

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.

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

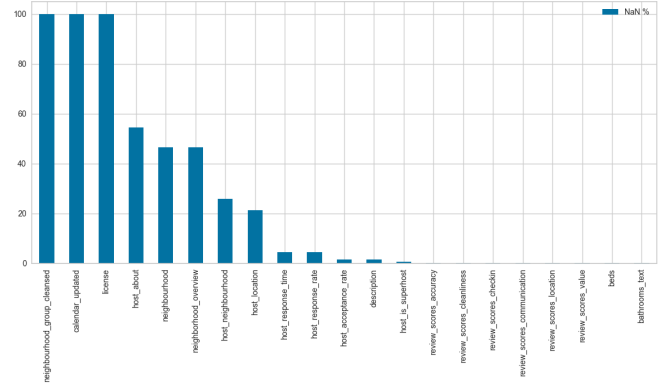


Fig. 1. %NaN values present in each feature columns from Listings dataset

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 cleaned', '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

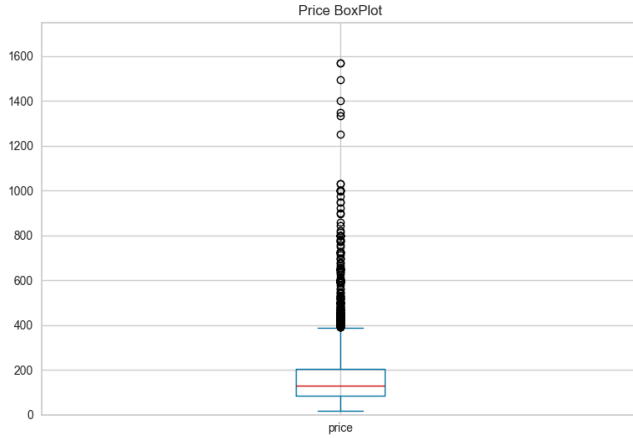


Fig. 2. Plot of Price Feature

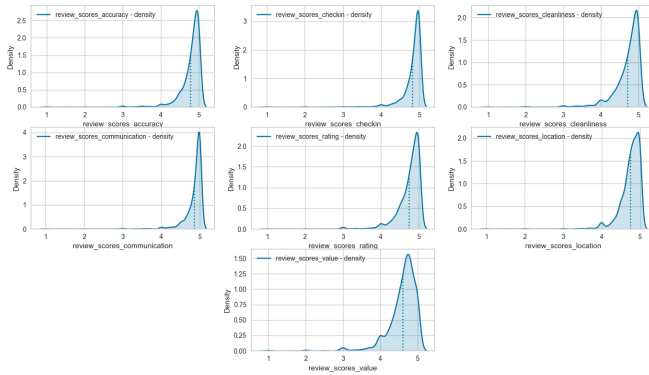


Fig. 3. Distribution Plot of all the Target Columns (Review Rating Columns)

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.

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

$$\text{tf}(t, d) = \frac{\text{terms } t \text{ occur in doc } d}{\text{total terms in } d}$$
$$\text{idf}(t) = 1 + \log\left(\frac{1 + \text{documents in corpus}}{1 + \text{df}(t)}\right)$$

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.

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.

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.

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

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.

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%.

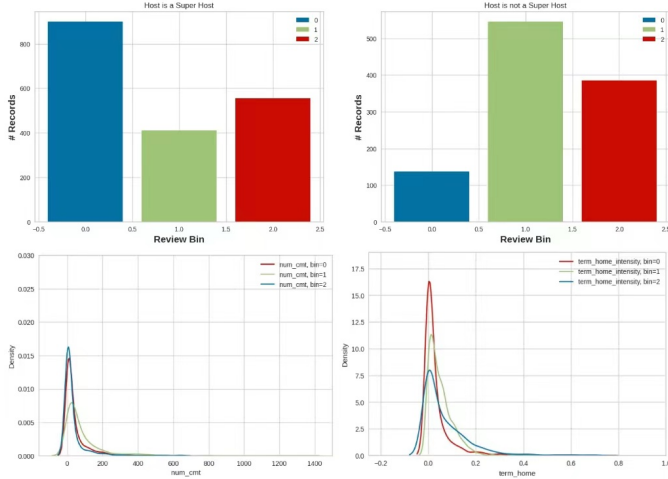


Fig. 6. Analysis Important Features

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

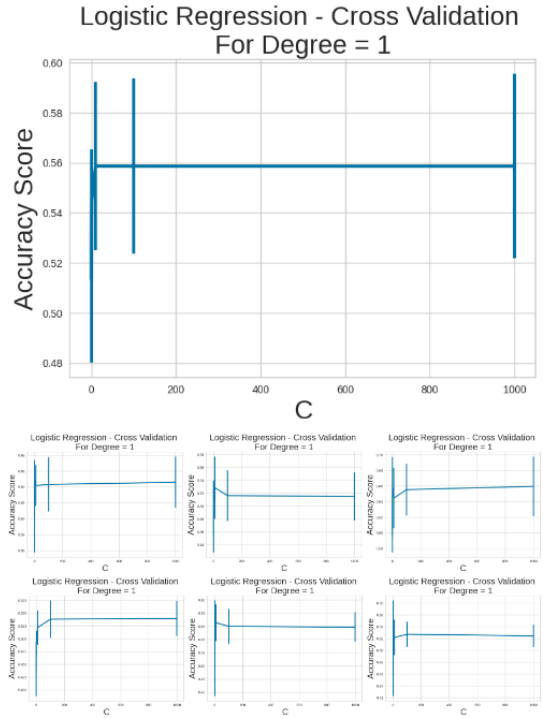


Fig. 7. Cross-Validation Accuracy Scores for Degree 1

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

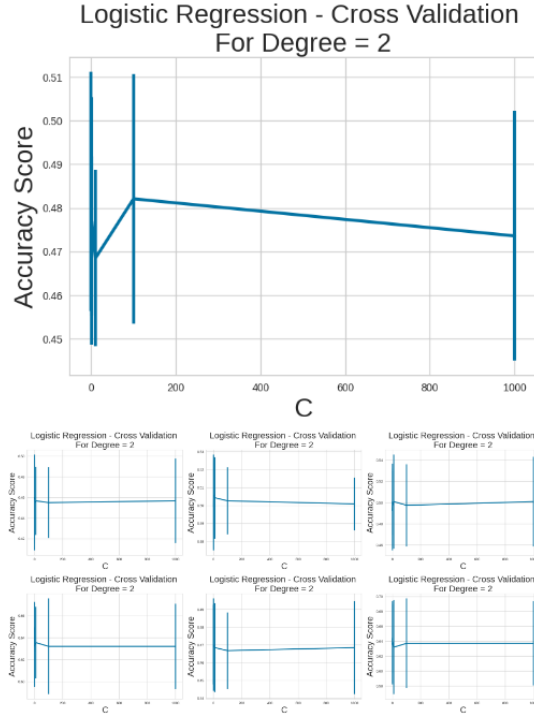


Fig. 8. Cross-Validation Accuracy Scores for Degree 2

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.

