CS7CS4 – Machine Learning Supplemental Assignment 2023-24

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

I. DATASET

Airbnb is an online marketplace that provides individuals with properties specializing in homestay, rental, and travel experiences. Based on the attributes of a given listing, the review rating scores for various domains such as cleanliness, communication, and value may differ. The main objective of this assignment is to determine the potential rating of any given Airbnb listing in Dublin based on its features and facilities. The raw dataset for this study was obtained from the Inside Airbnb website, and the model was built using the Listings and Reviews Data available on the site. The Listings data consists of 75 feature columns with a total of 7345 listings. The Reviews data consists of 6 feature columns and a total of 230,065 reviews.

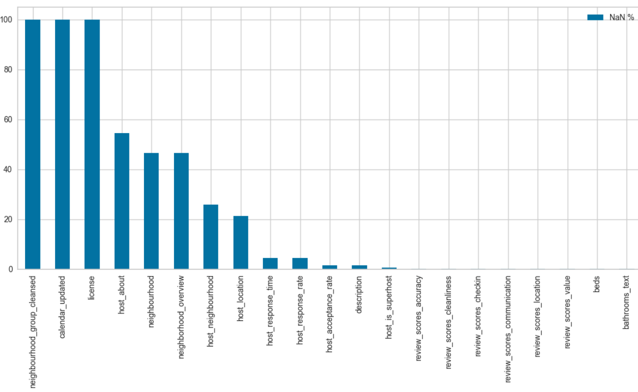


Fig. 1. %NaN values present in each feature columns from Listings dataset

II. FEATURE ENGINEERING

The listings data consists of 75 feature columns, including 10 columns containing data scraping information and 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 enhance comprehensibility and readability. Similarly, for the reviews dataset, out of the 6 features present in the raw data, only the **listing ID** and **comments** features were considered for further analysis. The detailed preprocessing steps for the respective datasets are described in subsequent sections.

A. *LISTINGS DATA PRE-PROCESSING*

The listings dataset included key amenities for each property mentioned under the amenities column. It was crucial for people to consider certain amenities when selecting accommodations for their vacation. Therefore, it was essential to split the consolidated amenities in one column into separate categories for each listing. In total, there are 77 distinct amenities possible for any given listing, which I have grouped into 9 different categories. The grouping of amenities into broader domains is based on logical clustering; for example, amenities such as ‘Hot shower’, ‘Shower gel’, ‘Hair dryer’, ‘Shampoo’, ‘Conditioner’, and ‘Body Soap’ are grouped under the ‘Bath Products’ category, while amenities such as ‘Oven’, ‘Hot water kettle’, ‘Cooking basics’, and ‘Microwave’ are grouped under the ‘Kitchen Appliances’ category. It is possible that a given listing has none of the amenities from the 9 broad categories or lacks any contact information in the host verification columns. In such cases, I populated all NaN values in the newly created feature columns with 0. Moving forward, it was important to understand the number of NaN values in each feature column. Figure 1 shows the percentage of NaN values present in each column of the Listings dataset.

Based on the values presented in the above graph, we can see that the features ’neighbourhood group cleansed,’ ’license,’ ’bathrooms,’ and ’calendar updated’ are composed of NaN values. Therefore, these four columns were completely dropped from the dataset for further analysis. Additionally, examining the seven Review Scores columns, which are our target variables, reveals that, on average, 20% of those rows contain NaN values. Since these are target values for our model, imputing those values is not the right choice as it may introduce bias. Therefore, for all other data records with NaN values, the entire row is dropped from the dataset for further analysis, as these would not add value to the model we are building for this particular use case. Closely examining the ’price’ feature, we find that prices are listed in dollars ($). To make the values machine-readable for further analysis, I replaced ’$’ and ’,’ with null and converted the string values to integers. This preprocessing step is crucial for ensuring data consistency and facilitating analysis. The ’price’ feature exhibits numerous outliers. The average price of a given listing is approximately $167, with the median being $110. In contrast, the maximum price of any listing is $99,549.

From the values presented in Figure 2, it is evident that the majority of the values lie within the range of $50 and 300$. Therefore, I dropped all rows with price values outside the mentioned range. For all seven target variables that our model needs to predict, the rating values are very close to each other

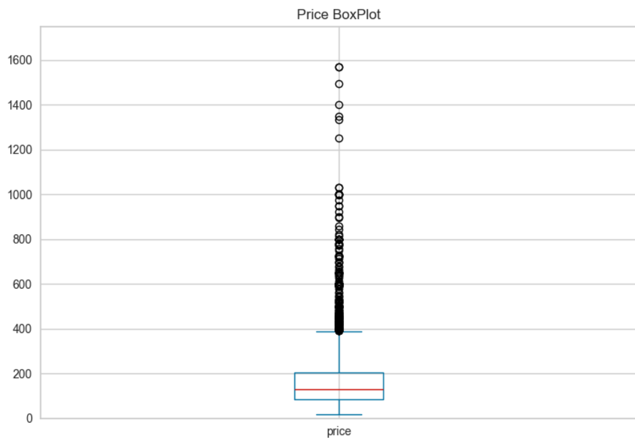


Fig. 2. Plot of Price Feature

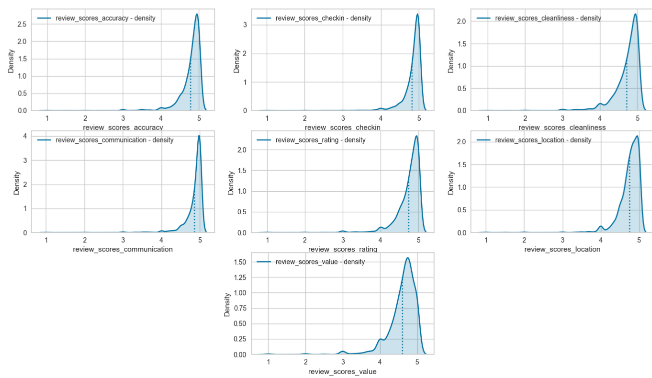


Fig. 3. Distribution Plot of all the Target Columns (Review Rating Columns)

and tend to be high. As shown in Figure 3, all the rating columns are heavily concentrated in the range between 4.7 and 5. To address this, I used the Binning Approach to handle such close numbers.

In the Binning Approach, we categorize continuous data into discrete bins based on specific criteria, facilitating further analysis. For this use case, I decided to bin the 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 based on its quantile range, we avoid bias in which the number of records in one bin is greater than others, making it a suitable choice for this problem. After the values are binned, we predict the rating bin a given listing may fall into rather than predicting the exact numeric rating, making this a classification problem where the model predicts a categorical variable given some input parameters. To categorize a given rating into a discrete bin, I used the Pandas DataFrame QCut functionality, which divides the underlying data into equal-sized bins. The function defines the bins using percentiles based on the data distribution, not the actual numeric edges of the bins. Note that the ratings are first converted to percentages before being categorized into bins. For the features 'host response rate' and 'host acceptance rate,' I replaced the '%' signs in the feature columns with null and converted the resultant figures into numeric for further processing. 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 to be machine-readable for the learning model chosen for this problem. For scaling, I used min-max scaling as it retains the shape of 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 consists of features such as listing ID, ID, date, reviewer ID, reviewer name, and comments. In total, there are 230,065 reviews, with one listing potentially having multiple reviews. For our analysis, I only considered the features listing ID and comments.

For the comments feature, there are non-English comments, and some comments contain only special characters, emojis, or HTML tags like br. To make these features usable, we need to preprocess by removing emojis and special characters, considering only English reviews.

First, we drop all review rows that do not have any comment associated with them. Next, we identify only the English reviews and discard reviews in other languages. To identify the language of the review text, I used the **fasttext** Python library and its pretrained model, which identifies the language of any given text. The **fasttext** library is an efficient language identification tool based on n-gram features, dimensionality reduction, and a fast approximation of the softmax classifier.

Next, we need to identify and remove all emojis from the text. To identify the number of emojis in a comment, I used the emoji Python library. I read every comment character by character and identified if the character matches any **EMOJI DATA** from the emoji library. If the length of the comment text is equal to the emoji count or double the emoji count, we drop that data point as the comment is composed entirely of emojis without significant text. I introduced the 2x condition for dropping data points because, for emojis with skin tone, **EMOJI DATA** splits the actual emoji and the skin tone, resulting in 2 counts for a single emoji.

