# CS7CS4 – Machine Learning Supplemental Assignment 2023-24

Xin Wang
School of Computer Science and Statistics
Trinity College Dublin
Dublin, Ireland
Email: wangx33@tcd.ie

***Abstract*—This study aims to predict the ratings of Airbnb listings in Dublin using machine learning models. By examining various features and services of the listings, we seek to create an accurate predictive model that can assist hosts in improving their offerings and provide renters with dependable information. Data from the Inside Airbnb website, including listings and reviews, were processed and analyzed. We employed and evaluated three machine learning models: Logistic Regression, k-Nearest Neighbors (k-NN), and Decision Tree Classifiers. Logistic Regression consistently outperformed the other models, demonstrating its effectiveness in this context. The findings emphasize the importance of feature engineering and model tuning in achieving high predictive accuracy. This research offers both academic insights into the application of machine learning in real-world situations and practical recommendations for enhancing the Airbnb platform.**

***Index Terms*—Airbnb ratings prediction,Machine learning,Logistic Regression,k-Nearest Neighbors (k-NN),Decision Tree Classifier,Feature engineering,Data preprocessing,Model evaluation**

## I. INTRODUCTION

Airbnb, a web-based service, has significantly transformed how people search for places to stay and travel. It provides a broad range of lodging options, from whole apartments to individual rooms, catering to diverse travelers' needs. Users can make reservations through the platform and rate their stays on various criteria like cleanliness, communication, overall experience, location, value, and accuracy. These ratings impact the listing's ranking and visibility and are essential in influencing prospective renters' choices.

However, Airbnb ad ratings are shaped by various factors, including the particular features and services of each listing. Predicting these ratings is crucial as it aids hosts in enhancing their offerings and gives renters a more dependable foundation for their decisions. Hence, this paper aims to forecast the possible ratings of Airbnb listings in Dublin by examining their features and services.

To achieve this goal, this study uses data from the Inside Airbnb website. The datasets used include the ads dataset and the reviews dataset. The ads dataset contains 75 feature columns with a total of 7,345 records, while the reviews dataset contains 6 feature columns with a total of 230,065 reviews. Through in-depth analysis and modeling of this data, we hope to build an effective predictive model that accurately predicts the ratings of Airbnb ads in Dublin.

The academic significance of this study stems from applying independent machine learning analyses to thoroughly and comprehensively explore the data. By utilizing various machine learning algorithms, we identified and validated the key factors affecting listing ratings. This approach not only highlights the potential of machine learning for real-world applications but also showcases the effectiveness of data-driven analytics in addressing complex issues.

This study holds both academic value and practical application. By uncovering the key factors that influence ratings, we can provide valuable insights for enhancing the Airbnb platform and offer a better service experience to both hosts and renters. In summary, this research not only presents a new analytical perspective for academia but also offers significant practical references.

## II. FEATURE ENGINEERING

The Listings dataset comprises 75 feature columns, 10 of which relate to data scraping details and audit information **(such as listing URL, scrape ID, last scraped, source, picture URL, host ID, host URL, host thumbnail URL, host picture URL, and calendar last scraped)**. These columns were excluded to enhance the dataset's clarity and comprehensibility. Similarly, from the Reviews dataset, only the **listing ID** and **comments** were retained for further analysis from the initial 6 features. Detailed preprocessing steps for each dataset are outlined in the following sections.

### *A. LISTINGS DATA PRE-PROCESSING*

The Listings dataset includes an amenities column that lists key amenities for each property. To make it easier for users to choose accommodations, these amenities were divided into distinct categories. There are 77 unique amenities, which were grouped into 9 broader categories. For example, amenities like 'Hot shower,' 'Shower gel,' 'Hair dryer,' 'Shampoo,' 'Conditioner,' and 'Body Soap' were categorized under 'Bath Products,' while 'Oven,' 'Hot water kettle,' 'Cooking basics,' and 'Microwave' were categorized under 'Kitchen Appliances.' If a listing lacked any amenities from the 9 categories or did not have host verification information, the missing values were filled with zeros. Figure 1 illustrates the percentage of NaN values in each column of the Listings dataset.

Next, it was crucial to evaluate the number of NaN values in each feature column. Figure 1 shows the percentage of NaN values in each column of the Listings dataset. The analysis identified that the columns 'neighbourhood group cleansed,' 'license,' 'bathrooms,' and 'calendar updated' contained NaN values, leading to their exclusion from further analysis. Additionally, the seven Review Scores columns, which are our

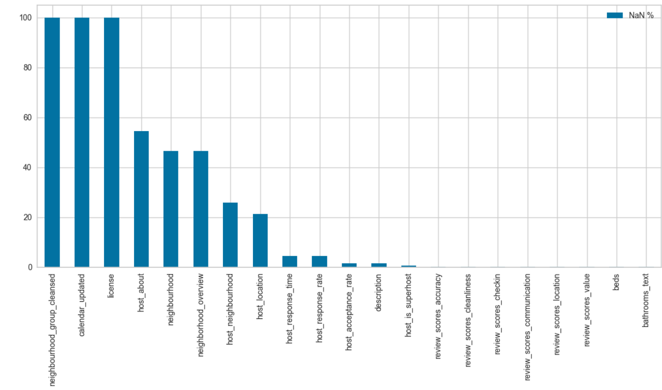



Fig. 1. %NaN values present in each feature columns from Listings dataset

target variables, had about 20% NaN values. Since these are essential for our model, imputing these values could introduce bias, so any rows with NaN values were removed from the dataset.

A detailed examination of the 'price' feature revealed that prices were listed in dollars ($). To ensure machine-readability, I removed the '$' and ',' symbols and converted the string values to integers. This preprocessing step was vital for maintaining data consistency. The 'price' feature displayed numerous outliers, with an average price of about $167 and a median of $110, while the maximum price reached $1,600. As shown in Figure 2, most prices fell within the $50 to $300 range, leading to the removal of rows with prices outside this range.

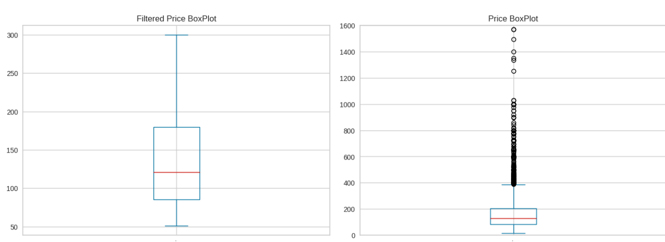



Fig. 2. Plot of Price Feature

Finally, the distribution of host response rates was plotted to understand the overall picture of host responsiveness in the dataset. As shown in Figure 3, the majority of hosts have a highly concentrated response rate of 100%, indicating they respond to guest inquiries very promptly. A small number of hosts have response rates between 0% and 20%, showing significantly lower responsiveness. There is also a smaller group of hosts with moderate response rates between 20% and 80%. This right-skewed distribution may result from the platform's rules or incentives, hosts' sense of responsibility, and potential biases in data collection. Overall, this high response rate enhances the platform's user experience and satisfaction, but improvement measures are needed for the few hosts with low response rates.

The rating values for the seven target variables were closely

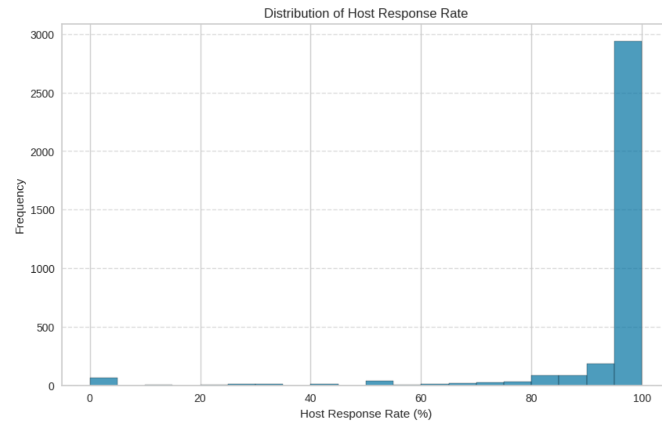



Fig. 3. Plot of Host Response Rate

clustered. Figure 4 illustrates the distribution of all rating columns, which predominantly range between 4 and 5. To address this clustering, I used a Binning Approach, categorizing continuous data into discrete bins based on quantile ranges. This method helps avoid bias by ensuring equal-sized bins. As a result, we predict the rating bin for a given listing rather than the exact numeric rating, transforming this into a classification problem where the model predicts a categorical outcome based on input parameters.

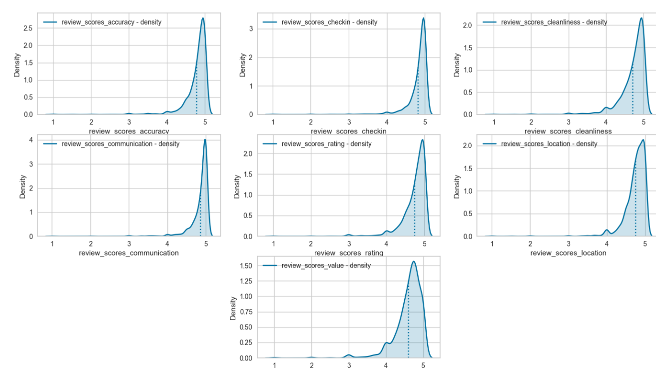



Fig. 4. Distribution Plot of all the Target Columns (Review Rating Columns)

In the Binning Approach, continuous data is categorized into discrete bins based on specific criteria, which facilitates further analysis. For this use case, I chose to bin individual rating values based on their quantile range. The primary reason for using this approach is that the quantile method allows for the creation of equal-sized bins.

By categorizing a given rating value into equal-sized bins according to its quantile range, we avoid bias where one bin has more records than others, making this method suitable for our problem. Once the values are binned, we predict the rating bin for a given listing instead of the exact numeric rating, transforming this into a classification problem where the model predicts a categorical variable based on input parameters.