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-

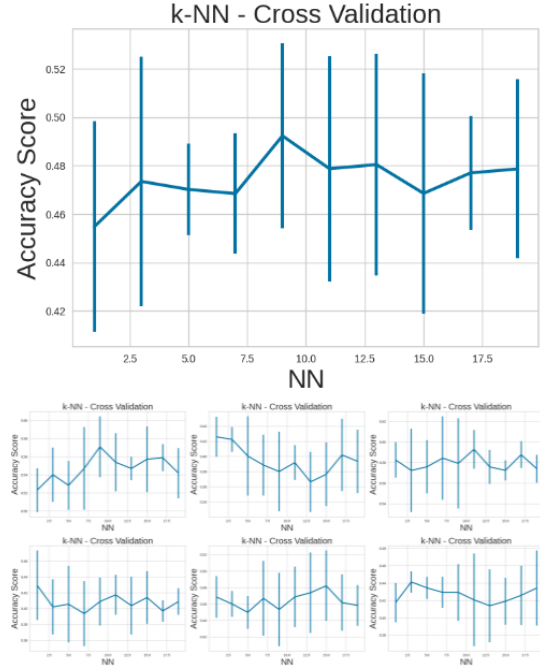


Fig. 9. k-NN Cross Validation for the target variable

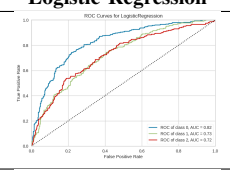
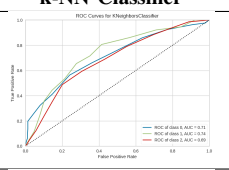
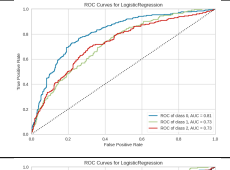
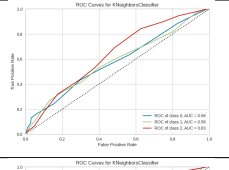
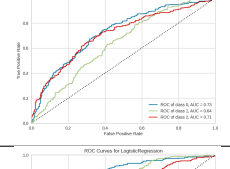
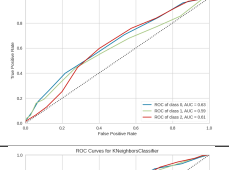
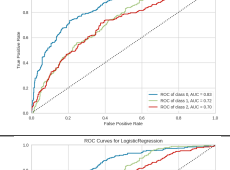
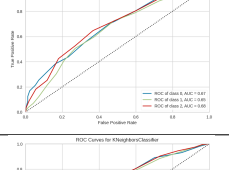
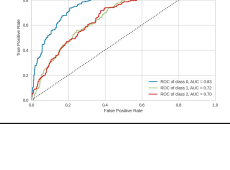
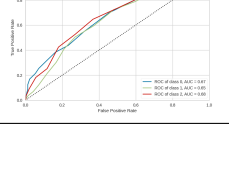
TABLE II
VALUE OF HYPER-PARAMETER K WITH THE CORRESPONDING TRAIN AND TEST ACCURACY FOR ALL THE 7 TARGETS

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

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		

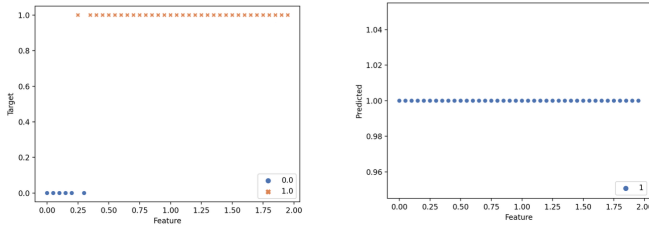


Fig. 10. Plot of Data Imbalance

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 1

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.

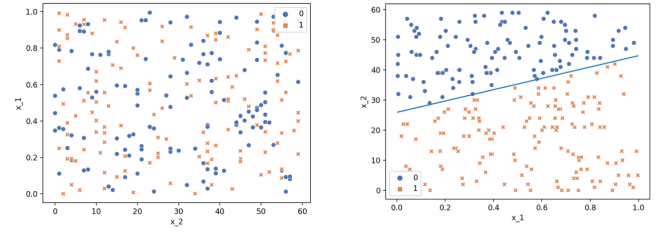


Fig. 11. Two Features and Target Values

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,

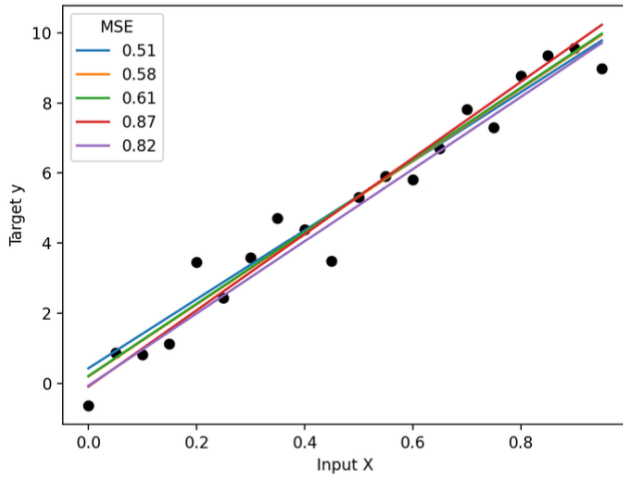


Fig. 12. Random data splits MSE

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 if 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

	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

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.

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

```

1 import warnings
2 warnings.filterwarnings('ignore')
3 import pandas as pd
4 from sklearn.feature_extraction.text import
   TfidfVectorizer
5 from nltk.corpus import stopwords
6 import pandas as pd
7 import numpy as np
8 import seaborn as sns
9 import matplotlib.pyplot as plt
10 import regex
11 import emoji
12 from pycountry import languages
13 import fasttext
14
15 def split_count(info):
16
17     emoji_list = []
18     data = regex.findall(r'\X', info)
19     for word in data:
20         if any(char in emoji.EMOJI_DATA for char in
           word):
21             emoji_list.append(word)
22
23     return len(emoji_list)
24
25
26 reviews_raw = pd.read_csv("reviews.csv")
27 PRETRAINED_MODEL_PATH = 'lid.176.bin'
28 model = fasttext.load_model(PRETRAINED_MODEL_PATH)
29
30 reviews_raw['lang'] = range(0, len(reviews_raw))
31
32 for index, row in reviews_raw.iterrows():
33     if type(row['comments']) == type('s'):
34         predictions = model.predict(row['comments'])
35         l = predictions[0][0].split('_label_')[1]