Further deep-diving into the comments from the review dataset reveals that some comments are very small and may contain ASCII characters such as ':)' or 'OK'. For a comment to be significant and relevant for model training, I only selected comments longer than 20 characters. After preprocessing, we are left with 220,551 out of 230,065 reviews. This means we dropped 4% of the reviews, leaving us with only relevant English language reviews for model training. The total number of comments for a given listing is updated after preprocessing is completed.

1) TF-IDF METHODOLOGY: In order to utilize the actual comment text from the Review Dataset, preprocessing was necessary to make it machine-readable. Given that we are dealing with actual review text, the initial step involved implementing the Term Frequency – Inverse Document Frequency (TF-IDF) methodology. TF-IDF helps determine the importance of a word in a collection of text, 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 review text set was to identify important tokens from each review and use them as features for the machine learning model developed for this use case. After completing the preprocessing steps mentioned in Section 2.1 on the review dataset, all stopwords were initially removed from the review comments.

For implementing TF-IDF, I utilized ‘TfidfVectorizer‘ from ‘sklearn‘’s text feature extraction library ‘sklearn.feature_extraction.text‘. Since the objective was to identify the most relevant words from a review comment, I configured the ‘TfidfVectorizer‘ to analyze words and restrict the maximum number of features returned to 33.

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 4, 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. Opting to restrict the maximum features to the median aligns with the central tendency of the data distribution.

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

Now that all the records from the Listings and Reviews dataset are cleansed and processed, we can move forward and identify the important features for a given machine learning model. To create a training set, I merged the listings dataset

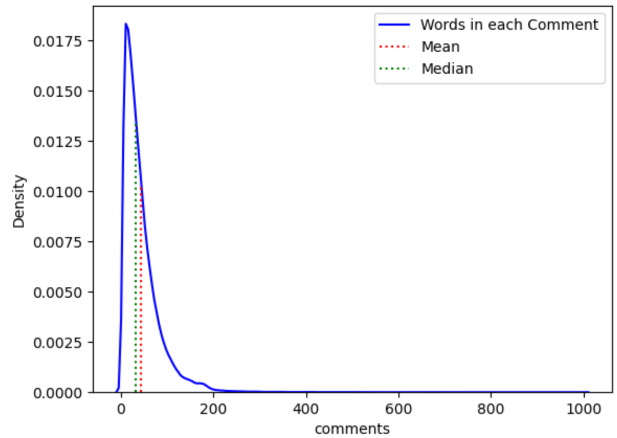


Fig. 4. Plot of Words in each comment

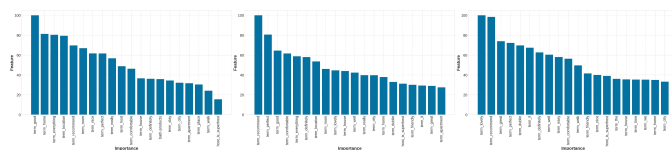


Fig. 5. Important Features for Different Bins - review_scores_accuracy

with the processed dataset created in Section 2.2.1 based on listing ID. To identify the most important features from the dataset, I initially fitted a Logistic Regression model and extracted the weights of the features from the model coefficients.

As shown in Figure 5, for a listing to achieve a very high rating (Bin 2), the most important terms or top 3 tokens in the review comments are ‘Home,’ ‘Perfect,’ and ‘Recommend.’ For a listing to achieve a rating in Bin 1, the review must include the terms ‘Would,’ ‘Lovely,’ or ‘Home’ and the listing must have Kitchen Appliances amenities. We can also see in Figure 6 that the intensity of the term ‘Home’ and the number of comments per bin differ for all three review bins. The number of records in each review bin is significantly different when the host is a Superhost compared to when the host is not a Superhost. This indicates that these features are important when building a model. Similar analysis was conducted for all the features, and the model was eventually built on a total of 60 features.

III. MODELS OF MACHINE LEARNING

Since we have converted the given use case into a classification problem, the main obvious choices to test and implement first are Logistic Regression and kNN Classifier. I 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 was split in a 75-25 ratio of 3741 data rows, where the models were trained on 75% of the dataset and tested on the remaining 25%.

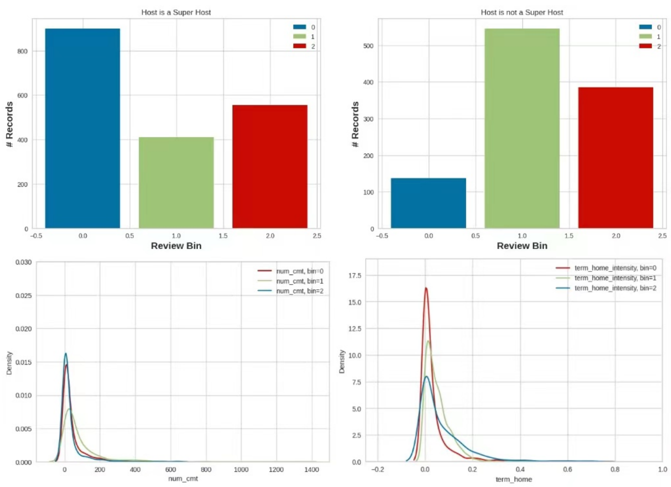


Fig. 6. Analysis Important Features

A. LOGISTIC REGRESSION AND PARAMETER TUNING

Logistic Regression is a supervised machine learning algorithm that utilizes labeled data to train the model. During training, the model assigns weights, also known as model coefficients, to each feature in the dataset. The model optimizes these weights to minimize error using a cost function. To prevent overfitting, a penalty (L1, L2, or None) is applied to the model as a hyperparameter.

For this specific use case, I explored a range of C values, which control the regularization strength, from 0.001 to 1000. The C values used were [0.001, 0.1, 1, 10, 100, 1000], exposing the model to varying levels of regularization, where smaller C values imply stronger regularization. Input parameters were randomized during training to avoid bias.

Given the multiclass nature of the problem, where the model predicts whether an Airbnb listing falls into review bins 0, 1, or 2, the 'newton-cg' solver was chosen. This solver is recommended by scikit-learn for multinomial loss problems, supports the L2 penalty, and computes the Hessian matrix, making it suitable for this use case. The 'multi_class' parameter was set to 'multinomial' to handle the multiclass classification.

To determine the optimal C value for each target variable, I plotted the cross-validation (CV = 5) graph. Figure 7 depicts the accuracy scores across a range of C values for the target variable review scores.

From the plot, it is observed that increasing the value of C (reducing the regularization penalty) tends to increase the accuracy of the model. However, after a certain point (C = 100 in this case), the accuracy improvement diminishes. The training accuracy reaches approximately 59.8%, while the testing accuracy stabilizes around 58.5% when C = 1000. Additionally, the standard error of the model is lowest at this point, indicating optimal model performance. Similar analyses were conducted to determine optimal C values for other target variables, as summarized in Table I.

For most of the target variables, we can see that the optimal

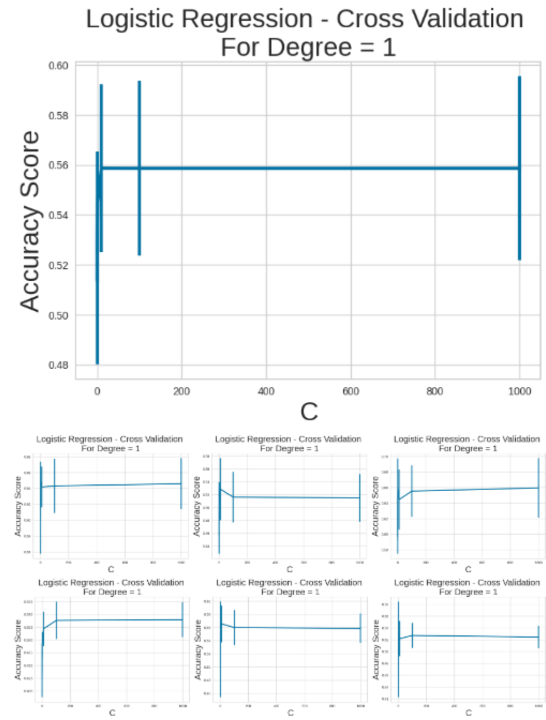


Fig. 7. Cross-Validation Accuracy Scores for Degree 1

TABLE I
VALUES OF C FOR DIFFERENT TARGET VARIABLES

Target Variable	C Value	Train Accuracy	Test accuracy
accuracy	1000	0.59	0.58
checkin	10	0.70	0.67
cleanliness	10	0.56	0.57
communication	1	0.72	0.66
location	100	0.54	0.48
rating	100	0.64	0.56
value	1000	0.61	0.56

value of C is 100 or less. If we continue increasing the value of C, the penalty applied to the model decreases, hence exposing the model to the risk of overfitting. This further supports the claim of the optimal C values presented in Table I. Additionally, the process of augmenting the features to a polynomial feature space was computationally expensive and hence not the right choice for such a dataset.

