# CS7CS4 – Machine Learning Final Assignment 2022
# Airbnb Listings Review Ratings Prediction

Abhik Subir Bhattacharjee
bhattaa2@tcd.ie
*22305544*

## 1. INTRODUCTION AND DATASETS

Airbnb is an online marketplace which focuses on providing individuals with properties which specialise on homestay, rental and travelling experiences. Based on the various attributes of a given listing, the review rating scores for various domains like cleanliness, communication, value, etc. may differ. The main intention of this assignment is to determine the potential rating of any given Airbnb listing in Dublin based on the features and facilities it provides. The raw dataset for this particular work was taken from the *Inside Airbnb* website and the model was built using the *Listings* and *Reviews* Data available from the website. The *Listings* data consists of total 75 feature columns with total of 7566 listings. The *Reviews* data consists of total 6 feature columns and a total of 243183 reviews.

## 2. FEATURE ENGINEERING

The listings data consists in total of **75 feature columns** of which **10 columns** consist of the **data scraping information** along with some **audit columns** *(listing_url, scrape_id, last_scraped, source, picture_url, host_id, host_url, host_thumbnail_url, host_picture_url, calendar_last_scraped)*. These columns were dropped from the dataset to make the data easy to comprehend and more readable. Similarly, for the reviews dataset, out of the total **6 features** present in the raw data, only the *listing_id* and *comments* feature were considered for further analysis. The detailed pre-processing steps for the respective datasets are described in sub-sequent sections.

### 2.1 LISTINGS DATA PRE-PROCESSING

The listings dataset had some key amenities listed for a given property mentioned under the *amenities* column. It was key that people look out for certain amenities while selecting their stay for their next vacation. Hence it was essential to split those different amenities consolidated together in one single column for a given listing into separate categories. In total there are **77 distinct amenities** possible for any given listing which I have further clubbed together into **9 different categories**. The clubbing of amenities into one broader domain is based on logical clustering like amenities such as *'Hot shower', 'Shower gel', 'Hair dryer', 'Shampoo', 'Conditioner', 'Body Soap', etc.* being clubbed together under the *'Bath Products'* category, while amenities such as *'Oven', 'Hot water kettle', 'Cooking basics', 'Microwave', etc.* being clubbed together under the *'Kitchen Appliances'* category.

Similarly, for a given host, the contact methods are listed down in a single column named '*host_verifications*'. The different types of contact methods namely *'host_email', 'host_phone' and 'host_work_email'* are separated and made independent features where if a given host has any of the aforementioned contact methods, the feature column is *populated as 1 else 0*. The details of the various amenities and contact features clubbed into a single category are further described in *Table 1*.

It is quite possible that a given listing has none of the amenities from the 9 broad categories or does not have any of the contact information listed in the host verification columns. In such cases, I have populated all *NaN* values in all the newly created feature columns as 0.

Moving forward, it was important to understand the number of *NaN* values present in each of the feature columns. *Figure 1* consists the *% NaN* values present in each of the columns present in the Listings dataset.

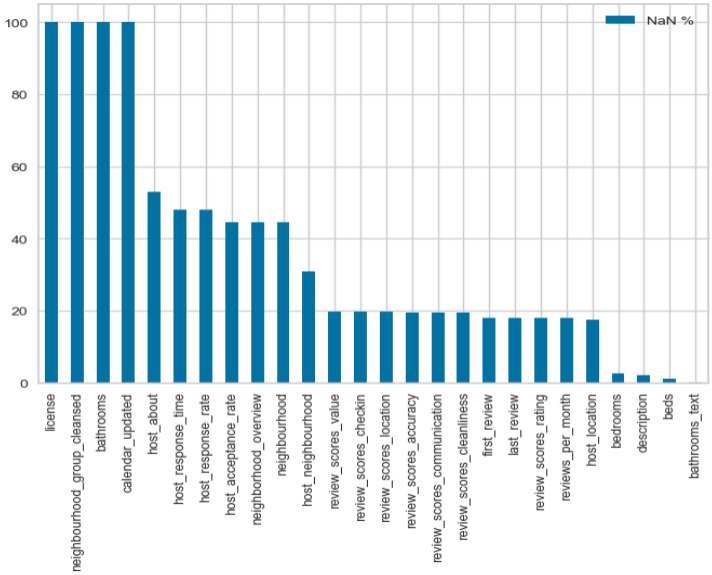



**Figure 1:** %NaN values present in each feature columns from Listings dataset

Based on the values presented in the above graph, we could

| Amenities | Broader Category – New Feature | Column Fill Value |
|---|---|---|
| Hot Water, Shower gel, Hair dryer, Bathtub, Shampoo, Essentials, Bidet, Conditioner, Body soap, Baby bath | bath-products | If any one of the mentioned amenities present, then 1<br>Else 0 |
| Bluetooth sound system, Ethernet connection, Heating, Pocket wifi, Cable TV, Wifi | electric-system | |
| Breakfast | food-services | |
| 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 | |
| Cleaning before checkout, Luggage dropoff allowed, Long term stays allowed | house-rules | |
| 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 | |
| Free parking on premises, Free street parking | parking | |
| 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 | |
| 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 | |

**Table 1:** Amenities to Broad Category Mapping

see that the features *neighbourhood_group_cleansed, license, bathrooms and calendar_updated* are composed of NaN values. Hence those 4 columns are completely dropped from the dataset for further analysis. Also if we see the 7 *Review Scores* columns which are our *target variables*, we can see that on an average 20% of those rows are *NaN* values. Since those are target values for our model, imputing those values isn't the right choice as there is a chance that the model would be *biased*. Hence, for all the other data records which have *NaN* values, the complete row is dropped from the dataset for further analysis as those won't add any value to the model we are trying to build for this particular use-case.

Now on closely looking at the feature *'price'*, we can see that the price is listed down in $ Dollars. In order to make the value machine readable and ready for further analysis, I have replaced '$' and ',' with null and *converted the string value to integer*. We also observe that the feature has a lot of outliers. The **average** price of any given listing is **~$182** with the **median** being **$105**. On the contrary, the **maximum price** of any listing is **$99,149**.

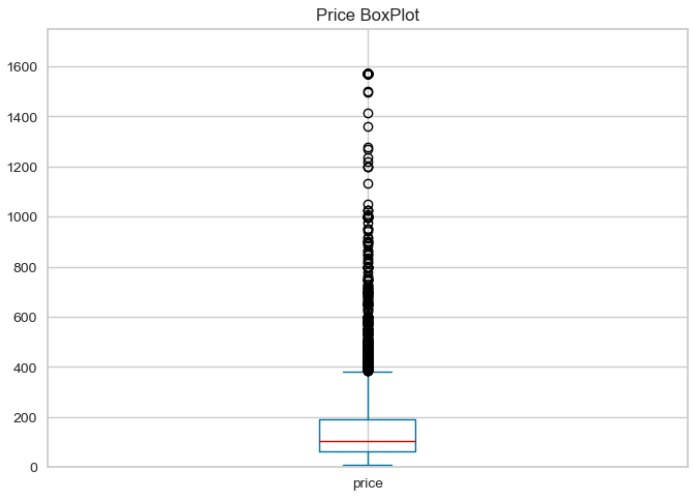



**Figure 2:** Box Plot of Price Feature

From the values presented in Figure 2, we can see that the majority of the values lie within the **range of $50 and $300**. Hence I have dropped all the rows which have a price value not in the range mentioned above. Now for all the 7 target variables which our model needs to predict, we can see that

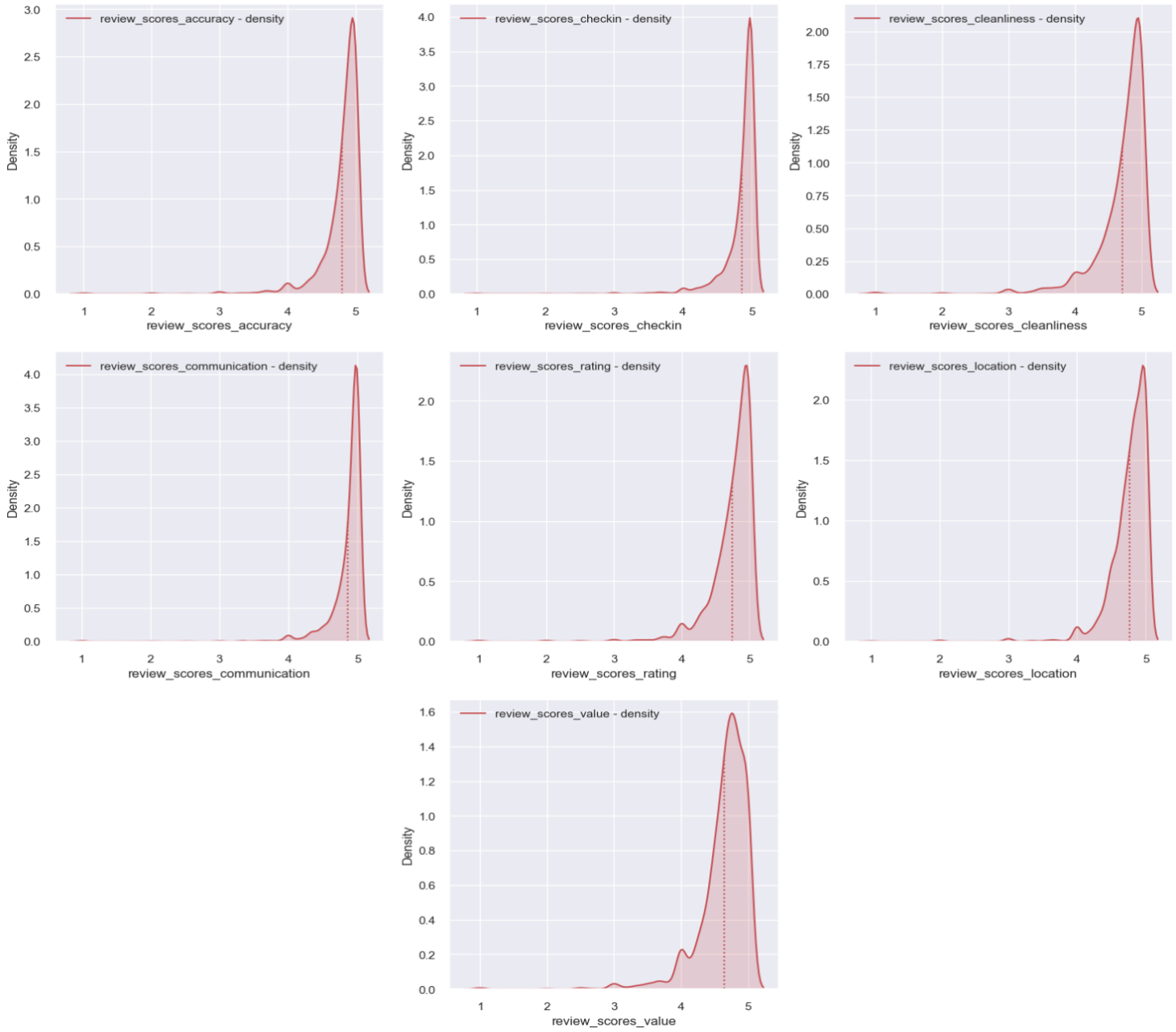



**Figure 3:** Distribution Plot of all the Target Columns (Review Rating Columns)

the rating values are very close to each other and all tend to be high. Please refer to Figure 3 above showing that all the rating columns are heavily concentrated in the **range between 4.5 and 5**. In order to tackle that, I have used the ***Binning Approach*** to handle such close numbers.

