Machine Learning I - Term Project Report

AirBnB Property Price Prediction

Salman Malik - ERP: 27256

Daniyal Akbar - ERP: 25363

May 29, 2023

Contents

Abstract

This project aimed to develop a machine learning model to predict the prices of Airbnb listings. The dataset used for the analysis consisted of housing data, including features such as location, number of rooms, availability, and reviews. The project followed a systematic approach, including data preprocessing, feature engineering, model selection, and evaluation.

The dataset was preprocessed by handling missing values, encoding categorical variables, and performing feature scaling. Exploratory data analysis was conducted to gain insights into the data distribution and relationships between variables. Feature engineering techniques were applied to extract meaningful information from the data, including one-hot encoding for categorical variables.

Multiple regression models were evaluated, including linear regression, random forest regression, gradient boosting regression, and XGBoost regression. The models were trained and evaluated using various performance metrics such as mean squared error (MSE) and coefficient of determination (R-squared). Hyperparameter tuning was performed using techniques such as grid search and cross-validation to optimize the model's performance. Additionally, feature importance analysis using methods like LIME and SHAP was conducted to gain insights into the model's decision-making process.

The results showed that the Gradient Boosting Regressor model (Stacking) achieved the highest R-squared value of 0.8645 on the training data, indicating a strong fit.

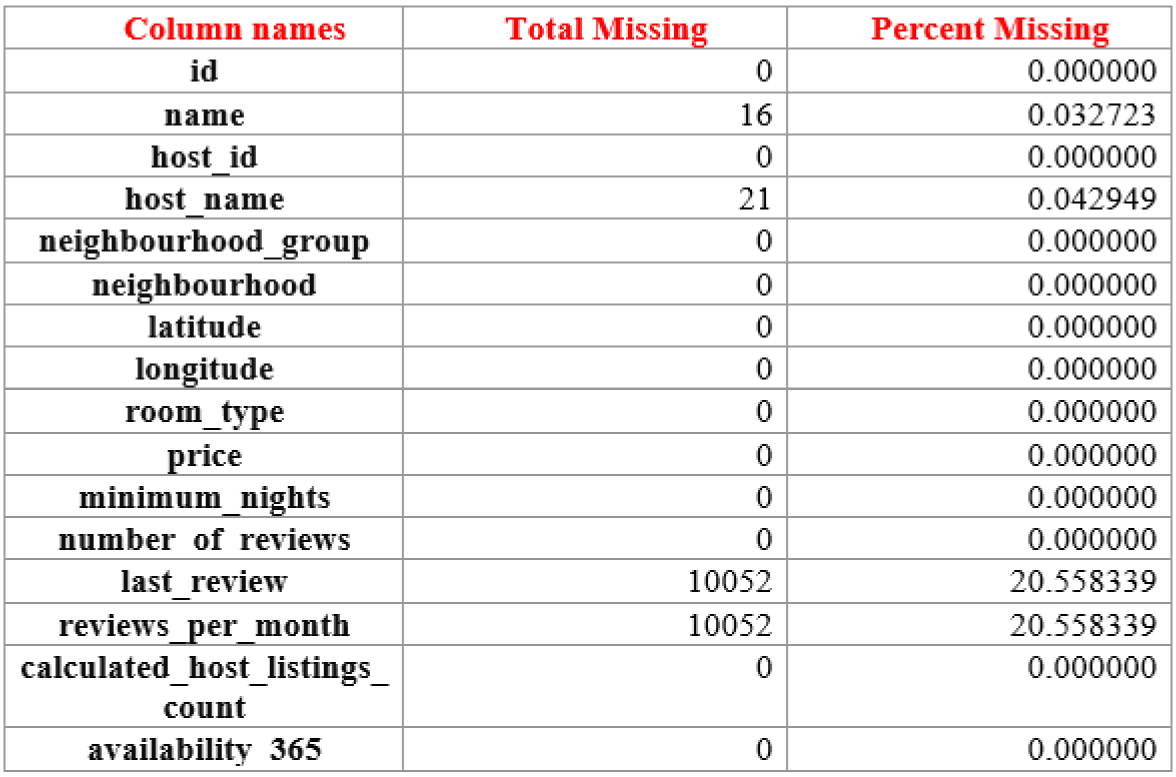
In conclusion, the developed machine learning model demonstrated promising predictive capabilities for estimating Airbnb listing prices. The findings highlight the importance of feature selection, model tuning, and interpretability techniques in improving the model's performance and understanding its behavior.

GitHub Repo Links:
Salman Malik: https://github.com/imsalmanmalik/XAI-PredictiveML-With-Streamlit.git
Danyal Akbar: https://github.com/DaniyalAkbar/streamlit_cost_prediction.git

1 Introduction

1.1 Data Loading

In this section of the code, we're importing a dataset into the Python environment using the pandas library. The data file, AB_NYC_2019.csv, is being read and loaded into two pandas DataFrame objects: df_orig and df. The Dataset contains 48895 rows and 16 columns.



Column names	Total Missing	Percent Missing
id	0	0.000000
name	16	0.032723
host_id	0	0.000000
host_name	21	0.042949
neighbourhood_group	0	0.000000
neighbourhood	0	0.000000
latitude	0	0.000000
longitude	0	0.000000
room_type	0	0.000000
price	0	0.000000
minimum_nights	0	0.000000
number_of_reviews	0	0.000000
last_review	10052	20.558339
reviews_per_month	10052	20.558339
calculated_host_listings_count	0	0.000000
availability_365	0	0.000000

Figure 1: Nulls in the data.

1.2 Data Inspection and Cleaning

In this section, we're inspecting the dataset for missing and unique values, performing data cleaning tasks, and transforming the data to make it more suitable for analysis. The first part of this section calculates the total number and percentage of missing values in each column of the dataframe, using df.isnull().sum() to count missing values, and len(df) to find the total number of entries. These calculations are then combined into a new dataframe, missing_data, for easy review.
Next, the code calculates the number and percentage of unique values in each column, again storing the results in a new dataframe, unique_data. The dataframe is then modified:
The 'id', 'host_id', 'host_name', and 'name' columns are dropped, as they are assumed to be non-essential for the subsequent analysis.

The 'last_review' column is converted from string type to DateTime using pd.to_datetime(), enabling date-based operations and comparisons.
Missing values in the 'reviews_per_month' column are replaced with 0 using fillna(0). This might be under the assumption that a missing review count means no reviews were made. The 'last_review' column is also dealt with missing values, filling them with the earliest date in the column.
It is then transformed from DateTime objects to integer representations of dates, or ordinal dates, by subtracting the ordinal date of the earliest review from all reviews. Finally, the script recalculates the number and percentage of missing values in each column, after the cleaning and transformation steps, and stores this information in missing_data.

2 Exploratory Data Analysis

This section of the code generates a scatter plot using the seaborn and matplotlib libraries. The scatter plot represents Airbnb listings in New York City, using their geographical coordinates for plotting. The 'latitude' and 'longitude' values of each listing are used as the y and x coordinates, respectively. Each point in the plot represents a single Airbnb listing. The 'neighbourhood_group' value of each listing is used to color-code the points. This allows the visualization to show not only the distribution of Airbnb listings across the city, but also the distribution of listings among different neighborhood groups.