Additionally, I augmented and 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 significant gap of 10% between the training and testing accuracy scores. The cross-validation plot for all the target variables to be predicted by the model is presented in Figure 8.

B. k-NN CLASSIFIER AND PARAMETER TUNING

k-NN, or k-Nearest Neighbors, is a supervised machine learning algorithm that groups the data by their target variables or classes. The model uses its hyperparameter (k neighbors) and the distance between the nearest class to classify new

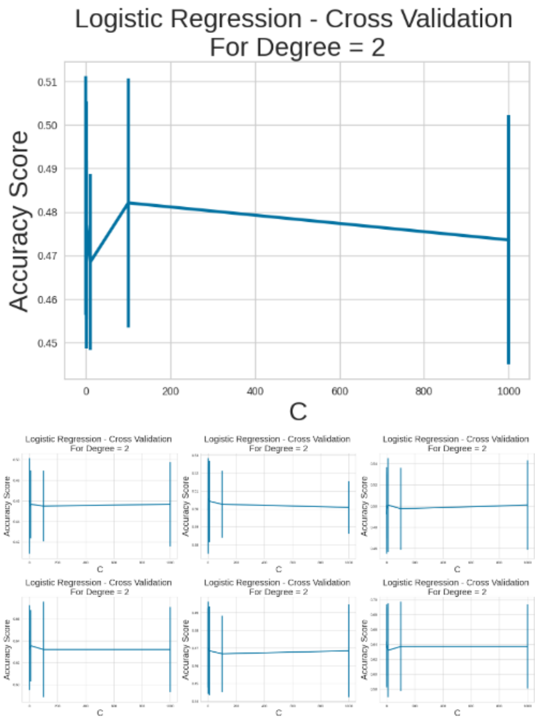


Fig. 8. Cross-Validation Accuracy Scores for Degree 2

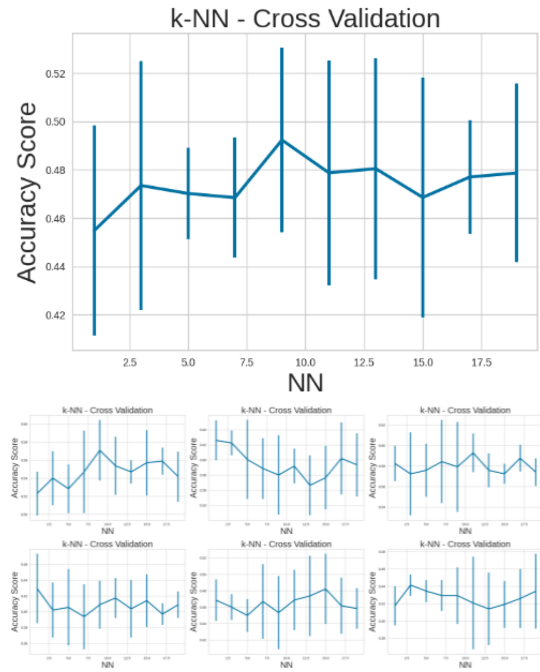


Fig. 9. k-NN Cross Validation for the target variable

data points. While predicting, the model calculates the distance (usually Cosine or Euclidean) between data points and identifies the 'k' nearest neighbors to assign a class to it. In this particular model, I only changed the value of k, which is the number of neighbors, and kept the weights uniform. I only selected odd numbers because selecting even numbers might result in ties, leading to many misclassified values. To verify the performance of the model, I implemented 5-fold cross-validation. The number of neighbors on which this model was built are as follows – N Neighbors Range: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19].

In Figure 9, the cross-validation plot for the target variable review scores accuracy is shown 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 neighbors is 9. Although the accuracy score of the 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 increase the value of k, the model becomes simpler, suppressing the noise in the dataset. Hence, the optimal value of k is 9 for the target variable review scores accuracy, with a training accuracy score of 60% and a testing accuracy score of 51%. Similarly, the optimal value of k and the training and testing accuracy scores are presented in Table II. The cross-validation plots for all the remaining six target variables are provided in Figure 9.

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

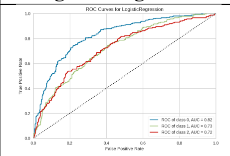
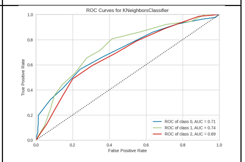
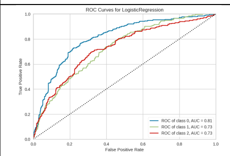
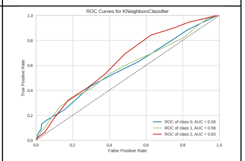
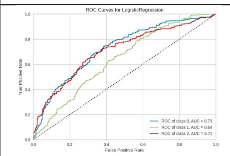
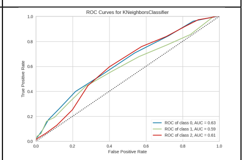
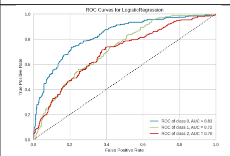
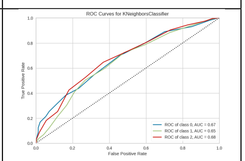
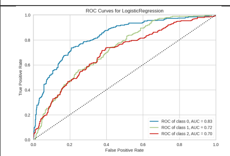
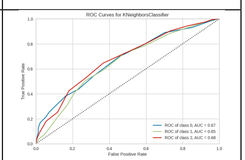
Target Variable	K Value	Train Accuracy	Test accuracy
accuracy	9	0.60	0.51
checkin	9	0.69	0.60
cleanliness	5	0.62	0.42
communication	7	0.73	0.63
location	5	0.60	0.40
rating	15	0.57	0.49
value	19	0.53	0.45

C. *RESULT EVALUATION AND MODELS COMPARISON*

In this section, to compare the results from both models, I have chosen to compare the scores of the target variable review scores accuracy. As a metric to assess the models, I used ROC-AUC curves. 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 both Logistic Regression and k-NN models in Table III.

From the values presented in the ROC-AUC curves, we can see that the Logistic Regression model is a better choice for this particular target variable compared to a k-NN classifier. The AUC for all three classes for the target variable review scores accuracy is higher for Logistic Regression compared to a k-NN classifier. Hence, if provided with a choice, I would select Logistic Regression as the appropriate model for this problem statement. The ROC-AUC curves for all the other target variables are presented in Table III. Also, when comparing both models with a baseline classifier, we see that both selected models outperform a dummy classifier that always predicts the most frequent value (accuracy of 32%)

TABLE III
ROC-AUC FOR DIFFERENT CLASSIFIERS AND TARGET VARIABLES

Target Variable	**Logistic Regression**	**k-NN Classifier**
accuracy		
cleanliness		
location		
rating		
value		

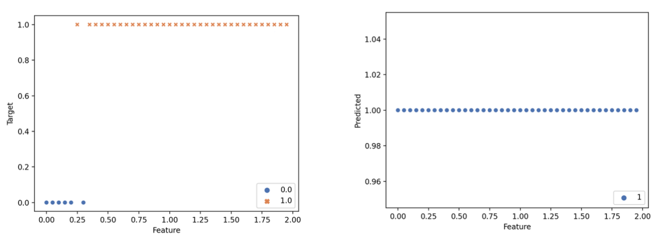


Fig. 10. Plot of Data Imbalance

and a dummy classifier that predicts the rating bin uniformly (accuracy of 33%). This proves that our model selection is accurate and that the performance of the model for this use case is optimal.

