

**Project -04 Group-15**

**New York City Airbnb Analysis**

Project Write-Up

Fatima Hamadi  
Gunel Garayeva  
Lina Blanco  
Kade Rivers

Professor Alexander Booth

**Data Science & Analysis Bootcamp**

**October 3, 2024**

### **Abstract**

This project focuses on visualizing and analyzing Airbnb listing data from New York City to uncover key insights into pricing trends, neighborhood popularity, and booking availability. Two interactive Tableau stories were created to explore the relationship between neighborhood location, price, availability, and customer reviews. Machine Learning was performed to utilize rental features and predict prices. A recommender was created to input features and output a list of the closest related current listings to those features. The project aims to help hosts, guests, and stakeholders in the Airbnb ecosystem make informed decisions about pricing strategies, booking trends, and market competitiveness. The project utilizes a cleaned and standardized dataset of Airbnb listings, offering intuitive, data-driven insights through dynamic visualizations. By leveraging interactive features, the dashboards and machine learning provide users with the ability to explore trends and draw actionable conclusions.

### **Inspiration Behind the Project**

The inspiration for this project came from the rising popularity of short-term rentals and the significant impact platforms like Airbnb have on both local housing markets and tourism. New York City, as one of the world's largest and most diverse travel destinations, presents a rich opportunity for analysis. Understanding how neighborhoods differ in terms of pricing, availability, and demand is crucial for both hosts and travelers.

The project seeks to address a common challenge for Airbnb hosts—setting competitive prices and optimizing availability while managing reviews and guest satisfaction. It offers insights into how prices vary across boroughs and how reviews may impact booking and hosting decisions. Policymakers could also use the data to understand the concentration and effects of Airbnb rentals in specific areas. The combination of data visualization and machine learning potential in this field makes it an ideal area for exploration.

## Project Links

**Web Deployment:** <https://project4group15.pythonanywhere.com/>

**Repository:** <https://github.com/musicmaj/project-4-group-15>

## Data Cleaning

The original dataset for this project was taken from Kaggle and two additional datasets from an open source Airbnb website were added to add in additional years for updated listings and increased prediction capabilities. The addition of datasets in October of 2023 and September of 2024 listings showed the impact of Local Law 18 in the state of New York that became a law in 2022, but wasn't fully enforced until September 2023. Due to adding on additional datasets and preparing for analysis and machine learning, a significant amount of data cleaning was required. This involved adjusting missing values that were logical to continue to include in the dataset and had reasonable inputs in place of null or removal, correcting data types, and removing duplicate entries. Outliers in price data were addressed by filtering listings with extreme prices that could skew the overall analysis. After cleaning, the dataset was shared and adjusted by each team member to suit the specific needs of their target and features for a well-rounded analysis and to avoid repetition throughout the visualizations.

## Design Concepts

The overall color palette for this project is the standard color palette for Airbnb that is available online, the Bootswatch theme titled, "Solar", and the Airbnb-themed slide deck from Slidechef. In more complex visualizations, contrasting colors were used to differentiate between various boroughs, while gradients were used to depict intensity—such as higher prices or review counts. Viewing accessibility was intended by choosing colors distinguishable for color-blind users.

## **Tableau Stories: Fatina Hamadi and Gunel Garayeva**

The project consists of two tableau stories with three or more interactive dashboards, each designed with a specific focus which include: pricing trends by neighborhood and room type, availability over time, review insights, popular neighborhood groups and top hosts, changes in price.

The Pricing Trends dashboard along with the map of listings by neighborhood and room type helps users visualize price differences across boroughs and neighborhoods, revealing trends such as the most listings and higher prices in Manhattan compared to the Bronx.

The Availability Over Time dashboard highlights seasonal trends, such as increased bookings during tourist seasons, helping hosts optimize their listings for peak demand.

The Review Insights dashboard and the Host Dashboard show highly-reviewed listings and higher rated hosts tend to be priced higher, giving potential hosts insights into how reviews might impact their earning potential.

The Changes in Avg Price and total bookings graphs showcase the surge of rentals prior to COVID, the drop-off of rentals mid-covid, the slow build following the lifting of restrictions, and then a hefty price drop in pricing after the Local Law 18 was enforced in September of 2023 making about 7,500 Airbnb listings no longer a legal listing to have on the site.

## **Machine Learning: Lina Blanco**

Due to the detail of the Machine Learning process, various definitions and blocks of code have been provided in the write up by making the text smaller, eliminating extra lines of space, and indented sections to differentiate definitions and code versus the results and trends.

Just as with all data analysis, the data cleaning step is an essential part of the data preprocessing phase that prepares the dataset for Exploratory Data Analysis (EDA). Find below the data cleaning and organization steps outlined for this project process:

Input and Output Paths:

INPUT\_PATH: Location of the raw dataset.

## Full-Stack Airbnb Data Analysis and Application

OUTPUT\_UNENCODED\_PATH and OUTPUT\_ENCODED\_PATH: Paths to save cleaned data. Helps organize workflow and manage raw vs. processed data.

### Dropping Columns:

Use COLUMNS\_TO\_DROP to remove unnecessary columns like 'id' and 'host id' to reduce noise and focus on relevant features for analysis.

### Renaming Columns:

COL\_RENAME dictionary renames columns for clarity (e.g., 'neighbourhood\_group' becomes 'boro'). Clear names improve readability and analysis.

### Correcting Values:

BORO\_CORRECTIONS and ROOM\_TYPE\_CORRECTIONS ensure consistency by fixing typos (e.g., 'brookln' to 'brooklyn'). Standardized values prevent errors during analysis.

### Handling Money Values:

Use MONEY\_REGEX to clean monetary values (e.g., removing \$ signs or commas), converting them into numbers. MONEY\_COLS lists which columns need this cleaning.

These steps ensure a clean, well-structured dataset that is ready for accurate and efficient analysis.

```
INPUT_PATH = "Airbnb_Open_Data_Original_2022_2024.csv"
OUTPUT_UNENCODED_PATH = "airbnb_2022_2024_clean_unencoded.csv"
OUTPUT_ENCODED_PATH = "airbnb_2022_2024_clean_encoded.csv"
COLUMNS_TO_DROP = ['id', 'NAME', 'host id', 'host name', 'neighbourhood', 'Construction year',
                    'lat', 'long', 'country', 'country code', 'number of reviews',
                    'last review', 'reviews per month', 'calculated host listings count',
                    'house_rules', 'availability 365', 'license']
COL_RENAME = {'neighbourhood_group': 'boro'}
BORO_CORRECTIONS = {'brookln': 'brooklyn', 'manhatan': 'manhattan', 'staten island': 'staten'}
ROOM_TYPE_CORRECTIONS = {'Entire home/apt': 'entire', 'Private room': 'privater',
                          'Shared room': 'sharedr', 'Hotel room': 'hotelr'}
MONEY_REGEX = {"$": "", "'": "", ",": ""}
MONEY_COLS = ('price', 'service_fee')
Code useful Tips
# Remove rows with minimum nights < 0
neg_min_night_mask_s = df2['minimum_nights'] < 0
print(f"Removing {neg_min_night_mask_s.sum()} rows")
df2 = df2[~neg_min_night_mask_s]
```

Removing invalid minimum nights entries, Converting categorical data into a numerical format with one-hot encoding. Freeing up memory by deleting the original unneeded DataFrame (df2). This keeps the process memory-efficient and ensures only clean, encoded data is used for further analysis or modeling.