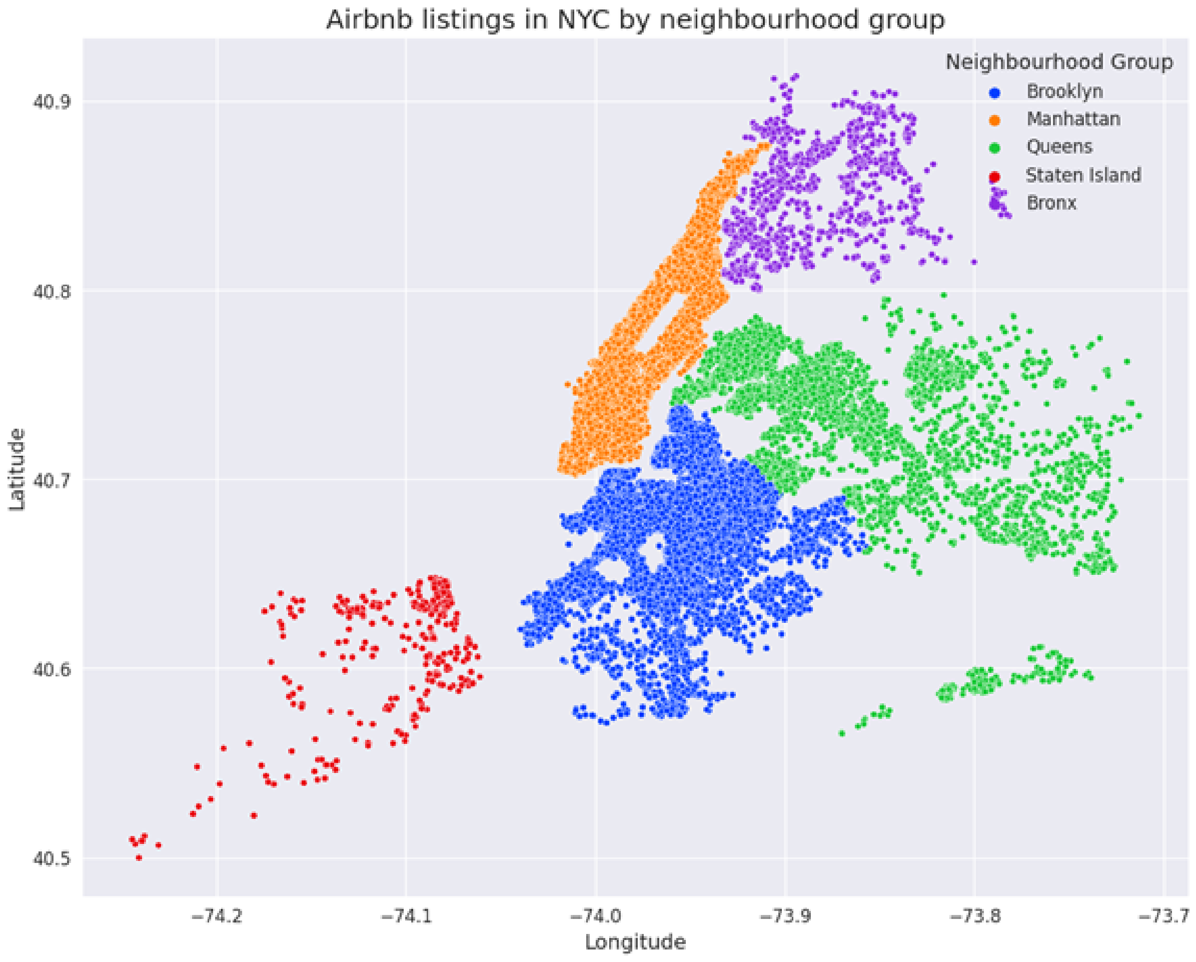


Figure 2: Data classifid by neighborhoods.

2.1 Distribution of Prices of Airbnb listings

This section of the code generates a series of three plots which aim to analyze the distribution of prices of Airbnb listings in the dataset.
The first plot is a histogram that displays the distribution of the 'price' values in the dataframe df. From this plot, you can identify the overall distribution, central tendency,

dispersion, and skewness of the 'price' values.
The second plot is also a histogram, but it shows the distribution of the natural logarithm of 'price' values, where a constant of 1 is added to each value before taking the logarithm (to avoid issues with zero values). Logarithmic transformation is often used to make heavy-tailed or right-skewed distributions more normal-like, which can be useful when applying techniques that assume a normal distribution.
The third plot is a QQ-plot (Quantile-Quantile plot) of the log-transformed 'price' values. A QQ-plot is a scatterplot created by plotting two sets of quantiles against one another. If both sets of quantiles came from the same distribution, the points should fall approximately along the 45-degree line. The QQ-plot is used to check if the data is distributed in a certain way, often to verify the assumption of normality in the data.
The subplots are arranged in a row, and each subplot is labeled with a title and x-axis label for clarity. The QQ-plot is also labeled with a y-axis label. All this combined makes for a comprehensive inspection of the 'price' variable in the dataset.

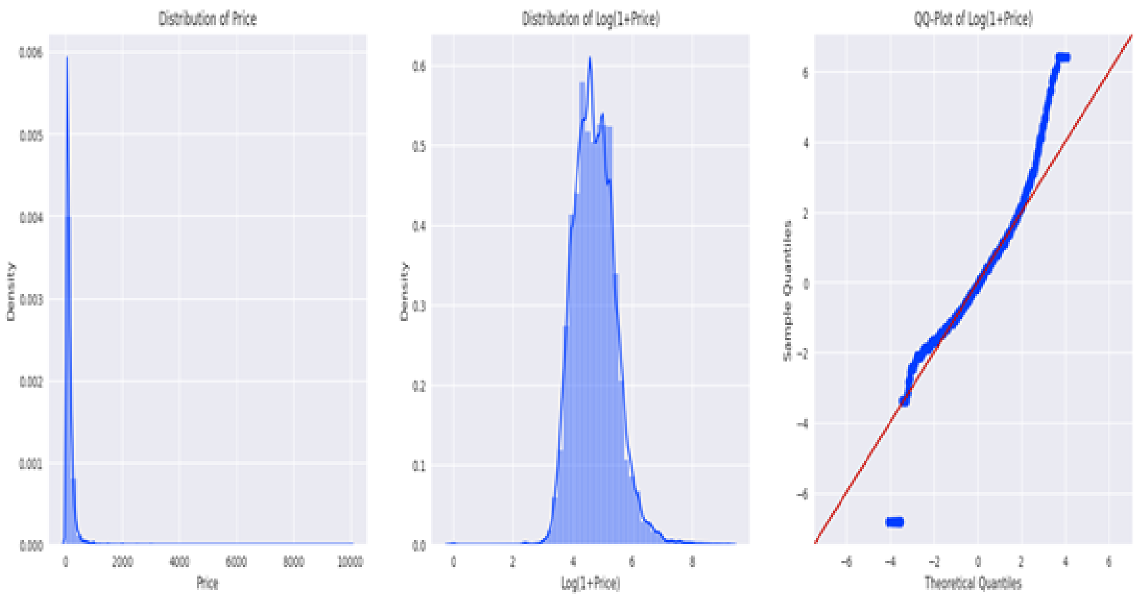


Figure 3: Distribution of Prices of Airbnb listings

2.2 Geographical Analysis of the Dataset

This section of the code is performing a geographical analysis on the dataset. Specifically, it's investigating the distributions of 'latitude' and 'longitude' values and displaying them as plots. Latitude and longitude are geographical coordinates that can represent the locations of Airbnb listings on the earth's surface.
The first plot is a histogram that shows the distribution of 'latitude' values in the dataframe df. This gives a sense of where along the north-south axis of the map the Airbnb listings tend to be concentrated.
The second plot is another histogram showing the distribution of 'longitude' values. This

provides an idea of where along the east-west axis the Airbnb listings are concentrated. The third plot is a scatter plot, where each point's coordinates are determined by a listing's 'latitude' (x-axis) and 'longitude' (y-axis). This plot essentially provides a geographical distribution of the Airbnb listings.