In the *Binning Approach*, we categorize the continuous data to be present in one discrete bin based on a particular criteria which makes it easier for further analysis. For this particular use case, I have decided to bin the individual rating values based on the *quantile range* they belong to. The major reason to use this approach is that the quantile method allows us to **create equal sized bins** in which the data needs to be divided.

By the virtue of this feature of categorizing a given rating value to a discrete equal sized bins based on its quantile range, we are not making the dataset biased in which number of records in one bin is more than others thus making it a good choice for this particular problem statement. After the values are binned, we now have to predict the rating bin which a given listing may lie into rather than predicting the exact numeric rating of a listing making this problem a ***Classification Problem*** where the model needs to predict a categorical variable given some input parameters. In order to categorize a given rating to a discrete bin, I have used the ***Pandas DataFrame QCut Functionality*** which divides up the underlying data into **equal**

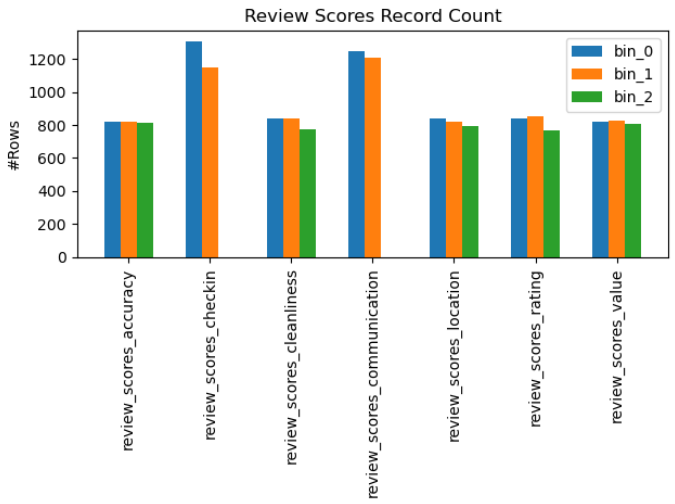



**Figure 4:** #Records in each bin for each Review Score/Target Variable

| Review Scores | Metric | Bin 0 | Bin 1 | Bin 2 |
|---|---|---|---|---|
| review_scores_accuracy | Min | 0 | 4.81 | 5 |
| | Max | 4.8 | 4.99 | 5 |
| review_scores_checkin | Min | 0 | 4.96 | N/A |
| | Max | 4.95 | 5 | |
| review_scores_cleanliness | Min | 0 | 4.68 | 4.94 |
| | Max | 4.67 | 4.93 | 5 |
| review_scores_communication | Min | 0 | 4.98 | N/A |
| | Max | 4.97 | 5 | |
| review_scores_location | Min | 1 | 4.71 | 4.94 |
| | Max | 4.7 | 4.93 | 5 |
| review_scores_rating | Min | 0 | 4.68 | 4.93 |
| | Max | 4.67 | 4.92 | 5 |
| review_scores_value | Min | 0 | 4.59 | 4.84 |
| | Max | 4.58 | 4.83 | 5 |

**Table 2:** Bin ranges for each review score category

**sized bins**. The function defines the bins using percentiles based on the distribution of the data, and not the actual numeric edges of the bins. Please refer to Figure 4 above showing the number of records in each bin for every review column that our model needs to predict. **Please note that the ratings are first converted to percentage before they are categorized to a bin.** For the features *host_response_rate and host_acceptance_rate* , I have replaced the *'%'* signs in the feature columns with *Null* and converted the resultant figure into numeric so that it can be further processed. The feature columns *host_response_time , host_is_superhost, host_identity_verified, instant_bookable, room_type, neighbourhood_cleansed, and has_availability* are **label encoded** using sklearn's preprocessing library so that it can be machine readable for the learning model we decide for this particular problem statement. For scaling, I have used **min-max scaling** as it *retains the shape* of the original distribution of the data. In min-max scaling, every data point is scaled to a new value using the formula –

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

The list of columns which are scaled and dropped from the listings dataset are mentioned in Appendix [1] and [2].

## 2.2 REVIEWS DATA PRE-PROCESSING

The reviews dataset consists of features such as *listing_id, id, date, reviewer_id, reviewer_name and comments.* In total there are **243183 reviews** wherein one listing may have multiple reviews tagged to it. For our further analysis, I have only considered the features *listing_id and comments.* For the feature comments, we can see that there are Non-English comments along with some comments which only contain special characters, emojis and some HTML tags like *<br/>*. In order to make those features usable, we have to do some pre-processing like removal of Emojis and special characters and only consider English reviews. First we drop all the review rows which do not have any comment associated with it. Then we have to identify only the English reviews and discard the reviews of other languages. In order to identify the language of the review text, I have used the ***fasttext Python Library*** along with its pretrained model which helps identify the language of any given text. The *fasttext* library is a very efficient language identification library which is based on n-gram features, dimensionality reduction and a fast approximation of the softmax classifier (Joulin et al., 2016).

Moving forward, we need to identify and remove all the emojis from a given text. In order to identify the number of emojis in a given comment, I have used the ***emoji Python Library***. Here I have read every comment character by character and identified if the character read matches any *EMOJI_DATA* from the *emoji* library. Now if the length of the characters in the comment text is equal to the emoji count or is double the emoji count, we need to drop that particular data point as the comment is composed fully of emojis without any actual text which holds some significant information. I have introduced the *2x condition* while dropping the data points since for emojis with skin tone, *EMOJI_DATA* splits the actual emoji and the skin tone which makes it 2 counts for a single emoji (Example *Table 3*).

Also on further deep-diving on the comments from the review dataset, we can see that some comments are very small and

might contain some ASCII characters such as *':)'* or '*OK*'. In order for a comment to hold a significance and relevant so that it can be considered as a feature on which a model can be trained, I have only selected comments which are greater than 20 in length. After all the pre-processing was completed, we are left **206287 out of 243183 reviews**. This means that we have dropped **~15%** of the reviews thus leaving us with only relevant English language reviews which we can use to train our model on. The total ***Number of Comments*** for a given listing is updated after the pre-processing is completed.

| Original Emoji | Python Interpretation | Emoji Count | Length String |
|---|---|---|---|
| 👍🏻 | 👍 | 1 | 2 |

**Table 3:** String length for Coloured Emojis

### 2.2.1 TF-IDF ON REVIEW COMMENTS

In order to use the actual comment text from the Review Dataset, we have to do some pre-processing on top of it in order to make it machine readable. Since we are dealing with actual review text, the first course of action taken to deal with English texts was to implement the *Term Frequency – Inverse Document Frequency methodology (TF-IDF)*. *TF-IDF* is used to determine how important a given word is to a given collection of text, in our case a review for a given listing. This would help us determine the user satisfaction or dis-satisfaction towards a particular listing when the guest had visited and experienced the Airbnb property. *TF-IDF* of term $t$ in a document collection $d$ is defined as –

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

$$Where\ tf(t,d) = \#times\ a\ term\ t\ occurs\ in\ document\ d$$

$$idf(t) = 1 + log\frac{1 + \#Documents\ in\ D}{1 + df(t)}$$

The main reason why *TF-IDF* was considered for this particular review text set was to identify the *Important Tokens* from every review text and have it as a feature to the machine learning model that we are trying to build for this particular use case. After all the pre-processing is completed as mentioned in *Section 2.1* on the review dataset, we first remove all the stop words from the comments of every review.

For this particular use case to implement *TF-IDF*, I have used ***TfidfVectorizer*** from sklearn's text feature extraction library *sklearn.feature_extraction.text*. Since we are trying to identify the most relevant words from a given review comment, I have used the *TfidfVectorizer* ***Word Analyzer*** in order to identify the most relevant features from a given comment. I have also restricted the number of **maximum features** returned by the *TfidfVectorizer* to **33**.

The number of features returned by the *TfidfVectorizer* are restricted in the code for 2 main reasons. Based on the number of words present in each review comment, we can see that there are a lot of outliers when the number of words in each review comment are considered. There are some very small comments of 10-15 words; on the other hand there are also some very huge comments which are in the range of 500-1000 words.