```

```

36         if l != 'ceb' and l != 'nds' and l != 'war'
37         and l != 'wuu':
38             reviews_raw['lang'][index] = l
39 reviews_raw = reviews_raw[reviews_raw['comments'].
40     notna()]
41 reviews_raw['emoji_count'] = reviews_raw.comments.
42     apply(split_count)
43 reviews_raw['lenstr'] = reviews_raw['comments'].str.
44     len()
45 reviews_raw['test_emoji'] = (((reviews_raw['
46     emoji_count'] == reviews_raw['lenstr']) |
47     (2*reviews_raw['
48     emoji_count'] == reviews_raw['lenstr'])) & (
49     reviews_raw['emoji_count']!=0))
50 reviews_raw = reviews_raw[reviews_raw.test_emoji ==
51     False]
52 reviews_raw = reviews_raw.drop(columns='test_emoji')
53 reviews_raw = reviews_raw.drop(columns='lenstr')
54 reviews_raw = reviews_raw.drop(columns='emoji_count'
55     )
56 reviews_raw.comments = reviews_raw.comments.apply(
57     lambda x: x.encode('ascii', 'ignore').decode('
58     ascii'))
59 reviews_raw['emoji_count'] = reviews_raw.comments.
60     apply(split_count)
61 reviews_raw = reviews_raw[(reviews_raw.comments.str.
62     len() > 20)]
63 reviews_raw['comments'] = reviews_raw['comments'].
64     str.replace('<br/>', '')
65 reviews_raw = reviews_raw[reviews_raw.lang == '_en']
66 reviews_raw.to_csv("reviews_processed.csv")
67 reviews = pd.read_csv("reviews_processed.csv",
68     lineterminator='\n')
69 print("Mean: ", reviews['comments'].str.split().str.
70     len().mean())
71 print("Median: ", reviews['comments'].str.split().
72     str.len().median())
73 i = sns.kdeplot(reviews['comments'].str.split().str.
74     len(),color="b", label='Words in each Comment')
75 xi = i.lines[0].get_xdata()
76 yi = i.lines[0].get_ydata()
77 meani = reviews['comments'].str.split().str.len().
78     mean()
79 mediani = reviews['comments'].str.split().str.len().
80     median()
81 heighti = np.interp(meani, xi, yi)
82 heighti2 = np.interp(mediani, xi, yi)
83 i.vlines(meani, 0, heighti, color='r', ls=':', label
84     ='Mean')
85 i.vlines(mediani, 0, heighti2, color='g', ls=':',
86     label='Median')
87 plt.legend()
88 plt.show()
89 stop = stopwords.words('english')
90 reviews['comments_stp_rem'] = reviews['comments'].
91     apply(lambda x: ' '.join([word.lower() for word
92     in x.split()
93         if word not in (stop)]))
94
95 tr_idf_model = TfidfVectorizer(analyzer = 'word',
96     max_features=33)
97 tf_idf_vector = tr_idf_model.fit_transform(reviews.
98     comments_stp_rem)
99 tf_idf_array = tf_idf_vector.toarray()
100 words_set = tr_idf_model.get_feature_names_out()
101 df_tf_idf = pd.DataFrame(tf_idf_array, columns =
102     words_set)
103 df_tf_idf['listing_id'] = reviews.listing_id
104 test = df_tf_idf.groupby('listing_id').size()
105 review_1 = pd.DataFrame({'listing_id':test.index, '
106     num_cmt':test.values})
107 test2 = pd.DataFrame([])
108 for i in words_set:
109     test2[f'term_{i}'] = df_tf_idf.groupby(['
110     listing_id'])[f'{i}'].mean()
111 review_final = review_1.merge(test2, on='listing_id'
112     )
113 review_final.to_csv("reviews_final.csv")
114
115 import warnings
116 warnings.filterwarnings('ignore')
117
118 import pandas as pd
119 import numpy as np
120 import seaborn as sns
121 import matplotlib.pyplot as plt
122 from sklearn import preprocessing
123 from sklearn.linear_model import Lasso,
124     LogisticRegression, Ridge
125 from sklearn.model_selection import train_test_split
126 from sklearn.metrics import mean_squared_error
127 from sklearn.preprocessing import MinMaxScaler
128 from sklearn.metrics import mean_squared_error,
129     accuracy_score, log_loss
130 from sklearn.dummy import DummyRegressor,
131     DummyClassifier
132 from yellowbrick.classifier import ROCAUC
133 from sklearn.metrics import confusion_matrix,
134     precision_score, recall_score, auc
135 from sklearn.metrics import f1_score,
136     classification_report, roc_curve
137 from sklearn.neighbors import KNeighborsClassifier
138 from sklearn.preprocessing import PolynomialFeatures
139 from sklearn import metrics
140 from sklearn.model_selection import cross_val_score
141
142 # Data Loading - Listings and Reviews
143
144 listings = pd.read_csv("listings.csv")
145 reviews = pd.read_csv("reviews_final.csv")
146
147 # Splitting the amenities and logically clustering
148     them into 9 different categories
149
150 amenities = listings['amenities'].str.split(',',
151     expand=True)
152 amenities = amenities.loc[amenities[77].notnull()]
153 amenities_list = amenities.iloc[0]
154 amenities_list = amenities_list.to_list()
155
156 listings.loc[listings['amenities'].str.contains('Hot
157     Water|Shower gel|Hair dryer|Bathtub|Shampoo|
158     Essentials|Bidet|Conditioner|Body soap|Baby bath
159     '),
160     'bath-products'] = 1

```



```

39 listings.loc[listings['amenities'].str.contains('
    Bluetooth sound system|Ethernet connection|
    Heating|Pocket wifi|Cable TV|Wifi'),
40             'electric-system'] = 1
41 listings.loc[listings['amenities'].str.contains('
    Breakfast'),
42             'food-services'] = 1
43 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'),
44             'house-furniture'] = 1
45 listings.loc[listings['amenities'].str.contains('
    Cleaning before checkout|Luggage dropoff allowed
    |Long term stays allowed'),
46             'house-rules'] = 1
47 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'),
48             'kitchen-appliances'] = 1
49 listings.loc[listings['amenities'].str.contains('
    Free parking on premises|Free street parking'),
50             'parking'] = 1
51 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'),
52             'recreation'] = 1
53 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'),
54             'safety'] = 1
55
56
57 # Splitting the host contact details into 3
    categories namely host_email, host_phone,
    host_work and host_work_email
58 host_verification = listings['host_verifications'].
    str.split(',', expand=True)
59
60 listings.loc[listings['host_verifications'].str.
    contains('email'),
61             'host_email'] = 1
62 listings.loc[listings['host_verifications'].str.
    contains('phone'),
63             'host_phone'] = 1
64 listings.loc[listings['host_verifications'].str.
    contains('work_email'),
65             'host_work_email'] = 1
66
67
68 # Handling of NaN values of New Feature Columns
    Created
69 new_feature_cols = listings.iloc[:,75:].columns
70 listings[new_feature_cols] = listings[
    new_feature_cols].fillna(0)
71
72
73 # Merging the Listings data with the Cleansed
    Reviews Dataset
74 listings = listings.merge(reviews, how='inner',
    left_on='id', right_on='listing_id')
75
76
77 # Dropping the columns which are not required
    while building a Machine Learning Model
78 # Plot %NaN Values
79 def plot_nas(df: pd.DataFrame):
80     if df.isnull().sum().sum() != 0:
81         na_df = (df.isnull().sum() / len(df)) * 100
82         na_df = na_df.drop(na_df[na_df == 0].index).
            sort_values(ascending=False)
83         missing_data = pd.DataFrame({'NaN %': na_df
            })
84         missing_data.plot(kind = "bar")
85         plt.show()
86     else:
87         print('No NAs found')
88 plot_nas(listings)
89
90
91 # Column Drop List
92 not_needed_columns = [
93     'id', 'listing_url', 'scrape_id', 'last_scraped',
94     'source', 'name',
95     'picture_url', 'host_id', 'host_url', 'host_name',
96     'host_location',
97     'host_about', 'host_thumbnail_url', '
98     host_picture_url', 'host_neighbourhood',
99     'neighbourhood', 'neighbourhood_group_cleansed',
100    'calendar_updated', 'first_review', 'last_review',
101    'license',
102    'calculated_host_listings_count', '
103    calculated_host_listings_count_entire_homes',
104    'calculated_host_listings_count_private_rooms',
105    'calculated_host_listings_count_shared_rooms',
106    'description', 'neighborhood_overview', '
107    host_verifications', 'host_since',
108    'bathrooms', 'bathrooms_text', 'amenities', '
109    availability_30',
110    'availability_60', 'availability_90', '
111    availability_365', 'calendar_last_scraped',
112    'number_of_reviews_ltm', 'number_of_reviews_l30d',
113    'host_has_profile_pic', 'property_type',
114    'minimum_minimum_nights', '
115    maximum_maximum_nights', 'minimum_nights_avg_ntm',
116    '
117    minimum_maximum_nights', '
118    maximum_minimum_nights', 'maximum_nights_avg_ntm',
119    '
120    ]
121 listings.drop(not_needed_columns, axis = 1, inplace
    = True)
122 listings = listings.dropna()
123
124 # Missing Data Imputation
125 imputation_cols = ['bedrooms', 'beds']
126 for i in imputation_cols:
127     listings.loc[listings.loc[:,i].isnull(),i] =
        listings.loc[:,i].median()
128
129
130 # Pre-Processing of Features 'price', '
    host_response_rate' and 'host_acceptance_rate'
131 listings['price'] = listings['price'].str.replace('$', '')
132 listings['price'] = listings['price'].str.replace(',', '')
133 listings['price'] = pd.to_numeric(listings['price'])
134 listings['host_response_rate'] = listings["
    host_response_rate"].str.replace("%", "")
135 listings['host_response_rate'] = pd.to_numeric(
    listings['host_response_rate'])
136 listings['host_acceptance_rate'] = listings["
    host_acceptance_rate"].str.replace("%", "")
137 listings['host_acceptance_rate'] = pd.to_numeric(
    listings['host_acceptance_rate'])