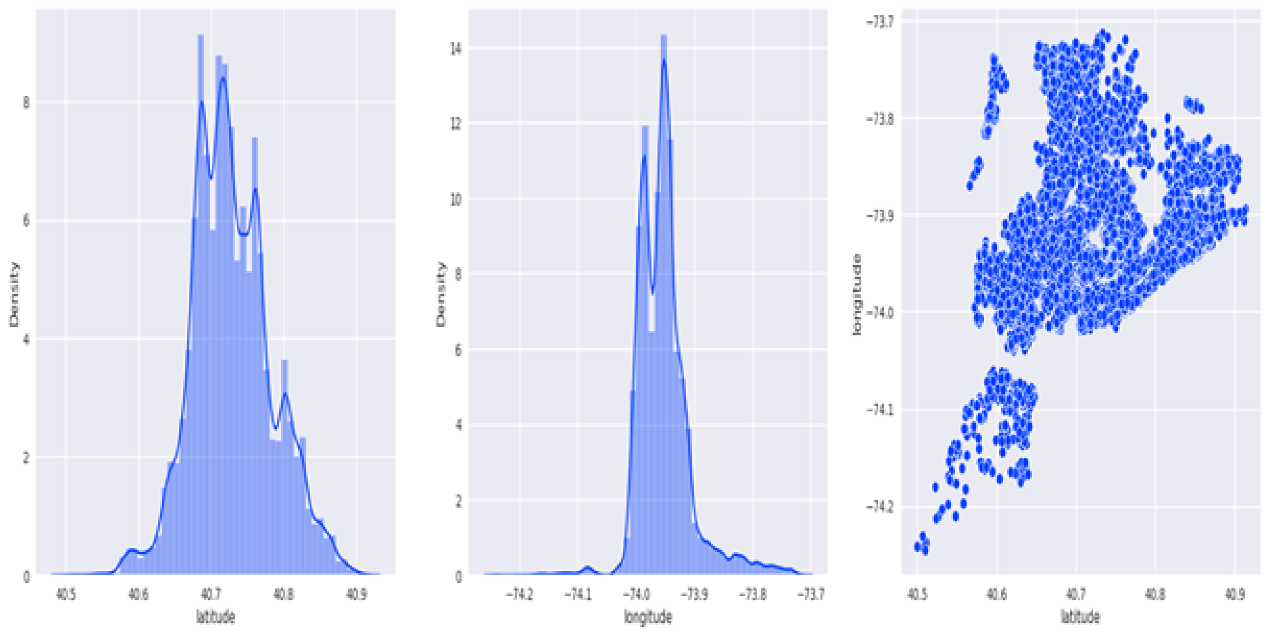


Figure 4: Geographical Analysis of the Dataset

2.3 Availability vs Reviews per Month

This section of the code generates a scatterplot with the 'availability_365' variable on the x-axis and 'reviews_per_month' on the y-axis. Each point on the scatterplot represents an Airbnb listing from the dataset.
The 'availability_365' variable represents the number of days in a year that a listing is available for booking. The 'reviews_per_month' variable, as its name suggests, represents the number of reviews a listing receives each month.
This scatterplot aims to visualize and potentially identify any relationship or correlation between the availability of Airbnb listings throughout the year and the frequency of reviews they receive per month. For instance, one might expect that listings that are available more often might receive more reviews, as they would have more guests.
However, this is not necessarily the case, as other factors can influence the frequency of reviews, such as the quality of the listing or the propensity of guests to leave reviews. This scatterplot will provide a visual exploration of these variables' potential relationship.

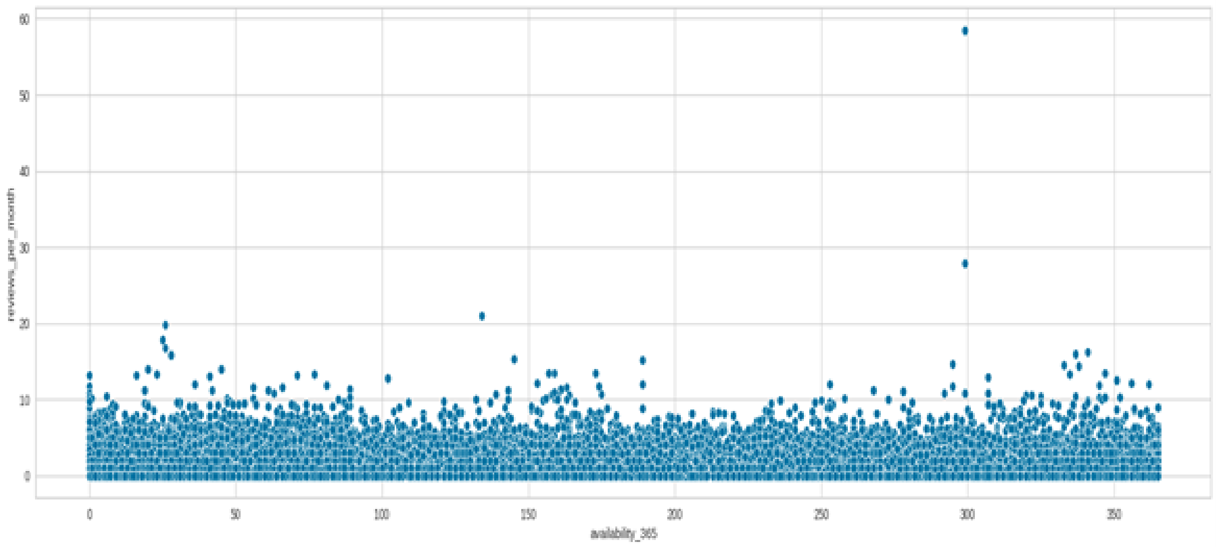


Figure 5: Availability vs Reviews per Month

2.4 Correlation Plot

This section of the code creates a correlation matrix heatmap, which is a visual representation of the correlation between different variables in the dataframe df. The df.corr() function calculates the pairwise correlation of all columns in the dataframe, returning a correlation matrix. A correlation matrix is a table showing the value of the correlation coefficient (Correlation coefficients range from -1 to 1, where -1 indicates a strong negative correlation, +1 indicates a strong positive correlation, and 0 indicates no correlation) between sets of variables. Each attribute of the dataset is correlated with all other attributes.