To categorize ratings into discrete bins, I used the Pandas

DataFrame QCut functionality, which segments the data into equal-sized bins based on distribution percentiles. Ratings were first converted to percentages before binning. For features like 'host response rate' and 'host acceptance rate,' I removed the '%' signs and converted the resulting figures into numeric format for further processing. Features such as 'host response time,' 'host is superhost,' 'host identity verified,' 'instant bookable,' 'room type,' 'neighbourhood cleansed,' and 'has availability' were label-encoded using sklearn's preprocessing library to ensure they are machine-readable. For scaling, I applied min-max scaling, which preserves the original data distribution. In min-max scaling, each data point is scaled to a new value using the formula.:

$$x_{scaled} = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

### *B. REVIEWS DATA PRE-PROCESSING*

The Reviews dataset includes features such as listing ID, ID, date, reviewer ID, reviewer name, and comments. With 230,065 reviews, a single listing may have multiple reviews. For analysis, I focused solely on the listing ID and comments. The comments feature contained non-English entries and some comments composed entirely of special characters, emojis, or HTML tags. To make these features usable, I needed to preprocess them by removing emojis and special characters, while retaining only English reviews.

Initially, I eliminated all review rows lacking associated comments. Then, I filtered for English reviews using the **fasttext** Python library and its pretrained model, which identifies the language of any text. Next, I removed emojis from the text using the **emoji** Python library. Each comment was analyzed character by character to check for matches with the emoji data. If a comment consisted entirely of emojis or was predominantly composed of them, it was discarded. I set a threshold of 20 characters for comments to ensure they were substantial enough for model training. After preprocessing, 220,551 reviews were retained out of the original 230,065, resulting in a 4% reduction of irrelevant reviews. The total number of comments for a given listing was updated after preprocessing was completed.

*1) TF-IDF METHODOLOGY:* To utilize the review text effectively, preprocessing was necessary to make it machine-readable. I implemented the Term Frequency-Inverse Document Frequency (TF-IDF) methodology, which assesses the significance of a word within a text collection. The TF-IDF of a term, such as a review for a listing, by considering both the frequency of the term in the document and its rarity across all documents.

TF-IDF of term $t$ in a document $d$ is defined as:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$$

where:

$$\text{tf}(t, d) = \frac{\text{terms } t \text{ occur in doc } d}{\text{total terms in } d}$$

$$\text{idf}(t) = 1 + \log\left(\frac{1 + \text{ documents in corpus}}{1 + \text{df}(t)}\right)$$

Here, $\text{tf}(t, d)$ represents the term frequency, and $\text{idf}(t)$ represents the inverse document frequency.

The rationale for using TF-IDF in this context was to extract important tokens from each review to serve as features in the machine learning model. After the preprocessing steps outlined in Section 2.1, I removed all stopwords from the review comments. For the TF-IDF implementation, I utilized the 'TfidfVectorizer' from sklearn's text feature extraction library, limiting the maximum number of features returned to 33. This limitation helps mitigate potential bias and overfitting, as the distribution of word counts per comment varies significantly.

The limitation on the number of features returned by 'TfidfVectorizer' serves two main purposes. First, considering the wide range of word counts per comment, there are significant outliers—from very short comments of 10-15 words to lengthy ones spanning 500-1000 words. Based on the distribution shown in Figure 5, most data falls within the 20-40 words per comment range. The statistical summary of the word counts per comment shows a mean of approximately 44 and a median of 33. Restricting the maximum features to the median aligns with the central tendency of the data distribution.

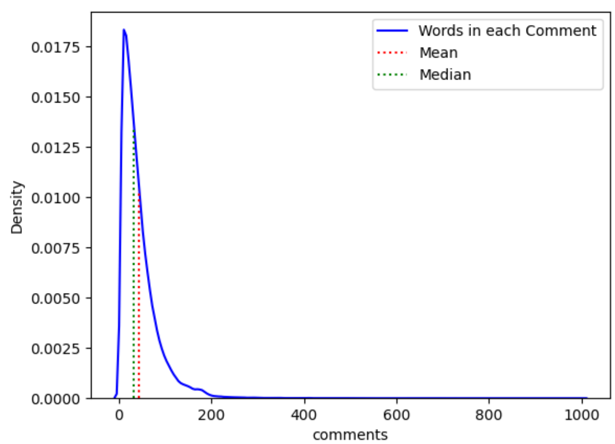



Fig. 5. Plot of Words in each comment

Moreover, an increase in the number of features could potentially bias the data and lead to overfitting, resulting in high training accuracy but poor testing accuracy.

Finally, after computing TF-IDF for all review comments, I calculated the mean of all features created by 'TfidfVectorizer', grouped by Listing ID. This approach ensures that each listing has only one row, facilitating seamless integration with the Listings dataset.

### *C. FEATURE SELECTION AND IMPORTANCE*

After cleaning and processing the Listings and Reviews datasets, I aimed to identify features that significantly impact the machine learning model. Using 'review scores accuracy' as an example, I calculated the correlation coefficients between the features and the target variables. A threshold of 0.01

was set, retaining only features with correlation coefficients exceeding this value.

As shown in Figure 6, features such as ’host is superhost’ (correlation coefficient of 0.257), ’host listings count’ (0.232), and ’host total listings count’ (0.215) have a strong positive correlation with review scores accuracy. Additionally, positive terms used in reviews, such as ’good’ (0.211), ’home’ (0.166), and ’recommend’ (0.166), also show high correlations, indicating that the content of the review influences the rating.

Regarding amenities and services, features such as parking (0.097), food service (0.078), and security (0.079) displayed notable correlations. Host response rate (0.070) and acceptance rate (0.033) also ranked high, showing the impact of hosts’ timely responses and acceptance of booking requests on ratings.

This analytical approach helped identify multiple features significantly affecting ratings, providing key inputs for subsequent machine learning models and enhancing the accuracy and reliability of predictions.

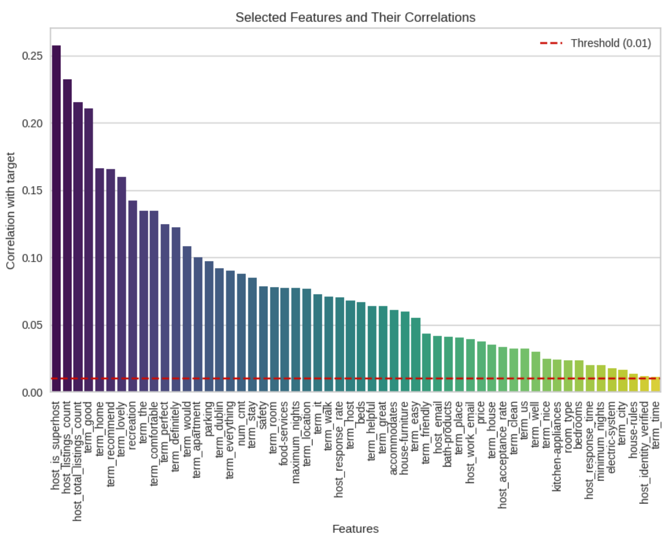



Fig. 6. The Correlations of Features - eg.review_scores_accuracy

Figure 6 shows that for a listing to achieve a very high rating (Bin 2), the most significant terms in the review comments are ‘Home,’ ‘Perfect,’ and ‘Recommend.’ For a rating in Bin 1, reviews must include terms like ‘Would,’ ‘Lovely,’ or ‘Home,’ along with amenities like Kitchen Appliances. The analysis also highlights differences in the intensity of the term ‘Home’ and the number of comments per bin, especially when comparing Superhosts to non-Superhosts. This suggests that these features are crucial in developing the model.

Figure 7 shows that the intensity of the term ‘Home’ and the number of comments per bin differ across all three review bins. The number of records in each review bin varies significantly when the host is a Superhost compared to when the host is not a Superhost. This highlights the importance of these features in model development. A similar analysis was conducted for all the features, and the final model was built using a total of 60 features.

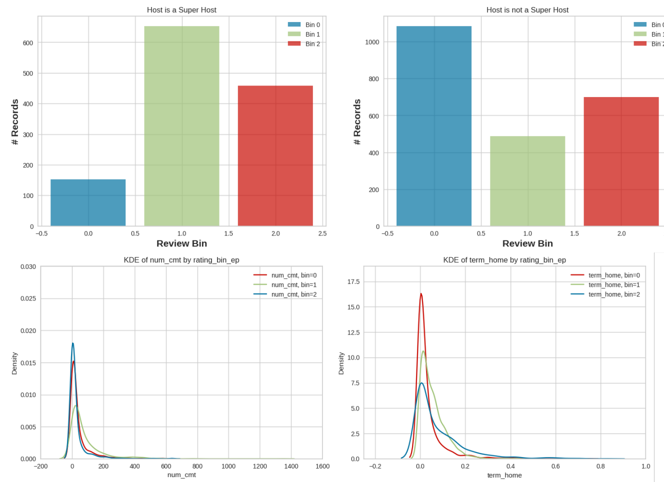



Fig. 7. Analysis Important Features

## III. MODELS OF MACHINE LEARNING

Given that this use case is framed as a classification problem, the initial models to test and implement include Logistic Regression, k-Nearest Neighbors (k-NN), and Decision Tree Classifier. All models were executed as part of this assignment, and their results were compared to determine the best fit. The combined dataset of Listings and Reviews was split into an 80-20 ratio, with 3741 data rows. The models were trained on 78% of the dataset and tested on the remaining 20%.

### *A. LOGISTIC REGRESSION AND PARAMETER TUNING*

Logistic Regression is a supervised learning algorithm that uses labeled data for training. During this process, the model assigns weights (coefficients) to each feature, optimizing these weights to minimize error through a cost function. To prevent overfitting, a penalty (L1, L2, or None) is applied as a hyperparameter. For this case, I explored a range of C values, controlling the regularization strength from 0.001 to 1000. The C values tested were [0.001, 0.1, 1, 10, 100, 1000], exposing the model to varying levels of regularization, where smaller C values indicate stronger regularization.

GGiven the multiclass nature of the problem, where the model predicts which review bin an Airbnb listing falls into (0, 1, or 2), I selected the ‘newton-cg’ solver recommended by scikit-learn for multinomial loss problems. This solver supports the L2 penalty and computes the Hessian matrix, making it suitable for this scenario. To identify the optimal C value for each target variable, I plotted the cross-validation (CV = 5) graph. Figure 8 presents accuracy scores across varying C values for the review scores target variable. The results indicate that increasing C (reducing regularization) generally enhances model accuracy, although improvements diminish after a certain point (C = 10). The optimal C value and corresponding accuracies for each target variable are summarized in Table I for degree 1 and Table II for degree 2.

For most target variables, the optimal value of C is 10 or less for degree 1 models and similar for degree 2 models.