```

```

126
127 listings[['price']].plot(kind='box', title='Price
128     BoxPlot')
129 plt.ylim(0,1750)
130
131 # Outlier Removal for the Feature 'price'
132 listings = listings[listings.price > 50]
133 listings = listings[listings.price <= 300]
134 listings[['price']].plot(kind='box', title='Price
135     BoxPlot')
136
137 # Neighbourhood Analysis
138 listings.neighbourhood_cleaned.unique()
139 listings.groupby('neighbourhood_cleaned').
140     host_response_time.count()
141 neighbourhood_DF=listings.groupby('
142     neighbourhood_cleaned').host_response_time.
143     count()
144 neighbourhood_DF=neighbourhood_DF.reset_index()
145 neighbourhood_DF=neighbourhood_DF.rename(columns={'
146     host_response_time': 'Number_Of_Listings'})
147 neighbourhood_DF.plot(kind='bar',
148     x='neighbourhood_cleaned',
149     y='Number_Of_Listings',
150     figsize=(18,8),
151     title = 'Dublin Neighborhood Frequency',
152     legend = False)
153
154 # Statistical Analysis
155 print(listings.review_scores_accuracy.mean())
156 print(listings.review_scores_checkin.mean())
157 print(listings.review_scores_cleanliness.mean())
158 print(listings.review_scores_communication.mean())
159 print(listings.review_scores_location.mean())
160 print(listings.review_scores_rating.mean())
161 print(listings.review_scores_value.mean())
162
163 print(listings.review_scores_accuracy.median())
164 print(listings.review_scores_checkin.median())
165 print(listings.review_scores_cleanliness.median())
166 print(listings.review_scores_communication.median())
167 print(listings.review_scores_location.median())
168 print(listings.review_scores_rating.median())
169 print(listings.review_scores_value.median())
170
171 print(listings.review_scores_accuracy.mode())
172 print(listings.review_scores_checkin.mode())
173 print(listings.review_scores_cleanliness.mode())
174 print(listings.review_scores_communication.mode())
175 print(listings.review_scores_location.mode())
176 print(listings.review_scores_rating.mode())
177 print(listings.review_scores_value.mode())
178
179 print(listings.review_scores_accuracy.min())
180 print(listings.review_scores_checkin.min())
181 print(listings.review_scores_cleanliness.min())
182 print(listings.review_scores_communication.min())
183 print(listings.review_scores_location.min())
184 print(listings.review_scores_rating.min())
185 print(listings.review_scores_value.min())
186
187 print(listings.review_scores_accuracy.max())
188 print(listings.review_scores_checkin.max())
189 print(listings.review_scores_cleanliness.max())
190 print(listings.review_scores_communication.max())
191 print(listings.review_scores_location.max())
192 print(listings.review_scores_rating.max())
193 print(listings.review_scores_value.max())
194
195 # Density Plots for all 7 Target Variables (Review
196
197 Scores)
198 plt.figure(figsize=(18, 18))
199 plt.subplot(3, 3, 1)
200 review_scores_accuracy_density = sns.kdeplot(
201     listings.review_scores_accuracy, color="b",
202     label='
203     review_scores_accuracy - density')
204 x_review_scores_accuracy =
205     review_scores_accuracy_density.lines[0].
206     get_xdata()
207 y_review_scores_accuracy =
208     review_scores_accuracy_density.lines[0].
209     get_ydata()
210 mean_review_scores_accuracy_density = listings.
211     review_scores_accuracy.mean()
212 height_review_scores_accuracy = np.interp(
213     mean_review_scores_accuracy_density,
214     x_review_scores_accuracy,
215     y_review_scores_accuracy)
216 review_scores_accuracy_density.vlines(
217     mean_review_scores_accuracy_density, 0,
218     height_review_scores_accuracy,
219     color='b', ls=
220     ':')
221 review_scores_accuracy_density.fill_between(
222     x_review_scores_accuracy, 0,
223     y_review_scores_accuracy,
224     facecolor='b', alpha=0.2)
225 plt.legend()
226
227 plt.subplot(3, 3, 2)
228 review_scores_checkin_density = sns.kdeplot(listings
229     .review_scores_checkin, color="b",
230     label='
231     review_scores_checkin - density')
232 x_review_scores_checkin =
233     review_scores_checkin_density.lines[0].get_xdata
234     ()
235 y_review_scores_checkin =
236     review_scores_checkin_density.lines[0].get_ydata
237     ()
238 mean_review_scores_checkin_density = listings.
239     review_scores_checkin.mean()
240 height_review_scores_checkin = np.interp(
241     mean_review_scores_checkin_density,
242     x_review_scores_checkin,
243     y_review_scores_checkin)
244 review_scores_checkin_density.vlines(
245     mean_review_scores_checkin_density, 0,
246     height_review_scores_checkin,
247     color='b', ls=
248     ':')
249 review_scores_checkin_density.fill_between(
250     x_review_scores_checkin, 0,
251     y_review_scores_checkin,
252     facecolor='b', alpha=0.2)
253 plt.legend()
254
255 plt.subplot(3, 3, 3)
256 review_scores_cleanliness_density = sns.kdeplot(
257     listings.review_scores_cleanliness, color="b",
258     label='
259     review_scores_cleanliness - density')
260 x_review_scores_cleanliness =
261     review_scores_cleanliness_density.lines[0].
262     get_xdata()
263 y_review_scores_cleanliness =