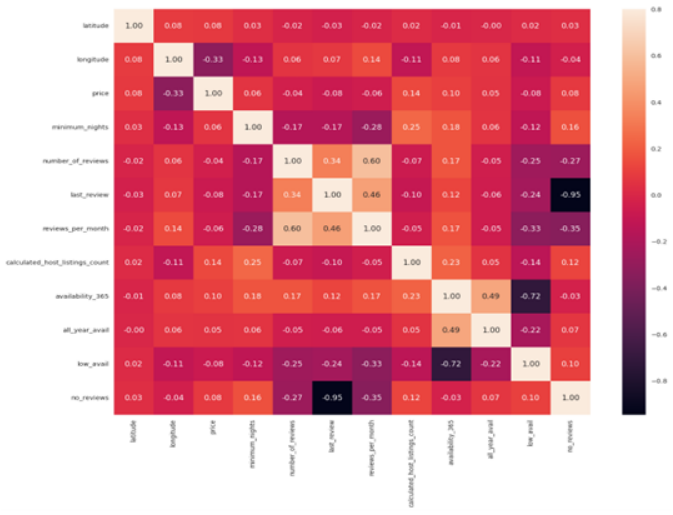


Figure 6: Correlation Heatmap

3 Feature Engineering

3.1 One-Hot Encoding of Categorical Variables

The selected data is stored in the variable categorical_features. The shape of this new dataframe is printed to give an idea of the number of categorical features (columns) and records (rows) we have.
The pd.get_dummies function is then used to perform one-hot encoding on these categorical features. This function converts categorical variable(s) into dummy/indicator variables for each unique category. In the resulting dataframe, each row corresponds to a record in the original dataframe, and each column corresponds to a unique category in the original categorical variable. The values in this dataframe are 0s and 1s, with 1 indicating that a record belongs to that category.
The new dataframe is stored in categorical_features_one_hot, and the first five rows are displayed using the head function to inspect the result of the transformation.

3.2 Preparing Data for Machine Learning Models

The target variable price is then separated from the numerical features and stored in y. The price column is dropped from the numerical_features dataframe using the drop function.
The numerical and one-hot encoded categorical features are then concatenated to form the full feature set X. This is done using the np.concatenate function for the numpy array X, and the pd.concat function for the dataframe X_df.
Finally, X_df and y are concatenated to form the final processed dataframe Processed_data, which contains all the preprocessed features and the target variable. This processed dataframe is saved to a .dat file named 'NYC_Airbnb_Processed.dat' using the to_csv function.

4 Applying Machine Learning Models

This section focuses on assessing the performance of different models using cross-validation and evaluating their predictive capabilities using various metrics such as R-squared (r2) score and root mean squared error (RMSE).

Model: Linear Regression, Ridge, Lasso, ElasticNet, RandomForestRegressor, XGBRegressor, HuberRegressor.

Interpretation:

performs cross-validation using the rmse_cv_r function for multiple regression models. The output shows the average RMSE and its standard deviation for each model. Lower RMSE values indicate better performance, so models with lower average RMSE and smaller standard deviation are preferred. Among the models, XGBRegressor and RandomForestRegressor show relatively better performance based on the RMSE values.

Model

Linear Regression, Ridge, Lasso, ElasticNet, RandomForestRegressor, XGBRegressor.

Output:

- LinearRegression: 17343533157159694227721024.00000 +/- 19042661515419211729469944.000
- Ridge: 0.19338 +/- 0.002994
- Lasso: 0.45868 +/- 0.003121
- ElasticNet: 0.45868 +/- 0.003121
- RandomForestRegressor: 0.17617 +/- 0.003449
- XGBRegressor: 0.17382 +/- 0.002767

Interpretation:

performs cross-validation using the rmse_cv_s function for multiple regression models. The output presents the average RMSE and its standard deviation for each model. lower RMSE values indicate better performance. Among the models, XGBRegressor, Ridge, and RandomForestRegressor exhibit relatively better performance based on the RMSE values.

Model: XGBRegressor

Output:

- r2 scores: [-0.037144722159469845, -0.043860175915658495, -0.027303277690762018, -0.029202139121270676, -0.03681140824865725]
- Average r2 score: -0.03486434462716366

Interpretation:

performs cross-validation using the XGBRegressor model and calculates the R-squared (r2) score for each fold. The r2 scores measure the proportion of the variance in the target variable that is predictable from the features. The negative r2 scores indicate that the model performs poorly in predicting the target variable. The average r2 score across all folds is -0.0349, indicating overall poor performance.

Model: Linear Regression

Output:

r2 scores: [-5.117273338547359e+25, -3.2118440724031025e+25, -7.338609093633118e+23, -4.55303718654612e+24, -2.1289947304377044e+27]
Average r2 score: -4.435145605286237e+26

Interpretation:

performs cross-validation using the Linear Regression model and calculates the R-squared (r2) score for each fold. The highly negative r2 scores indicate that the model performs extremely poorly in predicting the target variable. The average r2 score across all folds is also highly negative, indicating very poor performance.

Model: Ridge

Output:

r2 scores: [-0.00574919097169202, -0.008086363629714155, -0.006020682512763331, -0.01031060630941516, -0.006840666137115425] Average r2 score: -0.0074015019121400185

Interpretation:

performs cross-validation using the Ridge model and calculates the R-squared (r2) score for each fold. The negative r2 scores indicate that the model has limited predictive capability. The average r2 score across all folds is -0.0074, indicating modest but still relatively weak performance.

Model: ElasticNet

Output:

r2 scores: [-8.737995406105092e-05, -7.816759111434202e-05, -4.243262215530841e-05, -9.743178651211437e-06, -0.00010039282480578393]
Average r2 score: -6.362323415753934e-05

Interpretation:

performs cross-validation using the ElasticNet model and calculates the R-squared (r2) score for each fold. The r2 scores are close to zero, indicating a very weak relationship between the features and the target variable. The average r2 score across all folds is also close to zero, suggesting poor overall performance.

Model: RandomForestRegressor

Output:

r2 scores: [-0.06333888965751466, -0.05700658264628, -0.05458892769997914,
-0.059930681801769525, -0.05464619916505775]
Average r2 score: -0.05790225619412022

Interpretation:

performs cross-validation using the RandomForestRegressor model and calculates the R-squared (r2) score for each fold. The negative r2 scores indicate that the model has limited predictive capability. The average r2 score across all folds is -0.0579, indicating modest but still relatively weak performance.

Model: Linear Regression, Ridge, Lasso, ElasticNet, RandomForestRegressor, XGBRegressor

Output

- linear r: 0.435004274281806 77
- ridge r: 0.4346108208307634
- lasso r: 0.6724420567105368
- elastic net r: 0.6718649573448198
- randomforest r: 0.4183441740499073
- xgb r: 0.4120362040701081

Interpretation:

trains several regression models using the X_train_r dataset and evaluates their predictions on the validation set (X_val_r). The output displays the root mean squared error (RMSE) for each model. Lower RMSE values indicate better predictive performance. Among the models, XGBRegressor and RandomForestRegressor achieve relatively lower RMSE values, suggesting better performance in predicting the target variable.