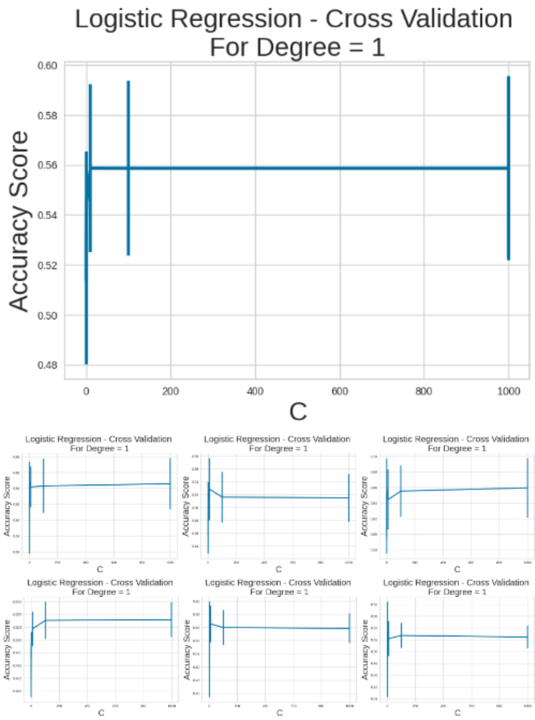



Fig. 8. Cross-Validation Accuracy Scores for Degree 1

TABLE I
VALUES OF C FOR DIFFERENT TARGET VARIABLES (DEGREE=1)

| Target Variable | C Value | Train Accuracy | Test accuracy |
|---|---|---|---|
| accuracy | 10 | 0.60 | 0.59 |
| checkin | 10 | 0.69 | 0.68 |
| cleanliness | 100 | 0.58 | 0.54 |
| communication | 10 | 0.71 | 0.68 |
| location | 10 | 0.60 | 0.60 |
| rating | 100 | 0.60 | 0.58 |
| value | 100 | 0.60 | 0.60 |

Increasing the value of C further reduces the regularization penalty, risking overfitting. This is reflected in the optimal C values shown in Tables I and II. Transforming the training dataset into a polynomial feature space of degree 2 was computationally expensive and did not significantly improve results, as there was still a notable gap between the training and testing accuracy scores.

When examining the results more closely, it is apparent that for certain variables like check-in and communication, higher C values (such as 10) yielded better generalization performance, indicating that moderate regularization was beneficial. Conversely, for variables like cleanliness and value, very high C values (such as 100 or 1000) were optimal, suggesting these features were more sensitive to regularization and benefitted from stronger penalization to avoid overfitting.

Additionally, I augmented and transformed the training dataset into a polynomial feature space of degree 2 to predict the review bins a given listing might belong to. This process yielded slightly better results in some cases but was generally not significantly better than the degree 1 models, as indicated

TABLE II
VALUES OF C FOR DIFFERENT TARGET VARIABLES (DEGREE=2)

| Target Variable | C Value | Train Accuracy | Test accuracy |
|---|---|---|---|
| accuracy | 10 | 0.70 | 0.60 |
| checkin | 10 | 0.77 | 0.68 |
| cleanliness | 100 | 0.64 | 0.55 |
| communication | 10 | 0.71 | 0.68 |
| location | 10 | 0.70 | 0.60 |
| rating | 100 | 0.65 | 0.60 |
| value | 1000 | 0.65 | 0.58 |

by the results in Table II. The increase in computational complexity did not justify the marginal improvement in accuracy. This was evident in variables like check-in and location, where both degree 1 and degree 2 models performed similarly.

The cross-validation plot for all the target variables predicted by the model is presented in Figure 9, illustrating how the model's performance varied across different regularization strengths and polynomial degrees. The plots show that the model's accuracy tends to stabilize or even decline beyond a certain C value, reinforcing the importance of selecting an appropriate regularization parameter.

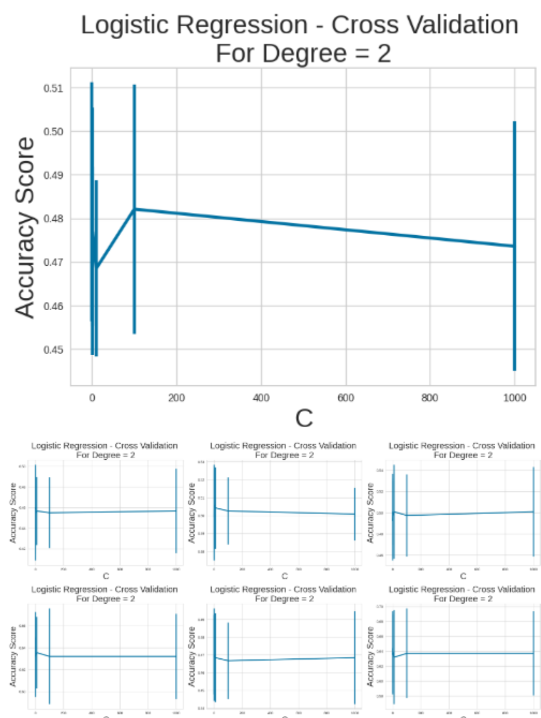



Fig. 9. Cross-Validation Accuracy Scores for Degree 2

### B. *k-NN CLASSIFIER AND PARAMETER TUNING*

The k-Nearest Neighbors (k-NN) algorithm is a supervised learning method that classifies data based on their target variables or classes. The model uses its hyperparameter (k neighbors) and the distance between classes to classify new data points. In this analysis, I varied the value of k while keeping weights uniform, selecting only odd numbers to avoid ties. I implemented 5-fold cross-validation to assess model performance, testing k values in the range of [1, 3, 5, 7, 9, 11, 13, 15, 17, 19].

The performance of the k-NN classifier depends significantly on the choice of k, the number of neighbors considered. A smaller k can lead to a model that is too complex, capturing noise in the dataset (overfitting), while a larger k can oversimplify the model, missing important trends (underfitting). To find the optimal k value, I conducted experiments across a range of odd values to avoid ties during classification.

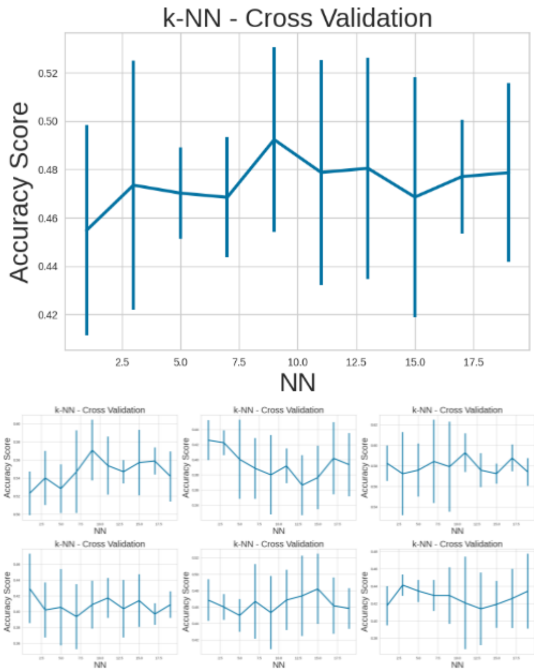



Fig. 10. k-NN Cross Validation for the target variable

Figure 10 illustrates the cross-validation plot for review scores accuracy across different k values. The analysis showed that the model reached its highest accuracy with k set to 7. Although k=1 produced similar accuracy, it was found to be too complex and hard to interpret. Increasing k simplifies the model, reducing dataset noise. Therefore, the optimal k value for review scores accuracy is 7, with training accuracy at 63.6% and testing accuracy at 49.6%. Similarly, the optimal k values for other target variables demonstrated a trend of balancing model complexity and performance.

The detailed results for each of the seven target variables (accuracy, checkin, cleanliness, communication, location, rating, and value) are summarized in Table III. These results highlight the importance of selecting an appropriate k value for each specific target to achieve optimal performance.

### C. DECISION TREE AND PARAMETER TUNING

The Decision Tree algorithm is a supervised learning method that divides data based on their features to make decisions. This analysis examines the performance of decision tree classifiers with varying tree depths (from 3 to 19) across multiple target variables: accuracy, check-in, cleanliness, communication, location, rating, and value. Each tree depth was evaluated using 5-fold cross-validation to assess model performance, testing depths in the range of [3, 5, 7, 9, 11, 13, 15, 17, 19].

The performance of the Decision Tree classifier is significantly influenced by the depth of the tree. A shallow tree (small depth) might underfit the data, missing important patterns, while a deep tree (large depth) might overfit, capturing noise in the dataset. To find the optimal tree depth, experiments were conducted across a range of depths.

From the Figure 11, for the accuracy target, a depth of 3 provided a train accuracy of 61% and a test accuracy of 58%, while a deeper tree at depth 7 resulted in a train accuracy of 74% but a lower test accuracy of 56%, indicating overfitting.

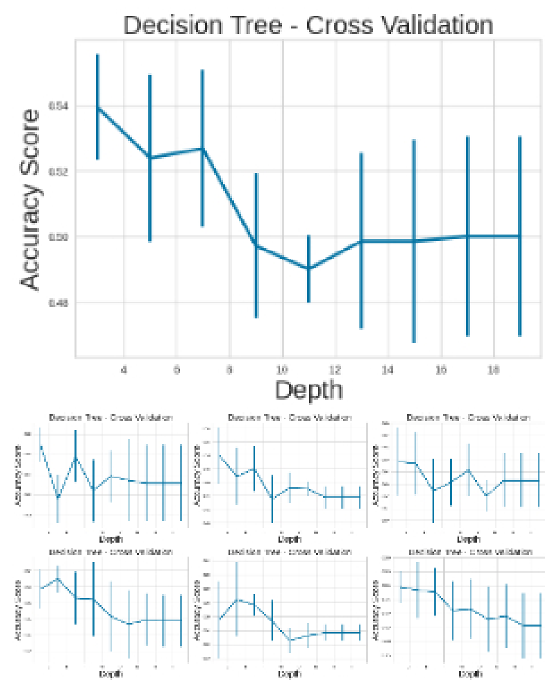



Fig. 11. Decision Tree Cross Validation for the target variable

The detailed results for each of the seven target variables highlight the importance of selecting an appropriate tree depth to achieve optimal performance.Table IV summarizes the value of hyper-parameter depth with the corresponding train and test accuracy for all the 7 targets:

TABLE III
VALUE OF HYPER-PARAMETER K WITH THE CORRESPONDING TRAIN AND TEST ACCURACY FOR ALL THE 7 TARGETS

| Target Variable | K Value | Train Accuracy | Test accuracy |
|---|---|---|---|
| accuracy | 7 | 0.63 | 0.50 |
| checkin | 19 | 0.67 | 0.61 |
| cleanliness | 7 | 0.60 | 0.44 |
| communication | 5 | 0.75 | 0.62 |
| location | 7 | 0.60 | 0.43 |
| rating | 7 | 0.64 | 0.50 |
| value | 19 | 0.53 | 0.46 |

TABLE IV
VALUE OF HYPER-PARAMETER DEPTH WITH THE CORRESPONDING TRAIN AND TEST ACCURACY FOR ALL THE 7 TARGETS

| Target Variable | Depth Value | Train Accuracy | Test accuracy |
|---|---|---|---|
| accuracy | 3 | 0.61 | 0.58 |
| checkin | 5 | 0.74 | 0.70 |
| cleanliness | 5 | 0.64 | 0.52 |
| communication | 5 | 0.70 | 0.67 |
| location | 5 | 0.59 | 0.54 |
| rating | 5 | 0.69 | 0.60 |
| value | 7 | 0.71 | 0.55 |

The figures and tables illustrate the impact of tree depth on model performance, emphasizing the importance of choosing the right depth to balance model complexity and accuracy.