IV. 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, represented in Figure 10 left. In this data, the target value consists of 34 '1s' and only 6 '0s'. On predicting this Feature, the model can only predict for the majority class, i.e. 1, as shown in Figure 10 right.

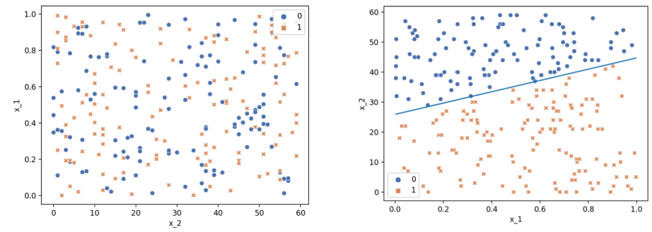


Fig. 11. Two Features and Target Values

2) **No Linear Correlation**
If data have no Linear Correlation between each other, the model will not be able to predict the target variable accurately. For example, I have created two features and one target variable randomly. Figure 11 left displays the two features and the respective target values. On predicting the target values from the features, it can be noticed from the following plot, Figure 11 right, that the values are predicted linearly and not in the scattered way as they were created.

B. QUESTION II

1) **KNN's Advantages**
The model relies on a straightforward calculation to determine the data points that belong to a specific class. The algorithm calculates the distance between the new data point and all the data points in the dataset and then selects the K data points closest to the new data point. Furthermore, it relies on the entire dataset to make predictions. The algorithm can be used simply on a new dataset without additional training. Lastly, it can be used for both Classification and Regression. For Classification, the KNN model classifies the target variables by a majority vote of its neighbours. For Regression, the output is the property value for an object. This value is the average of its K Nearest Neighbors.
2) **KNN's Disadvantages**
It can be slow at predicting the target class for a new data instance because it searches through the entire dataset to find the N-Neighbors. Furthermore, the algorithm is sensitive to irrelevant/outliers in the dataset. The data needs to be scaled appropriately to handle these outliers. Lastly, A model for the KNN algorithm is only created once a prediction is performed. An overhead when training data is enormous and prediction time is critical.
3) **MLP's Advantages**
MLP Classifiers can model complex relationships within the data. These classifiers can learn non-linear relationships between the input and output variables. They can also handle high-dimensional data like images, text and audio by learning distributed representations of the input data in the hidden layers. Furthermore, MLP Classifiers can be trained on large datasets and achieve high predictive accuracy on many real-world classification tasks. They can be used for predicting the target values quickly,

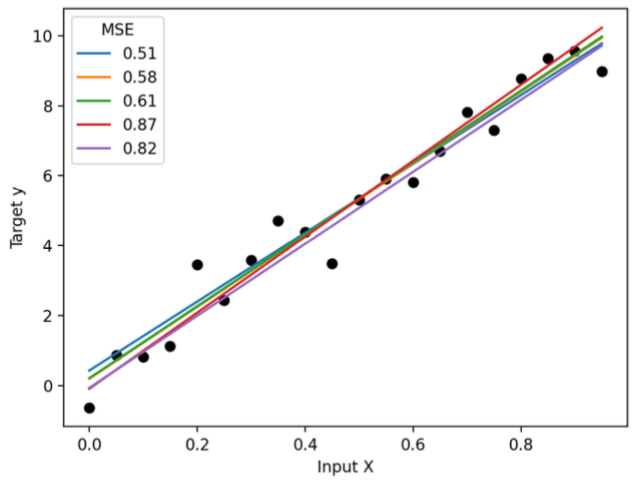


Fig. 12. Random data splits MSE

as the model gets stored after training on the data. Lastly, in MLP, several hyperparameters are available for tuning, such as the learning rate, activation function, number of hidden layers, amount of regularisation, dropout, and batch size.

4) **MLP's Disadvantages**
If the dataset is large, it can take much time to train, especially it there are many hidden layers in the model. Due to the many hidden layers available in the model, it needs a lot of hyperparameter tuning to get satisfactory results. Furthermore, MLP models can be challenging to interpret as the model becomes complex with an increase in the number of hidden layers. Hence, making it work as a black box model does not provide a clear insight into its working.

C. QUESTION III

In K-Fold Cross Validation, the dataset is resampled multiple times to train and evaluate the model on different subsets of the data, to understand how well the model generalises. When the model is trained on a single dataset split, it might perform poorly on other unseen data leading to an overly optimistic evaluation of the models' performance. Resampling the data multiple times across different folds can give a more reliable estimate of the model's performance on unseen data with different data splits. For example, I have created a feature X using np.arange with values ranging between 0 and 1, and target y from feature X. Figure 12 is created by randomly splitting the training data 5 times and predicting the target output for each iteration. The Mean Squared Error is calculated for these independent splits (folds). It can be seen that the MSE for each of the folds changes. It is understood that evaluating the model on a single train-test split is not a good option.

Selecting the Value of K: On selecting a small value of k, the model would be trained and evaluated on a limited number of folds, leading to a high bias, indicating that the model makes assumptions about the target variable. Nevertheless, on selecting a high value of k, the model will be trained on many folds, leading to a high variance, indicating that the model would overfit the data and be sensitive to small changes in the target variables.

	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. 13. Time Series Features for Product Usage

D. QUESTION VI

If we want to predict the value of a time series data at time t+q, q is the future value. We can use lagged output values to create features for the time series at time t, t-1, and t-2. These values contain valuable information to predict future values. For Example, if we want to create a feature of a times series to predict the number of product usage per month. The lagged values are created by shifting the current value by one or two steps. From Figure 13, it can be seen that for the feature tag_1, the data is shifted by one step; similarly, for tag_2, it is shifted by two steps.

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)
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")
```

```
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.preprocessing import PolynomialFeatures
from sklearn import metrics
from sklearn.model_selection import cross_val_score


#  Data Loading - Listings and Reviews

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

#  Splitting the amenities and logically clustering them into 9 different categories

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


#   Splitting the host contact details into 3
    categories namely host_email, host_phone,
    host_work and host_work_email
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


#   Handling of NaN values of New Feature Columns
    Created
new_feature_cols = listings.iloc[:,75:].columns
listings[new_feature_cols] = listings[
    new_feature_cols].fillna(0)


#   Merging the Listings data with the Cleansed
    Reviews Dataset
listings = listings.merge(reviews, how='inner',
    left_on='id', right_on='listing_id')


#   Dropping the columns which are not required
    while building a Machine Learning Model
#   Plot %NaN Values
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')
plot_nas(listings)


#   Column Drop List
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()


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


#   Pre-Processing of Features 'price', '
    host_response_rate' and 'host_acceptance_rate'
listings['price'] = listings['price'].str.replace('$
    ', '')
listings['price'] = listings['price'].str.replace(',
    ', '')
listings['price'] = pd.to_numeric(listings['price'])
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'])
```

```
listings[['price']].plot(kind='box', title='Price BoxPlot')
plt.ylim(0,1750)


#   Outlier Removal for the Feature 'price'
listings = listings[listings.price > 50]
listings = listings[listings.price <= 300]
listings[['price']].plot(kind='box', title='Price BoxPlot')


#   Neighbourhood Analysis
listings.neighbourhood_cleansed.unique()
listings.groupby('neighbourhood_cleansed').host_response_time.count()
neighbourhood_DF=listings.groupby('neighbourhood_cleansed').host_response_time.count()
neighbourhood_DF=neighbourhood_DF.reset_index()
neighbourhood_DF=neighbourhood_DF.rename(columns={'host_response_time':'Number_Of_Listings'})
neighbourhood_DF.plot(kind='bar',
           x='neighbourhood_cleansed',
           y='Number_Of_Listings',
           figsize =(18,8),
           title = 'Dublin Neighborhood Frequency',
           legend = False)