Model

Linear Regression, Ridge, Lasso, ElasticNet, RandomForestRegressor, XGBRegressor

Output

- linear s: 5666857890801.589
- ridge s: 0.6467829239822482
- lasso s: 0.6787572375106184
- elastic net s: 0.6787572375106184
- randomforest s: 0.5146659238284258
- xgb s: 0.5440981369891925

Interpretation:

trains several regression models using the X_train_s dataset and evaluates their predictions on the validation set (X_val_r). The output displays the root mean squared error (RMSE) for each model. Lower RMSE values indicate better predictive performance. Among the models, RandomForestRegressor achieves a relatively lower RMSE value, suggesting better performance in predicting the target variable compared to other models. However, it's worth noting that the RMSE values for Lasso, ElasticNet, and XGBRegressor are relatively close to each other.

5 Hyperparameter Tuning and Feature Selection

In this analysis, we explore the concepts of hyperparameter tuning and feature selection using Lasso and Ridge regression. The goal is to find the best hyperparameters and identify the most relevant features for predicting the target variable.

5.1 Feature Selection with Lasso Regression (alpha = 0.02)

The code performs feature selection using Lasso regression with an alpha value of 0.02. It creates new datasets (X_train_r_lpo2 and X_val_r_lpo2) that contain only the selected features.

Output

- Linear regression: RMSE = 0.4725
- Ridge regression: RMSE = 0.4725
- Lasso regression: RMSE = 0.6724
- Elastic Net: RMSE = 0.6719
- Random Forest: RMSE = 0.4477
- XGBoost: RMSE = 0.4366

Interpretation:

Among the evaluated models, Linear Regression and Ridge Regression perform the best with similar RMSE values of 0.4725. Lasso regression and Elastic Net yield higher RMSE values, indicating weaker performance in this context. Random Forest and XGBoost models show promising results with lower RMSE values than linear-based models.

5.2 Feature Selection with Lasso Regression (alpha = 0.05)

This code block performs feature selection using Lasso regression with an alpha value of 0.05. It creates new datasets (X_train_r_lpo5 and X_val_r_lpo5) with the selected features.

Output

- Linear regression: RMSE = 0.4881
- Ridge regression: RMSE = 0.4881
- Lasso regression: RMSE = 0.6724
- Elastic Net: RMSE = 0.6719
- Random Forest: RMSE = 0.5339
- XGBoost: RMSE = 0.4632

Interpretation

The performance of Linear Regression and Ridge Regression slightly deteriorates, with RMSE values of 0.4881. Lasso regression and Elastic Net continue to exhibit weaker performance.

Random Forest and XGBoost models also show higher RMSE values compared to Code1, indicating less accurate predictions.

5.3 Feature Selection with Lasso Regression (alpha = 0.1)

Feature selection is performed using Lasso regression with an alpha value of 0.1. New datasets (X_train_r_lp1 and X_val_r_lp1) are created based on the selected features.

Output

- Linear regression: RMSE = 0.5018
- Ridge regression: RMSE = 0.5018
- Lasso regression: RMSE = 0.6724
- Elastic Net: RMSE = 0.6719
- Random Forest: RMSE = 0.5518
- XGBoost: RMSE = 0.4782

Interpretation

Linear Regression and Ridge Regression exhibit consistent performance with RMSE values of 0.5018. Lasso regression and Elastic Net still show weaker performance. Random Forest and XGBoost models also demonstrate slightly higher RMSE values, indicating moderate prediction accuracy.

5.4 Feature Selection with Lasso Regression (alpha = 0.5)

Feature selection is performed using Lasso regression with an alpha value of 0.5. New datasets (X_train_r_lp5 and X_val_r_lp5) are created based on the selected features.

Output

- Linear regression: RMSE = 0.6717
- Ridge regression: RMSE = 0.6717
- Lasso regression: RMSE = 0.6724
- Elastic Net: RMSE = 0.6719
- Random Forest: RMSE = 0.6452
- XGBoost: RMSE = 0.6452

Interpretation

In this code block, Linear Regression and Ridge Regression produce identical RMSE values of 0.6717. However, their performance is weaker compared to previous code blocks. Lasso regression and Elastic Net exhibit similar RMSE values as before. Random Forest and XGBoost models also show slightly higher RMSE values than in previous code blocks.

5.5 Feature Selection with Lasso Regression (alpha = 1)

Feature selection is performed using Lasso regression with an alpha value of 1. New datasets (X_train_r_l1 and X_val_r_l1) are created based on the selected features.

Output

- Linear regression: RMSE = 0.6717
- Ridge regression: RMSE = 0.6717
- Lasso regression: RMSE = 0.6724
- Elastic Net: RMSE = 0.6719
- Random Forest: RMSE = 0.6452
- XGBoost: RMSE = 0.6452

Interpretation

Linear Regression and Ridge Regression produce identical RMSE values of 0.6717. Lasso regression and Elastic Net exhibit consistent weaker performance. Random Forest and XGBoost models show slightly higher RMSE values, indicating moderate prediction accuracy.

5.6 Hyperparameter Tuning with Lasso Regression

In this code block, hyperparameter tuning is performed using Lasso regression. A grid search is conducted to find the best alpha value from a predefined set of values.

Output

- Best alpha: 0.05
- Best score: 0.26
- RMSE: 0.5103

Interpretation

The hyperparameter tuning process suggests that the best alpha value for Lasso regression is 0.05. The corresponding best score is 0.26, with an RMSE value of 0.5103. This indicates a relatively moderate predictive performance.

5.7 Hyperparameter Tuning with Ridge Regression

Hyperparameter tuning is performed using Ridge regression. A grid search is conducted to find the best alpha value from a predefined set of values.

Output

- Best alpha: 3.0
- Best score: 0.19
- RMSE: 0.4393

Interpretation

The hyperparameter tuning process suggests that the best alpha value for Ridge regression is 3.0. The corresponding best score is 0.19, with an RMSE value of 0.4393. This indicates relatively improved predictive performance compared to Lasso regression.

5.8 Hyperparameter Tuning with XGBoost Regression using Optuna

The code defines the objective function to optimize, which includes the hyperparameters to tune. The hyperparameters include the number of estimators, maximum depth, learning rate, subsample ratio, colsample ratio, regularization alpha and lambda, and other relevant parameters. The XGBoost model is trained and evaluated with the current set of hyperparameters. The RMSE is calculated and used as the optimization criterion.