### D. RESULT EVALUATION AND MODELS COMPARISON

In this section, the performance of the Logistic Regression, k-NN, and Decision Tree models will be evaluated and compared to determine which model best fits the dataset and achieves the desired accuracy in predicting Airbnb listing ratings. The ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) metric is used to assess the models. An ROC curve is a plot of True Positive Rate vs. False Positive Rate, while AUC is the area under the ROC curve, providing a single value for each class rather than a curve. For an ideal classifier, the ROC curve gives a point in the top left corner, indicating 100% True Positives and 0% False Positives; similarly, for an ideal classifier, AUC = 1, and a random classifier has AUC = 0.5.

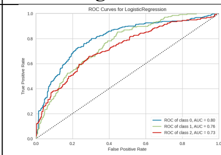
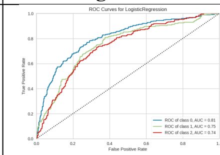
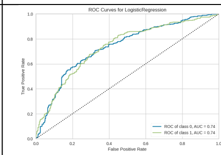
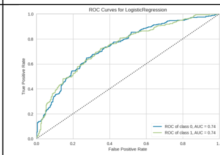
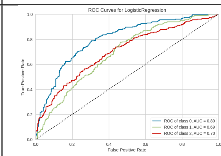
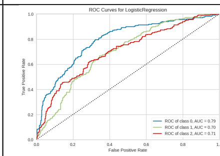
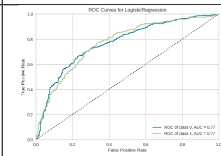
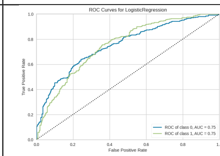
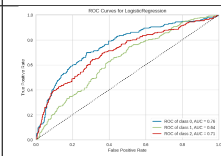
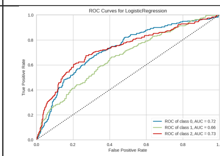
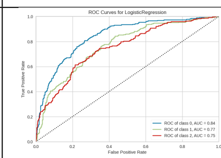
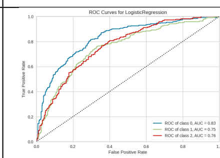
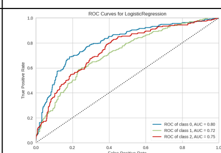
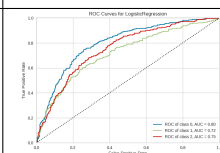
Please find the ROC-AUC curves of Logistic Regression, k-NN, and Decision Tree models for various target variables in the following Table VI and Table **??**.

From the values presented in the ROC-AUC curves, it is evident that the Logistic Regression model consistently outperforms both the k-NN and Decision Tree classifiers for all target variables. Comparing Logistic Regression with degree=1 and degree=2, we observe that the performance is relatively similar, with degree=1 slightly outperforming degree=2 in some instances, and vice versa in others. For instance, in the 'review_scores_accuracy' target variable, degree=1 achieves an AUC of 0.80, 0.76, and 0.73 for classes 0, 1, and 2 respectively, while degree=2 achieves 0.81, 0.75, and 0.74. Both degrees of Logistic Regression are robust and significantly better than k-NN and Decision Tree models.

The AUC for all classes across all target variables is higher for Logistic Regression compared to k-NN and Decision Tree classifiers. For example, in the 'review_scores_cleanliness' target variable, Logistic Regression (degree=1) achieves an AUC of 0.80, 0.69, and 0.70 for classes 0, 1, and 2 respectively, whereas k-NN achieves 0.62, 0.62, and 0.63, and Decision Tree achieves 0.65, 0.60, and 0.61.

The Dummy Classifiers further emphasize the superiority of the selected models. The Frequent Class Dummy achieved an accuracy of 36.2%, with all predictions skewed towards the most frequent class, and the Uniform Class Dummy achieved an accuracy of 33.8%, with predictions uniformly distributed

TABLE V
ROC-AUC FOR DIFFERENT DEGREE OF LOGISTIC REGRESSION

| Target Variable | Degree = 1 | Degree = 2 |
|---|---|---|
| accuracy |  |  |
| checkin |  |  |
| cleanliness |  |  |
| communication |  |  |
| location |  |  |
| rating |  |  |
| value |  |  |

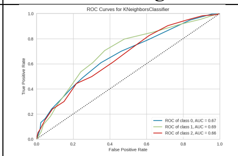
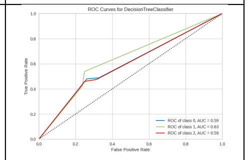
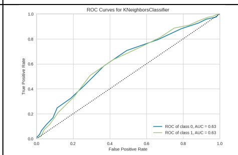
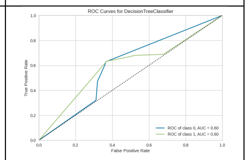
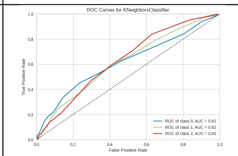
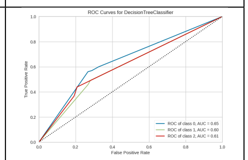
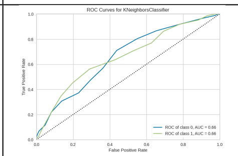
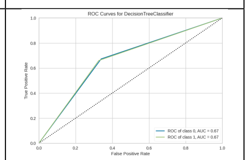
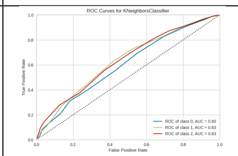
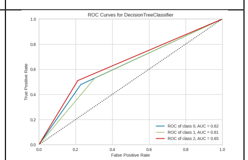
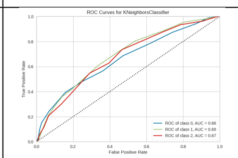
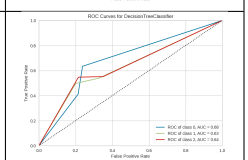
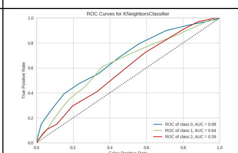
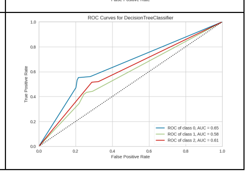
across classes. Both of these baselines perform significantly worse than the Logistic Regression, k-NN, and Decision Tree models.

Therefore, Logistic Regression, regardless of the degree, is the most suitable model for this dataset and problem statement. The selected models' performance is optimal for this use case, as evidenced by their superior AUC values and accuracy compared to the baseline classifiers.

## IV. CONCLUSION

In this research, our goal was to forecast possible ratings for Airbnb listings in Dublin by examining different features and services offered. We thoroughly processed the data and engineered features, categorizing amenities, addressing missing values, and effectively utilizing review text data. We carefully prepared datasets from the Inside Airbnb website to maintain

TABLE VI
ROC-AUC Between k-Nearest Neighbors and Decision Tree

| Target Variable | k-Nearest Neighbors | Decision Tree |
|---|---|---|
| accuracy |  |  |
| checkin |  |  |
| cleanliness |  |  |
| communication |  |  |
| location |  |  |
| rating |  |  |
| value |  |  |

data integrity and suitability for machine learning models.

We utilized three main machine learning models in our analysis: Logistic Regression, k-Nearest Neighbors (k-NN), and Decision Tree Classifiers. Each model underwent thorough testing and tuning to identify the optimal hyperparameters for the best predictive performance. The results from these models revealed several important insights:

1) **Logistic Regression:** Logistic Regression proved to be the strongest model for all target variables, consistently achieving higher accuracy and ROC-AUC scores compared to k-NN and Decision Tree models. The best regularization parameter (C) differed across target variables, with moderate regularization typically producing the best outcomes. Degree 1 polynomial features performed slightly better than degree 2, suggesting that adding complexity did not notably improve predictive accuracy.
2) **k-Nearest Neighbors (k-NN):** The k-NN model highlighted the significance of choosing the right k value. Striking a balance between underfitting and overfitting was essential, with k=7 yielding the best results for most target variables. Although k-NN is simple and easy to implement, it was less effective than Logistic Regression, especially for higher-dimensional data.
3) **Decision Tree Classifier:** The performance of the Decision Tree model was highly dependent on tree depth. Shallow trees tended to underfit, while deep trees often overfitted the data. The optimal tree depths varied for different target variables, but overall, the Decision Tree model did not perform as well as Logistic Regression in terms of predictive accuracy.

This research offers substantial academic and practical benefits. Academically, it showcases the application of various machine learning algorithms to real-world problems, particularly in predicting Airbnb listing ratings. The detailed feature engineering and model evaluation processes serve as a valuable guide for future studies aiming to predict categorical outcomes using complex datasets.

Practically, the study provides actionable insights for Airbnb hosts and platform developers. By pinpointing key features that affect listing ratings, hosts can focus on enhancing specific aspects of their services, such as cleanliness, communication, and amenities, to boost guest satisfaction. Additionally, the platform can use these predictive models to offer personalized recommendations and improve the overall user experience.

## V. Answer of Assignment 2

### A. Question I

Following are the two situations where Logistic Regression would give inaccurate results:

1) **Data Imbalance**
   If the dataset consists of imbalanced data for the target variable, as shown in Figure 12 left. The target value in this data is made up of 34 ”1s” and just 6 ”0s.” As seen in Figure 12 right, the model can only predict for the majority class, or 1, when predicting this feature.

