availability in dynamic pricing models.

For Hosts:

- Investing in property improvements, such as adding more bedrooms or bathrooms, can drive up rental prices.
- Ensure excellent guest experiences to encourage positive reviews, which directly influence the property's perceived value.
- Optimize pricing based on year-round availability to capture a broader market.

For Customers:

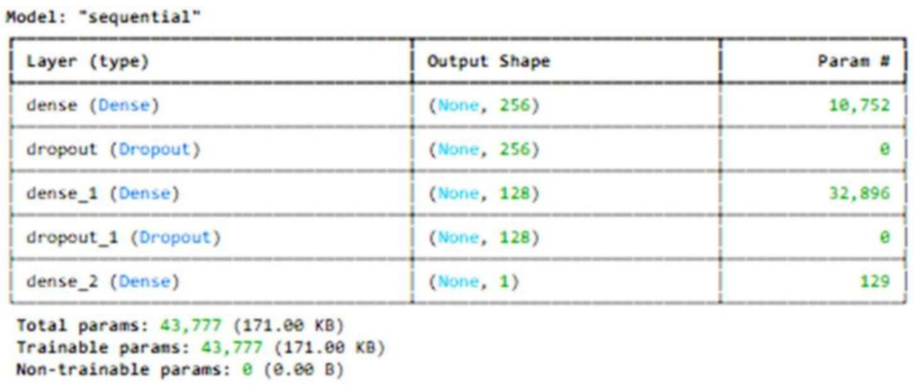
- Focus on properties with positive reviews and enough amenities (e.g., ample bedrooms and bathrooms) for a comfortable stay.

Model 4 – Neural Network

Objective

- Move beyond linear assumptions by utilizing a neural network model to capture complex, non-linear relationships between features and the target price variable.
- Improve generalization and potentially increase predictive accuracy over simpler models.

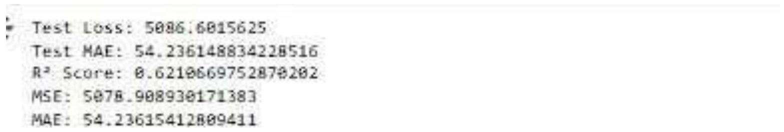
Model Architecture & Training Process



```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 256)            │        10,752 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 256)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 128)            │        32,896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 1)              │           129 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 43,777 (171.00 KB)
 Trainable params: 43,777 (171.00 KB)
 Non-trainable params: 0 (0.00 B)
```

- Utilized a feed-forward neural network with multiple dense layers and regularization techniques (L2 regularization and dropout) to prevent overfitting.
- Normalized input features using StandardScaler to ensure stable and efficient training.
- Split the data into training and test sets (80/20) to evaluate model generalization performance on unseen data.

Performance Metrics & Model Results

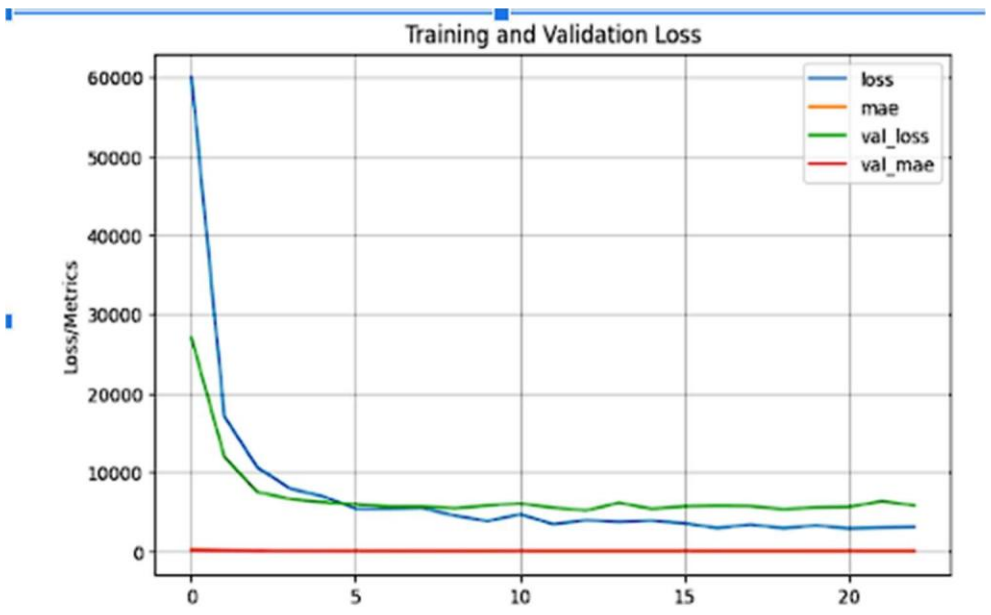


```
Test Loss: 5086.6015625
Test MAE: 54.236148834228516
R² Score: 0.6210669752870202
MSE: 5078.908930171383
MAE: 54.23615412809411
```