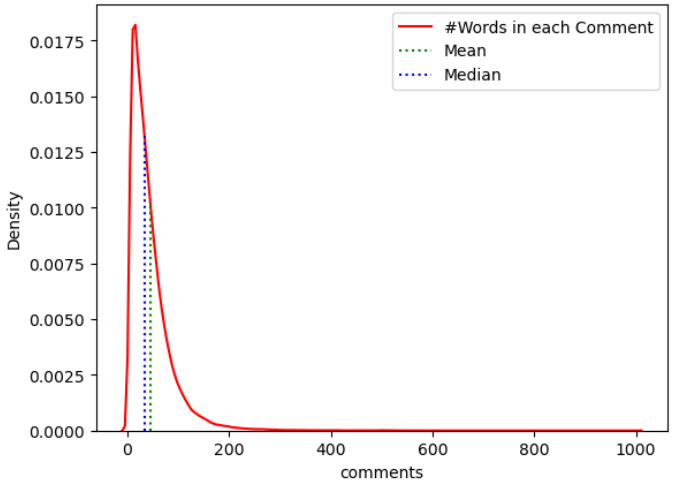



**Figure 5:** Density plot of #Words in each comment

Based on the values presented in *Figure 5*, we can see that most of the data is concentrated in the range between 20 and 40 words per comment. Based on the value statistics of the #Words per comment, we see that the **Mean is ~45 and Median is 33**. Since the Median identifies the *central tendency* of the data distribution, **restricting the max features returned by the Vectorizer to the Median** of the data would be the ideal choice.

Additionally, if we increase the number of features returned by the Vectorizer, there is a chance that the data maybe biased and the model would result into an overfitted model, which would result into higher training accuracy but very low testing accuracy.

Finally after the *TF-IDF* for all the review comments are completed, I have calculated the **Mean** of all the features created by the *TfidfVectorizer* by grouping it on the basis of a given *Listing ID*. By this, there is only one row for one listing so that there is no discrepancy while merging this dataset with the Listings dataset.

## 2.3 FEATURE SELECTION AND IMPORTANCE

Now all the records from the Listings and Reviews dataset are cleansed and processed, we can move forward and identify all the important features from the dataset for a given Machine

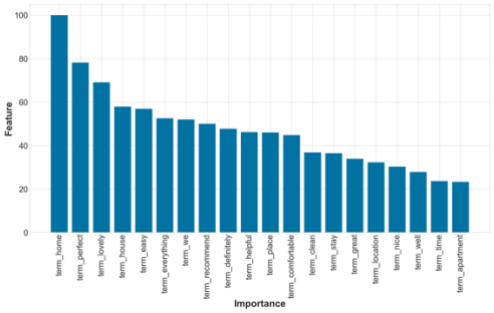



**Figure 6.1:** Important Features for Bin 0 - *review_scores_accuracy*

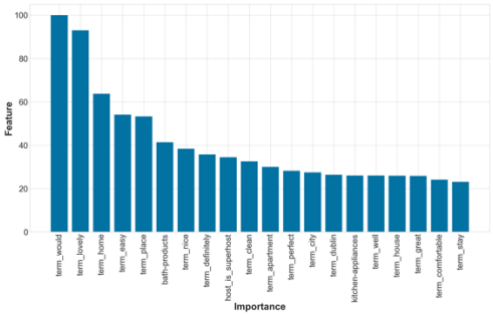



**Figure 6.2:** Important Features for Bin 1 - *review_scores_accuracy*

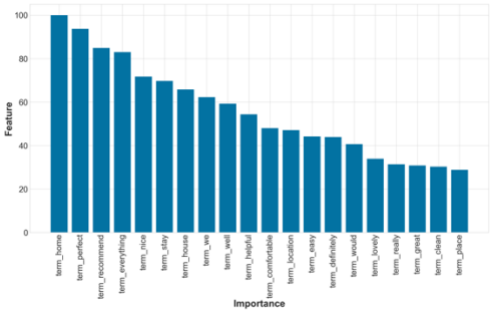



**Figure 6.3:** Important Features for Bin 2 - *review_scores_accuracy*

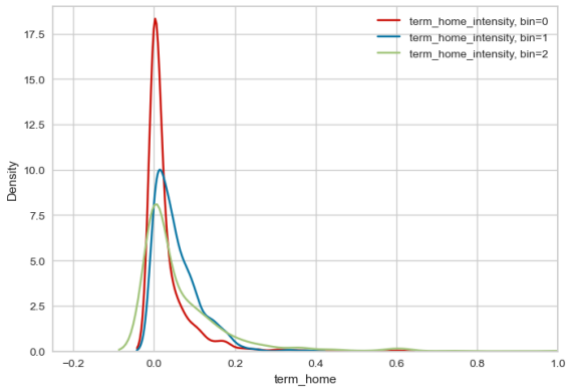



**Figure 7.1:** Intensity of Term *Home* for all 3 Bins

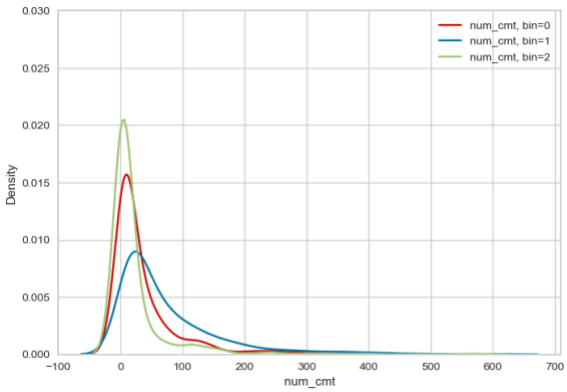



**Figure 7.2:** Intensity of #*Comments* for all 3 Bins

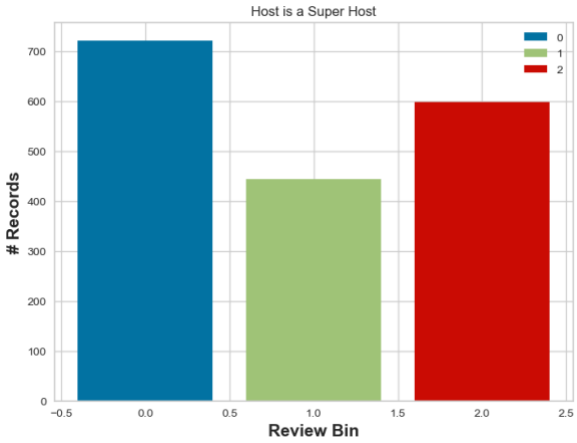



**Figure 7.3:** *#Records* in all 3 Bins when Host is *Super Host*

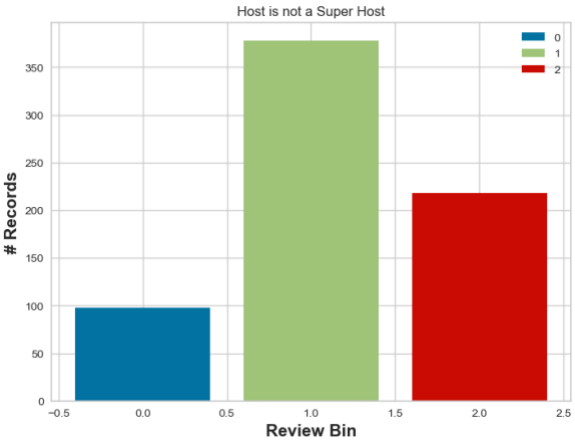



**Figure 7.4:** *#Records* in all 3 Bins when Host is Not a *Super Host*

Learning model. In order to create a training set, I have merged the listings dataset with the processed dataset created in *Section 2.2.1* on the basis of *listing_id.* In order to identify all the main important features from the dataset, I have fitted a Logistic Regression model initially and extracted the weights of the features from the model coefficients. As shown in *Figure 6*, we can see that in order to have a listing with very high rating (Bin 2), the most important terms or top 3 tokens in the review comments are *'Home', 'Perfect' and 'Recommend'* while the listing to have a rating in Bin 1, the review must have the terms *'Would', 'Lovely' or 'Home'* and the listing must have the *Kitchen Appliances* amenities. We can also see in *Figure 7* that the intensity of term *'Home' and #Comments* per bin is different for all the three review bins. The number of records in each review bin when the Host is Super Host is significantly different when the Host is not a Super Host. This means that these features are important when a model is being built. Similar kind of analysis is done for all the features and the model is eventually built on a total **60 Features** (Detailed column list is mentioned in Appendix [3]).

## 3. MACHINE LEARNING MODELS

Since we have converted the given use case into a classification problem, the main obvious choices to test and implement first is of Logistic Regression and kNN Classifier. I have implemented both as part of this assignment and compared their results to obtain the best fit for this use case. The combined dataset of *Listings and Reviews* are split in the **75-25 ratio of 2456 Data Rows** wherein the models are trained on 75% of the dataset and tested on the remaining 25% data.

### 3.1 LOGISTIC REGRESSION

*Logistic Regression* is a *supervised* machine learning algorithm that uses labelled data to train the model. During training, the model assigns *weights* also known as the model *coefficients* to each and every feature present in the training dataset. The model constantly tries to reduce the error by using a *cost function*. In order to avoid *overfitting*, a *penalty (L1, L2 or None)* is subjected to the model as a *hyper-parameter*.

For this particular use case, I have subjected the model to a range of *C* values between 0.001 and 1000. The *C* values subjected to the model are *[0.001, 0.1, 1, 10, 100, 1000]* which exposes the model to a range of regularisation wherein smaller *C* values lead to stronger regularisation. Additionally while training the model, I have randomized the input parameters in order to avoid bias. Since we are dealing with a multiclass problem wherein the trained model has to predict whether a given Airbnb listing lies in the review bin 0, 1 or 2, *'newton-cg'* solver is recommended by scikit-learn as it is equipped to handle multinomial loss. The *'newton-cg'* solver supports *L2* penalty and calculates the *Hessian* which is a square matrix of second order partial derivative rather than the normal *Stochastic Gradient Descent.* This solver proves to be a better choice and hence is finalised for this particular use case. Lastly, since this is a multiclass classification problem, I have set the parameter *'multi_class'* to '*multinomial'*.

In order to determine the optimal value of *C* for each target variable, I have plotted the *Cross-Validation* graph with *CV = 5.* Please see *Figure 8* depicting the accuracy scores for a range of *C* values for the target *review_scores_accuracy.*

From the plot we can see that on increasing the value of *C*, that is decreasing the penalty subjected to the model, we can see that accuracy of the model goes on increasing. When *C = 100*, the accuracy of the model does not increase significantly even when the value of *C* is increased drastically. The ***Training Accuracy*** of the model is **~61%** and the ***Testing Accuracy*** of the model is **~58%** when *C = 100*. Also, the *Standard Error* of the model is lowest when *C = 100*. This suggests that the optimal value of *C* for this particular use case is 100. Similarly the optimal values of *C* for other target variables are calculated. The same is recorded in *Table 3*.