```

```

        review_scores_cleanliness_density.lines[0].
        get_ydata()
231 mean_review_scores_cleanliness_density = listings.
        review_scores_cleanliness.mean()
232 height_review_scores_cleanliness = np.interp(
        mean_review_scores_cleanliness_density,
        x_review_scores_cleanliness,
233
        y_review_scores_cleanliness)
234 review_scores_cleanliness_density.vlines(
        mean_review_scores_cleanliness_density, 0,
        height_review_scores_cleanliness,
235
        color='b', ls=
        ':')
236 review_scores_cleanliness_density.fill_between(
        x_review_scores_cleanliness, 0,
        y_review_scores_cleanliness,
237
        facecolor='b', alpha=0.2)
238 plt.legend()
239
240 plt.subplot(3, 3, 4)
241 review_scores_communication_density = sns.kdeplot(
        listings.review_scores_communication, color="b",
242
        label='
        review_scores_communication - density')
243 x_review_scores_communication =
        review_scores_communication_density.lines[0].
        get_xdata()
244 y_review_scores_communication =
        review_scores_communication_density.lines[0].
        get_ydata()
245 mean_review_scores_communication_density = listings.
        review_scores_communication.mean()
246 height_review_scores_communication = np.interp(
        mean_review_scores_communication_density,
        x_review_scores_communication,
247
        y_review_scores_communication)
248 review_scores_communication_density.vlines(
        mean_review_scores_communication_density, 0,
        height_review_scores_communication,
249
        color='b', ls=
        ':')
250 review_scores_communication_density.fill_between(
        x_review_scores_communication, 0,
        y_review_scores_communication,
251
        facecolor='b', alpha=0.2)
252 plt.legend()
253
254 plt.subplot(3, 3, 5)
255 review_scores_rating_density = sns.kdeplot(listings.
        review_scores_rating, color="b",
256
        label='
        review_scores_rating - density')
257 x_review_scores_rating =
        review_scores_rating_density.lines[0].get_xdata
        ()
258 y_review_scores_rating =
        review_scores_rating_density.lines[0].get_ydata
        ()
259 mean_review_scores_rating_density = listings.
        review_scores_rating.mean()
260 height_review_scores_rating = np.interp(
        mean_review_scores_rating_density,
        x_review_scores_rating,
261
        y_review_scores_rating)
262 review_scores_rating_density.vlines(
        mean_review_scores_rating_density, 0,
        height_review_scores_rating,
263
        color='b', ls=
        ':')
264 review_scores_rating_density.fill_between(
        x_review_scores_rating, 0,
        y_review_scores_rating,
265
        facecolor='b', alpha=0.2)
266 plt.legend()
267
268 plt.subplot(3, 3, 6)
269 review_scores_location_density = sns.kdeplot(
        listings.review_scores_location, color="b",
270
        label='
        review_scores_location - density')
271 x_review_scores_location =
        review_scores_location_density.lines[0].
        get_xdata()
272 y_review_scores_location =
        review_scores_location_density.lines[0].
        get_ydata()
273 mean_review_scores_location_density = listings.
        review_scores_location.mean()
274 height_review_scores_location = np.interp(
        mean_review_scores_location_density,
        x_review_scores_location,
275
        y_review_scores_location)
276 review_scores_location_density.vlines(
        mean_review_scores_location_density, 0,
        height_review_scores_location,
277
        color='b', ls=
        ':')
278 review_scores_location_density.fill_between(
        x_review_scores_location, 0,
        y_review_scores_location,
279
        facecolor='b', alpha=0.2)
280 plt.legend()
281
282 plt.subplot(3, 3, 8)
283 review_scores_value_density = sns.kdeplot(listings.
        review_scores_value, color="b",
284
        label='
        review_scores_value - density')
285 x_review_scores_value = review_scores_value_density.
        lines[0].get_xdata()
286 y_review_scores_value = review_scores_value_density.
        lines[0].get_ydata()
287 mean_review_scores_value_density = listings.
        review_scores_value.mean()
288 height_review_scores_value = np.interp(
        mean_review_scores_value_density,
        x_review_scores_value,
289
        y_review_scores_value)
290 review_scores_value_density.vlines(
        mean_review_scores_value_density, 0,
        height_review_scores_value,
291
        color='b', ls=
        ':')
292 review_scores_value_density.fill_between(
        x_review_scores_value, 0, y_review_scores_value,
293
        facecolor='b', alpha=0.2)
294 plt.legend()
295
296 plt.show()
297
298
299
300 # Data Scaling
301 # Min Max Scaling
302 def minmax(X):
303     X_std = (X - X.min()) / (X.max() - X.min())
304     X_scaled = X_std * (X.max() - X.min()) + X.min()

```

```

306     return X_scaled
307
308
309 # List of Columns to be Scaled
310 scaling_data = ['host_response_rate', '
311     host_acceptance_rate', 'bedrooms', 'beds',
312     'host_listings_count', '
313     host_total_listings_count',
314     'latitude', 'longitude', '
315     accommodates', 'price', '
316     minimum_nights', 'maximum_nights',
317     'number_of_reviews', 'num_cmt', 'avg_senti',
318     'review_scores_rating', '
319     review_scores_accuracy',
320     'review_scores_cleanliness', '
321     review_scores_checkin',
322     'review_scores_communication', '
323     review_scores_location',
324     'review_scores_value', '
325     reviews_per_month', 'bath-products', 'electric-
326     system',
327     'food-services', 'house-furniture', '
328     house-rules',
329     'kitchen-appliances', 'parking', '
330     recreation', 'safety',
331     'host_email', 'host_work_email']
332
333 for i in scaling_data:
334     listings[i] = minmax(listings[i])
335
336 # Label Encoding
337 listings.dropna(axis = 1, inplace = True)
338 label_encoder = preprocessing.LabelEncoder()
339 listings.host_response_time = label_encoder.
340     fit_transform(listings.host_response_time)
341 listings.host_is_superhost = label_encoder.
342     fit_transform(listings.host_is_superhost)
343 listings.host_identity_verified = label_encoder.
344     fit_transform(listings.host_identity_verified)
345 listings.instant_bookable = label_encoder.
346     fit_transform(listings.instant_bookable)
347 listings.room_type = label_encoder.
348     fit_transform(listings.room_type)
349 listings.neighbourhood_cleansed = label_encoder.
350     fit_transform(listings.neighbourhood_cleansed)
351 listings.has_availability = label_encoder.
352     fit_transform(listings.has_availability)
353
354 # Correlation Matrix
355 test_corr = listings.corr()
356 test_corr.to_csv("test_corr.csv")
357
358 # Metrics Print Function
359 def print_metrics(y_test, y_pred):
360     print("-"*10+"CONFUSION-MATRIX"+"-"*10)
361     print(confusion_matrix(y_test, y_pred))
362
363     print("-"*10+"CLASSIFICATION-REPORT"+"-"*10)
364     print(classification_report(y_test, y_pred))
365
366 # Review Scores Accuracy
367 # Logistic Regression
368 # Defining the Input Variables
369 X = listings[
370     ['host_response_time', '
371     host_response_rate', 'host_acceptance_rate',
372     'bedrooms', 'beds', '
373     neighbourhood_cleansed', '
374     host_is_superhost', '
375     host_listings_count', 'host_total_listings_count',
376     '
377     'host_identity_verified', '
378     room_type',
379     'accommodates', 'price', '
380     minimum_nights', 'maximum_nights',
381     'bath-products', 'electric-system',
382     'food-services', 'house-furniture', '
383     house-rules',
384     'kitchen-appliances', 'parking', '
385     recreation', 'safety',
386     'host_email', 'host_work_email'] +
387     list(reviews.columns[2:])
388 ]
389
390 # Defining the Quantile Bins for the Target
391 Variables
392 y = listings[['review_scores_accuracy']]
393 y = (y/y.max())*100
394
395 y = y.assign(
396     rating_bin_ep = pd.qcut(
397         y['review_scores_accuracy'],
398         q=3,
399         duplicates='drop',
400         labels=[0,1,2]
401     )
402 )
403
404 # Min Max of Each Bin
405 y.groupby('rating_bin_ep').min()
406 y.groupby('rating_bin_ep').max()
407 y = y['rating_bin_ep']
408 X_train, X_test, y_train, y_test = train_test_split(
409     X, y, test_size=0.25)
410
411 # Number of Records in Each Bin
412 cnt_plt = sns.countplot(y)
413 cnt_plt.bar_label(cnt_plt.containers[0])
414 plt.show()
415
416 # Logistic Regression - Varied C Range, using '
417 newton-cg' solver and multi_class='multinomial'
418 c_range = [0.001, 0.1, 1, 10, 100, 1000]
419 mean_error = []
420 std_error = []
421 for c in sorted(c_range):
422     logit = LogisticRegression(C=c, random_state=0,
423         solver='newton-cg', multi_class='multinomial')
424     logit.fit(X_train, y_train)
425     y_pred = logit.predict(X_test)
426     print("C = ", c)
427     print('Train accuracy score:', logit.score(
428         X_train, y_train))
429     print('Test accuracy score:', logit.score(X_test,
430         y_test))
431     print("Mean Squared Error: ", mean_squared_error(
432         y_test, y_pred))
433     scores = cross_val_score(logit, X_test, y_test,
434         cv=5, scoring='accuracy')
435     mean_error.append(np.array(scores).mean())
436     std_error.append(np.array(scores).std())
437     plt.errorbar(c_range, mean_error, yerr = std_error,
438         linewidth=3)
439     plt.xlabel('C', fontsize=25)
440     plt.ylabel('Accuracy Score', fontsize=25)
441     title_cv = "Logistic Regression - Cross Validation \
442         nFor Degree = 1"
443     plt.title(title_cv, fontsize=25)
444     plt.show()
445
446 # ROC-AUC Curve for all three categories