5.9 XGBoost Model with Best Hyperparameters:

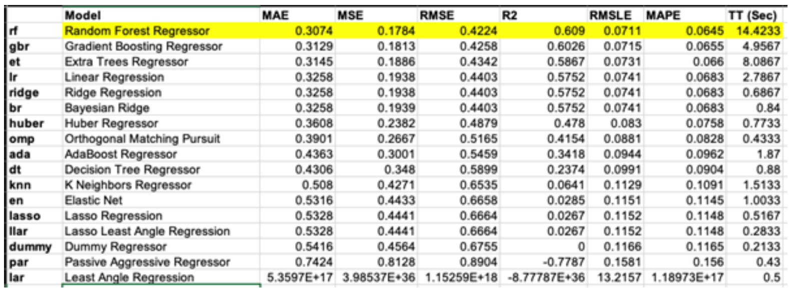
The code defines an XGBoost model with the best hyperparameters obtained from the hyperparameter search using Optuna. The model is initialized with the specified hyperparameters. XGBoost RMSE: The RMSE value obtained from evaluating the XGBoost model with the best hyperparameters on the validation dataset.The output shows the RMSE value obtained by evaluating the XGBoost model with the best hyperparameters on the validation dataset. This metric represents the model's performance in terms of prediction accuracy, with lower values indicating better performance. In this case, the XGBoost model achieves an RMSE of 0.4292, suggesting improved predictive.

5.10 Conclusion

The analysis demonstrates the impact of different alpha values on feature selection and predictive performance in Lasso and Ridge regression. It reveals that lower alpha values tend to result in better feature selection and prediction accuracy. **Ridge regression with an alpha value of 3.0** shows the best performance among the evaluated models, while Lasso regression performs comparatively weaker.

6 Automated Model Selection and Comparison using PyCaret

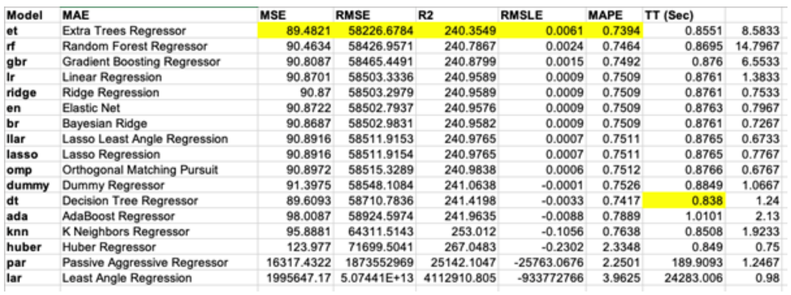
The highlighted output reveals the best model selected by PyCaret based on the evaluation metric used for the regression task. PyCaret compares various regression algorithms and selects the model with the highest performance. The specific details of the best model, such as the algorithm used, hyperparameters, and other relevant information, are not provided in the given code.



	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
rf	Random Forest Regressor	0.3074	0.1784	0.4224	0.609	0.0711	0.0645	14.4233
gbr	Gradient Boosting Regressor	0.3129	0.1813	0.4258	0.6026	0.0715	0.0655	4.9567
et	Extra Trees Regressor	0.3145	0.1886	0.4342	0.5867	0.0731	0.066	8.0867
lr	Linear Regression	0.3258	0.1938	0.4403	0.5752	0.0741	0.0683	2.7867
ridge	Ridge Regression	0.3258	0.1938	0.4403	0.5752	0.0741	0.0683	0.6867
br	Bayesian Ridge	0.3258	0.1939	0.4403	0.5752	0.0741	0.0683	0.84
huber	Huber Regressor	0.3608	0.2382	0.4879	0.478	0.083	0.0758	0.7733
omp	Orthogonal Matching Pursuit	0.3901	0.2667	0.5165	0.4154	0.0881	0.0828	0.4333
ada	AdaBoost Regressor	0.4363	0.3001	0.5459	0.3418	0.0944	0.0962	1.87
dt	Decision Tree Regressor	0.4306	0.348	0.5899	0.2374	0.0991	0.0904	0.88
knn	K Neighbors Regressor	0.508	0.4271	0.6535	0.0641	0.1129	0.1091	1.5133
en	Elastic Net	0.5316	0.4433	0.6658	0.0285	0.1151	0.1145	1.0033
lasso	Lasso Regression	0.5328	0.4441	0.6664	0.0267	0.1152	0.1148	0.5167
llar	Lasso Least Angle Regression	0.5328	0.4441	0.6664	0.0267	0.1152	0.1148	0.2833
dummy	Dummy Regressor	0.5416	0.4564	0.6755	0	0.1166	0.1165	0.2133
par	Passive Aggressive Regressor	0.7424	0.8128	0.8904	-0.7787	0.1581	0.156	0.43
lar	Least Angle Regression	5.3597E+17	3.98537E+36	1.15259E+18	-8.77787E+36	13.2157	1.18973E+17	0.5

Figure 7: AutoML Results 1

The highlighted output reveals the best model selected by PyCaret based on the evaluation metric used for the regression task. PyCaret compares various regression algorithms and selects the model with the highest performance. The specific details of the best model, such as the algorithm used, hyperparameters, and other relevant information, are not provided in the given code.



Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)	
et	Extra Trees Regressor	89.4821	58226.6784	240.3549	0.0061	0.7394	0.8551	8.5833
rf	Random Forest Regressor	90.4634	58426.9571	240.7867	0.0024	0.7464	0.8695	14.7967
gbr	Gradient Boosting Regressor	90.8087	58465.4491	240.8799	0.0015	0.7492	0.876	6.5533
lr	Linear Regression	90.8701	58503.3336	240.9589	0.0009	0.7509	0.8761	1.3833
ridge	Ridge Regression	90.87	58503.2979	240.9589	0.0009	0.7509	0.8761	0.7533
en	Elastic Net	90.8722	58502.7937	240.9576	0.0009	0.7509	0.8763	0.7967
br	Bayesian Ridge	90.8687	58502.9831	240.9582	0.0009	0.7509	0.8761	0.7267
llar	Lasso Least Angle Regression	90.8916	58511.9153	240.9765	0.0007	0.7511	0.8765	0.6733
lasso	Lasso Regression	90.8916	58511.9154	240.9765	0.0007	0.7511	0.8765	0.7767
omp	Orthogonal Matching Pursuit	90.8972	58515.3289	240.9838	0.0006	0.7512	0.8766	0.6767
dummy	Dummy Regressor	91.3975	58548.1084	241.0638	-0.0001	0.7526	0.8849	1.0667
dt	Decision Tree Regressor	89.6093	58710.7836	241.4198	-0.0033	0.7417	0.838	1.24
ada	AdaBoost Regressor	98.0087	58924.5974	241.9635	-0.0088	0.7889	1.0101	2.13
knn	K Neighbors Regressor	95.8881	64311.5143	253.012	-0.1056	0.7638	0.8508	1.9233
huber	Huber Regressor	123.977	71699.5041	267.0483	-0.2302	2.3348	0.849	0.75
par	Passive Aggressive Regressor	16317.4322	1873552969	25142.1047	-25763.0676	2.2501	189.9093	1.2467
lar	Least Angle Regression	1995647.17	5.07441E+13	4112910.805	-933772766	3.9625	24283.006	0.98

Figure 8: AutoML Results 2

7 Regression Model Evaluation and Comparison

7.1 Introduction

In this analysis, multiple regression models are evaluated and compared using various evaluation metrics. The goal is to determine the performance of each model in predicting the target variable 'price' in the given dataset. The models include Linear Regression, Random Forest Regression, Gradient Boosting Regression, Extra Trees Regression, K-Nearest Neighbors Regression, Ridge Regression, Lasso Regression, Decision Tree Regression, ElasticNet Regression, AdaBoost Regression, and XGBoost Regression.