**Figure 8:** Logistic Regression Cross Validation Plot for the target variable *review_scores_accuracy* (CV = 5)

| Target Variable | Optimal C Value | Train Accuracy | Test Accuracy |
|---|---|---|---|
| *review_scores_accuracy* | 100 | 61% | 58% |
| *review_scores_checkin* | 100 | 71% | 66% |
| *review_scores_cleanliness* | 100 | 58% | 53% |
| *review_scores_communication* | 100 | 69% | 68% |
| *review_scores_location* | 100 | 57% | 49% |
| *review_scores_rating* | 10 | 62% | 57% |
| *review_scores_value* | 100 | 58% | 53% |

**Table 3:** Optimal Value of Hyper-Parameter *C* with the corresponding *Train* and *Test* Accuracy for all the 7 Targets

The cross validation plots for all the remaining 6 Target Variables are provided in Appendix [4]. For most of the Target Variables, we can see that the optimal value of *C* is 100 or less. If we go on increasing the value of *C,* the penalty subjected to the model goes on decreasing and hence exposing the model to the risk of *Overfitting*. This is another evidence backing the claim of the optimal *C* values presented in *Table 3*.

Additionally, I have also augmented the transformed the training dataset into a *Polynomial Feature Space* of *Degree 2* and tried predicting the review bins a given listing might belong to. This process did not yield good results as there was a huge gap of **~10%** between the *Training and Testing* accuracy scores. Additionally the process of augmenting the features to a *Polynomial Feature Space* was *computationally expensive* and hence **isn't the right choice** for such a dataset. The *Cross Validation Plot* and the optimal value of *C* for all the target variables to be predicted by the model are presented in Appendix [5] and [6] respectively.

### 3.2 k-NN CLASSIFIER

*k-NN or k-Nearest Neighbours* is a *Supervised* machine learning algorithm which groups the data by their target variables or classes. The model uses its *Hyperparameter (k neighbours)* and the distance between the nearest class to classify the new data point exposed to the model. While predicting, the model calculates the distance (usually Cosine or Euclidian) between the data points and identifies the *'k'* Nearest Neighbours to assign a class to it.

In this particular model, I have only changed the value of *k* which is the number of neighbours and have kept the *weights uniform*. I have *only selected odd numbers* as selecting even numbers might result in situation of ties thus resulting in many mis-classified values. In order to verify the performance of the model, I have implemented *K-Fold Cross Validation* of 5 folds. The number of neighbours on which this model was built on are as follows – **N Neighbours Range:** [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

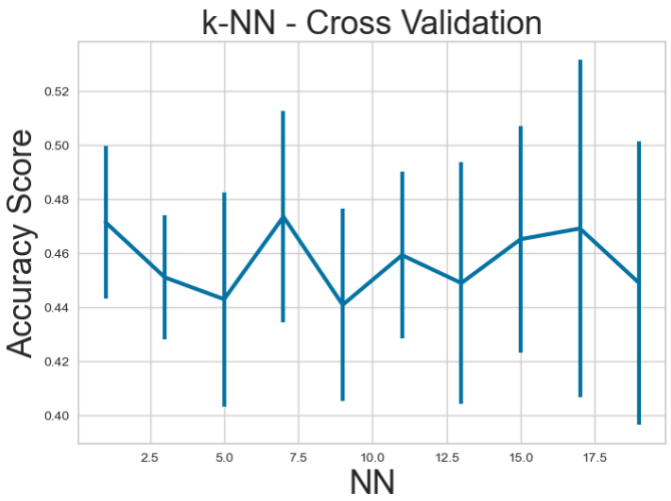



**Figure 9:** k-NN Cross Validation for the target variable *review_scores_accuracy* (CV = 5)

In *Figure 9* above, the *Cross Validation* plot for the target variable *review_scores_accuracy* is plotted for a range of *k* values. Based on the *Cross Validation* plot, we can see that the accuracy score of the model is highest when the number of **nearest neighbours is 7**. Although, the accuracy score of model when *k=1* is similar, we won't be using that as the resultant model is complex and difficult to comprehend. Also as we go on increasing the value of *k*, the model becomes simpler, supressing the noise in the dataset. Hence the optimal value of k is 7 for the target variable *review_scores_accuracy* where we get the *Training Accuracy Score* of **~63%** and *Testing Accuracy Score* of **~45%**. In a similar way, the optimal value of *k* and the *Training and Testing Accuracy Scores* are presented in *Table 4*. The cross validation plots for all the remaining 6 target variables are provided in Appendix [7]

| Target Variable | Optimal *k* Value | Train Accuracy | Test Accuracy |
|---|---|---|---|
| *review_scores_accuracy* | 7 | 63% | 45% |
| *review_scores_checkin* | 7 | 72% | 62% |
| *review_scores_cleanliness* | 9 | 57% | 42% |
| *review_scores_communication* | 7 | 72% | 63% |
| *review_scores_location* | 3 | 66% | 43% |
| *review_scores_rating* | 11 | 59% | 48% |
| *review_scores_value* | 13 | 54% | 44% |

**Table 4:** Optimal Value of Hyper-Parameter *k* with the corresponding *Train* and *Test* Accuracy for all the 7 Targets

### 3.3 RESULT EVALUATION AND MODEL COMPARISON

For this section to compare the results from both the model, I have selected to compare the scores of the target variable *review_scores_accuracy*. As a metric to assess the models, I have used ROC-AUC curves. A ROC curve is a plot of *True Positive Rate vs False Positive Rate*, while AUC is the area under the ROC curve which for every class gives out a single value rather than a curve. For an ideal classifier, ROC curve gives a point in the top left corner stating that the classifier has 100% True Positives and 0% False Positives; Similarly for ideal classifier, AUC = 1 and a random classifier has AUC = 0.5. Please find the ROC-AUC curve of both *Logistic Regression and k-NN* models in *Figure 10.1 and Figure 10.2* respectively.

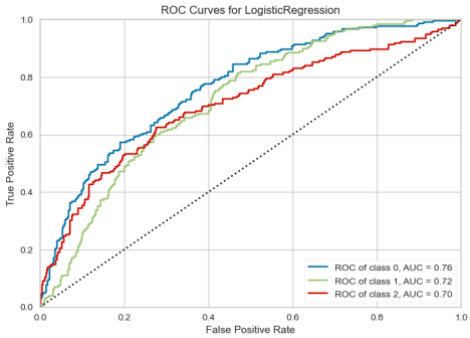



**Figure 10.1:** ROC-AUC for Logistic Regression

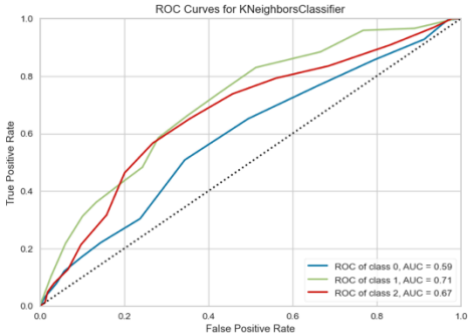



**Figure 10.2:** ROC-AUC for k-NN Classifier

From the values presented in the ROC-AUC curves above, we can see that the *Logistic Regression* model is a better choice for this particular target variable when compared to a *k-NN* classifier. The AUC for all the three classes for the target variable *review_scores_accuracy* is more for *Logistic Regression* when compared to a *k-NN* classifier. Hence if provided with a choice, I would select *Logistic Regression* as the appropriate model for this particular problem statement. The ROC-AUC curves for all the other target variables are presented in Appendix [8].

Also on comparing both the models with a **baseline classifier**, we can see that both the selected models outperform a Dummy Classifier that always the **most frequent** value which has an accuracy of **~32%**; and a Dummy Classifier that predicts the value of the rating bin **uniformly** which has an accuracy of **~33%**. This proves that our model selection is accurate and the performance of the model for this use case is optimal.

## QUESTION 2 (i)

*Logistic Regression* is the go to choice of learning algorithm for any classification problem that has a linear correlation between its input features. *Logistic Regression* works well when the different classes within the dataset is *linearly separable*. One of the major reasons where *Logistic Regression* would give inaccurate predictions is when it is subjected to a problem where in the classes in the dataset which the model has to predict is ***not linearly separable*.** Please take into consideration the following example –

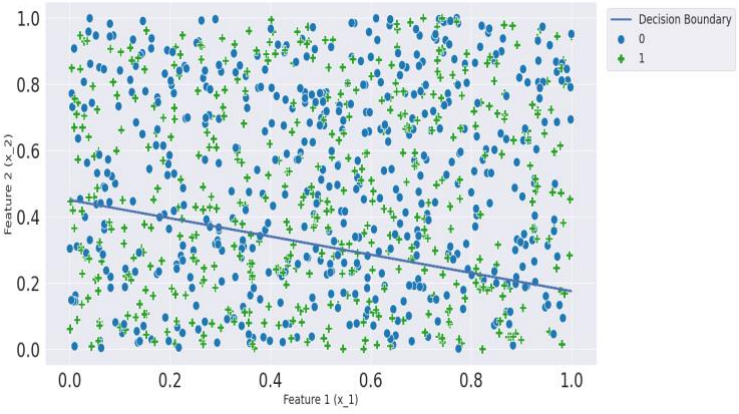



**Figure 11:** Linearly non-separable data

The data points shown in *Figure 11* is an arbitrary example depicting a scenario when the dataset in not linearly separable. In such a scenario, the *Logistic Regression* tries to fit a *Decision Boundary* which is a straight line in order to separate both the classes. In such a scenario, the trained model would misclassify a lot of values and give inaccurate predictions.

Another probable situation wherein *Logistic Regression* models fail to provide accurate prediction is when there is a huge range of features in which the model is to be trained and has less number of data rows to train. Having a huge feature means that the data is *sparse* and the model has to deal with a lot of *dimensions* and fit it accordingly. If both of those conditions are satisfied, that having more features and less data point to train on, the model starts giving importance to features which aren't that important and eventually *learns the data*.

In such a scenario, when a completely new data point is subjected to the model, there is a high chance of mis-classification as the model is *overfitted* and fails to classify the new data point accurately.

