Predicting Airbnb Prices - A Machine Learning Approach

August 21, 2024

1 Introduction

1.1 Project Overview

This project focused on predicting Airbnb prices in San Francisco using a machine learning approach. The goal was to create a model that could accurately estimate listing prices based on various features, providing insights for both hosts and potential guests.

1.2 Objective

The primary objective was to develop a machine learning model that predicts the price of Airbnb listings in San Francisco. The model aimed to provide accurate price estimates based on features such as the number of bedrooms, number of persons accommodated, room type, and review scores.

1.3 Tools and Technologies Used

The project used a variety of tools and technologies to ensure efficient data processing, model development, and analysis:

- **Python**: The core programming language used for the entire project, chosen for its extensive libraries and ease of use in data science workflows.
- **Pandas**: Utilized for data manipulation and preprocessing tasks, including handling missing values, encoding categorical variables, and transforming features.
- **NumPy**: Employed for numerical operations, particularly in feature transformation and matrix operations.
- **Matplotlib and Seaborn**: Used for data visualization, these libraries helped in understanding data distributions, feature relationships, and the results of feature engineering.
- **Scikit-learn**: The primary machine learning library used for model selection, training, and evaluation. It provided tools for model comparison, hyperparameter tuning (using `GridSearchCV`), and performance metrics like Mean Squared Error (MSE).
- **SHAP (SHapley Additive exPlanations)**: Applied for model interpretability, SHAP provided insights into how each feature influenced the predictions made by the model.
- **Joblib and Pickle**: Utilized for saving and loading trained models.

2 Data Collection and Preprocessing

2.1 Data Source

The dataset used in this project was sourced from Airbnb listings in San Francisco. It included a variety of features such as `price`, `longitude`, `host_is_superhost`, `bathrooms`, `cancellation_policy`, `property_type`, and several review scores. The dataset comprised 7,146 entries and 34 features.

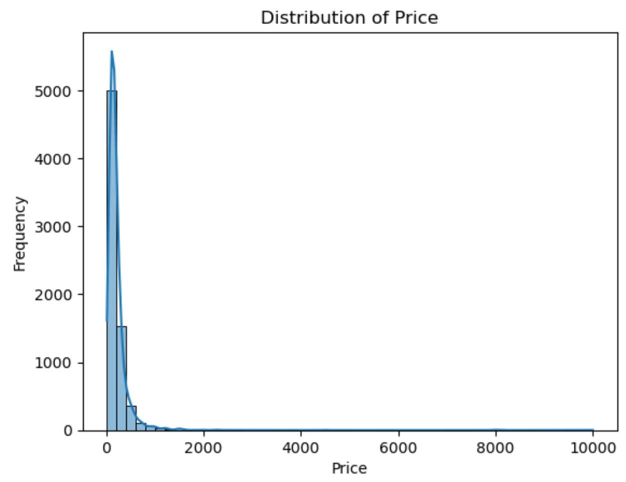
2.2 Initial Data Exploration

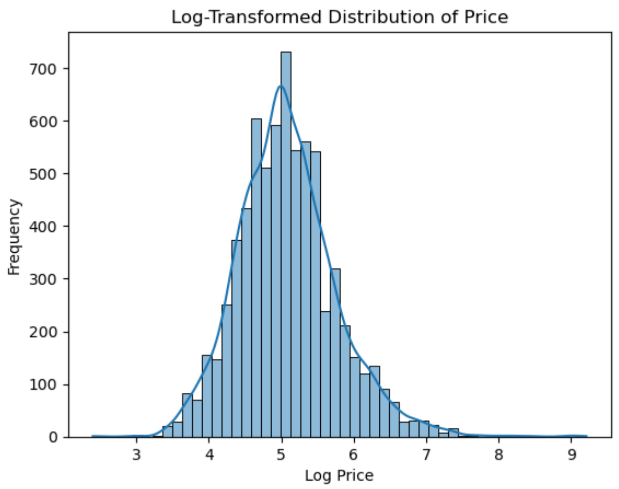
The initial exploration involved loading the dataset and performing basic statistical analysis to understand its structure. Summary statistics were generated to identify key features and to gain insights into the distribution of the data.