```

```

417 visualizer = ROCAUC(logit, classes=["0", "1", "2"],
418     macro=False, micro=False)
419 visualizer.fit(X_train, y_train)
420 visualizer.score(X_test, y_test)
421 visualizer.show()
422
423 # Feature Importance
424 from matplotlib import pyplot
425 cols = X.columns
426 cols = np.asarray(cols)
427
428 plt.figure(figsize=(30,15))
429 feature_importance = abs(logit.coef_[0])
430 feature_importance = 100.0 * (feature_importance /
431     feature_importance.max())
432 top_features = pd.DataFrame({'feature_imp':
433     feature_importance,
434     'features': cols},
435     columns=['feature_imp', 'features'])
436 top_features = top_features.sort_values(by='
437     feature_imp', ascending=False).head(20)
438 plt.bar(top_features.features, top_features.
439     feature_imp)
440 plt.xlabel('Importance', fontsize=35, fontweight='
441     bold')
442 plt.ylabel('Feature', fontsize=35, fontweight='bold'
443     )
444 plt.xticks(fontsize=30, rotation = 90)
445 plt.yticks(fontsize=30)
446 plt.show()
447
448 plt.figure(figsize=(30,15))
449 feature_importance = abs(logit.coef_[1])
450 feature_importance = 100.0 * (feature_importance /
451     feature_importance.max())
452 top_features = pd.DataFrame({'feature_imp':
453     feature_importance,
454     'features': cols},
455     columns=['feature_imp', 'features'])
456 top_features = top_features.sort_values(by='
457     feature_imp', ascending=False).head(20)
458 plt.bar(top_features.features, top_features.
459     feature_imp)
460 plt.xlabel('Importance', fontsize=35, fontweight='
461     bold')
462 plt.ylabel('Feature', fontsize=35, fontweight='bold'
463     )
464 plt.xticks(fontsize=30, rotation = 90)
465 plt.yticks(fontsize=30)
466 plt.show()
467
468 # Comparison With Baseline Classifier
469 dmfr = DummyClassifier(strategy='most_frequent').fit
470     (X_train, y_train)
471 dmun = DummyClassifier(strategy='uniform').fit(
472     X_train, y_train)
473
474 print("\n\nDUMMY CLASSIFIER - frequent")
475 print(dmfr.score(X_test, y_test))
476 y_pred = dmfr.predict(X_test)
477 print_metrics(y_test, y_pred)
478
479 print("\n\nDUMMY CLASSIFIER - Uniform")
480 print(dmun.score(X_test, y_test))
481 y_pred = dmun.predict(X_test)
482 print_metrics(y_test, y_pred)
483
484 # Density Plot for Term 'Home' for each Review
485 Bin
486 test = pd.concat([X, y], axis=1)
487 g = sns.kdeplot(test.loc[test['rating_bin_ep'] == 0,
488     'term_home'], color="r", label='
489     term_home_intensity, bin=0')
490 g.set(ylim=(0, 19))
491 g.set(xlim=(-0.25, 1))
492 plt.legend()
493
494 h = sns.kdeplot(test.loc[test['rating_bin_ep'] == 1,
495     'term_home'], color="g", label='
496     term_home_intensity, bin=1')
497 h.set(ylim=(0, 19))
498 h.set(xlim=(-0.25, 1))
499 plt.legend()
500
501 i = sns.kdeplot(test.loc[test['rating_bin_ep'] == 2,
502     'term_home'], color="b", label='
503     term_home_intensity, bin=2')
504 i.set(ylim=(0, 19))
505 i.set(xlim=(-0.25, 1))
506 plt.legend()
507
508 plt.show()
509
510 # Density Plots for Number of Comments against
511 each Review Bin
512 test = pd.concat([X, y], axis=1)
513
514 g = sns.kdeplot(test.loc[test['rating_bin_ep'] == 0,
515     'num_cmt'], color="r", label='num_cmt, bin=0')
516 g.set(ylim=(0, 0.03))
517 plt.legend()
518
519 h = sns.kdeplot(test.loc[test['rating_bin_ep'] == 1,
520     'num_cmt'], color="g", label='num_cmt, bin=1')
521 h.set(ylim=(0, 0.03))
522 plt.legend()
523
524 i = sns.kdeplot(test.loc[test['rating_bin_ep'] == 2,
525     'num_cmt'], color="b", label='num_cmt, bin=2')
526 i.set(ylim=(0, 0.03))
527 plt.legend()
528
529 plt.show()
530
531 # Number of Records in each Review Bin for the
532 type of Host (Superhost)
533 test.host_is_superhost.value_counts()
534 test1 = test.groupby(['host_is_superhost', '
535     rating_bin_ep']).size()
536
537 plt.figure(figsize=(18, 6))
538 plt.subplot(1, 2, 1)
539 for j in (0,1,2):