## QUESTION 2 (ii)

**MLP Neural Network –** A MLP (Multi-Layer Perceptron) Neural Network is a three layered network. The first one is an *input layer*, followed by a *hidden layer* and finally the *output layer*. The *hidden layer* can have multiple nodes and similarly there can be multiple nodes in the *output layer*. It can very well be used in a multi-class classification where in the model can multiple classes in its *output layer*.

In a traditional neural network, weights are applied to each neuron in order to transform the data from one layer to another layer. Then an *activation function* is applied to the *neuron* which decides whether the given *neuron* should be activated or not.

One of the **major advantages** of using MLP Neural Net is that it can be applied to complex input feature space and *non-linear* problems. Also this technique works very well even in cases where the number of input features are huge. In case of classification problems, the **other main advantage** of using a MLP is its ability and provisions to *define the weights* applied and the *number of neurons that can be added* to the hidden layers in order to fine-tune the model. This gives us better flexibility to adjust the decision regions which is key for some classification tasks.

The **main disadvantage** of using an MLP Neural Net is that is can get too complex too easily. Also as mentioned above, in order to activate a *Neuron*, a given neural network must have an *activation function* associated with it. One such activation function is *Rectified Linear Unit (ReLU)*. *ReLU* is a *non-linear activation function* which pushes any negative value to zero. The formula for *ReLU* is as follows –

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

The main drawback of *ReLU* is that it cannot learn from any example for which their activation is zero or less which leads to a situation of *"Dead Neuron"* where the output is always 0.

**K-Nearest Neighbours –** k-NN is a supervised learning algorithm which groups data given their target values using some distance metric. The *hyperparameters* that can be configured in k-NN are its number of nearest neighbours *k* and the type of distance metrics like *Euclidean, Cosine, etc*.

One of the **major advantage** of using k-NN when compared to MLP is that it is easy to use as it has very less hyperparameters to configure and it is very easy to comprehend as it classifies a new data point based on its distance to the nearest other classes. It is a good choice for datasets which are small as the model would have to calculate the distance of the test point over a small sample space and find the *k* nearest neighbours to classify the test point.

The same feature of k-NN is its **main disadvantage** when compared to MLP. Implementing k-NN becomes computationally expensive when the dataset is huge and on the other hand, MLP works satisfactorily in such cases. Additionally in k-NN, we can only change the *hyperparameter k* and the *distance metric* leaving it with less opportunity to fine tune the model when compared to a MLP Classifier.

## QUESTION 2 (iii)

In *k-fold Cross Validation*, the dataset is divided into *k* equal sized parts, then the model is tested iteratively on one of those parts and trained on the remaining *(k-1)* parts. By using this technique, the model would eventually be tested on each of the *k* blocks or parts of data in order to identify the *cost* J($\theta$).

In case of *Hold-Out* method, the dataset is split randomly into *x:y* ratio where *x%* of the data is used for training while *y%* is used for testing. Usually the data is split into 75:25 or 80:20 ratio. In this case, the model is never really trained on the *y%* of the dataset. Hence we are losing out on any important trend that may have been present in those *y%* of data points.

In *k-fold Cross Validation,* all the parts of data would be used in training at some point of time thus helping us to uncover all the trends present in the dataset thus *making the model more robust and generalised*.

Identifying the ideal value of *k* is very important to understand the performance of the model. If we have a very small value of *k,* like *k = 2*, we are introducing a lot of *bias* into the model. This is because the model would only be trained on the 50% of the data and would be tested on the remaining 50% since the dataset is divided into 2 equal parts. As we go on increasing the value of *k,* we are exposing the model to more training data and hence the performance of the model would in variably increase. We want the model to learn as much data as possible so that it can get the parameter values and weights right which would lead to less fluctuations within the model. This is due to the fact that the training datasets are highly correlated in higher values of *k* and the predictions are highly dependent on the values the model is trained at thus resulting in an *over-fitted* model. On the other hand, for very low values of *k,* although the bias is high, the variance is lower due to the fact that the training features are less correlated which might lead to a situation of *under-fitting.*

Thus it is important to have a *bias-variance* trade-off between the train and test splits. The most common choices of *k are 5 and 10*. When *k = 5*, the model is iteratively trained on 80% of the dataset and tested on the remaining 20%. Similarly when *k = 10*, the train-test ratio is 90:10. These values of *k* are often selected as good fit for any model due to the following 2 reasons – The bias-variance trade-off is optimal and also since the *k* value is not too large, the computation time is adequate. Also the model gets ample amount of data to learn representative parameter values and generalise the data.

## QUESTION 2 (iv)

In *Multi-Step* prediction, the prediction at step *k+q* depends on the predictions at step *k, k+1, k+2, …, k+q-1.* This means that the prediction at step *k+q* is serially dependent on events occurred in recent past. In other words, we *feedback* the model outputs from previous states to create a particular prediction in the current state.

In such a situation where a particular prediction is dependent on events occurred in recent past, *lagged* output values can be useful for creating *time-series* data. *Lagging* usually means to shift the current value or state of the input by one or more steps in the next input value in order to predict its target. Let us consider an example where we are trying to predict the temperature (in ºC).

| forecast_date | temperature_degree | temperature_lag1 | temperature_lag2 |
|---|---|---|---|
| 2023-01-01 | 13 | NaN | NaN |
| 2023-01-02 | 7 | 13.0 | NaN |
| 2023-01-03 | 8 | 7.0 | 13.0 |
| 2023-01-04 | 11 | 8.0 | 7.0 |
| 2023-01-05 | 7 | 11.0 | 8.0 |

**Table 5:** Sample Lagged Output Values

In time series data, by lagging we make the past values to appear in the same row that we are trying to predict. In this case highlighted above, we can see that for 01-Jan, we have no previous value of temperature recorded. But for 02-Jan, we are using the temperature predicted on 01-Jan, that is 13, to predict the target *temperature_degree*. In a similar way, the temperature forecast of future days would be nothing but some function of the temperature recorded in prior 2 days.

## APPENDIX

[1] List of Columns from *Listings* Dataset which are Scaled -

```
'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',
'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'
```

[2] List of Columns from *Listings* Dataset which are Dropped –

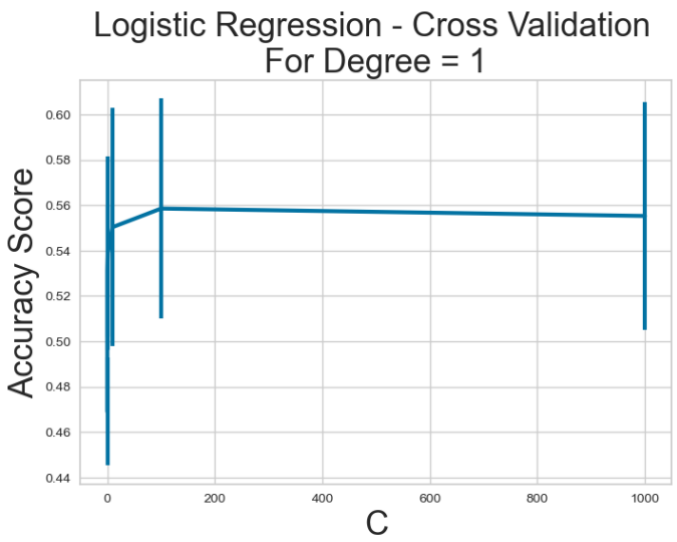
```
'id','listing_url',            'scrape_id',
'last_scraped',      'source',      'name',
'picture_url',    'host_id',    'host_url',
'host_name', 'host_location', 'host_about',
'host_thumbnail_url',   'host_picture_url',
'host_neighbourhood',      'neighbourhood',
'neighbourhood_group_cleansed',
'calendar_updated',         'first_review',
'last_review',                  'license',
'calculated_host_listings_count',
'calculated_host_listings_count_entire_hom
es',
'calculated_host_listings_count_private_ro
oms',
'calculated_host_listings_count_shared_roo
ms',                         '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'
```

[3] Feature List used to train the Machine Learning Models –

```
'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',     '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', 'num_cmt', 'term_also',
'term_apartment',             'term_city',
'term_clean',                'term_close',
'term_comfortable',     'term_definitely',
'term_dublin',                'term_easy',
'term_everything',            'term_good',
'term_great', 'term_helpful', 'term_home',
'term_host',   'term_house',   'term_it',
'term_location',           'term_lovely',
'term_nice', 'term_perfect', 'term_place',
'term_really',           'term_recommend',
'term_room',   'term_stay',   'term_the',
'term_time',    'term_us',    'term_walk',
'term_we', 'term_well', 'term_would'
```

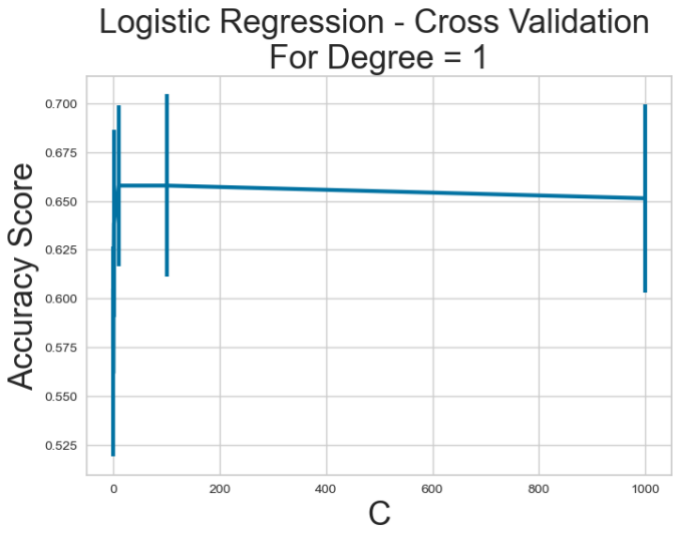
[4] *Logistic Regression* Cross Validation plots for the features

`review_scores_rating`,
`review_scores_cleanliness`,
`review_scores_checkin`,
`review_scores_communication`,
`review_scores_location`,
`review_scores_value` – For Degree 1