### Explanation:

#### Filtering Invalid Rows:

The code identifies rows where minimum nights is less than 0 using a boolean mask, prints the count of these rows, and then filters the DataFrame to keep only the valid rows (minimum\_nights >= 0). This ensures only meaningful data is retained.

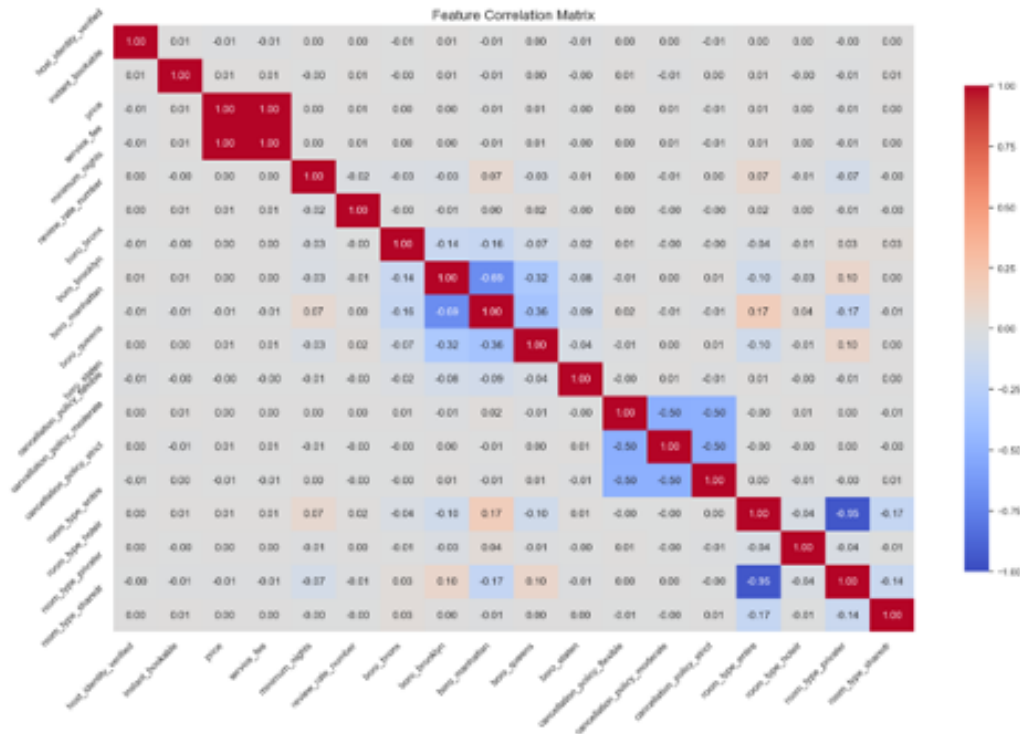
#### One-Hot Encoding:

## Full-Stack Airbnb Data Analysis and Application

`df_encoded = pd.get_dummies(df2)`: Converts categorical variables in `df2` into binary columns (0 or 1) to make them usable by machine learning models.

Deleting the Original DataFrame:

`del df2`: Safely deletes the original DataFrame to free up memory once it's no longer needed after encoding.



### Correlation

The correlation matrix indicates the strength and direction of relationships between various features, revealing that price and service\_fee are strongly positively correlated, while room\_type categories like private\_room (privater) and 'Entire home/apt' (entire) are strongly negatively correlated, suggesting distinct accommodation types.

The correlation between price and service\_fee, with a coefficient of 0.999991, indicating a strong positive correlation. For the relationship between room\_type\_privater and room\_type\_entire, the correlation coefficient is -0.949837, indicating a strong negative correlation. These numerical values demonstrate the significant relationships between these variables.

The numerical values of the correlation coefficients provide a quantitative measure of the relationships between the variables:

1. Strong Positive Correlation (Price and Service Fee): The correlation coefficient of 0.999991 between price and service\_fee is extremely close to +1, which indicates that as the price of a listing increases, the service fee also tends to increase almost perfectly. This suggests that the pricing structure for listings is closely tied to the service fees charged, reflecting a consistent pattern in how these two variables interact.
2. Strong Negative Correlation (Room Type - Private and Entire): The correlation coefficient of -0.949837 between room\_type\_private and room\_type\_entire is very close to -1, indicating a strong negative relationship. This means that when a listing is categorized as an entire home, it is less likely to be categorized as a private room, and vice versa. This reflects the distinct nature of these accommodation types, where one type's prevalence reduces the likelihood of the other type being present.

### Scaler

The use of a scaler is typically intended to standardize or normalize the feature variables (X) before training a machine learning model. Here's an explanation of the context and the importance of scaling:

#### Defining the Target Variable

```
X = df.drop(columns=['price'])
```

```
y = df['price']
```

- Here, X represents the feature set, which includes all columns in the DataFrame df except for the price column, which is defined as the target variable y. The goal is to predict price based on the features in X.

#### Splitting the Data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- The dataset is split into training and testing sets using train\_test\_split, with 80% of the data used for training and 20% for testing. This is essential for evaluating the model's performance on unseen data.

### Importance of Scaling

**Feature Scaling:** Before training a machine learning model, especially those that rely on distance calculations (like K-Nearest Neighbors or Support Vector Machines used in the project) or gradient descent optimization (like linear regression), it is important to scale the features. This ensures that all features contribute equally to the distance calculations and model training.

**Standardization:** Using a scaler like StandardScaler transforms the features to have a mean of 0 and a standard deviation of 1. This is particularly useful when features have different units or ranges, as it helps the model converge faster during training.

**Normalization:** Alternatively, normalization (using MinMaxScaler, for example) scales the features to a range of [0, 1]. This can be beneficial for algorithms sensitive to the scale of the data.

### Machine Learning Train and Test – Code Explanation

#### 1. List of Models to Test

```
models = {
```

```
    "LinearRegression": LinearRegression(),
```

```
    "Ridge": Ridge(),
```

```
    "Lasso": Lasso(),
```

```
    "ElasticNet": ElasticNet(),
```

```
    "BayesianRidge": BayesianRidge(),
```

```
    "HuberRegressor": HuberRegressor(),
```

```
    "RANSACRegressor": RANSACRegressor(),
```

```
"DecisionTreeRegressor": DecisionTreeRegressor(),
"RandomForestRegressor": RandomForestRegressor(),
"GradientBoostingRegressor": GradientBoostingRegressor(),
"AdaBoostRegressor": AdaBoostRegressor(),
"KNeighborsRegressor": KNeighborsRegressor(),
"SupportVectorRegressor": SVR(),
}
```

This dictionary defines a list of regression models to be tested, with the model names as keys and the instantiated model objects as values. This allows for easy iteration and evaluation of each model.

### 2. Dictionary to Hold Model Performance

```
mse_results = {}
trained_models = {}
mae_results = {}
rmse_results = {}
r2_results = {}
```

Several dictionaries are initialized to store the performance metrics (Mean Squared Error, Mean Absolute Error, Root Mean Squared Error, and  $R^2$  score) for both training and testing datasets, as well as to keep track of the trained models.

### 3. Loop Through Each Model

```
for model_name, model in models.items():
    print(f"Evaluating {model_name}...")
```

This loop iterates over each model in the models dictionary, printing the name of the model currently being evaluated.

### 4. Fit the Model

```
model.fit(X_train_scaled, y_train)
```

Each model is fitted to the scaled training data ( $X_{train\_scaled}$  and  $y_{train}$ ).

### 5. Make Predictions

```
y_pred_train = model.predict(X_train_scaled) # Predictions on training data
y_pred_test = model.predict(X_test_scaled)    # Predictions on test data
```

Predictions are made for both the training and testing datasets.