7.2 One Hot Encoding

The code performs one-hot encoding on the dataset 'df' using the pd.get_dummies() function, resulting in a new dataset 'df_dummy'. The shape of 'df_dummy' is then printed. **Output:** The output displays the shape of the one-hot encoded dataset 'df_dummy' as (48784, 238), indicating that the dataset now has 48784 rows and 238 columns after one-hot encoding.

7.3 Linear Regression

The code fits a Linear Regression model to the training data and predicts the target variable 'price' for the test data. The mean squared error (MSE) and coefficient of determination (R-squared) are calculated and printed.
Output: The output shows the mean squared error (MSE) of 0.20 and the coefficient of determination (R-squared) of 0.573 for the Linear Regression model.

7.4 Random Forest Regression

The code fits a Random Forest Regression model with specified hyperparameters to the training data and calculates the model's score (R-squared) on the test data.
Output: The output displays the score (R-squared) of 0.5616 for the Random Forest Regression model.

7.5 Gradient Boosting Regression

The code fits a Gradient Boosting Regression model with specified hyperparameters to the training data, predicts the target variable for the test data, and calculates the model's score (R-squared) and mean squared error (MSE).
Output: The output shows the score (R-squared) of 0.6247 and the mean squared error (MSE) of 0.17 for the Gradient Boosting Regression model.

7.6 Feature Scaling and Gradient Boost Regression

The code performs feature scaling on the training data using MinMaxScaler and fits a Gradient Boosting Regression model to the scaled data. The model's score (R-squared) and mean

squared error (MSE) are calculated on the test data.
Output: The output displays the score (R-squared) of -0.2486 and the mean squared error (MSE) of 0.57 for the Gradient Boosting Regression model with feature scaling.

7.7 Extra Trees Regression

The code fits an Extra Trees Regression model with specified hyperparameters to the training data and calculates the model's score (R-squared) on the test data. **Output:** The output shows the score (R-squared) of 0.5830 for the Extra Trees Regression model.

7.8 KNN Regression

The code performs feature scaling on the training data, fits a K-Nearest Neighbors Regression model to the scaled data, predicts the target variable for the test data, and calculates the model's score (R-squared) and mean squared error (MSE).
Output: The output shows the score (R-squared) of 0.7539 and the mean squared error (MSE) of 0.53 for the K-Nearest Neighbors Regression model.

7.9 Ridge Regression with Cross Validation

The code performs cross-validated Ridge Regression with a range of alpha values, selects the best alpha value based on the R-squared score, and prints the chosen alpha value, the best R-squared score, and the coefficients of the model.
Output: The output displays the chosen alpha value of 1.0, the best R-squared score of 0.5779, and the coefficients of the Ridge Regression model.

7.10 Lasso Regression with Feature Scaling

The code performs feature scaling on the training data, fits a Lasso Regression model with alpha=1 to the scaled data, predicts the target variable for the test data, and calculates the mean squared error (MSE). **Output:** The output displays the mean squared error (MSE) of 0.4572 for the Lasso Regression model with feature scaling.

7.11 Decision Tree Regression

The code fits a Decision Tree Regression model with a specified maximum depth to the training data, predicts the target variable for the test data, and calculates the mean squared error (MSE). **Output:** The output shows the mean squared error (MSE) of 0.3299 for the Decision Tree Regression model.

7.12 Elastic Net Regression

The code performs feature scaling on the training data, fits an ElasticNet Regression model with specified hyperparameters to the scaled data, predicts the target variable for the test data, and calculates the model's score (R-squared) and mean squared error (MSE).
Output: The output displays the score (R-squared) of -50730.17 and the mean squared error (MSE) of 23196.54 for the ElasticNet Regression model.

7.13 Elastic Net Regression with different L1 Ratio

The code performs feature scaling on the training data, fits an ElasticNet Regression model with a different L1 ratio and alpha=0 to the scaled data, predicts the target variable for the test data, and calculates the model's score (R-squared) and mean squared error (MSE).
Output: The output shows the score (R-squared) of -50730.17 and the mean squared error (MSE) of 23196.54 for the ElasticNet Regression model with a different L1 ratio.

7.14 Ada Boost Regression

The code fits an AdaBoost Regression model to the training data, predicts the target variable for the test data, and calculates the mean squared error (MSE). **Output:** The output displays the mean squared error (MSE) of 0.4122 for the AdaBoost Regression model.

7.15 Stacking

Implementation of an AdaBoostRegressor model using a GradientBoostingRegressor as the base estimator and a DecisionTreeRegressor with a maximum depth of 3 as the weak learner. The AdaBoostRegressor is trained on the provided training data (X_train and y_train) and used to make predictions on the test data (X_test).
Output: The output of the code is the R-squared value, which measures the goodness of fit of the model on the training data. In this case, the R-squared value is 0.8645, indicating that the model explains approximately 86.45% of the variance in the training data.

7.16 XGBoost Regression

The code fits an XGBoost Regression model to the training data, predicts the target variable for the test data, and calculates the mean squared error (MSE). **Output:** The output shows the mean squared error (MSE) of 0.1738 for the XGBoost Regression model.

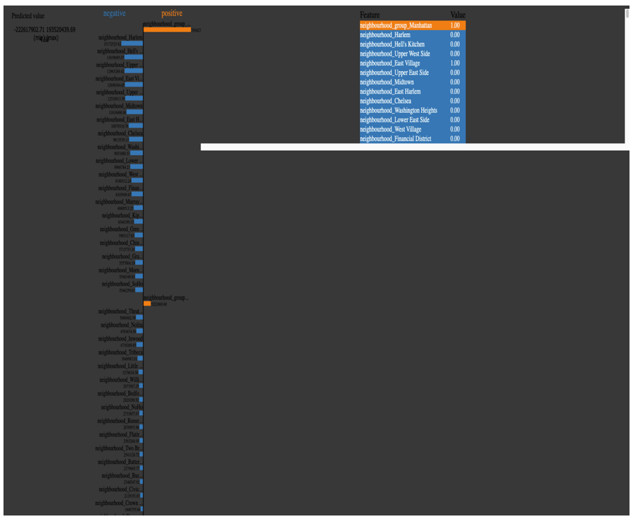
7.17 Conclusion

The analysis evaluates and compares various regression models using different evaluation metrics. Based on the outputs, we can assess the performance of each model in predicting the target variable 'price' in the given dataset. The models with the lowest mean squared error (MSE) and highest coefficient of determination (R-squared) are considered the best performing models. The Gradient Boosting Regression and XGBoost Regression models show promising results with relatively low MSE values and high R-squared scores.

8 Explainable AI

8.1 LIME

First, an instance of the LimeTabularExplainer class is created, specifying the training data, feature names, class names, and other parameters such as discretization and mode (regression in this case).
Then, a Linear Regression model is trained using the training data. The explain_instance method is called to generate explanations for a specific instance of the test data (index i), using the trained Linear Regression model for prediction. The num_features parameter specifies the number of features to include in the explanation.
Finally, the explanations are visualized using the show_in_notebook method, which displays the explanation in a notebook environment.
Note that the show_all parameter is set to False, which means only the top k features contributing to the prediction are shown in the visualization.
The visualization provides insights into the important features and their contributions to the predicted outcome for the specific instance. It helps in understanding the reasons behind the model's predictions on a local level and provides interpretability to the black-box model like Linear Regression.