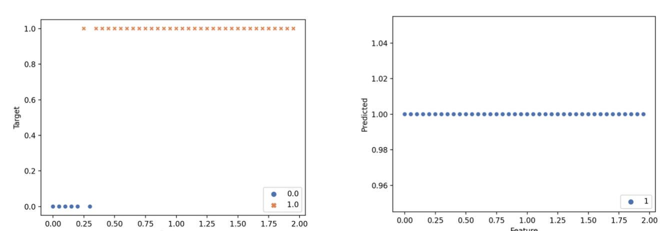



Fig. 12. Plot of Data Imbalance

2) **No Linear Correlation**
   The model cannot correctly forecast the target variable if there is no linear correlation between the data. For instance, I arbitrarily generated two features and one target variable. The two features and their corresponding

goal values are shown in Figure 13 left. The following plot, Figure 13 right, shows that the values are predicted linearly rather than scatteredly as they were formed when estimating the target values from the attributes.

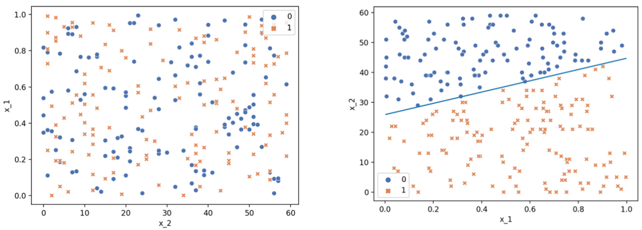



Fig. 13. Two Features and Target Values

## B. QUESTION II

1) **KNN's Advantages**
To identify the data points that are part of a particular class, the model uses a simple calculation. The K data points that are closest to the new data point are chosen by the algorithm after it has calculated the distance between the new data point and every other data point in the dataset. Moreover, the prediction process is dependent on the complete dataset. Without more training, the method can be applied simply to a new dataset. Finally, it can be applied to regression as well as classification. The KNN model uses a majority vote from its neighbors to classify the target variables. The property value of an item is the result of regression. The average of its K Nearest Neighbors is this value.
2) **KNN's Disadvantages**
Because it scans the complete dataset to find the N-Neighbors, it may take a while to forecast the target class for a new data instance. Additionally, the technique is susceptible to anomalies or irrelevant data in the dataset. To deal with these anomalies, the data must be scaled properly. Finally, a prediction is the only step in the creation of a KNN algorithm model. When training data is vast and prediction time is crucial, there is a significant overhead.
3) **MLP's Advantages**
MLP classifiers are capable of modeling intricate relationships found in data. Non-linear correlations between the input and output variables can be learned by these classifiers. They may also process high-dimensional data, such as text, audio, and photos, by teaching the hidden layers to learn distributed representations of the input data. In addition, MLP classifiers may be trained on big datasets and excel in a variety of real-world classification tasks with respect to predicted accuracy. Given that the model is stored once it has been trained on data, they can be utilized to rapidly forecast the target values. The learning rate, activation function, number of hidden layers, degree of regularization, dropout, and batch size are among the hyperparameters in MLP that can be adjusted.
4) **MLP's Disadvantages**
Training a large dataset can be time-consuming, particularly if the model has a lot of hidden layers. The model has numerous hidden layers, therefore fine-tuning its hyperparameters is necessary to achieve acceptable results. Additionally, as the number of hidden layers in an MLP model increases, the model gets increasingly complex and difficult to understand. As a result, using it as a black box model obscures the details of how it functions.

## C. QUESTION III

The dataset is resampled several times in K-Fold Cross Validation in order to train and assess the model on various subsets of the data and determine how well the model generalizes. A model's performance may be overestimated if it is trained on a single dataset split and then performs poorly on additional, unknown data. A more accurate estimation of the model's performance on unseen data with various data splits can be obtained by resampling the data several times across different folds. For instance, I used np.arange to generate a feature X with values ranging from 0 to 1, and I used feature X to build a target Y. The training data is split five times at random, and each time the desired output is predicted, Figure 14 is produced. For each of these separate splits, or folds, the Mean Squared Error is determined. The MSE varies for every fold, as can be observed. It is acknowledged that using a single train-test split to evaluate the model is not a recommended approach.

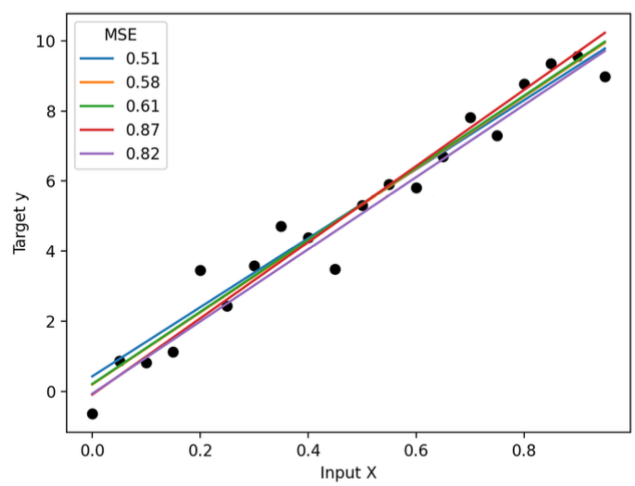



Fig. 14. Random data splits MSE

**Selecting the Value of K:** The model would be trained and assessed on a restricted number of folds if a small value of k was chosen. This would result in a significant bias, which would suggest that the model makes assumptions about the target variable. However, choosing a large number for k will cause the model to be trained numerous times, which will result in a high variance. This means that the model will overfit the data and become sensitive to even slight changes in the target variables.

### D. QUESTION VI

Q is the future value that we wish to anticipate for a time series data set at time t+q. Lagged output values can be utilized to generate features for the time series at t, t-1, and t-2. These values provide useful information for making future value predictions. For instance, let's say we wish to develop a times series feature that allows us to forecast how many products would be used each month. The current value is shifted by one or two steps to produce the lag values. The data for feature tag_1 is moved by one step, as shown in Figure 15, and the data for tag_2 is shifted by two steps.

| | Date | Product Usage | tag_1 | tag_2 |
|---|---|---|---|---|
| 0 | 2024-07-15 | 1000 | NaN | NaN |
| 1 | 2024-08-15 | 2000 | 1000.0 | NaN |
| 2 | 2024-09-15 | 4000 | 2000.0 | 1000.0 |
| 3 | 2024-10-15 | 8000 | 4000.0 | 2000.0 |
| 4 | 2024-11-15 | 16000 | 8000.0 | 4000.0 |
| 5 | 2024-12-15 | 32000 | 16000.0 | 8000.0 |

Fig. 15. Time Series Features for Product Usage

## APPENDIX

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import regex
import emoji
from pycountry import languages
import fasttext

def split_count(info):

    emoji_list = []
    data = regex.findall(r'\X', info)
    for word in data:
        if any(char in emoji.EMOJI_DATA for char in word):
            emoji_list.append(word)

    return len(emoji_list)

reviews_raw = pd.read_csv("reviews.csv")
PRETRAINED_MODEL_PATH = 'lid.176.bin'
model = fasttext.load_model(PRETRAINED_MODEL_PATH)

reviews_raw['lang'] = range(0, len(reviews_raw))

for index, row in reviews_raw.iterrows():
    if type(row['comments']) == type('s'):
      predictions = model.predict(row['comments'])
      l = predictions[0][0].split('_label_')[1]
      if l != 'ceb' and l != 'nds' and l != 'war' and l != 'wuu':
          reviews_raw['lang'][index] = l

reviews_raw = reviews_raw[reviews_raw['comments'].notna()]

reviews_raw['emoji_count'] = reviews_raw.comments.apply(split_count)

reviews_raw['lenstr'] = reviews_raw['comments'].str.len()

reviews_raw['test_emoji'] = (((reviews_raw['emoji_count'] == reviews_raw['lenstr']) |
                              (2*reviews_raw['emoji_count'] == reviews_raw['lenstr'])) & (reviews_raw['emoji_count']!=0))

reviews_raw = reviews_raw[reviews_raw.test_emoji == False]

reviews_raw = reviews_raw.drop(columns='test_emoji')
reviews_raw = reviews_raw.drop(columns='lenstr')
reviews_raw = reviews_raw.drop(columns='emoji_count')

reviews_raw.comments = reviews_raw.comments.apply(lambda x: x.encode('ascii', 'ignore').decode('ascii'))
reviews_raw['emoji_count'] = reviews_raw.comments.apply(split_count)

reviews_raw = reviews_raw[(reviews_raw.comments.str.len() > 20)]
reviews_raw['comments'] = reviews_raw['comments'].str.replace('<br/>', '')

reviews_raw = reviews_raw[reviews_raw.lang == '_en']

reviews_raw.to_csv("reviews_processed.csv")


reviews = pd.read_csv("reviews_processed.csv", lineterminator='\n')

print("Mean: ", reviews['comments'].str.split().str.len().mean())
print("Median: ", reviews['comments'].str.split().str.len().median())
i = sns.kdeplot(reviews['comments'].str.split().str.len(),color="b", label='Words in each Comment')
xi = i.lines[0].get_xdata()
yi = i.lines[0].get_ydata()
meani = reviews['comments'].str.split().str.len().mean()
mediani = reviews['comments'].str.split().str.len().median()
heighti = np.interp(meani, xi, yi)
heighti2 = np.interp(mediani, xi, yi)
i.vlines(meani, 0, heighti, color='r', ls=':', label='Mean')
i.vlines(mediani, 0, heighti2, color='g', ls=':', label='Median')
plt.legend()
plt.show()

stop = stopwords.words('english')
reviews['comments_stp_rem'] = reviews['comments'].apply(lambda x: ' '.join([word.lower() for word in x.split()
                             if word not in (stop)]))

tr_idf_model  = TfidfVectorizer(analyzer = 'word', max_features=33)
```

```python
tf_idf_vector = tr_idf_model.fit_transform(reviews.comments_stp_rem)

tf_idf_array = tf_idf_vector.toarray()
words_set = tr_idf_model.get_feature_names_out()
df_tf_idf = pd.DataFrame(tf_idf_array, columns = words_set)
df_tf_idf['listing_id'] = reviews.listing_id

test = df_tf_idf.groupby('listing_id').size()
review_1 = pd.DataFrame({'listing_id':test.index, 'num_cmt':test.values})