### 6. Calculate Metrics for Training and Test Data

```
mse_train = mean_squared_error(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)
rmse_train = np.sqrt(mse_train)
r2_train = r2_score(y_train, y_pred_train)
mse_test = mean_squared_error(y_test, y_pred_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
rmse_test = np.sqrt(mse_test)
r2_test = r2_score(y_test, y_pred_test)
```

Various performance metrics are calculated for both the training and testing datasets:

- Mean Squared Error (MSE): Measures the average squared difference between actual and predicted values.
- Mean Absolute Error (MAE): Measures the average absolute difference between actual and predicted values.
- Root Mean Squared Error (RMSE): The square root of MSE, providing error in the same units as the target variable.
- $R^2$  Score: Indicates the proportion of variance in the dependent variable that can be explained by the independent variables.

### 7. Store Results in Dictionaries

```
mse_results[model_name] = {'train': mse_train, 'test': mse_test}
mae_results[model_name] = {'train': mae_train, 'test': mae_test}
rmse_results[model_name] = {'train': rmse_train, 'test': rmse_test}
r2_results[model_name] = {'train': r2_train, 'test': r2_test}
```



```
trained_models[model_name] = model
```

The calculated metrics for each model are stored in their respective dictionaries, along with the trained model itself for potential future use.

8. Prepare Results for DataFrame

```
results_list = []
for model_name in mse_results.keys():
    results_list.append({
        'model_name': model_name,
        'mse_train': mse_results[model_name]['train'],
        'mse_test': mse_results[model_name]['test'],
        'mae_train': mae_results[model_name]['train'],
        'mae_test': mae_results[model_name]['test'],
        'rmse_train': rmse_results[model_name]['train'],
        'rmse_test': rmse_results[model_name]['test'],
        'r2_train': r2_results[model_name]['train'],
        'r2_test': r2_results[model_name]['test']
    })
```

## Results

	model_name	mse_train	mse_test	mae_train	mae_test	rmse_train	rmse_test	r2_train	r2_test
4	BayesianRidge	2.023923	2.012806	1.215402	1.210960	1.422646	1.418734	0.999981	0.999982
6	RANSACRegressor	2.024072	2.012977	1.215194	1.210726	1.422699	1.418794	0.999981	0.999982
0	LinearRegression	2.024227	2.013122	1.215422	1.210981	1.422763	1.418846	0.999981	0.999982
1	Ridge	2.024178	2.014129	1.216942	1.212402	1.422736	1.419200	0.999981	0.999982
5	HuberRegressor	2.024412	2.014449	1.218235	1.214038	1.422819	1.419313	0.999981	0.999982
8	RandomForestRegressor	0.321134	2.142369	0.459879	1.212877	0.566887	1.483883	0.999997	0.999981
2	Lasso	3.025647	3.092090	1.439049	1.447748	1.739439	1.758434	0.999972	0.999972
7	DecisionTreeRegressor	0.042158	3.622379	0.033538	1.428755	0.205323	1.803255	1.000000	0.999967
9	GradientBoostingRegressor	5.857011	5.734523	1.901724	1.925587	2.378447	2.394868	0.999948	0.999948
10	AdaBoostRegressor	1710.661380	1709.179010	35.876054	35.811427	41.380142	41.342218	0.964358	0.964543
11	KNeighborsRegressor	1760.406534	2604.782399	22.718422	28.062655	41.857223	52.960196	0.963803	0.874635
12	SupportVectorRegressor	7371.402268	8862.182223	41.116885	40.801537	85.856871	82.838168	0.832596	0.937942
3	ElasticNet	12225.687212	12379.465858	95.701088	98.035208	110.568830	111.263048	0.868208	0.868046

Once again plotting predictions against actual values, as well as creating residual plots, serves several important purposes in evaluating the performance of regression models. Here's an explanation of why these visualizations are useful:

Visual Comparison of Predictions and Actual Values

```
plt.scatter(y_test, y_pred, color='blue', label='Predicted vs Actual')
```

```
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Ideal Line (y=x)')
```

Scatter Plot: This plot shows the predicted values ( $y_{\text{pred}}$ ) on the y-axis against the actual values ( $y_{\text{test}}$ ) on the x-axis. It helps visualize how closely the predictions align with the actual values.

Ideal Line: The red dashed line represents the ideal scenario where predicted values perfectly match actual values (i.e.,  $y = x$ ). Points close to this line indicate good model performance.

### Assessing Model Performance

By examining the scatter plot, you can easily identify:

Accuracy: How closely the predicted values follow the actual values.

Bias: If the points systematically fall above or below the ideal line, it indicates a bias in the predictions.

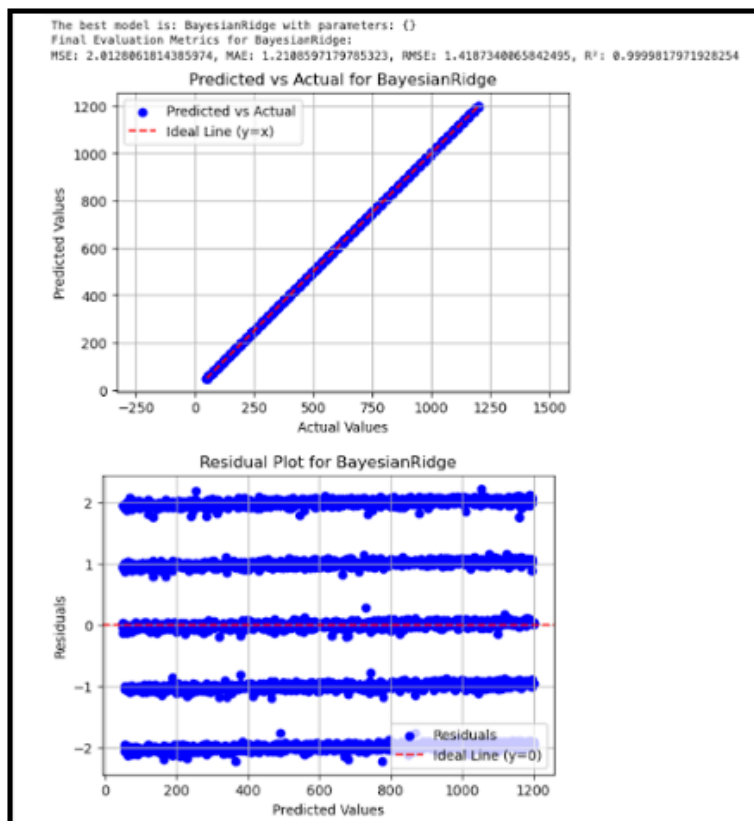
Variance: The spread of the points can indicate how well the model generalizes to unseen data.