#   Statistical Analysis
print(listings.review_scores_accuracy.mean())
print(listings.review_scores_checkin.mean())
print(listings.review_scores_cleanliness.mean())
print(listings.review_scores_communication.mean())
print(listings.review_scores_location.mean())
print(listings.review_scores_rating.mean())
print(listings.review_scores_value.mean())

print(listings.review_scores_accuracy.median())
print(listings.review_scores_checkin.median())
print(listings.review_scores_cleanliness.median())
print(listings.review_scores_communication.median())
print(listings.review_scores_location.median())
print(listings.review_scores_rating.median())
print(listings.review_scores_value.median())

print(listings.review_scores_accuracy.mode())
print(listings.review_scores_checkin.mode())
print(listings.review_scores_cleanliness.mode())
print(listings.review_scores_communication.mode())
print(listings.review_scores_location.mode())
print(listings.review_scores_rating.mode())
print(listings.review_scores_value.mode())

print(listings.review_scores_accuracy.min())
print(listings.review_scores_checkin.min())
print(listings.review_scores_cleanliness.min())
print(listings.review_scores_communication.min())
print(listings.review_scores_location.min())
print(listings.review_scores_rating.min())
print(listings.review_scores_value.min())

print(listings.review_scores_accuracy.max())
print(listings.review_scores_checkin.max())
print(listings.review_scores_cleanliness.max())
print(listings.review_scores_communication.max())
print(listings.review_scores_location.max())
print(listings.review_scores_rating.max())
print(listings.review_scores_value.max())


#   Density Plots for all 7 Target Variables (Review Scores)
plt.figure(figsize=(18, 18))
plt.subplot(3, 3, 1)
review_scores_accuracy_density = sns.kdeplot(listings.review_scores_accuracy, color="b",
                                              label='review_scores_accuracy - density')
x_review_scores_accuracy = review_scores_accuracy_density.lines[0].get_xdata()
y_review_scores_accuracy = review_scores_accuracy_density.lines[0].get_ydata()
mean_review_scores_accuracy_density = listings.review_scores_accuracy.mean()
height_review_scores_accuracy = np.interp(mean_review_scores_accuracy_density, x_review_scores_accuracy,
                                          y_review_scores_accuracy)
review_scores_accuracy_density.vlines(mean_review_scores_accuracy_density, 0, height_review_scores_accuracy,
                                      color='b', ls=':')
review_scores_accuracy_density.fill_between(x_review_scores_accuracy, 0, y_review_scores_accuracy,
                                            facecolor='b', alpha=0.2)
plt.legend()


plt.subplot(3, 3, 2)
review_scores_checkin_density = sns.kdeplot(listings.review_scores_checkin, color="b",
                                              label='review_scores_checkin - density')
x_review_scores_checkin = review_scores_checkin_density.lines[0].get_xdata()
y_review_scores_checkin = review_scores_checkin_density.lines[0].get_ydata()
mean_review_scores_checkin_density = listings.review_scores_checkin.mean()
height_review_scores_checkin = np.interp(mean_review_scores_checkin_density, x_review_scores_checkin,
                                          y_review_scores_checkin)
review_scores_checkin_density.vlines(mean_review_scores_checkin_density, 0, height_review_scores_checkin,
                                      color='b', ls=':')
review_scores_checkin_density.fill_between(x_review_scores_checkin, 0, y_review_scores_checkin,
                                            facecolor='b', alpha=0.2)
plt.legend()



plt.subplot(3, 3, 3)
review_scores_cleanliness_density = sns.kdeplot(listings.review_scores_cleanliness, color="b",
                                              label='review_scores_cleanliness - density')
x_review_scores_cleanliness = review_scores_cleanliness_density.lines[0].get_xdata()
y_review_scores_cleanliness =
```

```
review_scores_cleanliness_density.lines[0].get_ydata()
mean_review_scores_cleanliness_density = listings.review_scores_cleanliness.mean()
height_review_scores_cleanliness = np.interp(mean_review_scores_cleanliness_density, x_review_scores_cleanliness,
                                             y_review_scores_cleanliness)
review_scores_cleanliness_density.vlines(mean_review_scores_cleanliness_density, 0, height_review_scores_cleanliness,
                                          color='b', ls=':')
review_scores_cleanliness_density.fill_between(x_review_scores_cleanliness, 0, y_review_scores_cleanliness,
                                               facecolor='b', alpha=0.2)
plt.legend()


plt.subplot(3, 3, 4)
review_scores_communication_density = sns.kdeplot(listings.review_scores_communication, color="b",
                                                  label='review_scores_communication - density')
x_review_scores_communication = review_scores_communication_density.lines[0].get_xdata()
y_review_scores_communication = review_scores_communication_density.lines[0].get_ydata()
mean_review_scores_communication_density = listings.review_scores_communication.mean()
height_review_scores_communication = np.interp(mean_review_scores_communication_density, x_review_scores_communication,
                                               y_review_scores_communication)
review_scores_communication_density.vlines(mean_review_scores_communication_density, 0, height_review_scores_communication,
                                            color='b', ls=':')
review_scores_communication_density.fill_between(x_review_scores_communication, 0, y_review_scores_communication,
                                                 facecolor='b', alpha=0.2)
plt.legend()

plt.subplot(3, 3, 5)
review_scores_rating_density = sns.kdeplot(listings.review_scores_rating, color="b",
                                           label='review_scores_rating - density')
x_review_scores_rating = review_scores_rating_density.lines[0].get_xdata()
y_review_scores_rating = review_scores_rating_density.lines[0].get_ydata()
mean_review_scores_rating_density = listings.review_scores_rating.mean()
height_review_scores_rating = np.interp(mean_review_scores_rating_density, x_review_scores_rating,
                                        y_review_scores_rating)
review_scores_rating_density.vlines(mean_review_scores_rating_density, 0, height_review_scores_rating,
                                     color='b', ls=':')
review_scores_rating_density.fill_between(x_review_scores_rating, 0, y_review_scores_rating,
                                          facecolor='b', alpha=0.2)
plt.legend()


plt.subplot(3, 3, 6)
review_scores_location_density = sns.kdeplot(listings.review_scores_location, color="b",
                                             label='review_scores_location - density')
x_review_scores_location = review_scores_location_density.lines[0].get_xdata()
y_review_scores_location = review_scores_location_density.lines[0].get_ydata()
mean_review_scores_location_density = listings.review_scores_location.mean()
height_review_scores_location = np.interp(mean_review_scores_location_density, x_review_scores_location,
                                          y_review_scores_location)
review_scores_location_density.vlines(mean_review_scores_location_density, 0, height_review_scores_location,
                                       color='b', ls=':')
review_scores_location_density.fill_between(x_review_scores_location, 0, y_review_scores_location,
                                            facecolor='b', alpha=0.2)
plt.legend()

plt.subplot(3, 3, 8)
review_scores_value_density = sns.kdeplot(listings.review_scores_value, color="b",
                                          label='review_scores_value - density')
x_review_scores_value = review_scores_value_density.lines[0].get_xdata()
y_review_scores_value = review_scores_value_density.lines[0].get_ydata()
mean_review_scores_value_density = listings.review_scores_value.mean()
height_review_scores_value = np.interp(mean_review_scores_value_density, x_review_scores_value,
                                       y_review_scores_value)
review_scores_value_density.vlines(mean_review_scores_value_density, 0, height_review_scores_value,
                                    color='b', ls=':')
review_scores_value_density.fill_between(x_review_scores_value, 0, y_review_scores_value,
                                         facecolor='b', alpha=0.2)
plt.legend()

plt.show()


#   Data Scaling
#   Min Max Scaling
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


#   List of Columns to be Scaled
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])


#   Label Encoding
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)


#   Correlation Matrix
test_corr = listings.corr()
test_corr.to_csv("test_corr.csv")

#   Metrics Print Function
def print_metrics(y_test, y_pred):
    print("-"*10+"CONFUSION-MATRIX"+"-"*10)
    print(confusion_matrix(y_test, y_pred))

    print("-"*10+"CLASSIFICATION-REPORT"+"-"*10)
    print(classification_report(y_test, y_pred))

#   Review Scores Accuracy
#   Logistic Regression
#    Defining the Input Variables
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', '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:])
]


#    Defining the Quantile Bins for the Target Variables
y = listings[['review_scores_accuracy']]
y = (y/y.max())*100

y = y.assign(
    rating_bin_ep = pd.qcut(
        y['review_scores_accuracy'],
        q=3,
        duplicates='drop',
        labels=[0,1,2]
    )
)


#    Min Max of Each Bin
y.groupby('rating_bin_ep').min()
y.groupby('rating_bin_ep').max()
y = y['rating_bin_ep']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)