2.3 Data Cleaning and Transformation

The data cleaning process included the following steps:

- **Missing Values**: There were no significant missing values, which simplified the preprocessing phase.
- **Outlier Treatment**: Outliers in numerical features such as `accommodates`, `bedrooms`, and `bathrooms` were capped at the 99th percentile to reduce their influence.
- **Feature Transformation**: The `price` variable was log-transformed to stabilize variance and reduce skewness. This transformation helped in achieving a more normally distributed target variable.





- **Categorical Encoding**: Binary categorical variables, like `host_is_superhost` and `instant_bookable`, were encoded as 0 and 1. Multi-class categorical variables, including `cancellation_policy`, `neighbourhood_cleansed`, `property_type`, and `room_type`, were one-hot encoded to convert them into numerical format.

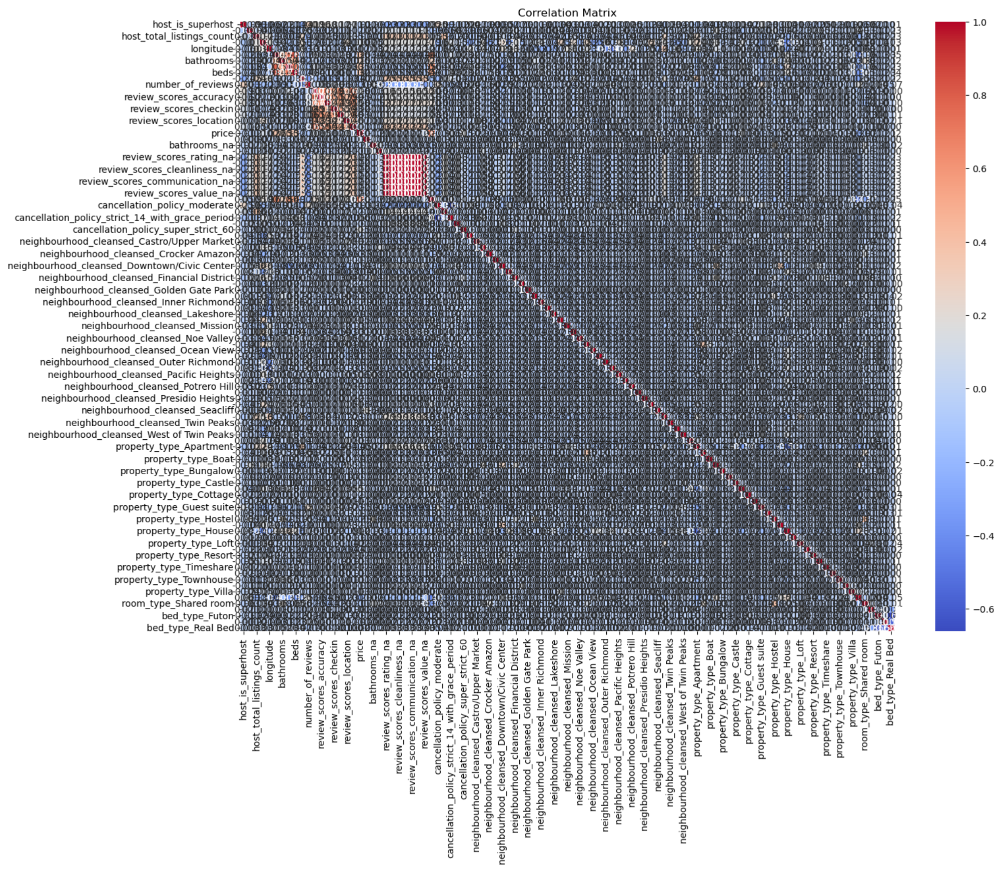
2.4 Final Preprocessed Data

After preprocessing, the dataset was transformed into a format suitable for model training. The final dataset contained 101 features after encoding and transformations, and it was saved as `preprocessed_airbnb_data.csv` for further analysis.