The best model based on MSE is: BayesianRidge

The best model based on MAE is: BayesianRidge

The best model based on RMSE is: BayesianRidge

The best model based on R2 is: BayesianRidge



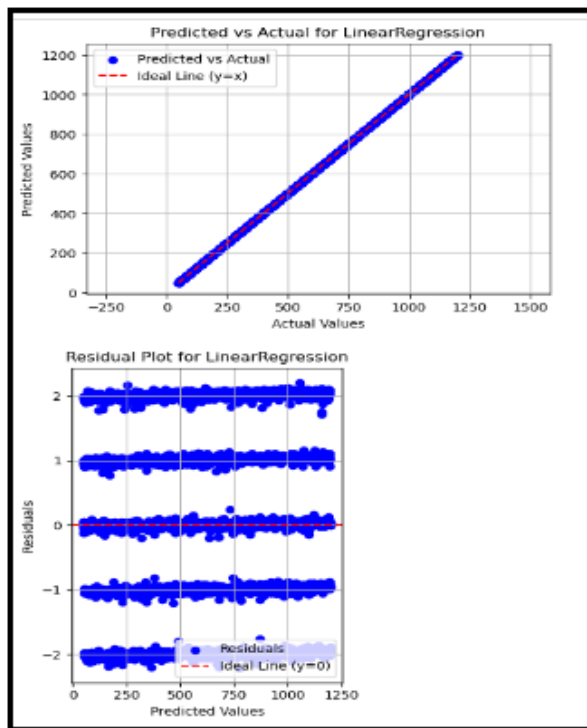
### 3. Residual Analysis

$$\text{residuals} = y_{\text{test}} - y_{\text{pred}}$$

## Full-Stack Airbnb Data Analysis and Application

Residuals: The difference between actual and predicted values. Analyzing residuals helps in understanding the model's performance.

```
plt.scatter(y_pred, residuals, color='blue', label='Residuals')  
plt.axhline(y=0, color='red', linestyle='--', label='Ideal Line (y=0)')
```



**Residual Plot:** This plot shows the residuals on the y-axis against the predicted values on the x-axis. It helps to Identify Patterns: Ideally, residuals should be randomly scattered around the horizontal line at 0, indicating that the model's errors are random and not systematic.

**Detect Non-Linearity:** If there are patterns in the residuals (e.g., a curve), it suggests that the model may not have captured some underlying relationships in the data.

**Check Homoscedasticity:** The spread of residuals should remain constant across all predicted values. If the spread increases or decreases, it indicates heteroscedasticity, which can violate regression assumptions. This particular dataset shows low homoscedasticity possible because factors like well-distributed input features, high model accuracy, effectiveness, and big attribute was the low noise in the data, which can help to ensure balanced errors across predictions.

Evaluate the trained and tested results

The evaluation results indicate that the model performs very similarly on both the training and test sets, suggesting it has a good fit and generalizes well:

Mean Absolute Error (MAE):

- Training MAE: 1.2154
- Test MAE: 1.2109

These values are nearly identical, indicating that the model makes similarly sized errors on both the training and test data. The average error is slightly over 1 unit, which shows a strong fit.

Mean Squared Error (MSE):

- Training MSE: 2.0239
- Test MSE: 2.0128

The MSE values are also very close, indicating that there's no major overfitting (the model isn't performing significantly better on the training data compared to the test data). The low MSE reflects that larger errors are rare.

R-squared ( $R^2$ ):

- Training  $R^2$ : 0.999981
- Test  $R^2$ : 0.999982

Both  $R^2$  values are extremely close to 1, meaning the model explains almost all of the variance in both the training and test datasets. This indicates a highly accurate model that predicts the target variable very well.

Patterns:

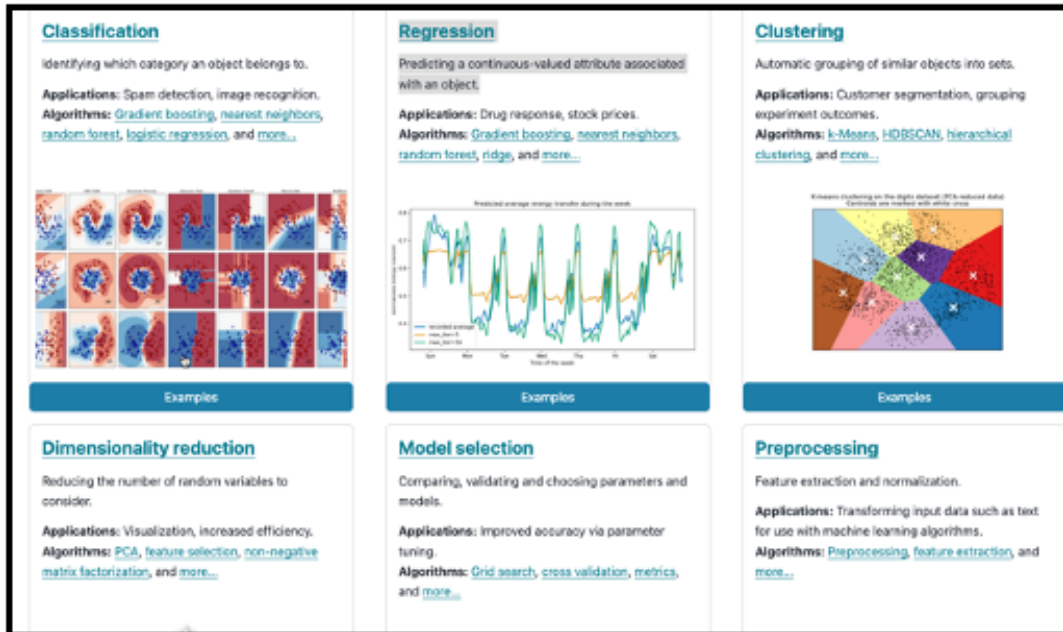
No Overfitting: Since the errors (MAE, MSE) and  $R^2$  values are almost identical for both training and test sets, the model generalizes well to unseen data.

High Accuracy: The near-perfect  $R^2$  values suggest that the model captures almost all of the underlying patterns in the data, leaving very little unexplained variance.

Small Prediction Errors: The MAE and MSE values are low, reflecting that the model makes small prediction errors consistently across both training and test sets.

Deep Dive into Regression

Our team chose `price` as the target given that Airbnb customer choice is driven by price in comparison with other lodging alternatives when choosing a location to travel to like NYC.



<https://scikit-learn.org/stable/index.html>

## 1. Data Manipulation and Numerical Operations

```
import pandas as pd
pd.set_option('display.max_columns', None)
import numpy as np
```

Pandas: A powerful library for data manipulation and analysis. It provides data structures like DataFrames for handling structured data.

Numpy: A library for numerical operations in Python, providing support for arrays and matrices, along with a collection of mathematical functions.

## 2. Visualization Libraries

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Matplotlib: A plotting library used for creating static, interactive, and animated visualizations in Python.

Seaborn: Built on top of Matplotlib, Seaborn provides a high-level interface for drawing attractive statistical graphics.

## 3. Preprocessing and Scaling

```
from sklearn.preprocessing import LabelEncoder, RobustScaler
from sklearn import preprocessing
```

LabelEncoder: Converts categorical labels into numerical format, which is essential for machine learning algorithms.

RobustScaler: A scaler that is robust to outliers, scaling features using statistics that are robust to outliers.

## 4. Machine Learning - Train/Test Split

```
from sklearn.model_selection import train_test_split
```

train\_test\_split: A function that splits the dataset into training and testing sets, which is crucial for evaluating the performance of machine learning models.

## 5. Pre-processing Tools

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

```
from sklearn.impute import SimpleImputer
```

```
from sklearn.decomposition import PCA
```

StandardScaler: Standardizes features by removing the mean and scaling to unit variance.

Pipeline: A tool for streamlining a series of data processing steps, ensuring that the same transformations are applied to both training and test data. (not use in this code, but important step worth mentioned) .

ColumnTransformer: Allows different preprocessing for different columns in a DataFrame.

SimpleImputer: Used for handling missing values by replacing them with a specified strategy (mean, median, etc.).

PCA (Principal Component Analysis): A dimensionality reduction technique that transforms the data into a lower-dimensional space while preserving as much variance as possible.