#    Number of Records in Each Bin
cnt_plt = sns.countplot(y)
cnt_plt.bar_label(cnt_plt.containers[0])
plt.show()


#    Logistic Regression - Varied C Range, using 'newton-cg' solver and multi_class='multinomial'
c_range = [0.001, 0.1, 1, 10, 100, 1000]
mean_error = []
std_error = []
for c in sorted(c_range):
    logit = LogisticRegression(C=c, random_state=0, solver='newton-cg',multi_class='multinomial')
    logit.fit(X_train, y_train)
    y_pred = logit.predict(X_test)
    print("C = ",c)
    print('Train accuracy score:',logit.score(X_train, y_train))
    print('Test accuracy score:',logit.score(X_test, y_test))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    scores = cross_val_score(logit, X_test, y_test, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())
plt.errorbar(c_range, mean_error, yerr = std_error, linewidth=3)
plt.xlabel('C', fontsize=25)
plt.ylabel('Accuracy Score', fontsize=25)
title_cv = "Logistic Regression - Cross Validation \nFor Degree = 1"
plt.title(title_cv, fontsize=25)
plt.show()


#    ROC-AUC Curve for all three categories
```

```python
visualizer = ROCAUC(logit, classes=["0", "1", "2"], macro=False, micro=False)

visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()

#    Feature Importance
from matplotlib import pyplot
cols = X.columns
cols = np.asarray(cols)

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                             'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[1])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                             'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[2])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                             'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()


#    Comparison With Baseline Classifier
dmfr = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
dmun = DummyClassifier(strategy='uniform').fit(X_train, y_train)

print("\n\nDUMMY CLASSIFIER - frequent")
print(dmfr.score(X_test, y_test))
y_pred = dmfr.predict(X_test)
print_metrics(y_test, y_pred)

print("\n\nDUMMY CLASSIFIER - Uniform")
print(dmun.score(X_test, y_test))
y_pred = dmun.predict(X_test)
print_metrics(y_test, y_pred)


#    Density Plot for Term 'Home' for each Review Bin
test = pd.concat([X, y], axis=1)
g = sns.kdeplot(test.loc[test['rating_bin_ep'] == 0, 'term_home'], color="r", label='term_home_intensity, bin=0')
g.set(ylim=(0, 19))
g.set(xlim=(-0.25, 1))
plt.legend()

h = sns.kdeplot(test.loc[test['rating_bin_ep'] == 1, 'term_home'], color="g", label='term_home_intensity, bin=1')
h.set(ylim=(0, 19))
h.set(xlim=(-0.25, 1))
plt.legend()

i = sns.kdeplot(test.loc[test['rating_bin_ep'] == 2, 'term_home'], color="b", label='term_home_intensity, bin=2')
i.set(ylim=(0, 19))
i.set(xlim=(-0.25, 1))
plt.legend()

plt.show()


#    Density Plots for Number of Comments against each Review Bin
test = pd.concat([X, y], axis=1)

g = sns.kdeplot(test.loc[test['rating_bin_ep'] == 0, 'num_cmt'], color="r", label='num_cmt, bin=0')
g.set(ylim=(0, 0.03))
plt.legend()

h = sns.kdeplot(test.loc[test['rating_bin_ep'] == 1, 'num_cmt'], color="g", label='num_cmt, bin=1')
h.set(ylim=(0, 0.03))
plt.legend()

i = sns.kdeplot(test.loc[test['rating_bin_ep'] == 2, 'num_cmt'], color="b", label='num_cmt, bin=2')
i.set(ylim=(0, 0.03))
plt.legend()

plt.show()


#    Number of Records in each Review Bin for the type of Host (Superhost)
test.host_is_superhost.value_counts()
test1 = test.groupby(['host_is_superhost', 'rating_bin_ep']).size()

plt.figure(figsize=(18, 6))
plt.subplot(1, 2, 1)
for j in (0,1,2):
```

```python
    plt.bar(j, test1[0][j], label = str(j))
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 (0,1,2):
    plt.bar(j, test1[1][j], label = str(j))
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()


#    Neighbourhood Type Analysis
test.neighbourhood_cleansed.value_counts()
test2 = test.groupby(['neighbourhood_cleansed', 'rating_bin_ep']).size()

plt.figure(figsize=(18, 6))
plt.subplot(1, 2, 1)
for j in (0,1,2):
    plt.bar(j, test2[0][j], label = str(j))
plt.title("Neighbourhood 0",fontsize=25)
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 (0,1,2):
    plt.bar(j, test2[1][j], label = str(j))
plt.title("Neighbourhood 1",fontsize=25)
plt.legend()
plt.xlabel('Review Bin', fontweight ='bold', fontsize = 15)
plt.ylabel('# Records', fontweight ='bold', fontsize = 15)
plt.show()

plt.figure(figsize=(18, 6))
plt.subplot(1, 2, 1)
for j in (0,1,2):
    plt.bar(j, test2[2][j], label = str(j))
plt.title("Neighbourhood 2",fontsize=25)
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 (0,1,2):
    plt.bar(j, test2[3][j], label = str(j))
plt.title("Neighbourhood 3",fontsize=25)
plt.legend()
plt.xlabel('Review Bin', fontweight ='bold', fontsize = 15)
plt.ylabel('# Records', fontweight ='bold', fontsize = 15)
plt.show()


#    Polynomial Degree and Error Plots
c_range = [0.001, 0.1, 1, 10, 100, 1000]
degree_range = [2]

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, y, 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 = metrics.confusion_matrix(y_test, y_pred)
        f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx[1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])

        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(" Classification Report\n", classification_report(y_test, y_pred))
        print("\n")

    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()


#   k-NN Classifier
nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
x_train_nn, x_test_nn, y_train_nn, y_test_nn = train_test_split(X, y, 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_nn, y_train_nn)
    y_pred_nn = knn_model.predict(x_test_nn)
    print("NN = ", nn)
    print('Train accuracy score:',knn_model.score(x_train_nn, y_train_nn))
    print('Test accuracy score:',knn_model.score(x_test_nn, y_test_nn))

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

plt.errorbar(nn_range, merr, yerr = serr, linewidth=3)
plt.xlabel('NN', fontsize=25)
plt.ylabel('Accuracy Score', fontsize=25)
```

```python
title_cv = f"k-NN - Cross Validation"
plt.title(title_cv, fontsize=25)
plt.show()


#    ROC-AUC Curve
visualizer = ROCAUC(knn_model, classes=["0", "1", "2"], macro=False, micro=False)

visualizer.fit(x_train_nn, y_train_nn)
visualizer.score(x_test_nn, y_test_nn)
visualizer.show()


#   Review Scores Checkin
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', '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[['review_scores_checkin']]
y = (y/y.max())*100

y = y.assign(
    rating_bin_ep = pd.qcut(
        y['review_scores_checkin'],
        q=2,
        duplicates='drop',
        labels=[0,1]
    )
)


#    Min Max of Each Bin
y.groupby('rating_bin_ep').min()
y.groupby('rating_bin_ep').max()


#    Splitting Data in 75-25 Ratio
y = y['rating_bin_ep']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)


#    Number of Records in Each Bin
cnt_plt = sns.countplot(y)
cnt_plt.bar_label(cnt_plt.containers[0])
plt.show()


#    Logistic Regression - Varied C Range, using 'newton-cg' solver and multi_class='multinomial'
c_range = [0.001, 0.1, 1, 10, 100, 1000]
mean_error = []
std_error = []
for c in sorted(c_range):
    logit = LogisticRegression(C=c, random_state=0, solver='newton-cg',multi_class='multinomial')
    logit.fit(X_train, y_train)
    y_pred = logit.predict(X_test)
    print("C = ",c)
    print('Train accuracy score:',logit.score(X_train, y_train))
    print('Test accuracy score:',logit.score(X_test, y_test))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    scores = cross_val_score(logit, X_test, y_test, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())
plt.errorbar(c_range, mean_error, yerr = std_error, linewidth=3)
plt.xlabel('C', fontsize=25)
plt.ylabel('Accuracy Score', fontsize=25)
title_cv = "Logistic Regression - Cross Validation \nFor Degree = 1"
plt.title(title_cv, fontsize=25)
plt.show()


#    Feature Importance
cols = X.columns
cols = np.asarray(cols)

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                              'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()