```



```

528 plt.bar(j, test1[0][j], label = str(j))
529 plt.title("Host is a Super Host")
530 plt.legend()
531 plt.xlabel('Review Bin', fontweight = 'bold',
532           fontsize = 15)
533 plt.ylabel('# Records', fontweight = 'bold', fontsize
534           = 15)
535 plt.subplot(1, 2, 2)
536 for j in (0,1,2):
537     plt.bar(j, test1[1][j], label = str(j))
538     plt.title("Host is not a Super Host")
539     plt.legend()
540     plt.xlabel('Review Bin', fontweight = 'bold',
541               fontsize = 15)
542     plt.ylabel('# Records', fontweight = 'bold', fontsize
543               = 15)
544     plt.show()
545
546 # Neighbourhood Type Analysis
547 test.neighbourhood_cleansed.value_counts()
548 test2 = test.groupby(['neighbourhood_cleansed', '
549                      rating_bin_ep']).size()
550
551 plt.figure(figsize=(18, 6))
552 plt.subplot(1, 2, 1)
553 for j in (0,1,2):
554     plt.bar(j, test2[0][j], label = str(j))
555     plt.title("Neighbourhood 0", fontsize=25)
556     plt.legend()
557     plt.xlabel('Review Bin', fontweight = 'bold',
558               fontsize = 15)
559     plt.ylabel('# Records', fontweight = 'bold', fontsize
560               = 15)
561     plt.subplot(1, 2, 2)
562     for j in (0,1,2):
563         plt.bar(j, test2[1][j], label = str(j))
564         plt.title("Neighbourhood 1", fontsize=25)
565         plt.legend()
566         plt.xlabel('Review Bin', fontweight = 'bold',
567               fontsize = 15)
568         plt.ylabel('# Records', fontweight = 'bold', fontsize
569               = 15)
570         plt.show()
571
572 plt.figure(figsize=(18, 6))
573 plt.subplot(1, 2, 1)
574 for j in (0,1,2):
575     plt.bar(j, test2[2][j], label = str(j))
576     plt.title("Neighbourhood 2", fontsize=25)
577     plt.legend()
578     plt.xlabel('Review Bin', fontweight = 'bold',
579               fontsize = 15)
580     plt.ylabel('# Records', fontweight = 'bold', fontsize
581               = 15)
582     plt.show()
583
584 # Polynomial Degree and Error Plots
585 c_range = [0.001, 0.1, 1, 10, 100, 1000]
586 degree_range = [2]
587
588 for i in degree_range:
589     trans = PolynomialFeatures(degree = i)
590     x_poly = trans.fit_transform(X)
591     x_train, x_test, y_train, y_test =
592     train_test_split(x_poly, y, test_size = 0.2,
593                     random_state=(1))
594     mean_error = []
595     std_error = []
596     for c in c_range:
597         log_reg = LogisticRegression(C = c,
598                                     random_state=0, solver='newton-cg', multi_class='
599 multinomial')
600         log_reg.fit(x_train, y_train)
601         y_pred = log_reg.predict(x_test)
602
603         cnf_mtx = metrics.confusion_matrix(y_test,
604                                           y_pred)
605         f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx
606 [1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])
607
608         scores = cross_val_score(log_reg, x_test,
609 y_test, cv=5, scoring='accuracy')
610         mean_error.append(np.array(scores).mean())
611         std_error.append(np.array(scores).std())
612
613         print(" Logistic Regression")
614         print(" For Degree = ", i)
615         print(" For C = ", c)
616         print(" Confusion Matrix - \n", cnf_mtx)
617         print(' Train accuracy score: ', log_reg.
618 score(x_train, y_train))
619         print(' Test accuracy score: ', log_reg.
620 score(x_test, y_test))
621         print(" F1 Score = ", f1_score)
622         print(" Classification Report\n",
623               classification_report(y_test, y_pred))
624         print("\n")
625
626 plt.errorbar(c_range, mean_error, yerr =
627 std_error, linewidth=3)
628 plt.xlabel('C', fontsize=25)
629 plt.ylabel('Accuracy Score', fontsize=25)
630 title_cv = f"Logistic Regression - Cross
631 Validation \nFor Degree = {i}"
632 plt.title(title_cv, fontsize=25)
633 plt.show()
634
635 # k-NN Classifier
636 nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
637 x_train_nn, x_test_nn, y_train_nn, y_test_nn =
638 train_test_split(X, y, test_size = 0.2,
639                 random_state=(1))
640 merr = []
641 serr = []
642
643 for nn in nn_range:
644     knn_model = KNeighborsClassifier(n_neighbors=nn,
645                                     weights='uniform')
646     knn_model.fit(x_train_nn, y_train_nn)
647     y_pred_nn = knn_model.predict(x_test_nn)
648     print("NN = ", nn)
649     print('Train accuracy score:', knn_model.score(
650 x_train_nn, y_train_nn))
651     print('Test accuracy score:', knn_model.score(
652 x_test_nn, y_test_nn))
653
654     scores_knn = cross_val_score(knn_model,
655 x_test_nn, y_test_nn, cv=5, scoring='accuracy')
656     merr.append(np.array(scores_knn).mean())
657     serr.append(np.array(scores_knn).std())
658
659 plt.errorbar(nn_range, merr, yerr = serr, linewidth
660 =3)
661 plt.xlabel('NN', fontsize=25)
662 plt.ylabel('Accuracy Score', fontsize=25)

```

```

644 title_cv = f"k-NN - Cross Validation"
645 plt.title(title_cv, fontsize=25)
646 plt.show()
647
648
649 # ROC-AUC Curve
650 visualizer = ROCAUC(knn_model, classes=["0", "1", "2",
651     "], macro=False, micro=False)
652
653 visualizer.fit(x_train_nn, y_train_nn)
654 visualizer.score(x_test_nn, y_test_nn)
655 visualizer.show()
656
657 # Review Scores Checkin
658 X = listings[
659     ['host_response_time', '
660     host_response_rate', 'host_acceptance_rate',
661     'bedrooms', 'beds',
662     'neighbourhood_cleansed',
663     'host_is_superhost',
664     'host_listings_count', 'host_total_listings_count',
665     'room_type',
666     'host_identity_verified',
667     'accommodates', 'price',
668     'minimum_nights', 'maximum_nights',
669     'bath-products', 'electric-system',
670     'food-services', 'house-furniture',
671     'house-rules',
672     'kitchen-appliances', 'parking',
673     'recreation', 'safety',
674     'host_email', 'host_work_email'] +
675     list(reviews.columns[2:])]
676
677 y = listings[['review_scores_checkin']]
678 y = (y/y.max())*100
679
680 y = y.assign(
681     rating_bin_ep = pd.qcut(
682         y['review_scores_checkin'],
683         q=2,
684         duplicates='drop',
685         labels=[0,1]
686     )
687 )
688
689 # Min Max of Each Bin
690 y.groupby('rating_bin_ep').min()
691 y.groupby('rating_bin_ep').max()
692
693 # Splitting Data in 75-25 Ratio
694 y = y[['rating_bin_ep']]
695 X_train, X_test, y_train, y_test = train_test_split(
696     X, y, test_size=0.25)
697
698 # Number of Records in Each Bin
699 cnt_plt = sns.countplot(y)
700 cnt_plt.bar_label(cnt_plt.containers[0])
701 plt.show()
702
703 # Logistic Regression - Varied C Range, using '
704     newton-cg' solver and multi_class='multinomial'
705 c_range = [0.001, 0.1, 1, 10, 100, 1000]
706 mean_error = []
707 std_error = []
708 for c in sorted(c_range):
709     logit = LogisticRegression(C=c, random_state=0,
710         solver='newton-cg', multi_class='multinomial')
711     logit.fit(X_train, y_train)
712     y_pred = logit.predict(X_test)
713     print("C = ", c)
714     print('Train accuracy score:', logit.score(
715         X_train, y_train))
716     print('Test accuracy score:', logit.score(X_test,
717         y_test))
718     print("Mean Squared Error: ", mean_squared_error(
719         y_test, y_pred))
720     scores = cross_val_score(logit, X_test, y_test,
721         cv=5, scoring='accuracy')
722     mean_error.append(np.array(scores).mean())
723     std_error.append(np.array(scores).std())
724 plt.errorbar(c_range, mean_error, yerr = std_error,
725     linewidth=3)
726 plt.xlabel('C', fontsize=25)
727 plt.ylabel('Accuracy Score', fontsize=25)
728 title_cv = "Logistic Regression - Cross Validation \
729     nFor Degree = 1"
730 plt.title(title_cv, fontsize=25)
731 plt.show()
732
733 # Feature Importance
734 cols = X.columns
735 cols = np.asarray(cols)
736
737 plt.figure(figsize=(30,15))
738 feature_importance = abs(logit.coef_[0])
739 feature_importance = 100.0 * (feature_importance /
740     feature_importance.max())
741 top_features = pd.DataFrame({'feature_imp':
742     feature_importance,
743     'features': cols},
744     columns=['feature_imp', 'features'])
745 top_features = top_features.sort_values(by='
746     feature_imp', ascending=False).head(20)
747 plt.bar(top_features.features, top_features.
748     feature_imp)
749 plt.xlabel('Importance', fontsize=35, fontweight='
750     bold')
751 plt.ylabel('Feature', fontsize=35, fontweight='bold')
752 plt.xticks(fontsize=30, rotation = 90)
753 plt.yticks(fontsize=30)
754 plt.show()
755
756 # Polynomial Degree and Error Plots
757 c_range = [0.001, 0.1, 1, 10, 100, 1000]
758 degree_range = [2]
759 for i in degree_range:
760     trans = PolynomialFeatures(degree = i)
761     x_poly = trans.fit_transform(X)
762     x_train, x_test, y_train, y_test =
763     train_test_split(x_poly, y, test_size = 0.2,
764         random_state=1)
765     mean_error = []
766     std_error = []
767     for c in c_range:
768         log_reg = LogisticRegression(C = c,
769             random_state=0, solver='newton-cg', multi_class='
770             multinomial')
771         log_reg.fit(x_train, y_train)
772         y_pred = log_reg.predict(x_test)
773         cnf_mtx = metrics.confusion_matrix(y_test,
774             y_pred)
775         f1_score = (2*(cnf_mtx[1][1])/((2*(cnf_mtx[
776             1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0]))
777         scores = cross_val_score(log_reg, x_test,
778             y_test, cv=5, scoring='accuracy')