### 6. Importing Linear Regression Models

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet, BayesianRidge, HuberRegressor, RANSACRegressor
```

Various regression models from the sklearn.linear\_model module, including:

LinearRegression: Basic linear regression model.

Ridge: Linear regression with L2 regularization.

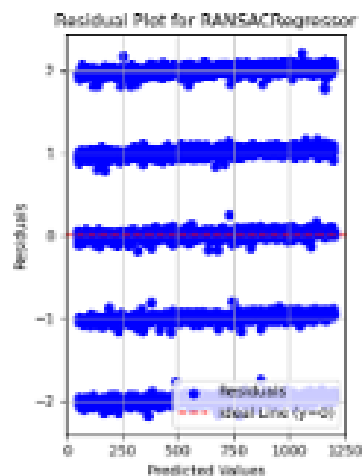
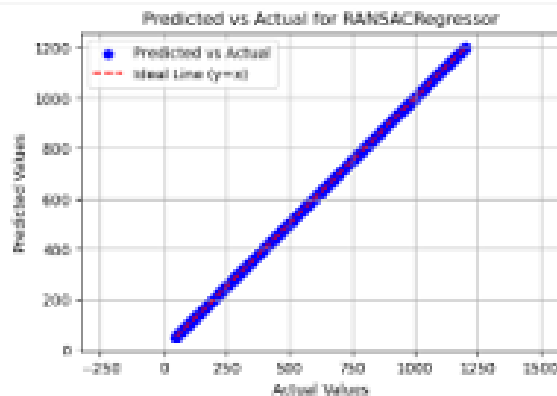
Lasso: Linear regression with L1 regularization.

ElasticNet: Combines L1 and L2 regularization.

BayesianRidge: A Bayesian approach to linear regression.

HuberRegressor: A robust regression model that is less sensitive to outliers.

RANSACRegressor: A regression model that iteratively fits the data to find inliers and outliers.



### 7. Importing Non-linear Regression Models

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
```

Non-linear regression models, including:

DecisionTreeRegressor: A decision tree-based regression model.

RandomForestRegressor: An ensemble method that uses multiple decision trees.

GradientBoostingRegressor: Builds models in a stage-wise fashion to improve accuracy.

AdaBoostRegressor: An adaptive boosting method for regression.

KNeighborsRegressor: A non-parametric method that predicts based on the nearest neighbors.

SVR (Support Vector Regression): A regression technique based on support vector machines.

8. Classification Models (Not needed for regression, however needed for other kind of ML experimentation)

```
from sklearn.naive_bayes import GaussianNB
from sklearn import tree # For Decision Trees
```

Importing classification models that may be useful depending on the analysis.

9. Metrics

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_absolute_percentage_error,
mean_squared_error
```

Refine the model for Cross validation and Inference

### Results Interpretation

- Training Set Results:
  - MAE: 1.2154
  - MSE: 2.0239
  - R<sup>2</sup>: 0.99998

These results indicate that the model performs exceptionally well on the training set, with a very low MAE and MSE, and an R<sup>2</sup> value extremely close to 1, suggesting that the model explains nearly all the variance in the training data.

### 2. Test Set Evaluation

```
test_predictions = best_model.predict(X_test_scaled)
test_mae = mean_absolute_error(y_test, test_predictions)
test_mse = mean_squared_error(y_test, test_predictions)
test_r2 = r2_score(y_test, test_predictions)
print("\nTest Set Evaluation:")
print(f"Mean Absolute Error: {test_mae}")
print(f"Mean Squared Error: {test_mse}")
print(f"R-squared: {test_r2}")
```

- Test Set Results:
  - MAE: 1.2109
  - MSE: 2.0128
  - R<sup>2</sup>: 0.99998

- The very low MAE and MSE values suggest that the model's predictions are very close to the actual values, indicating high accuracy.
- The  $R^2$  values close to 1 for both training and test sets suggest that the model explains almost all the variance in the data, indicating a strong fit.
- The slight differences in performance metrics between the training and test sets indicate that the model is not overfitting, as it performs well on both datasets.

### GridSearch

from sklearn.model\_selection import train\_test\_split, GridSearchCV

- train\_test\_split: This function is used to split the dataset into training and testing sets, but it is not explicitly shown in this snippet.
- GridSearchCV: This class performs an exhaustive search over specified hyperparameter values for a given model using cross-validation. It helps in finding the best combination of hyperparameters to improve model performance.

#### 2. List of Models and Their Hyperparameter Grids

```
models = {
    "LinearRegression": (LinearRegression(), {}),
    "Ridge": (Ridge(), {'alpha': [0.1, 1.0, 10.0]}),
    ...
    "SupportVectorRegressor": (SVR(), {'C': [0.1, 1, 10], 'epsilon': [0.1, 0.2]})
}
```

This dictionary defines a list of regression models along with their corresponding hyperparameter grids. The hyperparameter grids specify the values to be tested during the grid search. For example, for the Ridge regression model, the alpha parameter will be tested with values 0.1, 1.0, and 10.0.

#### 3. Dictionary to Hold the Best Models and Their Performance

```
results = []
```

An empty list is initialized to store the results of the model evaluations, including the best parameters and performance metrics for each model.

#### 4. Loop Through Each Model and Apply Grid Search

```
for model_name, (model, param_grid) in models.items():
```

```
    print(f"Evaluating {model_name}...")
```

```
    # Set up the Grid Search with cross-validation
```

```
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
                               scoring='neg_mean_squared_error', cv=5, n_jobs=-1, verbose=2)
```

```
    # Fit the Grid Search to the training data
```

```
    grid_search.fit(X_train_scaled, y_train)
```

This loop iterates through each model in the models dictionary. For each model:

A GridSearchCV object is created, specifying the model, hyperparameter grid, scoring method (negative mean squared error), number of cross-validation folds (cv=5), and parallel processing options (n\_jobs=-1).

The grid search is then fitted to the scaled training data (X\_train\_scaled and y\_train).

#### 5. Get the Best Model and Make Predictions

```
    best_model = grid_search.best_estimator_
```

```
    y_pred = best_model.predict(X_test_scaled)
```

After fitting, the best model (with optimal hyperparameters) is obtained using grid\_search.best\_estimator\_.



Predictions are made on the scaled test set (`X_test_scaled`) using the best model.

6. Calculate Metrics once again

```
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

7. Store the Results

```
results.append({
    'Model': model_name,
    'Best Parameters': grid_search.best_params_,
    'Mean Squared Error (MSE)': mse,
    'Mean Absolute Error (MAE)': mae,
    'Root Mean Squared Error (RMSE)': rmse,
    'R^2 Score': r2
})
```