#    Polynomial Degree and Error Plots
c_range = [0.001, 0.1, 1, 10, 100, 1000]
degree_range = [2]

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, y, 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 = metrics.confusion_matrix(y_test, y_pred)
        f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx[1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])

        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(" Classification Report\n", classification_report(y_test, y_pred))
        print("\n")

    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()


#   k-NN Classifier
nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
x_train_nn, x_test_nn, y_train_nn, y_test_nn = train_test_split(X, y, 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_nn, y_train_nn)
    y_pred_nn = knn_model.predict(x_test_nn)
    print("NN = ", nn)
    print('Train accuracy score:',knn_model.score(x_train_nn, y_train_nn))
    print('Test accuracy score:',knn_model.score(x_test_nn, y_test_nn))

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

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()


#   Review Scores Cleanliness
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', '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[['review_scores_cleanliness']]
y = (y/y.max())*100

y = y.assign(
    rating_bin_ep = pd.qcut(
        y['review_scores_cleanliness'],
        q=3,
        duplicates='drop',
        labels=[0,1,2]
    )
)


#    Min Max of Each Bin
y.groupby('rating_bin_ep').min()
y.groupby('rating_bin_ep').max()


#    Splitting Data in 75-25 Ratio
y = y['rating_bin_ep']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)


#    Number of Records in Each Bin
cnt_plt = sns.countplot(y)
cnt_plt.bar_label(cnt_plt.containers[0])
plt.show()


#    Logistic Regression - Varied C Range, using 'newton-cg' solver and multi_class='multinomial'
c_range = [0.001, 0.1, 1, 10, 100, 1000]
mean_error = []
std_error = []
for c in sorted(c_range):
    logit = LogisticRegression(C=c, random_state=0, solver='newton-cg',multi_class='multinomial')
    logit.fit(X_train, y_train)
    y_pred = logit.predict(X_test)
    print("C = ",c)
    print('Train accuracy score:',logit.score(X_train, y_train))
    print('Test accuracy score:',logit.score(X_test, y_test))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    scores = cross_val_score(logit, X_test, y_test, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())
plt.errorbar(c_range, mean_error, yerr = std_error, linewidth=3)
plt.xlabel('C', fontsize=25)
plt.ylabel('Accuracy Score', fontsize=25)
title_cv = "Logistic Regression - Cross Validation \nFor Degree = 1"
plt.title(title_cv, fontsize=25)
plt.show()


#    Feature Importance
cols = X.columns
cols = np.asarray(cols)

plt.figure(figsize=(30,15))
```

```
feature_importance = abs(logit.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                              'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[1])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                              'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[2])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                              'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()


#    ROC-AUC Curve for all three categories
visualizer = ROCAUC(logit, classes=["0", "1", "2"], macro=False, micro=False)

visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()


#    Polynomial Degree and Error Plots
c_range = [0.001, 0.1, 1, 10, 100, 1000]
degree_range = [2]

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, y, 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 = metrics.confusion_matrix(y_test, y_pred)
        f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx[1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])

        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(" Classification Report\n", classification_report(y_test, y_pred))
        print("\n")

    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()


#    k-NN Classifier
nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
x_train_nn, x_test_nn, y_train_nn, y_test_nn = train_test_split(X, y, 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_nn, y_train_nn)
    y_pred_nn = knn_model.predict(x_test_nn)
    print("NN = ", nn)
    print('Train accuracy score:',knn_model.score(x_train_nn, y_train_nn))
    print('Test accuracy score:',knn_model.score(x_test_nn, y_test_nn))

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

plt.errorbar(nn_range, merr, yerr = serr, linewidth=3)
```

```python
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()


#    ROC-AUC Curve
visualizer = ROCAUC(knn_model, classes=["0", "1", "2"], macro=False, micro=False)

visualizer.fit(x_train_nn, y_train_nn)
visualizer.score(x_test_nn, y_test_nn)
visualizer.show()


#   Review Scores Communication
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', '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[['review_scores_communication']]
y = (y/y.max())*100

y = y.assign(
    rating_bin_ep = pd.qcut(
        y['review_scores_communication'],
        q=2,
        duplicates='drop',
        labels=[0,1]
    )
)


#    Min Max of Each Bin
y.groupby('rating_bin_ep').min()
y.groupby('rating_bin_ep').max()


#    Splitting Data in 75-25 Ratio
y = y['rating_bin_ep']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)


#    Number of Records in Each Bin
cnt_plt = sns.countplot(y)
cnt_plt.bar_label(cnt_plt.containers[0])
plt.show()


#    Logistic Regression - Varied C Range, using 'newton-cg' solver and multi_class='multinomial'
c_range = [0.001, 0.1, 1, 10, 100, 1000]
mean_error = []
std_error = []
for c in sorted(c_range):
    logit = LogisticRegression(C=c, random_state=0, solver='newton-cg',multi_class='multinomial')
    logit.fit(X_train, y_train)
    y_pred = logit.predict(X_test)
    print("C = ",c)
    print('Train accuracy score:',logit.score(X_train, y_train))
    print('Test accuracy score:',logit.score(X_test, y_test))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    scores = cross_val_score(logit, X_test, y_test, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())
plt.errorbar(c_range, mean_error, yerr = std_error, linewidth=3)
plt.xlabel('C', fontsize=25)
plt.ylabel('Accuracy Score', fontsize=25)
title_cv = "Logistic Regression - Cross Validation \nFor Degree = 1"
plt.title(title_cv, fontsize=25)
plt.show()


#    Feature Importance
cols = X.columns
cols = np.asarray(cols)

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                             'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()


#    Polynomial Degree and Error Plots
c_range = [0.001, 0.1, 1, 10, 100, 1000]
degree_range = [2]

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, y, 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 = metrics.confusion_matrix(y_test, y_pred)
        f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx[1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])
```

```
        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(" Classification Report\n", classification_report(y_test, y_pred))
        print("\n")

    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()


#   k-NN Classifier
nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
x_train_nn, x_test_nn, y_train_nn, y_test_nn = train_test_split(X, y, 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_nn, y_train_nn)
    y_pred_nn = knn_model.predict(x_test_nn)
    print("NN = ", nn)
    print('Train accuracy score:',knn_model.score(x_train_nn, y_train_nn))
    print('Test accuracy score:',knn_model.score(x_test_nn, y_test_nn))

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

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()


#   Review Scores Location
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', '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[['review_scores_location']]
y = (y/y.max())*100

y = y.assign(
    rating_bin_ep = pd.qcut(
        y['review_scores_location'],
        q=3,
        duplicates='drop',
        labels=[0,1,2]
    )
)


#    Min Max of Each Bin
y.groupby('rating_bin_ep').min()
y.groupby('rating_bin_ep').max()


#    Splitting Data in 75-25 Ratio
y = y['rating_bin_ep']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)


#    Number of Records in Each Bin
cnt_plt = sns.countplot(y)
cnt_plt.bar_label(cnt_plt.containers[0])
plt.show()


#    Logistic Regression - Varied C Range, using 'newton-cg' solver and multi_class='multinomial'
c_range = [0.001, 0.1, 1, 10, 100, 1000]
mean_error = []
std_error = []
for c in sorted(c_range):
    logit = LogisticRegression(C=c, random_state=0, solver='newton-cg',multi_class='multinomial')
    logit.fit(X_train, y_train)
    y_pred = logit.predict(X_test)
    print("C = ",c)
    print('Train accuracy score:',logit.score(X_train, y_train))
    print('Test accuracy score:',logit.score(X_test, y_test))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    scores = cross_val_score(logit, X_test, y_test, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())
plt.errorbar(c_range, mean_error, yerr = std_error, linewidth=3)
plt.xlabel('C', fontsize=25)
plt.ylabel('Accuracy Score', fontsize=25)
title_cv = "Logistic Regression - Cross Validation \nFor Degree = 1"
plt.title(title_cv, fontsize=25)
plt.show()


#    Feature Importance
cols = X.columns
cols = np.asarray(cols)
```

```
plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                             'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[1])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                             'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[2])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                             'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()


#    ROC-AUC Curve for all three categories
visualizer = ROCAUC(logit, classes=["0", "1", "2"], macro=False, micro=False)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()