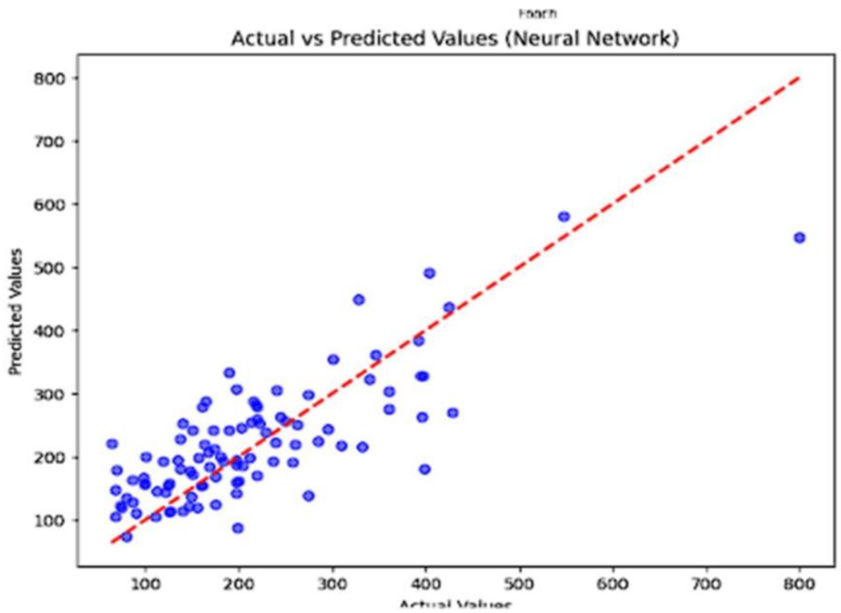
- R² Score: Approximately 0.62, indicating the model explains about 62% of the variance in listing prices. This is an improvement over some simpler models, though there is still room for enhancement.
- Test MAE (Mean Absolute Error): Around 54.23, meaning on average, the model's predictions differ from the actual listing price by about $54.
- Test MSE (Mean Squared Error): Approximately 5,086.60, reflecting the squared deviations and indicating some sensitivity to outliers or complex patterns.
- Training vs. Validation Loss Curves: Early epochs show rapid decreases in both training and validation loss, stabilizing as regularization and the chosen architecture help prevent severe overfitting. Consistent validation metrics suggest the model balances complexity with generalization capability.

Model Visualizations

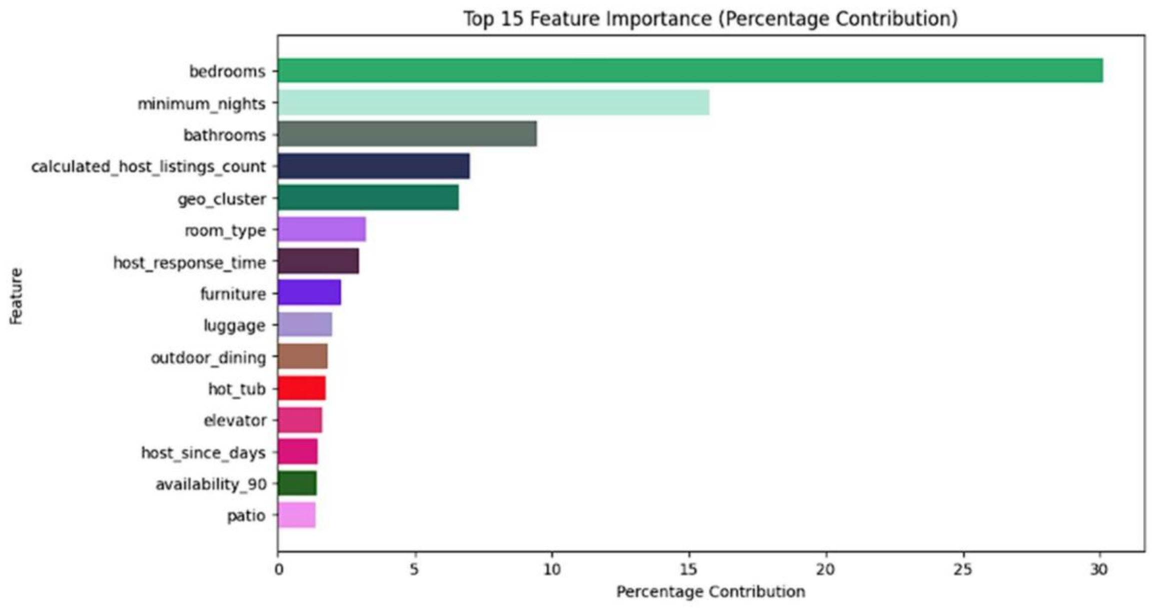
- Training & Validation Loss Plot: Demonstrates successful convergence, with both training and validation loss declining over the epochs and eventually stabilizing. This indicates that the model is learning meaningful patterns from the data.