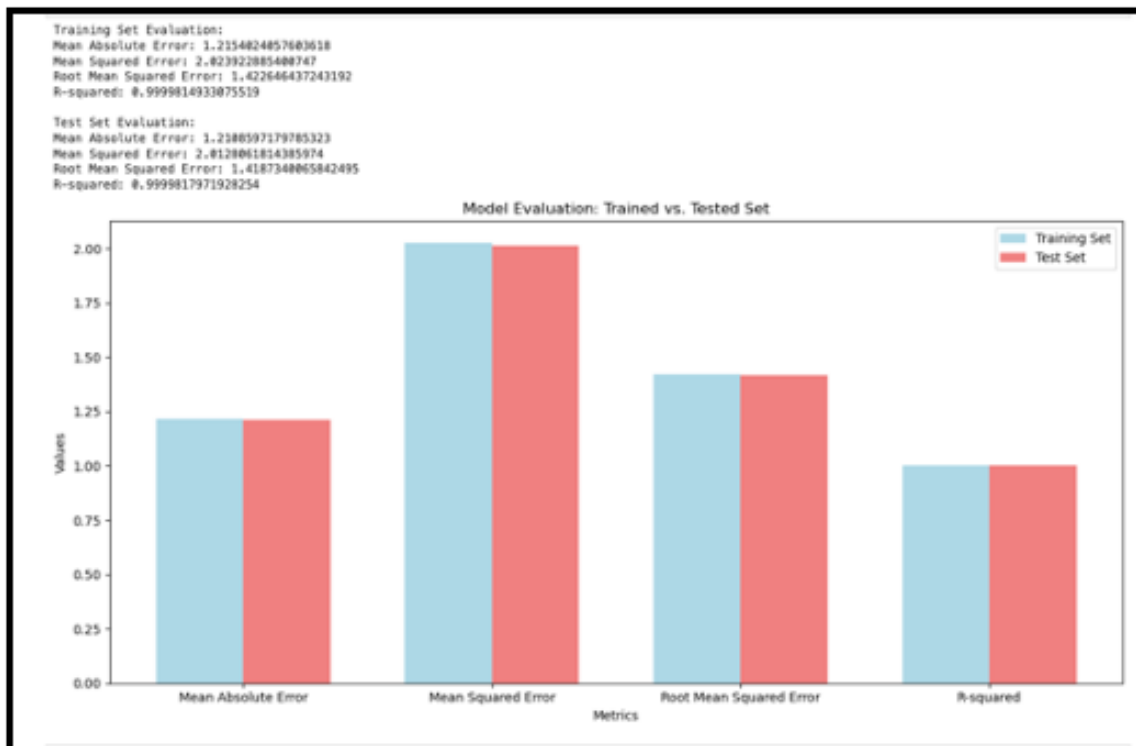
The results for each model, including the model name, best parameters, and performance metrics, are appended to the results list.

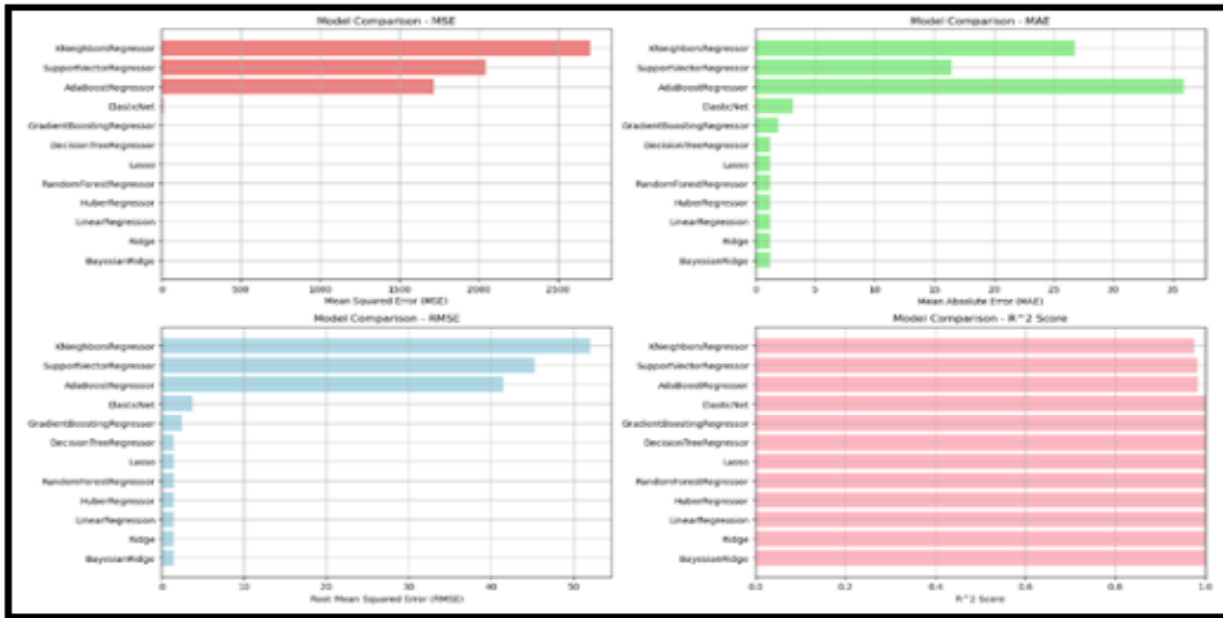
8. Create a DataFrame for Results

```
results_df = pd.DataFrame(results).sort_values(by='Mean Squared Error (MSE)')
print("\nModel Evaluation Results:")
print(results_df)
```

A DataFrame is created from the results list and sorted by MSE to easily compare model performances. The final results are printed, showing which models performed best based on the specified

### Comparison of Models & Metrix Performance





## Cross-validation

Predictions on the Validation Set

```
y_val_pred = best_model.predict(X_test_scaled) # X_val validation features
```

**Making Predictions:** The model (identified as `best_model` from previous steps) is used to make predictions on the validation set, which in this case is represented by `X_test_scaled`. Here, `X_test_scaled` serves as the features for the validation dataset.

**Validation Set:** This is typically a separate portion of the data that the model has not seen during training. It is used to evaluate how well the model generalizes to new, unseen data.

**Calculate Evaluation Metrics**

```
val_mse = mean_squared_error(y_test, y_val_pred) # y_val validation target
```

```
val_mae = mean_absolute_error(y_test, y_val_pred)
```

```
val_rmse = np.sqrt(val_mse)
```

```
val_r2 = r2_score(y_test, y_val_pred)
```

**Print Validation Metrics**

```
print("Validation Evaluation Metrics:")
```

```
print(f"MSE: {val_mse}, MAE: {val_mae}, RMSE: {val_rmse}, R2: {val_r2}")
```

The calculated metrics are printed to provide a summary of the model's performance on the validation set.

**Importance of Cross-Validation**

**Generalization:** Cross-validation helps in assessing how well the model generalizes to an independent dataset. By evaluating the model on a validation set that was not used during training, you can get a better estimate of its performance on unseen data.

**Model Selection:** Cross-validation allows for the comparison of multiple models or hyperparameter configurations. The model that performs best on the validation set can be selected as the final model.

**Avoiding Overfitting:** By using cross-validation, you can detect if a model is overfitting the training data. If the model performs significantly better on the training data than on the validation data, it may be capturing noise rather than the underlying pattern.

**Robustness:** Cross-validation provides a more robust estimate of model performance by averaging the results over multiple folds or splits of the data, reducing the variance associated with a single train-test split.

The provided cross-validation results indicate the performance of your regression model on the validation set. Here's a breakdown of each metric and what it signifies:

### Mean Squared Error (MSE)

Value: 2.0128

Interpretation: This metric measures the average squared difference between the actual target values and the predicted values. A lower MSE indicates better model performance. In this case, an MSE of approximately 2.01 suggests that the model's predictions are quite close to the actual values, but there is still some error present.

### Mean Absolute Error (MAE)

Value: 1.2109

Interpretation: MAE measures the average absolute difference between the actual and predicted values. It provides a straightforward interpretation of the average error in the same units as the target variable. An MAE of around 1.21 indicates that, on average, the model's predictions are off by about 1.21 units from the actual values, which is relatively low.

### Root Mean Squared Error (RMSE)

Value: 1.4187

Interpretation: RMSE is the square root of MSE and provides a measure of error in the same units as the target variable. An RMSE of approximately 1.42 suggests that the model's predictions deviate from the actual values by about 1.42 units on average. RMSE is particularly sensitive to larger errors due to the squaring of differences.