3 Feature Selection

3.1 Correlation Analysis

To identify the most relevant features for predicting Airbnb prices, a correlation matrix was generated. This matrix helped in understanding the linear relationships between various features and the target variable, `log_price`. Features like `accommodates`, `bedrooms`, and `review_scores_rating` showed strong positive correlations with `log_price`, indicating that as these values increase, the price tends to increase as well. Conversely, features such as `room_type_Private room` and `room_type_Shared room` demonstrated negative correlations, suggesting that listings with private or shared rooms are generally priced lower than entire homes or apartments.



This correlation analysis provided initial insights into which features might be the most influential in predicting the target variable, guiding the next steps in feature selection.

3.2 Multicollinearity Check

While correlation analysis helped identify relationships between individual features and `log_price`, it was also essential to check for multicollinearity — a scenario where two or more features are highly correlated with each other. High multicollinearity can lead to overfitting and reduced model interpretability.

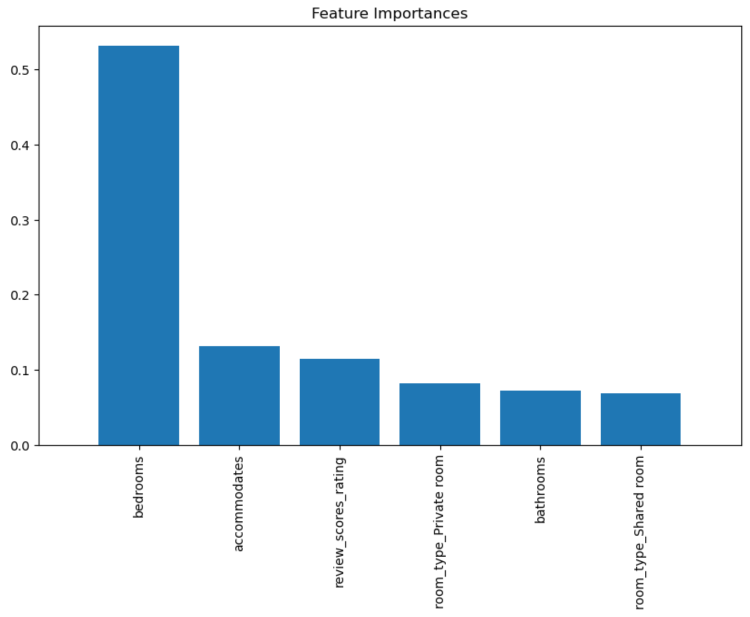
Pairs of features with correlation coefficients greater than 0.8 were flagged as having high multicollinearity. For instance, `accommodates` and `bedrooms` showed a high correlation, indicating potential redundancy. However, both features were retained in the model because they capture different, yet complementary, aspects of a listing's capacity. `Accommodates` reflects the overall capacity of the listing, indicating how many guests it can host, while `bedrooms` gives specific information about the sleeping arrangements, which can be a critical factor in determining price. The inclusion of both features allows the model to leverage the nuances of each factor, contributing to more accurate predictions.

This analysis ensured that the features chosen for the final model were not only relevant but also non-redundant.

3.3 Feature Importance Analysis

To further refine the feature selection process, a Random Forest model was used to evaluate the importance of each feature in predicting `log_price`. Random Forest was chosen due to its ability to handle a large number of features and rank them based on their contribution to the model's predictions.

The analysis confirmed that `bedrooms`, `accommodates`, and `review_scores_rating` were among the most influential features. The `room_type` variables also played a significant role, particularly in distinguishing between different types of accommodations. This feature importance analysis validated the earlier findings from the correlation analysis and provided a data-driven basis for the final feature selection.