test2 = pd.DataFrame([])
for i in words_set:
        test2[f'term_{i}'] = df_tf_idf.groupby(['listing_id'])[f'{i}'].mean()

review_final = review_1.merge(test2, on='listing_id')

review_final.to_csv("reviews_final.csv")
```

```python
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.linear_model import Lasso, LogisticRegression, Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, accuracy_score, log_loss
from sklearn.dummy import DummyRegressor, DummyClassifier
from yellowbrick.classifier import ROCAUC
from sklearn.metrics import confusion_matrix, precision_score, recall_score, auc
from sklearn.metrics import f1_score, classification_report, roc_curve
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics
from sklearn.model_selection import cross_val_score

def processing_and_merge(listings, reviews):

  amenities = listings['amenities'].str.split(',', expand=True)
  amenities = amenities.loc[amenities[77].notnull()]
  amenities_list = amenities.iloc[0]
  amenities_list = amenities_list.to_list()

  listings.loc[listings['amenities'].str.contains('Hot Water|Shower gel|Hair dryer|Bathtub|Shampoo|Essentials|Bidet|Conditioner|Body soap|Baby bath'),
               'bath-products'] = 1
  listings.loc[listings['amenities'].str.contains('Bluetooth sound system|Ethernet connection|Heating|Pocket wifi|Cable TV|Wifi'),
               'electric-system'] = 1
  listings.loc[listings['amenities'].str.contains('Breakfast'),
               'food-services'] = 1
  listings.loc[listings['amenities'].str.contains('Outdoor furniture|Dining table|Hangers|High chair|Crib|Clothing storage: wardrobe|Dedicated workspace|Drying rack for clothing|Bed linens|Extra pillows and blankets'),
               'house-furniture'] = 1
  listings.loc[listings['amenities'].str.contains('Cleaning before checkout|Luggage dropoff allowed|Long term stays allowed'),
               'house-rules'] = 1
  listings.loc[listings['amenities'].str.contains('Oven|Hot water kettle|Kitchen|Cooking basics|Microwave|Fire pit|Dishes and silverware|Barbecue utensils|Cleaning products|Baking sheet|Free washer|Free dryer|Iron|Dishwasher|Freezer|Coffee maker|Refrigerator|Toaster|dinnerware|BBQ grill|Stove|Wine glasses'),
                            'kitchen-appliances'] = 1
  listings.loc[listings['amenities'].str.contains('Free parking on premises|Free street parking'),
               'parking'] = 1
  listings.loc[listings['amenities'].str.contains('Board games|Indoor fireplace|Bikes|Shared patio or balcony|Private fenced garden or backyard|crib|books and toys|Outdoor dining area|Private gym in building|Piano|HDTV with Netflix|premium cable|standard cable'),
               'recreation'] = 1
  listings.loc[listings['amenities'].str.contains('Fire extinguisher|Carbon monoxide alarm|Window guards|Fireplace guards|First aid kit|Baby monitor|Private entrance|Lockbox|Smoke alarm|Room-darkening shades|Baby safety gates'),
               'safety'] = 1

  host_verification = listings['host_verifications'].str.split(',', expand=True)

  listings.loc[listings['host_verifications'].str.contains('email'),
               'host_email'] = 1
  listings.loc[listings['host_verifications'].str.contains('phone'),
               'host_phone'] = 1
  listings.loc[listings['host_verifications'].str.contains('work_email'),
               'host_work_email'] = 1

  new_feature_cols = listings.iloc[:,75:].columns
  listings[new_feature_cols] = listings[new_feature_cols].fillna(0)
  listings = listings.merge(reviews, how='inner', left_on='id', right_on='listing_id')

  return listings

def plot_nas(df: pd.DataFrame):
    if df.isnull().sum().sum() != 0:
        na_df = (df.isnull().sum() / len(df)) * 100
        na_df = na_df.drop(na_df[na_df == 0].index).sort_values(ascending=False)
        missing_data = pd.DataFrame({'NaN %' :na_df})
        missing_data.plot(kind = "bar")
        plt.show()
    else:
        print('No NAs found')

def drop_useless_cols(listings):
  not_needed_columns = [
    'id','listing_url', 'scrape_id', 'last_scraped', 'source', 'name',
    'picture_url', 'host_id', 'host_url', 'host_name', 'host_location',
    'host_about', 'host_thumbnail_url', 'host_picture_url', 'host_neighbourhood',
```

```
    'neighbourhood', 'neighbourhood_group_cleansed',
    'calendar_updated', 'first_review', 'last_review', 'license',
    'calculated_host_listings_count', 'calculated_host_listings_count_entire_homes',
    'calculated_host_listings_count_private_rooms', 'calculated_host_listings_count_shared_rooms',
    'description', 'neighborhood_overview', 'host_verifications','host_since',
    'bathrooms', 'bathrooms_text', 'amenities', 'availability_30',
    'availability_60', 'availability_90', 'availability_365', 'calendar_last_scraped',
    'number_of_reviews_ltm', 'number_of_reviews_l30d', 'host_has_profile_pic', 'property_type',
    'minimum_minimum_nights', 'maximum_maximum_nights', 'minimum_nights_avg_ntm',
    'minimum_maximum_nights', 'maximum_minimum_nights', 'maximum_nights_avg_ntm'
    ]

  listings.drop(not_needed_columns, axis = 1, inplace = True)
  listings = listings.dropna()

  imputation_cols = ['bedrooms', 'beds']
  for i in imputation_cols:
      listings.loc[listings.loc[:,i].isnull(),i] = listings.loc[:,i].median()

  return listings

def plot_price_box(listings):
  # Step 1: Clean the `price` column
  listings['price'] = listings['price'].astype(str).str.replace('$', '').str.replace(',', '')
  listings['price'] = pd.to_numeric(listings['price'])

  # Step 2: Clean the `host_response_rate` and `host_acceptance_rate` columns
  listings['host_response_rate'] = listings["host_response_rate"].str.replace("%","")
  listings['host_response_rate'] = pd.to_numeric(listings['host_response_rate'])
  listings['host_acceptance_rate'] = listings["host_acceptance_rate"].str.replace("%","")
  listings['host_acceptance_rate'] = pd.to_numeric(listings['host_acceptance_rate'])

  # Step 3: Plot the initial price boxplot
  plt.figure(figsize=(10, 5))
  listings[['price']].plot(kind='box', title='Price BoxPlot')
  plt.ylim(0, 1600)
  plt.show()

  # Step 4: Filter the `price` column data
  listings_filtered = listings[(listings.price > 50) & (listings.price <= 300)]

  # Step 5: Plot the filtered price boxplot
  plt.figure(figsize=(10, 5))
  listings_filtered[['price']].plot(kind='box', title='Filtered Price BoxPlot')
  plt.show()

  listings_filtered.head()

  # Plot the distribution of host_response_rate
  plt.figure(figsize=(10, 6))
  listings['host_response_rate'].plot(kind='hist', bins=20, edgecolor='black', alpha=0.7)
  plt.title('Distribution of Host Response Rate')
  plt.xlabel('Host Response Rate (%)')
  plt.ylabel('Frequency')
  plt.grid(axis='y', linestyle='--', alpha=0.7)
  plt.show()

  return listings

def plot_review_cols(listings):
  # Convert review scores to numeric and fill NaNs with the mean value of each column
  review_score_columns = [
    'review_scores_accuracy',
    'review_scores_checkin',
    'review_scores_cleanliness',
    'review_scores_communication',
    'review_scores_location',
    'review_scores_rating',
    'review_scores_value'
  ]

  # Ensure all columns are numeric and fill NaNs with the mean value
  for col in review_score_columns:
      listings[col] = pd.to_numeric(listings[col], errors='coerce')
      listings[col].fillna(listings[col].mean(), inplace=True)

  # Plot the distribution of each review score column
  fig, axes = plt.subplots(3, 3, figsize=(18, 18))

  # Define positions to leave a gap in the grid
  positions = [1, 2, 3, 4, 5, 6, 8]  # Subplot positions in a 3x3 grid

  for i, (col, pos) in enumerate(zip(review_score_columns, positions)):
      ax = plt.subplot(3, 3, pos)
      listings[col].plot(kind='hist', bins=20, edgecolor='black', alpha=0.7, ax=ax)
      ax.set_title(f'Distribution of {col}')
      ax.set_xlabel(f'{col}')
      ax.set_ylabel('Frequency')
      ax.grid(axis='y', linestyle='--', alpha=0.7)

  plt.tight_layout()
  plt.show()

def plot_neighborhood(listings):
  # Calculate the number of listings in each neighborhood
  neighbourhood_DF = listings.groupby('neighbourhood_cleansed').host_response_time.count().reset_index()
  neighbourhood_DF = neighbourhood_DF.rename(columns={'host_response_time': 'Number_Of_Listings'})

  # Sort by the number of listings
  neighbourhood_DF = neighbourhood_DF.sort_values(by='Number_Of_Listings', ascending=False)

  # Plot the bar chart
  plt.figure(figsize=(18, 8))
  plt.bar(neighbourhood_DF['neighbourhood_cleansed'], neighbourhood_DF['Number_Of_Listings'])
  plt.title('Dublin Neighborhood Frequency')
  plt.xlabel('Neighborhood')
  plt.ylabel('Number of Listings')
  plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
  plt.show()
```

```
def statics_review_score(listings):
  # Define the list of review score columns
  review_score_columns = [
      'review_scores_accuracy',
      'review_scores_checkin',
      'review_scores_cleanliness',
      'review_scores_communication',
      'review_scores_location',
      'review_scores_rating',
      'review_scores_value'
  ]

  # Calculate and print statistics for each review score column
  for col in review_score_columns:
      print(f"{col} statistics:")
      print(f"Mean: {listings[col].mean()}")
      print(f"Median: {listings[col].median()}")
      print(f"Mode: {listings[col].mode()[0]}")
      print(f"Min: {listings[col].min()}")
      print(f"Max: {listings[col].max()}")
      print("\n")

def plot_Kernel_Density_Estimate(listings):
  plt.figure(figsize=(18, 18))