### R-squared ( $R^2$ )

Value: 0.99998

Interpretation:  $R^2$  indicates the proportion of variance in the dependent variable that can be explained by the independent variables in the model. An  $R^2$  value of 0.99998 is extremely high, suggesting that the model explains nearly all the variance in the validation set. This indicates a very good fit, meaning the model is effectively capturing the underlying relationship in the data.

## Preparing for Deployment

```
Importing the Pickle Library  
import pickle
```

Pickle: This is a Python library used for serializing and deserializing Python objects. In the context of machine learning, it is commonly used to save trained models to disk so that they can be reused later without needing to retrain them.

## Full-Stack Airbnb Data Analysis and Application

### Saving the Trained Model

with open('trained\_model.pkl', 'wb') as file:

```
pickle.dump(best_model, file)
```

```
print("Model saved successfully!")
```

Opening a File: The open function is used to create a new file named trained\_model.pkl in write-binary mode ('wb').

Dumping the Model: The pickle.dump function is used to serialize the best\_model object (the trained machine learning model) and save it to the file. This allows the model to be stored in a format that can be easily loaded later.

Confirmation Message: A message is printed to confirm that the model has been saved successfully.

### Loading the Trained Model

with open('trained\_model.pkl', 'rb') as file:

```
loaded_model = pickle.load(file)
```

Opening the File: The open function is used again, this time in read-binary mode ('rb'), to open the previously saved model file.

Loading the Model: The pickle.load function is used to deserialize the model from the file, restoring it to the variable loaded\_model. This allows you to use the model without needing to retrain it.

### Making Predictions with the Loaded Model

```
predictions = loaded_model.predict(new_data_scaled)
```

Making Predictions: The loaded model (loaded\_model) is used to make predictions on new data (new\_data\_scaled). This demonstrates that the model can be reused after being saved.

### Output of Predictions

```
predictions
```

The output shows the predictions made by the loaded model. In this case, it returns an array with a single value, approximately 498.96. This value represents the model's predicted output for the input data provided in new\_data\_scaled.

### Saving and Loading Models provides...

Efficiency: Saving trained models allows you to avoid the time and computational resources required to retrain the model each time you want to use it. This is especially important for complex models or large datasets.

Reproducibility: It ensures that you can reproduce results consistently. When you save a model, you can load it later and expect it to perform the same way on the same input data.

Deployment: In production environments, saving models is essential for deploying machine learning applications. Once a model is trained and validated, it can be saved and integrated into applications for real-time predictions.

### Logging

```
import logging
```

This is a built-in Python module that provides a flexible framework for emitting log messages from Python programs. It is commonly used for tracking events that happen during execution, which can be helpful for debugging and monitoring purposes.

### Setting Up Logging

```
logging.basicConfig(filename='model_performance.log', level=logging.INFO)
```

Basic Configuration: The basicConfig function is used to configure the logging system. In this case:

filename: Specifies the name of the log file where log messages will be saved (model\_performance.log).

level: Sets the logging level to INFO, which means that all messages at this level and above (WARNING, ERROR, CRITICAL) will be logged. This helps filter out less important messages (like DEBUG messages).

### Logging Predictions and Actual Values

```
logging.info(f"Predictions: {predictions}, Actual: {y_val}")
```

Logging Information: This line logs a message at the INFO level, which includes the model's predictions and the actual values (y\_val) for comparison.

String Formatting: The f-string is used to format the message, embedding the values of predictions and y\_val directly into the log message.

## Monitoring Model Performance

**Tracking Performance:** Logging predictions and actual values helps in monitoring how well the model performs over time. This is particularly useful in production environments where models may be updated or retrained periodically.

**Debugging:** If the model's predictions are not as expected, having a log of previous predictions and actual values can help identify issues or patterns that need to be addressed.

**Model Drift Detection:** Over time, the performance of a model may degrade due to changes in the underlying data distribution (a phenomenon known as "model drift"). By logging performance metrics and predictions, you can track changes and determine when a model may need retraining or adjustment.

**Transparency and Accountability:** Keeping a log of model predictions and actual outcomes provides a record that can be reviewed later, which is important for accountability, especially in regulated industries.

## Recommender - Kade Rivers

A great addition to this project was to not only take into account the trends and current data available from the visualizations and machine learning, but to also utilize a recommender feature for travelers or hosts to be able to input features such as: neighborhood group or neighborhood, room type, price, minimum nights, and number of reviews and the use of KNeighbors to output the desired length of current 2024 listings into a DataFrame for the consumer to see the most closely related listings for the features they have input.

To make this recommender, only the 2024 dataset was used and cleaned to display specific features:

	rental_id	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews
0	1189243425411300671	Manhattan	Murray Hill	Entire home/apt	58.0	30	1
1	651593916026998398	Brooklyn	Flatlands	Private room	80.0	30	0
2	310325	Manhattan	Harlem	Private room	75.0	30	31
3	572612125615500056	Brooklyn	Sunset Park	Private room	45.0	30	6
4	1020282701018874374	Brooklyn	Bedford-Stuyvesant	Private room	47.0	30	0

After the cleaning of the 2024 dataset, a function was created called “recommendation” which required input from the user for their preferred list length, the neighborhood group, the room type, and the price, and after the user input, the response would output the closest recommendations into a data frame.

```
# User input closest_rentals_length, neighbourhood_group, room_type, price
closest_rentals_length = 10
neighbourhood_group = "Brooklyn"
room_type = "Private room"
price = 80

response = recommendation(closest_rentals_length, neighbourhood_group, room_type, price) # used for the flask app, returns JSON

# to test
pd.DataFrame(response)
```

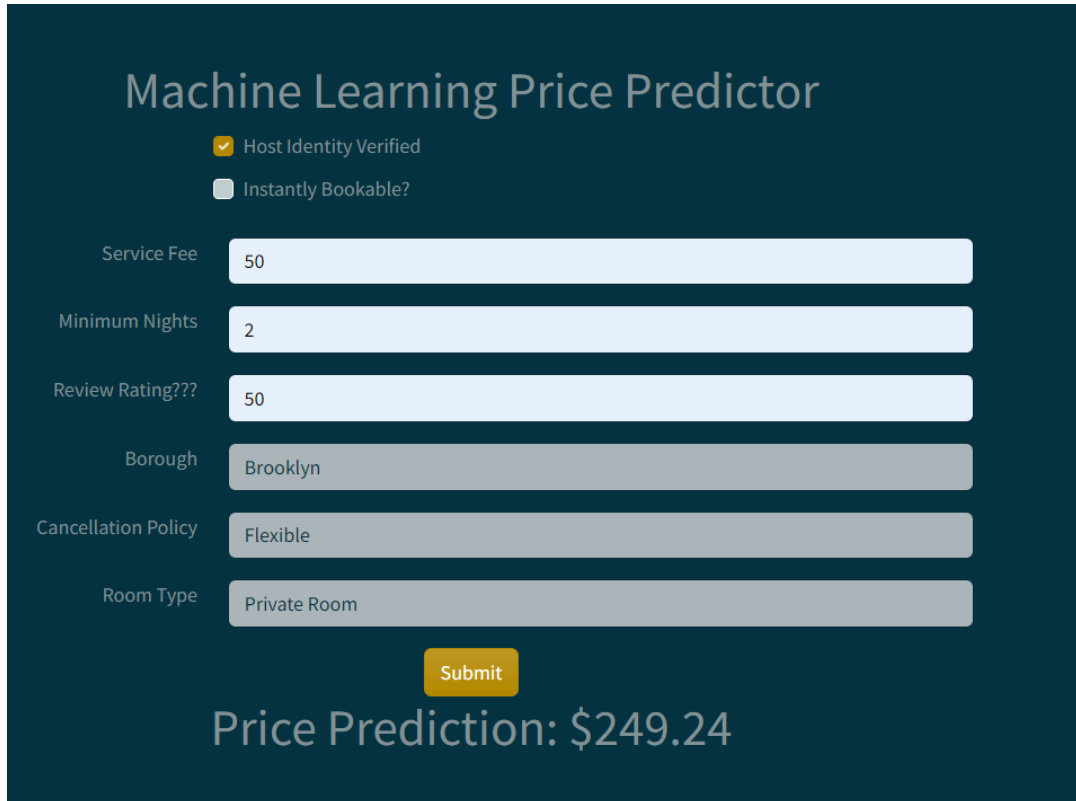
	rental_id	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	distance
0	17160286	Brooklyn	Bedford-Stuyvesant	Private room	100000.0	30	29	0.000000
1	605115521796576121	Brooklyn	Bedford-Stuyvesant	Private room	100000.0	30	9	0.000004
2	1004416200240866150	Manhattan	Upper East Side	Entire home/apt	15618.0	31	0	0.002764
3	904524100106225881	Manhattan	SoHo	Private room	20000.0	1	0	0.003593
4	904523946643334652	Manhattan	SoHo	Private room	20000.0	1	0	0.003593
5	904524223667771043	Manhattan	SoHo	Private room	20000.0	1	0	0.003593
6	52862058	Manhattan	Financial District	Entire home/apt	10000.0	30	1	0.006624
7	830682282357157632	Manhattan	Midtown	Entire home/apt	10000.0	30	0	0.006692
8	692813905111173363	Manhattan	Battery Park City	Entire home/apt	10000.0	30	0	0.006692
9	830656153550799267	Manhattan	Midtown	Entire home/apt	10000.0	30	0	0.006692