3.4 Final Feature Set

Based on the correlation analysis, multicollinearity check, and feature importance evaluation, the following features were selected for the final model:

- **`bedrooms`**: Strongly correlated with price, indicating that more bedrooms typically lead to higher prices. Despite its high correlation with `accommodates`, it was retained because it provides specific information about sleeping arrangements, a key determinant in pricing.
- **`accommodates`**: Reflects the overall capacity of the listing, another key determinant of price. It complements `bedrooms` by providing a broader perspective on how many guests a listing can host.
- **`review_scores_rating`**: Captures the overall quality of the listing as perceived by guests, influencing the price.
- **`room_type_Private room`**: Indicates whether the listing offers a private room, generally leading to lower prices compared to entire homes.
- **`bathrooms`**: Although less influential than bedrooms or accommodates, the number of bathrooms is still relevant for pricing.
- **`room_type_Shared room`**: Identifies listings with shared rooms, which are usually priced lower.

3.4.1 Feature Matrix and Target Variable

The next step involved creating the feature matrix `X` and the target variable `y` from the dataset.

- **Feature Matrix `X`**: This matrix consists of the selected features (`bedrooms`, `accommodates`, `review_scores_rating`, `room_type_Private room`, `bathrooms`, `room_type_Shared room`).
- **Target Variable `y`**: The target variable `y` is the `log_price`. Using `log_price` as the target helps in normalizing the distribution of prices and mitigating the effects of extreme values.

3.4.2 Splitting the Dataset

To evaluate the model's performance and ensure its ability to generalize to unseen data, the dataset was split into training and testing sets.

- **Training Set (80%)**: The majority of the data (80%) was allocated to the training set. This subset was used to train the model, allowing it to learn the relationships between the features and the target variable. The training set included 5,716 samples, each with the six selected features.
- **Testing Set (20%)**: The remaining 20% of the data was reserved as the testing set, consisting of 1,430 samples.

4 Model Selection and Training

4.1 Model Comparison

The first step in selecting the best model for predicting Airbnb prices was to compare several machine learning algorithms. Four different models were tested:

- **Linear Regression**: A baseline model that assumes a linear relationship between features and the target variable.

- **Decision Tree**: A model capable of capturing non-linear relationships but prone to overfitting.
- **Random Forest**: An ensemble model that averages multiple decision trees to reduce overfitting and improve predictive accuracy.
- **Gradient Boosting**: Another ensemble model that builds trees sequentially, with each new tree correcting the errors of the previous ones.

Each model was trained on the preprocessed training set, and their performance was evaluated on the test set using Mean Squared Error (MSE) as the metric.

[11]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

# Initialize models
lr = LinearRegression()
dt = DecisionTreeRegressor(random_state=42)
rf = RandomForestRegressor(random_state=42)
gb = GradientBoostingRegressor(random_state=42)

# Dictionary to hold models
models = {'Linear Regression': lr, 'Decision Tree': dt, 'Random Forest': rf,␣
↪'Gradient Boosting': gb}

# Train each model and evaluate
for name, model in models.items():
    model.fit(X_train, y_train.values.ravel())  # Convert y_train to 1D array
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test.values.ravel(), y_pred)  # Convert y_test␣
↪to 1D array
    print(f"{name} - MSE: {mse:.4f}")
```

```
Linear Regression - MSE: 0.2080
Decision Tree - MSE: 0.2497
Random Forest - MSE: 0.2062
Gradient Boosting - MSE: 0.1897
```

The Gradient Boosting model achieved the lowest MSE, indicating its superior performance in predicting Airbnb prices compared to the other models.

4.2 Hyperparameter Tuning

To further optimize the model, hyperparameter tuning was performed on the best-performing models: Gradient Boosting and Random Forest.

GridSearchCV was used to search for the best combination of hyperparameters for the Gradient Boosting model. The parameters tuned included the number of estimators, learning rate, maximum depth, and minimum samples required to split a node.

[12]:
```
from sklearn.model_selection import GridSearchCV

# Tuning Gradient Boosting
param_grid_gb = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10]
}

grid_search_gb = GridSearchCV(gb, param_grid_gb, cv=5,␣
 ↪scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_gb.fit(X_train, y_train.values.ravel())