  # Define the list of review score columns and their subplot positions
  review_score_columns = [
      'review_scores_accuracy',
      'review_scores_checkin',
      'review_scores_cleanliness',
      'review_scores_communication',
      'review_scores_location',
      'review_scores_rating',
      'review_scores_value'
  ]

  positions = [1, 2, 3, 4, 5, 6, 8]  # Subplot positions in a 3x3 grid

  for col, pos in zip(review_score_columns, positions):
      plt.subplot(3, 3, pos)
      density = sns.kdeplot(listings[col], color="b", label=f'{col} - density')
      x = density.lines[0].get_xdata()
      y = density.lines[0].get_ydata()
      mean = listings[col].mean()
      height = np.interp(mean, x, y)
      density.vlines(mean, 0, height, color='b', ls=':')
      density.fill_between(x, 0, y, facecolor='b', alpha=0.2)
      plt.legend()

  plt.tight_layout()
  plt.show()

def minmax(X):
    X_std = (X - X.min()) / (X.max() - X.min())
    X_scaled = X_std * (X.max() - X.min()) + X.min()
    return X_scaled

def scaling_data(listings):
  scaling_data = ['host_response_rate', 'host_acceptance_rate', 'bedrooms', 'beds',
                 'host_listings_count', 'host_total_listings_count',
                 'latitude', 'longitude', 'accommodates', 'price',
                 'minimum_nights', 'maximum_nights', 'number_of_reviews', 'num_cmt', #'avg_senti',
                 'review_scores_rating', 'review_scores_accuracy',
                 'review_scores_cleanliness', 'review_scores_checkin',
                 'review_scores_communication', 'review_scores_location',
                 'review_scores_value', 'reviews_per_month','bath-products','electric-system',
                 'food-services','house-furniture','house-rules',
                 'kitchen-appliances','parking','recreation','safety',
                 'host_email','host_work_email']

  for i in scaling_data:
    listings[i] = minmax(listings[i])

  listings.dropna(axis = 1, inplace = True)
  label_encoder = preprocessing.LabelEncoder()
  listings.host_response_time     = label_encoder.fit_transform(listings.host_response_time)
  listings.host_is_superhost      = label_encoder.fit_transform(listings.host_is_superhost)
  listings.host_identity_verified = label_encoder.fit_transform(listings.host_identity_verified)
  listings.instant_bookable       = label_encoder.fit_transform(listings.instant_bookable)
  listings.room_type              = label_encoder.fit_transform(listings.room_type)
  listings.neighbourhood_cleansed = label_encoder.fit_transform(listings.neighbourhood_cleansed)
  listings.has_availability       = label_encoder.fit_transform(listings.has_availability)

  test_corr = listings.corr()
  test_corr.to_csv("test_corr.csv")

  return listings

def plot_confusion_matrix(y_test, y_pred, title):
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(title)
    plt.show()

def ROCAUC_visualizer(model, bin_count, X, y):
    y = y['rating_bin_ep'].astype(int)
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
    y_train = y_train.astype(int)
    y_test = y_test.astype(int)

    visualizer = ROCAUC(model, classes=list(range(bin_count)), macro=False, micro=False)
    visualizer.fit(x_train, y_train)
    visualizer.score(x_test, y_test)
    visualizer.show()

def bin_column(listings, col, n_bins):
  X = listings[
                ['host_response_time', 'host_response_rate', 'host_acceptance_rate',
                 'bedrooms', 'beds','neighbourhood_cleansed',
                 'host_is_superhost', 'host_listings_count', 'host_total_listings_count',
                 'host_identity_verified', '
```

```python
    room_type',
                  'accommodates','price', 'minimum_nights', 'maximum_nights',
                  'bath-products','electric-system',
                  'food-services','house-furniture','house-rules',
                  'kitchen-appliances','parking','recreation','safety',
                  'host_email','host_work_email'] + list(reviews.columns[2:])
  ]

  y = listings[[col]]
  y = (y/y.max())*100

  y = y.assign(
      rating_bin_ep = pd.qcut(
          y[col],
          q=n_bins,
          duplicates='drop',
          labels=list(range(n_bins))
      )
  )

  y.groupby('rating_bin_ep').min()
  y.groupby('rating_bin_ep').max()

  return X , y

def select_important_features(X, y, threshold=0.05):

    correlations = X.corrwith(y).abs().sort_values(ascending=False)
    selected_features = correlations[correlations > threshold].index

    plt.figure(figsize=(10, 6))
    sns.barplot(x=selected_features, y=correlations[selected_features], palette="viridis")
    plt.axhline(y=threshold, color='r', linestyle='--', label=f'Threshold ({threshold})')
    plt.xlabel('Features')
    plt.ylabel('Correlation with target')
    plt.title('Selected Features and Their Correlations')
    plt.xticks(rotation=90)
    plt.legend()
    plt.show()

    return selected_features, correlations

def check_bins(y):
  y1 = y['rating_bin_ep']
  cnt_plt = sns.countplot(y1)
  cnt_plt.bar_label(cnt_plt.containers[0])
  plt.show()

def evaluate_logistic_regression(X, y, c_range, degree_range=[1], bin_count=2):

    y1 = y['rating_bin_ep']
    for i in degree_range:
        trans = PolynomialFeatures(degree=i)
        x_poly = trans.fit_transform(X)
        x_train, x_test, y_train, y_test = train_test_split(x_poly, y1, test_size=0.2, random_state=1)
        mean_error = []
        std_error = []
        for c in c_range:
            log_reg = LogisticRegression(C=c, random_state=0, solver='newton-cg', multi_class='multinomial')
            log_reg.fit(x_train, y_train)
            y_pred = log_reg.predict(x_test)

            cnf_mtx = confusion_matrix(y_test, y_pred)
            f1_score = metrics.f1_score(y_test, y_pred, average='weighted')

            scores = cross_val_score(log_reg, x_test, y_test, cv=5, scoring='accuracy')
            mean_error.append(np.array(scores).mean())
            std_error.append(np.array(scores).std())

            print(" Logistic Regression")
            print(" For Degree = ", i)
            print(" For C = ", c)
            print(" Confusion Matrix - \n", cnf_mtx)
            print(' Train accuracy score: ', log_reg.score(x_train, y_train))
            print(' Test accuracy score: ', log_reg.score(x_test, y_test))
            print(" F1 Score = ", f1_score)
            print(" Mean Squared Error = ", mean_squared_error(y_test, y_pred))
            print(" Classification Report\n", classification_report(y_test, y_pred))
            print("\n")

            title = f"Logistic Regression Confusion Matrix\nFor Degree = {i} and C = {c}"
            plot_confusion_matrix(y_test, y_pred, title)

        plt.errorbar(c_range, mean_error, yerr=std_error, linewidth=3)
        plt.xlabel('C', fontsize=25)
        plt.ylabel('Accuracy Score', fontsize=25)
        title_cv = f"Logistic Regression - Cross Validation \nFor Degree = {i}"
        plt.title(title_cv, fontsize=25)
        plt.show()

        ROCAUC_visualizer(log_reg, bin_count, x_poly, y)

def evaluate_knn(X, y, nn_range, bin_count=2):
    y1 = y['rating_bin_ep']
    x_train, x_test, y_train, y_test = train_test_split(X, y1, test_size=0.2, random_state=1)
    merr = []
    serr = []

    for nn in nn_range:
        knn_model = KNeighborsClassifier(n_neighbors=nn, weights='uniform')
        knn_model.fit(x_train, y_train)
        y_pred_nn = knn_model.predict(x_test)

        cnf_mtx = confusion_matrix(y_test, y_pred_nn)
        f1_score = metrics.f1_score(y_test, y_pred_nn, average='weighted')

        scores_knn = cross_val_score(knn_model, x_test, y_test, cv=5, scoring='accuracy')
        merr.append(np.array(scores_knn).mean())
        serr.append(np.array(scores_knn).std())

        print(" K Neighbors Classifier")
        print(" For NN = ", nn)
        print(" Confusion Matrix - \n", cnf_mtx)
        print(' Train accuracy score: ', knn_model.score(x_train, y_train))
```

```
        print(' Test accuracy score: ', knn_model.score(x_test, y_test))
        print(" F1 Score = ", f1_score)
        print(" Mean Squared Error = ", mean_squared_error(y_test, y_pred_nn))
        print(" Classification Report\n", classification_report(y_test, y_pred_nn))
        print("\n")

        title = f"k-NN Confusion Matrix\nFor NN = {nn}"
        plot_confusion_matrix(y_test, y_pred_nn, title)

    plt.errorbar(nn_range, merr, yerr=serr, linewidth=3)
    plt.xlabel('NN', fontsize=25)
    plt.ylabel('Accuracy Score', fontsize=25)
    title_cv = f"k-NN - Cross Validation"
    plt.title(title_cv, fontsize=25)
    plt.show()

    ROCAUC_visualizer(knn_model, bin_count, X, y)

def evaluate_decision_tree(X, y, depth_range, bin_count=2):
    y1 = y['rating_bin_ep']
    x_train, x_test, y_train, y_test = train_test_split(X, y1, test_size=0.2, random_state=1)
    merr_dt = []
    serr_dt = []

    for depth in depth_range:
        dt_model = DecisionTreeClassifier(max_depth=depth, random_state=1)
        dt_model.fit(x_train, y_train)
        y_pred_dt = dt_model.predict(x_test)

        cnf_mtx = confusion_matrix(y_test, y_pred_dt)
        f1_score = metrics.f1_score(y_test, y_pred_dt, average='weighted')

        scores_dt = cross_val_score(dt_model, x_test, y_test, cv=5, scoring='accuracy')
        merr_dt.append(np.array(scores_dt).mean())
        serr_dt.append(np.array(scores_dt).std())

        print(" Decision Tree Classifier")
        print(" For Depth = ", depth)
        print(" Confusion Matrix - \n", cnf_mtx)
        print(' Train accuracy score: ', dt_model.score(x_train, y_train))
        print(' Test accuracy score: ', dt_model.score(x_test, y_test))
        print(" F1 Score = ", f1_score)
        print(" Mean Squared Error = ", mean_squared_error(y_test, y_pred_dt))
        print(" Classification Report\n", classification_report(y_test, y_pred_dt))
        print("\n")

        title = f"Decision Tree Confusion Matrix\nFor Depth = {depth}"
        plot_confusion_matrix(y_test, y_pred_dt, title)

    plt.errorbar(depth_range, merr_dt, yerr=serr_dt, linewidth=3)
    plt.xlabel('Depth', fontsize=25)
    plt.ylabel('Accuracy Score', fontsize=25)
    title_cv = f"Decision Tree - Cross Validation"
    plt.title(title_cv, fontsize=25)
    plt.show()