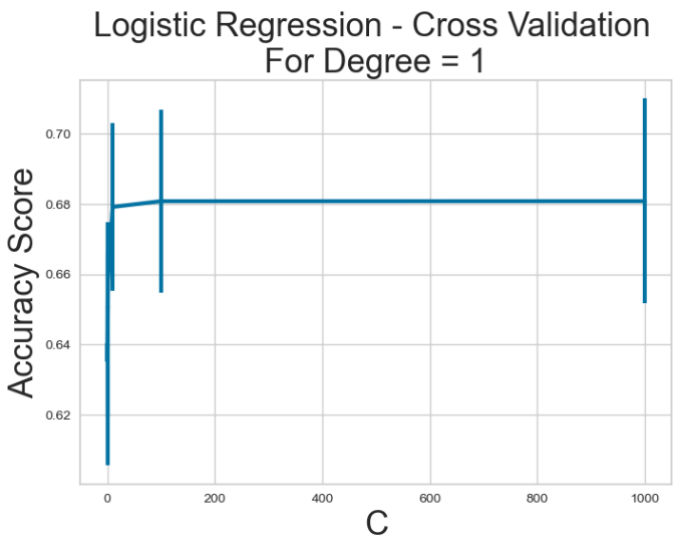



**4.1:** Cross Validation Plot of review_scores_rating

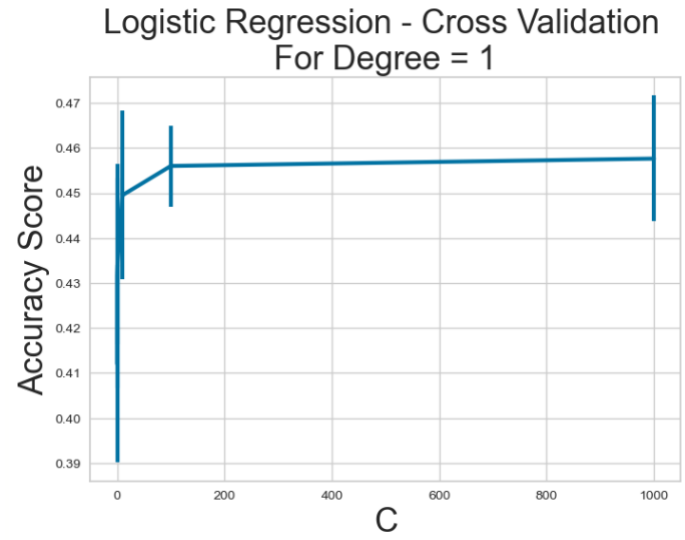

**4.2:** Cross Validation Plot of review_scores_cleanliness



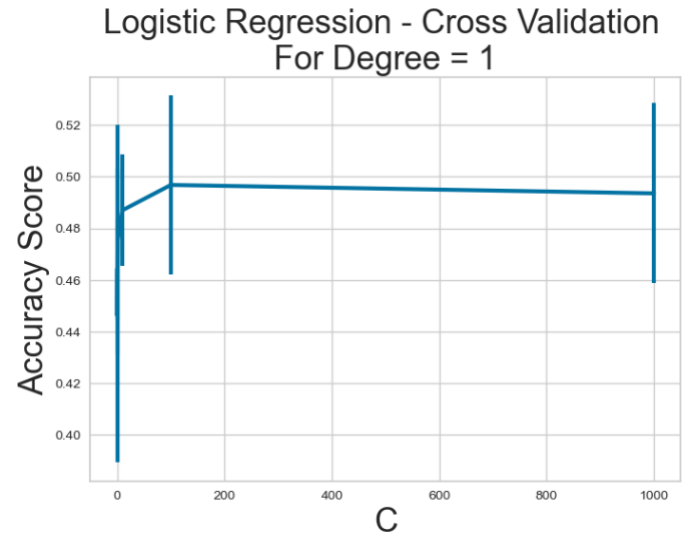

**4.3:** Cross Validation Plot of review_scores_checkin




**4.4:** Cross Validation Plot of review_scores_communication



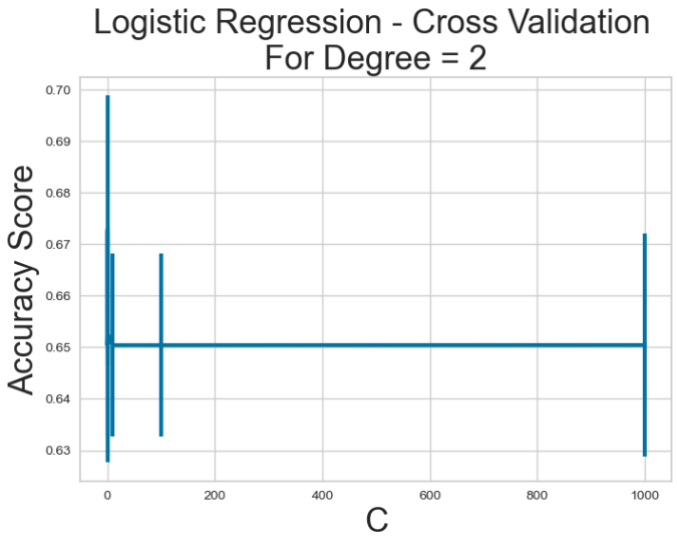

**4.5:** Cross Validation Plot of review_scores_location



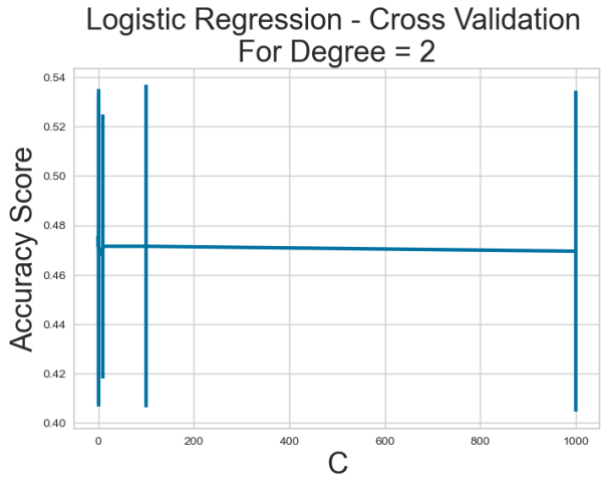

**4.6:** Cross Validation Plot of review_scores_value

[5] *Logistic Regression* Cross Validation plots for all 7 features (Degree = 2) –

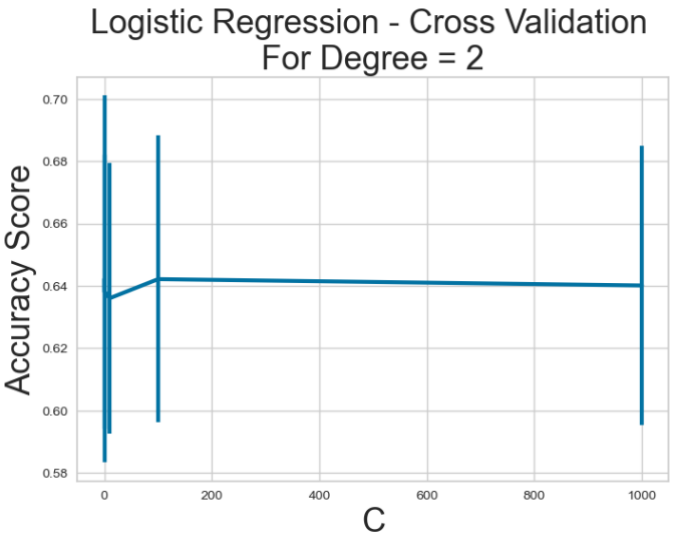

**5.1:** Cross Validation Plot - review_scores_accuracy (Degree 2)




**5.2:** Cross Validation Plot - review_scores_checkin (Degree 2)



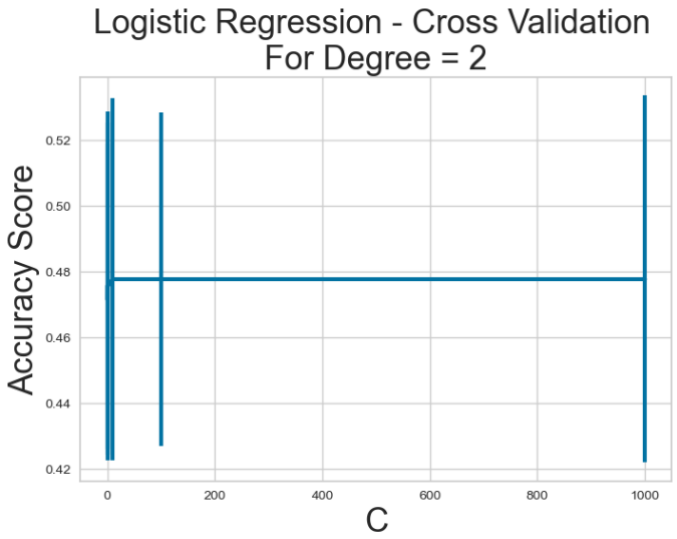

**5.3:** Cross Validation Plot - review_scores_cleanliness (Degree 2)




**5.4:** Cross Validation Plot - review_scores_communication (Degree 2)


**5.5:** Cross Validation Plot - review_scores_location (Degree 2)


**5.6:** Cross Validation Plot - review_scores_rating (Degree 2)

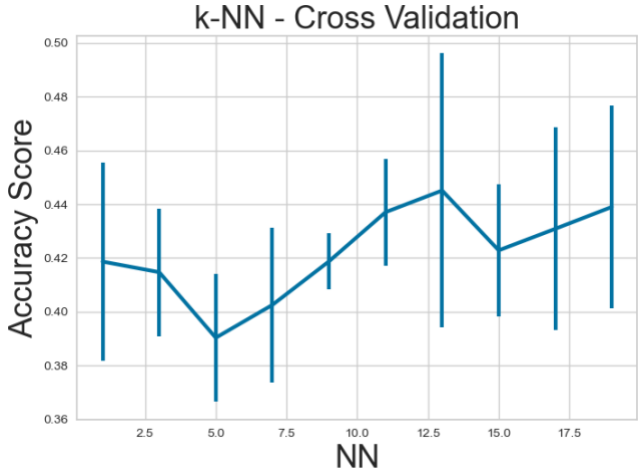



**5.7:** Cross Validation Plot - review_scores_value (Degree 2)

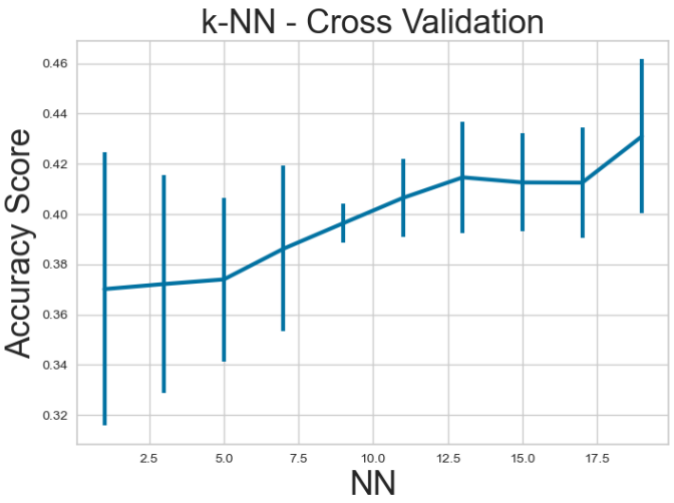
[6] Optimal Value of *C* for all the Target Variables in the Polynomial Feature Space –