# Best parameters and score
print("Best Parameters for Gradient Boosting:", grid_search_gb.best_params_)
print("Best CV Score for Gradient Boosting:", -grid_search_gb.best_score_)
```

```
Best Parameters for Gradient Boosting: {'learning_rate': 0.1, 'max_depth': 3,
'min_samples_split': 2, 'n_estimators': 100}
Best CV Score for Gradient Boosting: 0.21203240531466613
```

Similarly, GridSearchCV was used to tune the hyperparameters of the Random Forest model, including the number of trees, maximum depth, and minimum samples required at each node.

[13]:
```
# Tuning Random Forest
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5,␣
 ↪scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_rf.fit(X_train, y_train.values.ravel())

# Best parameters and score
print("Best Parameters for Random Forest:", grid_search_rf.best_params_)
print("Best CV Score for Random Forest:", -grid_search_rf.best_score_)
```

```
Best Parameters for Random Forest: {'max_depth': 10, 'min_samples_leaf': 4,
'min_samples_split': 10, 'n_estimators': 300}
Best CV Score for Random Forest: 0.21502721028418487
```

4.3 Final Model Selection

The final Gradient Boosting model was evaluated on the test set, yielding the following results:

[14]:
```
# Evaluate Gradient Boosting on Test Set
final_gb = grid_search_gb.best_estimator_
gb_test_pred = final_gb.predict(X_test)
gb_test_mse = mean_squared_error(y_test.values.ravel(), gb_test_pred)
print(f"Gradient Boosting Test MSE: {gb_test_mse:.4f}")
```

```
Gradient Boosting Test MSE: 0.1897
```

The identical MSE for both the initial and tuned Gradient Boosting models suggests that the initial model's parameters were already well-suited to the data, resulting in minimal performance improvement from tuning. This indicates that the tuning process did not significantly alter the model's behavior or that the data's inherent complexity limits further reduction in MSE.

The Random Forest model, while well-tuned, did not outperform the Gradient Boosting model.

[15]:
```
# Evaluate Random Forest on Test Set
final_rf = grid_search_rf.best_estimator_
rf_test_pred = final_rf.predict(X_test)
rf_test_mse = mean_squared_error(y_test.values.ravel(), rf_test_pred)
print(f"Random Forest Test MSE: {rf_test_mse:.4f}")
```

```
Random Forest Test MSE: 0.1912
```

The R-squared value of the final Gradient Boosting model on the test set was calculated to assess how well the model explains the variance in the target variable.

[16]:
```
from sklearn.metrics import r2_score