#    Polynomial Degree and Error Plots
c_range = [0.001, 0.1, 1, 10, 100, 1000]
degree_range = [2]

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, y, 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 = metrics.confusion_matrix(y_test, y_pred)
        f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx[1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])

        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(" Classification Report\n", classification_report(y_test, y_pred))
        print("\n")

    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()


#    k-NN Classifier
nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
x_train_nn, x_test_nn, y_train_nn, y_test_nn = train_test_split(X, y, 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_nn, y_train_nn)
    y_pred_nn = knn_model.predict(x_test_nn)
    print("NN = ", nn)
    print('Train accuracy score:',knn_model.score(x_train_nn, y_train_nn))
    print('Test accuracy score:',knn_model.score(x_test_nn, y_test_nn))

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

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()


#    ROC-AUC Curve
visualizer = ROCAUC(knn_model, classes=["0", "1", "2"], macro=False, micro=False)

visualizer.fit(x_train_nn, y_train_nn)
visualizer.score(x_test_nn, y_test_nn)
visualizer.show()


#   Review Scores Rating
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', '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[['review_scores_rating']]
y = (y/y.max())*100

y = y.assign(
    rating_bin_ep = pd.qcut(
        y['review_scores_rating'],
        q=3,
        duplicates='drop',
        labels=[0,1,2]
    )
)


#    Min Max of Each Bin
y.groupby('rating_bin_ep').min()
y.groupby('rating_bin_ep').max()


#    Splitting Data in 75-25 Ratio
y = y['rating_bin_ep']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)


#    Number of Records in Each Bin
cnt_plt = sns.countplot(y)
cnt_plt.bar_label(cnt_plt.containers[0])
plt.show()


#    Logistic Regression - Varied C Range, using 'newton-cg' solver and multi_class='multinomial'
c_range = [0.001, 0.1, 1, 10, 100, 1000]
mean_error = []
std_error = []
for c in sorted(c_range):
    logit = LogisticRegression(C=c, random_state=0, solver='newton-cg',multi_class='multinomial')
    logit.fit(X_train, y_train)
    y_pred = logit.predict(X_test)
    print("C = ",c)
    print('Train accuracy score:',logit.score(X_train, y_train))
    print('Test accuracy score:',logit.score(X_test, y_test))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    scores = cross_val_score(logit, X_test, y_test, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())
plt.errorbar(c_range, mean_error, yerr = std_error, linewidth=3)
plt.xlabel('C', fontsize=25)
plt.ylabel('Accuracy Score', fontsize=25)
title_cv = "Logistic Regression - Cross Validation \nFor Degree = 1"
plt.title(title_cv, fontsize=25)
plt.show()


#    Feature Importance
cols = X.columns
cols = np.asarray(cols)

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                             'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[1])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp': feature_importance,
                             'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[2])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
top_features = pd.DataFrame({'feature_imp':
```

```python
    feature_importance,
                            'features': cols}, columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold')
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()


#    ROC-AUC Curve for all three categories
visualizer = ROCAUC(logit, classes=["0", "1", "2"], macro=False, micro=False)

visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()


#    Polynomial Degree and Error Plots
c_range = [0.001, 0.1, 1, 10, 100, 1000]
degree_range = [2]

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, y, 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 = metrics.confusion_matrix(y_test, y_pred)
        f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx[1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])

        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(" Classification Report\n", classification_report(y_test, y_pred))
        print("\n")

    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()


#    k-NN Classifier
nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
x_train_nn, x_test_nn, y_train_nn, y_test_nn = train_test_split(X, y, 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_nn, y_train_nn)
    y_pred_nn = knn_model.predict(x_test_nn)
    print("NN = ", nn)
    print('Train accuracy score:',knn_model.score(x_train_nn, y_train_nn))
    print('Test accuracy score:',knn_model.score(x_test_nn, y_test_nn))

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

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()


#    ROC-AUC Curve
visualizer = ROCAUC(knn_model, classes=["0", "1", "2"], macro=False, micro=False)

visualizer.fit(x_train_nn, y_train_nn)
visualizer.score(x_test_nn, y_test_nn)
visualizer.show()


#   Review Scores Value
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', '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[['review_scores_value']]
y = (y/y.max())*100

y = y.assign(
    rating_bin_ep = pd.qcut(
        y['review_scores_value'],
```

```
        q=3,
        duplicates='drop',
        labels=[0,1,2]
    )
)


#    Min Max of Each Bin
y.groupby('rating_bin_ep').min()
y.groupby('rating_bin_ep').max()


#    Splitting Data in 75-25 Ratio
y = y['rating_bin_ep']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25)


#    Number of Records in Each Bin
cnt_plt = sns.countplot(y)
cnt_plt.bar_label(cnt_plt.containers[0])
plt.show()


#    Logistic Regression - Varied C Range, using '
    newton-cg' solver and multi_class='multinomial'
c_range = [0.001, 0.1, 1, 10, 100, 1000]
mean_error = []
std_error = []
for c in sorted(c_range):
    logit = LogisticRegression(C=c, random_state=0,
    solver='newton-cg',multi_class='multinomial')
    logit.fit(X_train, y_train)
    y_pred = logit.predict(X_test)
    print("C = ",c)
    print('Train accuracy score:',logit.score(
    X_train, y_train))
    print('Test accuracy score:',logit.score(X_test,
     y_test))
    print("Mean Squared Error: ", mean_squared_error
    (y_test, y_pred))
    scores = cross_val_score(logit, X_test, y_test,
    cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())
plt.errorbar(c_range, mean_error, yerr = std_error,
    linewidth=3)
plt.xlabel('C', fontsize=25)
plt.ylabel('Accuracy Score', fontsize=25)
title_cv = "Logistic Regression - Cross Validation \
    nFor Degree = 1"
plt.title(title_cv, fontsize=25)
plt.show()


#    Feature Importance
cols = X.columns
cols = np.asarray(cols)

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[0])
feature_importance = 100.0 * (feature_importance /
    feature_importance.max())
top_features = pd.DataFrame({'feature_imp':
    feature_importance,
                               'features': cols},
    columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='
    feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.
    feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='
    bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold'
    )
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[1])
feature_importance = 100.0 * (feature_importance /
    feature_importance.max())
top_features = pd.DataFrame({'feature_imp':
    feature_importance,
                               'features': cols},
    columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='
    feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.
    feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='
    bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold'
    )
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()

plt.figure(figsize=(30,15))
feature_importance = abs(logit.coef_[2])
feature_importance = 100.0 * (feature_importance /
    feature_importance.max())
top_features = pd.DataFrame({'feature_imp':
    feature_importance,
                               'features': cols},
    columns=['feature_imp', 'features'])
top_features = top_features.sort_values(by='
    feature_imp', ascending=False).head(20)
plt.bar(top_features.features, top_features.
    feature_imp)
plt.xlabel('Importance', fontsize=35, fontweight='
    bold')
plt.ylabel('Feature', fontsize=35, fontweight='bold'
    )
plt.xticks(fontsize=30, rotation = 90)
plt.yticks(fontsize=30)
plt.show()


#    ROC-AUC Curve for all three categories
visualizer = ROCAUC(logit, classes=["0", "1", "2"],
    macro=False, micro=False)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()

#    Polynomial Degree and Error Plots
c_range = [0.001, 0.1, 1, 10, 100, 1000]
degree_range = [2]

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, y, 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 = metrics.confusion_matrix(y_test,
    y_pred)
```

```
        f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx[1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])

        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(" Classification Report\n", classification_report(y_test, y_pred))
        print("\n")

    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()


#    k-NN Classifier
nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
x_train_nn, x_test_nn, y_train_nn, y_test_nn = train_test_split(X, y, 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_nn, y_train_nn)
    y_pred_nn = knn_model.predict(x_test_nn)
    print("NN = ", nn)
    print('Train accuracy score:',knn_model.score(x_train_nn, y_train_nn))
    print('Test accuracy score:',knn_model.score(x_test_nn, y_test_nn))

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

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()


#    ROC-AUC Curve
visualizer = ROCAUC(knn_model, classes=["0", "1", "2"], macro=False, micro=False)
visualizer.fit(x_train_nn, y_train_nn)
visualizer.score(x_test_nn, y_test_nn)
visualizer.show()
```

```
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)
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()
```

```python
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)
```