| Degree | Target Variable | Optimal *C* Value |
|---|---|---|
| 2 | *review_scores_accuracy* | 0.1 |
| | *review_scores_checkin* | 100 |
| | *review_scores_cleanliness* | 10 |
| | *review_scores_communication* | 100 |
| | *review_scores_location* | 100 |
| | *review_scores_rating* | 10 |
| | *review_scores_value* | 100 |

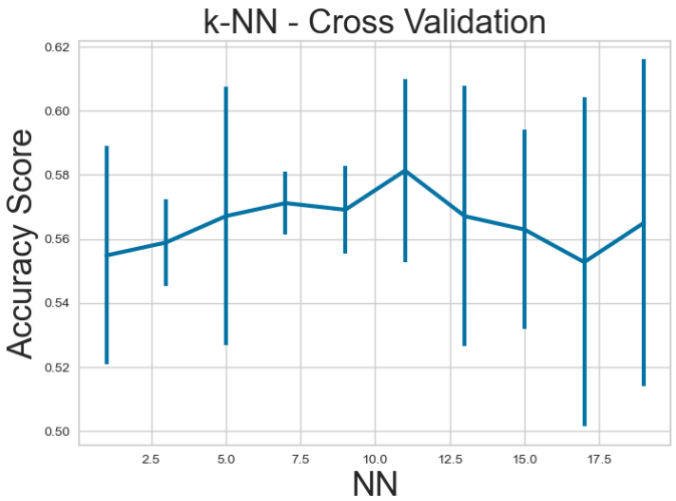
[7] k-NN Cross Validation Plots for the features `review_scores_rating`, `review_scores_cleanliness`, `review_scores_checkin`, `review_scores_communication`, `review_scores_location`, `review_scores_value`



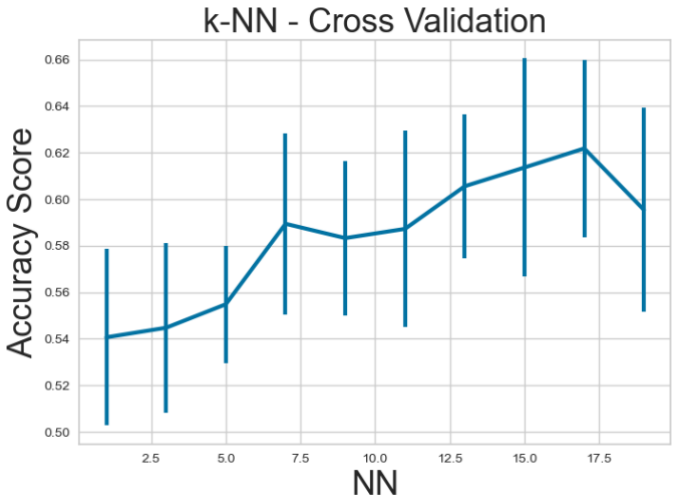

**7.1:** Cross Validation Plot of review_scores_rating



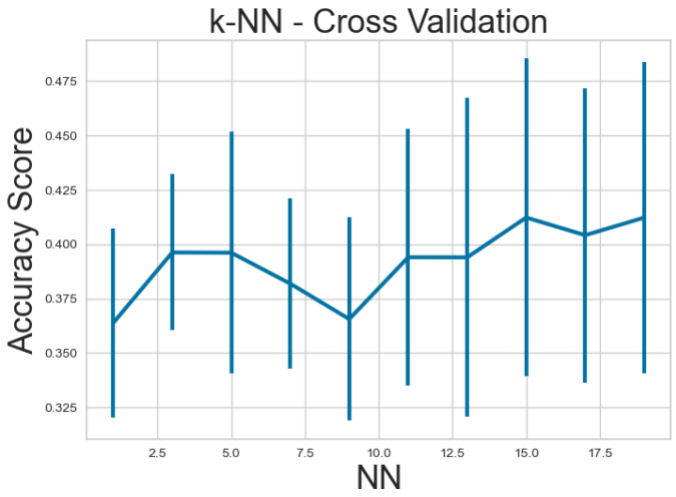

**7.2:** Cross Validation Plot of review_scores_cleanliness



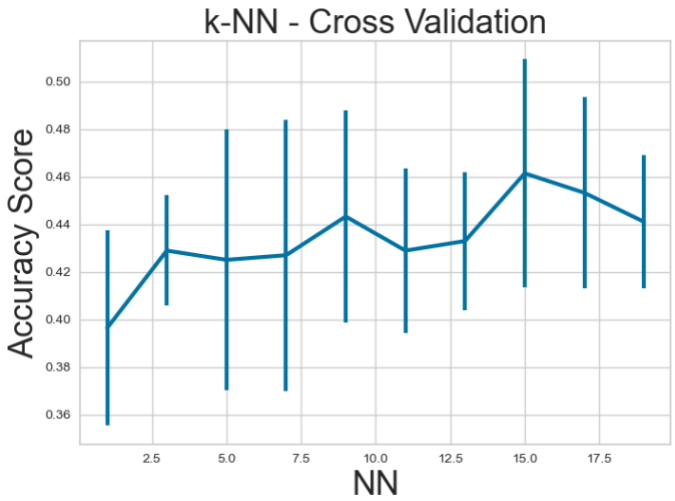

**7.3:** Cross Validation Plot of review_scores_checkin




**7.4:** Cross Validation Plot of review_scores_communication

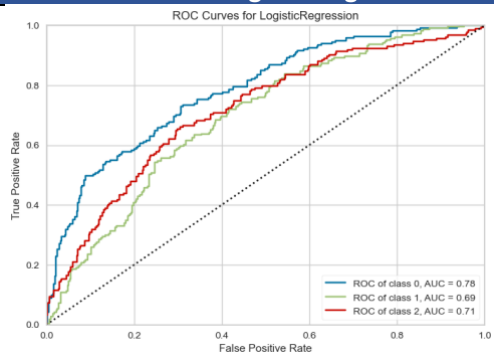
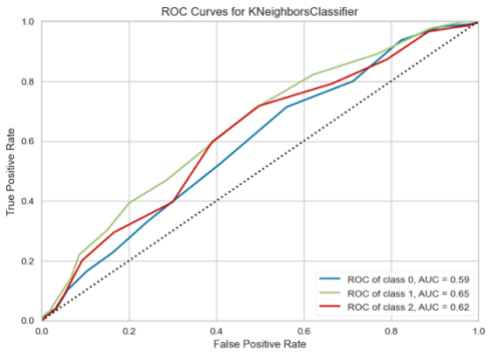
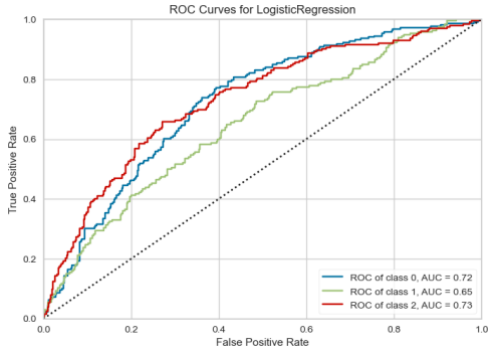
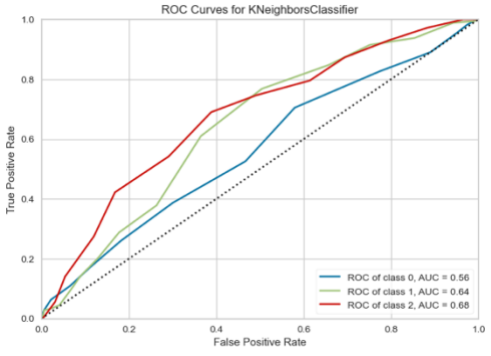
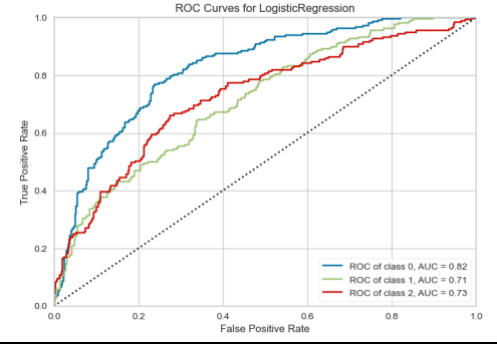
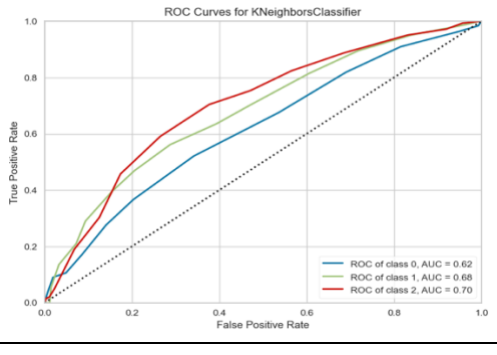