# Make predictions on the test set
y_test_pred = final_gb.predict(X_test)

# Calculate R-squared
r2 = r2_score(y_test.values.ravel(), y_test_pred)

print(f"R-squared for the final Gradient Boosting model: {r2:.4f}")
```

```
R-squared for the final Gradient Boosting model: 0.5952
```

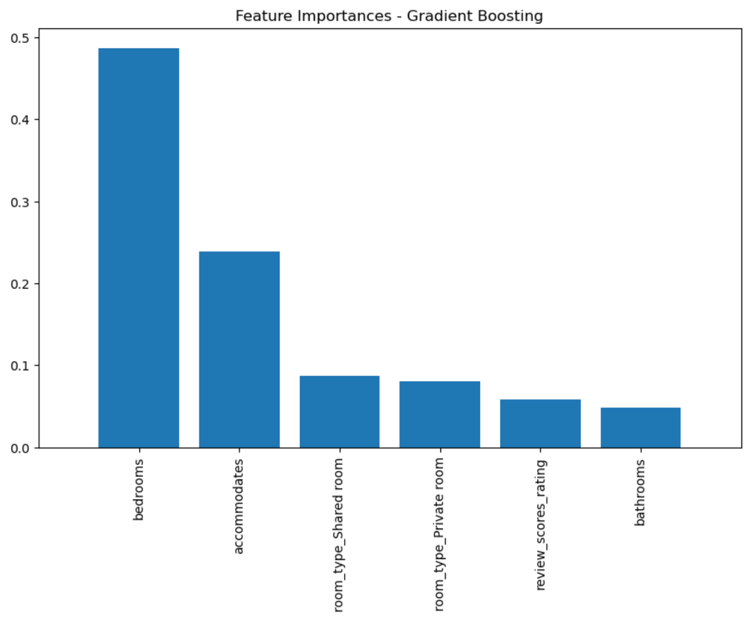
The R-squared value of 0.5952 indicates that approximately 59.5% of the variance in `log_price` can be explained by the model. While the model captures a significant portion of the variability, the remaining unexplained variance likely reflects the complexity and unpredictability typical of real estate markets, including location-specific factors, unique property features, and seasonal demand shifts that are difficult to quantify.

The Gradient Boosting model, with its strong performance metrics, was ultimately selected as the final model for predicting Airbnb prices. The Gradient Boosting model's ability to sequentially correct the errors of previous trees allowed it to capture complex patterns in the data more effectively than the other models.

5 Model Interpretation

5.1 Feature Importance

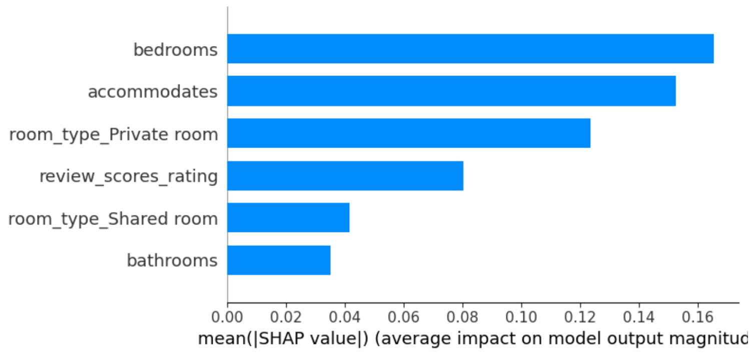
The feature importances were extracted from the final Gradient Boosting model, providing insights into how each feature contributed to the model's decision-making process.



For instance, features like `bedrooms` and `accommodates` were identified as highly influential, reflecting their strong correlation with higher listing prices. These features likely capture the core aspects of a property's value proposition—namely, its capacity to host guests, which is directly tied to revenue potential. Meanwhile, `review_scores_rating`, while not as influential as the primary features, still played a crucial role by capturing guest satisfaction, which can affect both demand and pricing.

5.1.1 SHAP Analysis

To provide transparency and explainability, SHAP (SHapley Additive exPlanations) was used to interpret the model's predictions. SHAP values indicate the impact of each feature on the model's output.



The SHAP analysis confirmed the importance of features such as bedrooms and accommodates in the model's predictions, providing further validation for their inclusion in the final model.

6 Model Testing

To ensure the model's reliability, it was tested on various scenarios, including edge cases, realistic cases, and a direct comparison with actual data.

6.1 Edge Cases

The model was tested on edge cases involving listings with extremely high or low values for certain features, such as the number of bedrooms or review scores. While the Gradient Boosting model generally handled these cases, some discrepancies were observed, particularly with unusual property types or combinations of features. For instance, properties with a very high number of bedrooms but low overall ratings led to predictions that deviated from expected prices. These discrepancies suggest that while the model can manage typical variations in the data, it may struggle with more atypical cases, highlighting potential areas for further model refinement.

6.2 Realistic Cases

The model's performance was also evaluated on more typical, realistic listings that represent the majority of Airbnb properties. In these scenarios, the predictions were generally accurate, closely aligning with the actual prices of the listings.

6.3 Comparison with Actual Data

To further assess the model's performance, a veriety of listings were selected from the original dataset, and their prices were predicted using the trained model. These predictions were then compared to the actual prices to evaluate how well the model performed on real-world data. Ten of the listings that were selected for comparison are shown below, along with the predicted and actual price for each.

[24]:
```python
import joblib

final_gb = joblib.load('final_gradient_boosting_model.pkl')