While the recommender is able to fully function in the Jupyter Notebook located in the GitHub Repository, unfortunately, the conflict of libraries between group members and PythonAnywhere prevented the proper unpickling of the data without breaking the Machine Learning portion of the web deployment, and ultimately the decision was made to utilize the Machine Learning Price Predictor to launch our project.

## Web Deployment

The web app for this project was deployed using Python Anywhere with the link provided toward the top of this write-up. The landing page is deployed and a navigation bar with tabs for:

About Us, Tableau 1, Tableau 2, Machine Learning, Recommender, and Works Cited are all included. The Tableau stories are both interactive as well as the Machine Learning Price Predictor pictured below with a price prediction:



The image shows a web form titled "Machine Learning Price Predictor" on a dark blue background. The form includes several input fields and checkboxes. At the top, there are two checkboxes: "Host Identity Verified" (checked) and "Instantly Bookable?" (unchecked). Below these are six input fields: "Service Fee" (50), "Minimum Nights" (2), "Review Rating???" (50), "Borough" (Brooklyn), "Cancellation Policy" (Flexible), and "Room Type" (Private Room). A yellow "Submit" button is located below the input fields. At the bottom, the text "Price Prediction: \$249.24" is displayed in a large, light-colored font.

Field	Value
Host Identity Verified	<input checked="" type="checkbox"/>
Instantly Bookable?	<input type="checkbox"/>
Service Fee	50
Minimum Nights	2
Review Rating???	50
Borough	Brooklyn
Cancellation Policy	Flexible
Room Type	Private Room

Submit

Price Prediction: \$249.24

While the recommender page looks similar, it is currently not interactive on the web application due to the lack of containerization of libraries and the conflicts that emerged with limited time to re-work the environments for this particular project.

### Bias/Limitations

While this project provides valuable insights, there are several limitations. The dataset only includes active Airbnb listings, which means the analysis might not reflect inactive or seasonal listings. Additionally, external factors like local events, policy changes, or global events (such as COVID-19) are not reflected in the dataset but can have a significant impact on the Airbnb market. Another source of potential bias is the lack of socioeconomic data tied to neighborhoods, which could affect the interpretation of price trends. For example, high prices in

certain neighborhoods might be driven by luxury markets rather than overall economic health. This dataset was also narrowed to the location of New York City and does not indicate overall trends in pricing for Airbnb.

The limitations of time constraint and lack of containerization for library consistency ultimately hindered the full deployment of the web application to ensure the Price Predictor and Recommender were both fully functioning, however, this limitation will be easily solvable in future projects now that the knowledge has been gained.

### **Conclusions and Future Work**

While Airbnb has changed the way we travel for business or pleasure, it's also called for updated regulations and laws to ensure its continuation and the verification of hosts and safety for renters. With the open source data available, we were able to visualize the geographic locations of rentals, discover important and impactful features of top performing rentals and neighborhoods as well as use machine learning to make predictions on the prices of rentals which can be used by consumers or future hosts. This information was able to be uploaded and usable in Python Anywhere. The recommender is an additional feature which can be implemented after the completion of this project with additional time and repurposing in order to adjust the libraries conflict that was encountered.

The options are limitless when it comes to adding onto this project and continuing on with additional datasets, other short-term rental companies like Vrbo, further insight into how often each rental is rented out throughout the year, etc... This was a fantastic project to show the skills we have learned and how we can transform, visualize, and predict using data.



### Collaboration Credits

Alexander Booth – Bootcamp Instructor  
Antonio Laverghetta Jr. - Bootcamp TA  
Christopher Madden Bootcamp TA  
Justin Moore – Bootcamp Central Tutor Support  
Kourt Bailey – Bootcamp TA & Central Tutor Support  
Sharon Colson – Bootcamp Central Tutor Support  
Anna Poulakos – Software Engineer

### Works Cited

Original Dataset:

<https://www.kaggle.com/datasets/arianazmoudeh/airbnbopendata>

Additional Datasets:

<https://insideairbnb.com/explore/>

Color Palette:

<https://coolors.co/ff5a5f-00a699-fc642d-484848-767676>

Slides Template:

<https://slidechef.net/templates/free-airbnb-pitch-deck-template/>

Web App Theme:

<https://bootswatch.com/solar/>

### Additional References

MSE vs RMSE vs MAE vs MAPE vs R-Squared: When to Use?

<https://vitalflux.com/mse-vs-rmse-vs-mae-vs-mape-vs-r-squared-when-to-use/>

Logging – Logging facility for Python

<https://docs.python.org/3/library/logging.html>

Regression

<https://scikit-learn.org/stable/index.html>

OpenAI. "ChatGPT." OpenAI, 2024, <https://www.openai.com/chatgpt>.

Xpert Learning Assistant Bootcamp: DATA-PT-EAST-APRIL-041524-MTTH-CONS

Sayan Roy · 2y ago · 5,217 views

<https://www.kaggle.com/code/sayanroy729/airbnb-open-data-eda-step-by-step>

Alexander Booth

Bootcamp activities

[https://git.bootcampcontent.com/boot-camp-consortium-east-coast/DATA-PT-EAST-APRIL-041524/-/tree/main/01-Lesson-Plans/20-Supervised-Learning/1/Activities/07-BOOTH\\_Does\\_Rocks?ref\\_type=heads](https://git.bootcampcontent.com/boot-camp-consortium-east-coast/DATA-PT-EAST-APRIL-041524/-/tree/main/01-Lesson-Plans/20-Supervised-Learning/1/Activities/07-BOOTH_Does_Rocks?ref_type=heads)

Airbnb Price Prediction (ML)

<https://www.kaggle.com/code/abdelrahmanraslan/airbnb-price-prediction-ml>