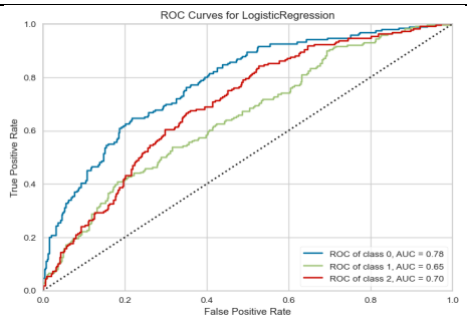
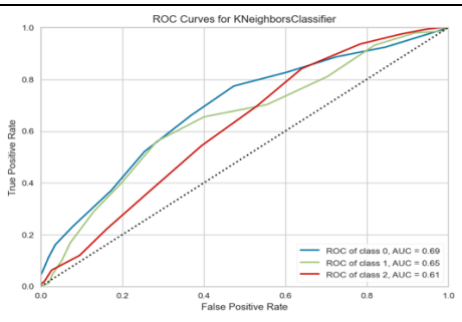

**7.5:** Cross Validation Plot of review_scores_location



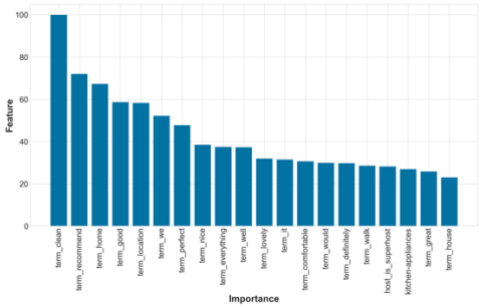

**7.6:** Cross Validation Plot of review_scores_value

[8] ROC-AUC Curves For all the remaining Target Variables –

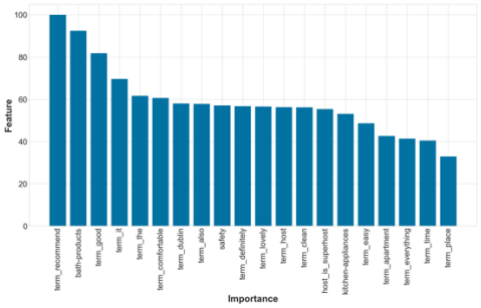
| Target Variable | ROC-AUC For Logistic Regression | ROC-AUC For k-NN Classifier |
|---|---|---|
| review_scores_cleanliness |   |   |
| review_scores_location |   |   |
| review_scores_rating |   |   |

| | | |
|---|---|---|
| review_scores_value |  |  |

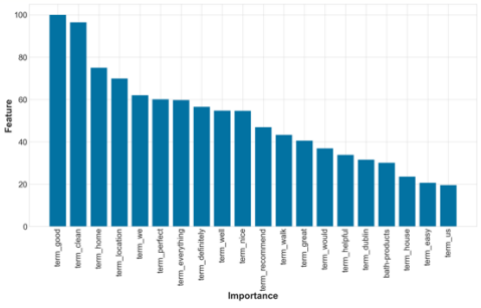
[9] Top 10 Features for each review bin in remaining features



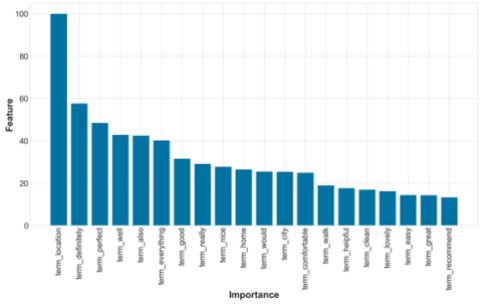

review_scores_cleanliness – Bin 0



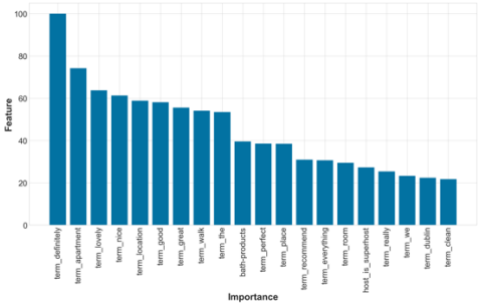

review_scores_cleanliness – Bin 1



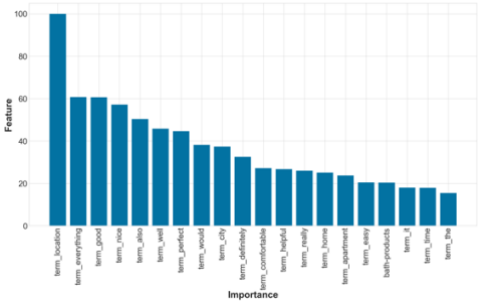

review_scores_cleanliness – Bin 2



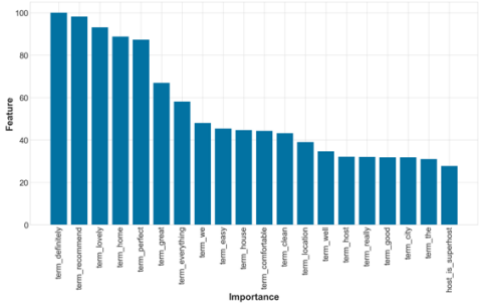

review_scores_location – Bin 0



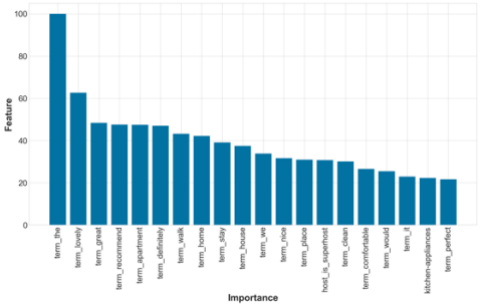

review_scores_location – Bin 1



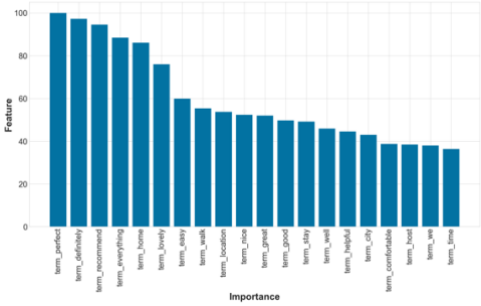

review_scores_location – Bin 2



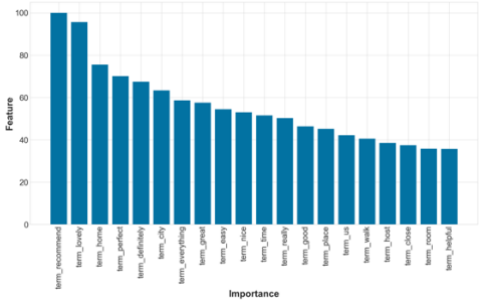

review_scores_rating – Bin 0



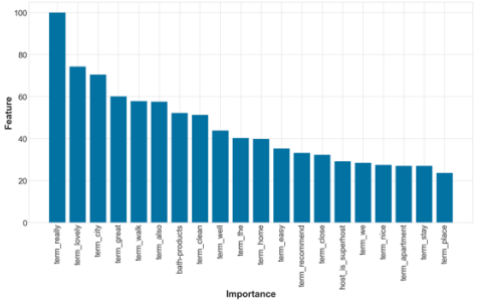

review_scores_rating – Bin 1



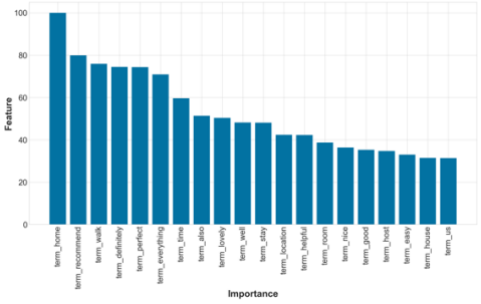

review_scores_rating – Bin 2




review_scores_value – Bin 0




review_scores_value – Bin 1




review_scores_value – Bin 2

## PYTHON SCRIPTS

The Complete Code for the Airbnb Ratings Prediction is present in GitHub at the following link - https://github.com/abhikbhattacharjee/AirBnB_Rating_Prediction

Following are the description of the respective .ipynb files –

- ML_Final_Assignment_2022_23_22305544.ipynb – Contains the *Listings* Data Pre-Processing code, independent Feature Plots, Scripts for Logistic Regression and k-NN Models
- Reviews_Preprocessing.ipynb – Script for *Reviews* Data Pre-Processing

### Python Script for Question 2.1 –

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
import seaborn as sns
import pandas as pd

x = np.random.rand(1000,2)
df = pd.DataFrame(np.random.randint(0,2,size=(1000, 1)))
y = df.iloc[:,0].values

log_reg = LogisticRegression()
log_reg.fit(x, y)

sns.set(rc={'figure.figsize':(20, 10)})
sctr_plt = sns.scatterplot(x=x[:,0], y=x[:,1], style=y, hue=y,
palette=['tab:blue', 'tab:green'],
                           markers=('o', 'P'), s=200)
sctr_plt.set(xlabel='Feature 1 (x_1)', ylabel='Feature 2 (x_2)')
sctr_plt.xaxis.get_label().set_fontsize(20)
sctr_plt.yaxis.get_label().set_fontsize(20)
y_ab1 = -(log_reg.intercept_ +
(log_reg.coef_[:,0]*x[:,0]))/log_reg.coef_[:,1]
plt.plot(x[:,0],y_ab1, label='Decision Boundary', linewidth=3)
plt.legend(fontsize=20, markerscale = 2,bbox_to_anchor=(1.01, 1))
plt.yticks(fontsize=30)
plt.xticks(fontsize=30)
plt.show()
```

### Python Script for Question 2.4 –

```
import numpy as np
import pandas as pd

begin_date = '2023-01-01'

weather_forecast =
pd.DataFrame({'forecast_date':pd.date_range(begin_date, periods=10)})
```

```
weather_forecast['temperature_degree'] =
np.random.randint(5,20,size=len(weather_forecast))

weather_forecast['temperature_lag1'] =
weather_forecast.temperature_degree.shift(1)
weather_forecast['temperature_lag2'] =
weather_forecast.temperature_degree.shift(2)
```