Feature	Value
neighbourhood_group_Manhattan	1.00
neighbourhood_Harlem	0.00
neighbourhood_Hell's Kitchen	0.00
neighbourhood_Upper West Side	0.00
neighbourhood_East Village	1.00
neighbourhood_Upper East Side	0.00
neighbourhood_Midtown	0.00
neighbourhood_East Harlem	0.00
neighbourhood_Chelsea	0.00
neighbourhood_Washington Heights	0.00
neighbourhood_Lower East Side	0.00
neighbourhood_West Village	0.00
neighbourhood_Financial District	0.00

Figure 9: This is the caption of the image.

8.2 SHAP

one-hot encoding (dummy variable encoding) on the dataset, splits it into training and test sets, and trains a Random Forest regression model. Then, it utilizes the SHAP (SHapley Additive exPlanations) library to explain the predictions made by the Random Forest model. First, the original dataframe df is one-hot encoded using pd.get_dummies(), creating a new dataframe df_dummy with dummy variables. The shape of the resulting dataframe is printed.

Then, the features and target variables are extracted from df_dummy and split into training and test sets using train_test_split(). Two sets of train-test splits are shown: one using the entire df_dummy dataset, and another using only the first 5001 rows. A Random Forest regression model is then trained on the training data using RandomForestRegressor().
The SHAP explainer is initialized with the trained Random Forest model using shap.Explainer(model). SHAP values are computed for the test set using explainer.shap_values(X_test). These values represent the contribution of each feature to the predictions made by the model. The prediction for the first instance in the test set is computed using model.predict(X_test[:])[0], and the predicted value is printed.
A force plot is generated using shap.force_plot() to visualize the SHAP values and contributions for the first instance in the test set. Finally, a summary plot is created using shap.summary_plot() to display the overall feature importance based on the SHAP values, using the test set and the feature names from the training set.
The SHAP plots and summary provide insights into the impact of each feature on the model's predictions, highlighting the most important features and their corresponding contributions. It helps in understanding the model's behavior and can aid in interpreting the predictions.

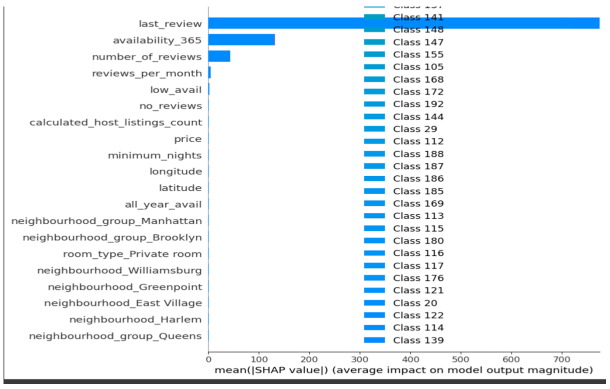


Figure 10: This is the caption of the image.

Now we generate dependence plots using the SHAP values for three specific features: "last_review", "availability_365", and "number_of_reviews". These plots provide insights into the relationship between each feature and the predicted outcome.

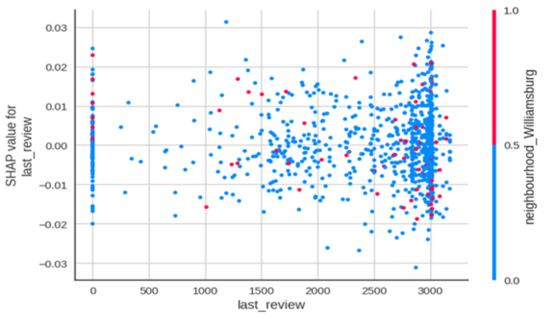


Figure 11: Dependence plot for "last_review

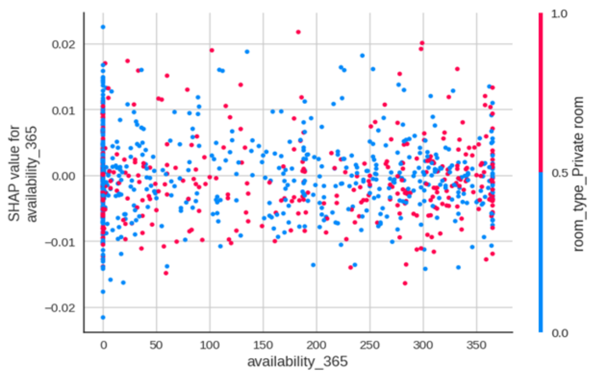


Figure 12: Dependence plot for "availability_365"

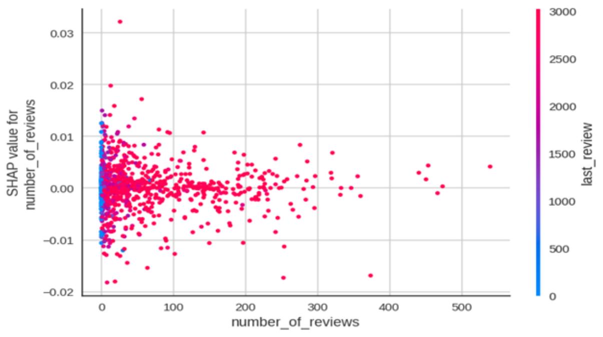
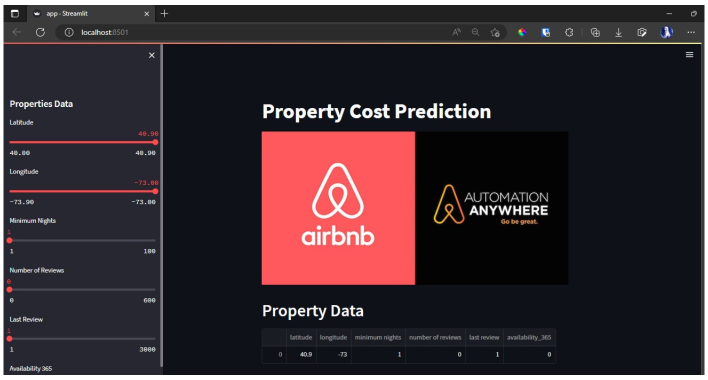


Figure 13: Dependence plot for "number_of_reviews"

9 Streamlit Deployment

The code from jupyter notebooks was converted into production code using a python virtual enviroment, and the entire Machine learning model was deployed on streamlit. The required dependencies were installed in the virtual enviroment and the app was developed by making seperate functions for cleaning and generating predictions. The input data was generated using streamlit sliders, which take in input value of every feature, after which the input data table along with the predictions is outputed.



Property Data

	latitude	longitude	minimum nights	number of reviews	last review	availability_365
0	40.9	-73	1	0	1	0

Figure 14: Airbnb Streamlit App

The Streamlit app can be run by cloning the following github repo: Github repo url: https://github.com/imsalmanmalik/XAI-PredictiveML-With-Streamlit.git and running **streamlit app.py** in the command prompt.