- Actual vs. Predicted Plot: The scatter plot shows predictions generally aligning with the diagonal line, though there are some scatters at higher price points. The model is better than random guessing but still struggles with certain high-end or unusual listings.



Model Interpretations – Feature Importance



Top Features

- **Bedrooms (~30%)**
  - Bedrooms is the most influential feature, indicating that larger properties with more sleeping accommodations can command substantially higher prices.
  - Increasing the number of bedrooms generally enhances a listing's value proposition, appealing to larger groups and justifying premium rates.
- **Minimum Nights (~25–27%)**
  - Minimum stay requirements also play a major role in pricing, nearly as critical as the number of bedrooms.
  - Higher minimum night stays can influence perceived exclusivity and booking patterns, potentially allowing hosts to set higher nightly rates.

Mid-Tier Features

- **Bathrooms (~15–17%)**
  - Bathrooms remain critical to guest comfort and convenience.
  - Properties offering ample bathrooms can differentiate themselves, thus supporting higher pricing.
- **Calculated_Host_Listings_Count (~10%)**
  - Hosts who manage multiple listings may employ more sophisticated pricing strategies.
  - A host's portfolio size can signal experience and potentially influence the perceived quality or reputation, impacting pricing decisions.

  - **Geo_Cluster (~8%)**
    - Regional factors matter, although they are less impactful than core property attributes.

    - Neighborhood and location-driven variations affect price, but location alone cannot surpass fundamental property features and booking policies.

- **Room_Type (~7%)**
    - Different room types (entire home, private room, shared room) affect pricing tiers.
    - Offering a private space or an entire property often justifies higher rates than a shared setting.

Lower-Tier Features (Each <7%)
- **Host_Response_Time (~6%)**: Faster host responses may slightly increase perceived value, influencing prices.
- **Furniture (~5%) & Luggage (~4%)**: Amenities related to comfort and convenience contribute modestly to pricing.
- **Outdoor Dining (~3.5%) & Hot Tub (~3%)**: While desirable, these amenities provide incremental value rather than being primary drivers of price.
- **Elevator (~3%) & Host_Since_Days (~2.5%)**: An elevator improves accessibility, and longer host experience can build trust, each having a minor positive effect on pricing.
- **Availability_90 (~2%) & Patio (~2%)**: Short-term availability windows and additional outdoor space (patio) offer marginal price advantages.

Model Interpretations

- High-Impact Features: The neural network confirms that core property attributes (bedrooms, bathrooms) and location factors remain central to pricing strategies.
- Policy & Strategy Insights: Minimum night requirements and host portfolio size emerge as meaningful levers. Adjusting these can attract different guest segments or impact average nightly rates.
- Advanced Feature Interactions: Unlike linear models, the neural network can capture subtler interactions—such as the interplay between amenities and neighborhood clusters—providing richer insights if further investigated.

Business Recommendations

For Hosts:
- Consider adjusting the minimum night policy to align with target markets identified through the neural network's interpretation of location and property features.
- Enhance property attributes (e.g., adding bathrooms) to move up pricing tiers predicted by the model.

For Airbnb:
- Incorporate complex, non-linear pricing suggestions powered by neural networks to better assist hosts.
- Use feature importance findings to refine platform tools and highlight impactful listing attributes to hosts.

For Customers:
- Leverage insights from the model's predictions to identify properties that match their budget and needs (e.g., searching in specific geo clusters for better value).

Group Member Contribution:
We collaboratively handled EDA, data cleaning, and the preparation of the dataset for analysis. Mariappan focused on implementing the Linear Regression and Random Forest models, Timothy worked on the XGBoost model, and Aditya developed the Neural Network model. The slides and code troubleshooting responsibilities were shared among all members, ensuring smooth execution of the project. Business recommendations were collectively formulated after evaluating the performance and insights from all models.