# Predict the log-transformed prices
X_test_cases = selected_rows.drop(columns=['log_price'])
predicted_log_prices = final_gb.predict(X_test_cases)

# Convert the predicted log prices back to actual prices
predicted_prices = np.exp(predicted_log_prices)

# Compare the predicted prices with the actual prices
comparison = pd.DataFrame({
    'Actual Log Price': selected_rows['log_price'],
    'Predicted Log Price': predicted_log_prices,
    'Actual Price': np.exp(selected_rows['log_price']),
    'Predicted Price': predicted_prices,
    'Difference (Price Scale)': np.exp(selected_rows['log_price']) -␣
 ↪predicted_prices
})

# Display the comparison
print(comparison)
```

```
    Actual Log Price  Predicted Log Price  Actual Price  Predicted Price  \
91          3.806662             3.911337          45.0        49.965711
33          4.700480             4.482176         110.0        88.426899
27          5.043425             4.752407         155.0       115.862809
56          4.753590             4.787325         116.0       119.980046
22          4.174387             4.283259          65.0        72.476221
48          5.135798             5.136988         170.0       170.202410
34          6.109248             6.072205         450.0       433.635717
67          4.753590             5.102832         116.0       164.487109
78          4.442651             4.502719          85.0        90.262199
44          5.068904             4.996385         159.0       147.877609

    Difference (Price Scale)
91                 -4.965711
33                 21.573101
27                 39.137191
56                 -3.980046
22                 -7.476221
48                 -0.202410
34                 16.364283
67                -48.487109
78                 -5.262199
44                 11.122391
```

The results showed that the model performed well in most cases, accurately predicting prices close to the actual values. However, there were instances where the model significantly overpredicted or underpredicted the price. These discrepancies were often due to unique neighborhood characteristics, which the model could not adequately account for. This highlights the challenges in modeling real estate data, where numerous unquantifiable factors can influence pricing. Despite these challenges, the model generally provided reliable predictions, making it a valuable tool for estimating Airbnb prices.

7 Conclusion

After completing the model development and testing phases, the following conclusions were reached:

7.1 Model Strengths

- **Accuracy**: The model performed reasonably well, particularly for typical properties in well-represented neighborhoods. The predictions closely aligned with actual prices, demonstrating the model's reliability in these scenarios.

- **Feature Selection**: The features chosen for the model proved to be effective predictors of Airbnb prices. In particular, `bedrooms` and `accommodates` were highly influential, reflecting their critical role in determining listing prices.

7.2 Limitations

- **Neighborhood Features**: The absence of more granular neighborhood data likely contributed to some of the discrepancies observed in the model's predictions. Including additional features, such as average rent prices in the area or proximity to landmarks, could enhance the model's accuracy by better capturing the influence of location on pricing.

7.3 Future Improvements

- **Feature Engineering**: Incorporating additional features related to neighborhood characteristics, such as local amenities, crime rates, or school quality, could significantly enhance the model's predictive accuracy. These features would provide a more comprehensive view of the factors influencing Airbnb prices.
- **Data Augmentation**: Expanding the dataset to include more recent or diverse listings would improve the model's ability to generalize to new data. This could involve adding listings from different cities or regions, as well as updating the dataset with the latest available data.

7.4 Final Steps

With the model finalized, the following steps were taken to ensure it was ready for deployment or further development:

- **Model Saving**: The final model was saved using `joblib`, ensuring that it can be reloaded and used in future applications. This step preserves the model's trained state, making it easy to deploy or further refine.
- **Documentation**: The entire process was thoroughly documented, including detailed explanations of data preprocessing steps, feature selection rationale, model evaluation metrics, and limitations.
- **Testing Documentation**: The discrepancies identified during the testing phase were carefully documented, along with potential reasons for these discrepancies and suggested improvements.