    ROCAUC_visualizer(dt_model, bin_count, X, y)

def plot_kde_by_bin(X, y, feature, bin_column, bins=3, ylim=(0, 19), xlim=(-0.25, 1)):
    test = pd.concat([X, y], axis=1)
    colors = ['r', 'g', 'b']
    labels = [f'{feature}, bin={i}' for i in range(bins)]

    for i in range(bins):
        sns.kdeplot(test.loc[test[bin_column] == i, feature], color=colors[i], label=labels[i]).set(ylim=ylim, xlim=xlim)
        plt.legend()

    plt.xlabel(feature)
    plt.ylabel('Density')
    plt.title(f'KDE of {feature} by {bin_column}')
    plt.show()

def plot_super_host(X, y):
  test = pd.concat([X, y], axis=1)
  test = test.groupby(['host_is_superhost', 'rating_bin_ep']).size().unstack()

  plt.figure(figsize=(18, 6))

  plt.subplot(1, 2, 1)
  for j in range(3):
     plt.bar(j, test.loc[1, j], label=f'Bin {j}', alpha=0.7)
  plt.title("Host is a Super Host")
  plt.legend()
  plt.xlabel('Review Bin', fontweight='bold', fontsize=15)
  plt.ylabel('# Records', fontweight='bold', fontsize=15)

  plt.subplot(1, 2, 2)
  for j in range(3):
     plt.bar(j, test.loc[0, j], label=f'Bin {j}', alpha=0.7)
  plt.title("Host is not a Super Host")
  plt.legend()
  plt.xlabel('Review Bin', fontweight='bold', fontsize=15)
  plt.ylabel('# Records', fontweight='bold', fontsize=15)

  plt.show()

# load csv
listings = pd.read_csv("listings.csv")
reviews = pd.read_csv("reviews_final.csv")

# listings pre-processing
listings = processing_and_merge(listings, reviews)
plot_nas(listings)
listings = drop_useless_cols(listings)
listings = plot_price_box(listings)
plot_review_cols(listings)
plot_neighborhood(listings)
statics_review_score(listings)
plot_Kernel_Density_Estimate(listings)
listings = scaling_data(listings)

# model parmater
c_range = [0.001, 0.1, 1, 10, 100, 1000]
degree_range = [1, 2]
nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
depth_range = [3, 5, 7, 9, 11, 13, 15, 17, 19]
```

```
# review_scores_checkin evaluating
X,y = bin_column(listings, 'review_scores_checkin', 2)
check_bins(y)
yy = listings['review_scores_checkin']
select_important_features(X, yy, threshold=0.01)
evaluate_logistic_regression(X, y, c_range, degree_range, 2)
evaluate_knn(X, y, nn_range, 2)
evaluate_decision_tree(X, y, depth_range, 2)

# review_scores_communication evaluating
X,y = bin_column(listings, 'review_scores_communication', 2)
check_bins(y)
yy = listings['review_scores_communication']
select_important_features(X, yy, threshold=0.01)
evaluate_logistic_regression(X, y, c_range, degree_range, 2)
evaluate_knn(X, y, nn_range, 2)
evaluate_decision_tree(X, y, depth_range, 2)

# review_scores_cleanliness evaluating
X,y = bin_column(listings, 'review_scores_cleanliness', 3)
check_bins(y)
yy = listings['review_scores_cleanliness']
select_important_features(X, yy, threshold=0.01)
evaluate_logistic_regression(X, y, c_range, degree_range, 3)
evaluate_knn(X, y, nn_range, 3)
evaluate_decision_tree(X, y, depth_range, 3)

# review_scores_location evaluating
X,y = bin_column(listings, 'review_scores_location', 3)
check_bins(y)
yy = listings['review_scores_location']
select_important_features(X, yy, threshold=0.01)
evaluate_logistic_regression(X, y, c_range, degree_range, 3)
evaluate_knn(X, y, nn_range, 3)
evaluate_decision_tree(X, y, depth_range, 3)

# review_scores_rating evaluating
X,y = bin_column(listings, 'review_scores_rating', 3)
check_bins(y)
yy = listings['review_scores_rating']
select_important_features(X, yy, threshold=0.01)
evaluate_logistic_regression(X, y, c_range, degree_range, 3)
evaluate_knn(X, y, nn_range, 3)
evaluate_decision_tree(X, y, depth_range, 3)

# review_scores_value evaluating
X,y = bin_column(listings, 'review_scores_value', 3)
check_bins(y)
yy = listings['review_scores_value']
select_important_features(X, yy, threshold=0.01)
evaluate_logistic_regression(X, y, c_range, degree_range, 3)
evaluate_knn(X, y, nn_range, 3)
evaluate_decision_tree(X, y, depth_range, 3)

# review_scores_accuracy evaluating
X,y = bin_column(listings, 'review_scores_accuracy', 3)
check_bins(y)
yy = listings['review_scores_accuracy']
select_important_features(X, yy, threshold=0.01)
evaluate_logistic_regression(X, y, c_range, degree_range, 3)
evaluate_knn(X, y, nn_range, 3)
evaluate_decision_tree(X, y, depth_range, 3)

# kde term home and num cmt
plot_kde_by_bin(X, y, 'term_home', 'rating_bin_ep', bins=3, ylim=(0, 19), xlim=(-0.25, 1))
plot_kde_by_bin(X, y, 'num_cmt', 'rating_bin_ep', bins=3, ylim=(0, 0.03), xlim=(-200, 1600))

# superhost
plot_super_host(X, y)
```

Listing 2. Python code of supplement.py

```
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

def binary_step(x, thresh=10):
  return np.where(x<thresh, 0, 1)

def decision_boundary(model):
    b = model.intercept_[0]
    w1, w2 = model.coef_[0]

    c = -b / w2
    m = -w1 / w2

    xd = np.linspace(X.x_1.min(), X.x_1.max())
    yd = m * xd + c

    return xd, yd

size = 250
# NO LINEAR CORELATION
X = pd.DataFrame()
X['x_1'] = np.random.rand(size, )
X['x_2'] = np.random.randint(0, 60, size=size)

y = binary_step(10 + np.random.normal(0.0,1.0,size))

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

model_no_linear_corr = LogisticRegression().fit(X_train, y_train)
ypred = model_no_linear_corr.predict(X)

print("Logistic Regression - No Colinearity")
print(f"Model Score: {model_no_linear_corr.score(X, y)}")
print(confusion_matrix(y, model_no_linear_corr.predict(X)))
print(classification_report(y, model_no_linear_corr.predict(X)))

pred_actual_logi = pd.DataFrame({'x_1': X['x_1'],
               'x_2': X['x_2'],
             'Ground Truth': y,
             'Prediction': ypred})

# sns.scatterplot(x=X.x_2, y=X.x_1, style=y, hue=y, palette='deep')

xd, yd = decision_boundary(model_no_linear_corr)
```

```python
for idx, val in enumerate(yd):
    if val < 0:
        yd[idx] = 0
    elif val > 50:
        yd[idx] = 50

sns.scatterplot(x=X.x_1, y=X.x_2, style=ypred, hue=ypred, palette='deep')
plt.plot(xd, yd)

# DATA IMBALANCE
X = pd.DataFrame()
size = 2000
X['x_1'] = np.arange(0,2,0.05)

y = 10 * X.x_1 + np.random.normal(0.0,1.0,X.size)

for idx, val in enumerate(y):
    if val < 2:
        y[idx] = int(0)
    else:
        y[idx] = int(1)
y = y.astype('int')

model_data_imbalance = LogisticRegression().fit(X, y)

print("Logistic Regression - Data Imbalance")
print(f"Model Score: {model_data_imbalance.score(X, y)}")

print(confusion_matrix(y, model_data_imbalance.predict(X)))
print(classification_report(y, model_data_imbalance.predict(X)))

sns.scatterplot(x=X.x_1, y=y, style=y, hue=y, palette='deep')
plt.legend(loc='lower right')
plt.ylabel('Target')
plt.xlabel('Feature')

ypred = model_data_imbalance.predict(X)
sns.scatterplot(x=X.x_1, y=ypred, style=ypred, hue=ypred, palette='deep')
plt.legend(loc='lower right')
ypred = model_data_imbalance.predict(X)
plt.ylabel('Predicted')
plt.xlabel('Feature')

pred_data_imbalance_logi = pd.DataFrame({'x_1': X['x_1'],
            'Ground Truth': y,
            'Prediction': model_data_imbalance.predict(X)})

# DUMMY CLASSIFIER
model_dummy = DummyClassifier(strategy='most_frequent').fit(X, y)

print("Dummy Classifier - Data Imbalance")
print(f"Model Score: {model_dummy.score(X, y)}")
print(confusion_matrix(y, model_dummy.predict(X)))
print(classification_report(y, model_dummy.predict(X)))

pred_data_imbalance_dummy = pd.DataFrame({'x_1': X['x_1'],
            'Ground Truth': y,
            'Prediction': model_dummy.predict(X)})

# HOLD OUT METHOD
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd

X = np.arange(0,1,0.05).reshape(-1, 1)
y = 10 * X + np.random.normal(0.0,1.0,X.size).reshape(-1, 1)

intercept = []
slope = []
mean_error = []

for i in range(5):
    Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,test_size=0.2)

    model = LinearRegression().fit(Xtrain, ytrain)
    ypred = model.predict(Xtest)

    intercept.append(model.intercept_[0])
    slope.append(model.coef_[0][0])
    mean_error.append(mean_squared_error(ytest, ypred))

    print('Intercept: {:.2f}\nSlope: {:.2f}\nSquared Error: {:.2f}'.format(model.intercept_[0],
                                                                 model.coef_[0][0],
                                                                 mean_squared_error(ytest, ypred)))
    print('\n\n')

    y_vals = model.intercept_ + X * model.coef_
    plt.plot(X, y_vals, label='{:.2f}'.format(mean_squared_error(ytest, ypred)))

vals = pd.DataFrame({
    'intercept': intercept,
    'slope': slope,
    'mean_error': mean_error})

plt.scatter(X, y, c='black')
plt.xlabel('Input X')
plt.ylabel('Target y')
plt.legend(title="MSE", fancybox=True)
plt.show();

import numpy as np
arr = np.arange(3.0, 5.5, 0.05)

# LAGGED OUTPUT
import pandas as pd
data = pd.DataFrame({
    'Date': ['2024-07-15','2024-08-15','2024-09-15','2024-10-15','2024-11-15','2024-12-15'],
    'Product Usage': [1000,2000,4000,8000,16000,32000],
    'tag_1': [ None,1000,2000,4000,8000,16000],
    'tag_2': [None,None,1000,2000,4000,8000],
    })
print(data)
```