```

```

759     mean_error.append(np.array(scores).mean())
760     std_error.append(np.array(scores).std())
761
762
763     print(" Logistic Regression")
764     print(" For Degree = ", i)
765     print(" For C = ", c)
766     print(" Confusion Matrix - \n", cnf_mtx)
767     print(' Train accuracy score: ', log_reg.
score(x_train, y_train))
768     print(' Test accuracy score: ', log_reg.
score(x_test, y_test))
769     print(" F1 Score = ", f1_score)
770     print(" Classification Report\n",
classification_report(y_test, y_pred))
771     print("\n")
772
773     plt.errorbar(c_range, mean_error, yerr =
std_error, linewidth=3)
774     plt.xlabel('C', fontsize=25)
775     plt.ylabel('Accuracy Score', fontsize=25)
776     title_cv = f"Logistic Regression - Cross
Validation \nFor Degree = {i}"
777     plt.title(title_cv, fontsize=25)
778     plt.show()
779
780
781 # k-NN Classifier
782 nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
783 x_train_nn, x_test_nn, y_train_nn, y_test_nn =
train_test_split(X, y, test_size = 0.2,
random_state=1)
784 merr = []
785 serr = []
786
787 for nn in nn_range:
788     knn_model = KNeighborsClassifier(n_neighbors=nn,
weights='uniform')
789     knn_model.fit(x_train_nn, y_train_nn)
790     y_pred_nn = knn_model.predict(x_test_nn)
791     print("NN = ", nn)
792     print('Train accuracy score:', knn_model.score(
x_train_nn, y_train_nn))
793     print('Test accuracy score:', knn_model.score(
x_test_nn, y_test_nn))
794
795     scores_knn = cross_val_score(knn_model,
x_test_nn, y_test_nn, cv=5, scoring='accuracy')
796     merr.append(np.array(scores_knn).mean())
797     serr.append(np.array(scores_knn).std())
798
799 plt.errorbar(nn_range, merr, yerr = serr, linewidth
=3)
800 plt.xlabel('NN', fontsize=25)
801 plt.ylabel('Accuracy Score', fontsize=25)
802 title_cv = f"k-NN - Cross Validation"
803 plt.title(title_cv, fontsize=25)
804 plt.show()
805
806
807 # Review Scores Cleanliness
808 X = listings[
809     ['host_response_time', '
host_response_rate', 'host_acceptance_rate',
810     'bedrooms', 'beds', '
neighbourhood_cleansed',
811     'host_is_superhost', '
host_listings_count', 'host_total_listings_count
',
812     'host_identity_verified', '
room_type',
813     'accommodates', 'price', '
minimum_nights', 'maximum_nights',
814     'bath_products', 'electric_system',
815     'food-services', 'house-furniture', '
house-rules',
816     'kitchen-appliances', 'parking', '
recreation', 'safety',
817     'host_email', 'host_work_email'] +
list(reviews.columns[2:])]
818 ]
819
820 y = listings[['review_scores_cleanliness']]
821 y = (y/y.max())*100
822
823 y = y.assign(
824     rating_bin_ep = pd.qcut(
825         y['review_scores_cleanliness'],
826         q=3,
827         duplicates='drop',
828         labels=[0,1,2]
829     )
830 )
831
832
833 # Min Max of Each Bin
834 y.groupby('rating_bin_ep').min()
835 y.groupby('rating_bin_ep').max()
836
837
838 # Splitting Data in 75-25 Ratio
839 y = y['rating_bin_ep']
840 X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.25)
841
842
843 # Number of Records in Each Bin
844 cnt_plt = sns.countplot(y)
845 cnt_plt.bar_label(cnt_plt.containers[0])
846 plt.show()
847
848
849 # Logistic Regression - Varied C Range, using '
newton-cg' solver and multi_class='multinomial'
850 c_range = [0.001, 0.1, 1, 10, 100, 1000]
851 mean_error = []
852 std_error = []
853 for c in sorted(c_range):
854     logit = LogisticRegression(C=c, random_state=0,
solver='newton-cg', multi_class='multinomial')
855     logit.fit(X_train, y_train)
856     y_pred = logit.predict(X_test)
857     print("C = ", c)
858     print('Train accuracy score:', logit.score(
X_train, y_train))
859     print('Test accuracy score:', logit.score(X_test,
y_test))
860     print("Mean Squared Error: ", mean_squared_error
(y_test, y_pred))
861     scores = cross_val_score(logit, X_test, y_test,
cv=5, scoring='accuracy')
862     mean_error.append(np.array(scores).mean())
863     std_error.append(np.array(scores).std())
864     plt.errorbar(c_range, mean_error, yerr = std_error,
linewidth=3)
865     plt.xlabel('C', fontsize=25)
866     plt.ylabel('Accuracy Score', fontsize=25)
867     title_cv = "Logistic Regression - Cross Validation \
nFor Degree = 1"
868     plt.title(title_cv, fontsize=25)
869     plt.show()
870
871
872 # Feature Importance
873 cols = X.columns
874 cols = np.asarray(cols)
875
876 plt.figure(figsize=(30,15))

```

```

877 feature_importance = abs(logit.coef_[0])
878 feature_importance = 100.0 * (feature_importance /
879     feature_importance.max())
879 top_features = pd.DataFrame({'feature_imp':
880     feature_importance,
881     'features': cols},
882     columns=['feature_imp', 'features'])
881 top_features = top_features.sort_values(by='
882     feature_imp', ascending=False).head(20)
882 plt.bar(top_features.features, top_features.
883     feature_imp)
883 plt.xlabel('Importance', fontsize=35, fontweight='
884     bold')
884 plt.ylabel('Feature', fontsize=35, fontweight='bold'
885     )
885 plt.xticks(fontsize=30, rotation = 90)
886 plt.yticks(fontsize=30)
887 plt.show()
888
889 plt.figure(figsize=(30,15))
890 feature_importance = abs(logit.coef_[1])
891 feature_importance = 100.0 * (feature_importance /
892     feature_importance.max())
892 top_features = pd.DataFrame({'feature_imp':
893     feature_importance,
894     'features': cols},
895     columns=['feature_imp', 'features'])
894 top_features = top_features.sort_values(by='
895     feature_imp', ascending=False).head(20)
895 plt.bar(top_features.features, top_features.
896     feature_imp)
896 plt.xlabel('Importance', fontsize=35, fontweight='
897     bold')
897 plt.ylabel('Feature', fontsize=35, fontweight='bold'
898     )
898 plt.xticks(fontsize=30, rotation = 90)
899 plt.yticks(fontsize=30)
900 plt.show()
901
902 plt.figure(figsize=(30,15))
903 feature_importance = abs(logit.coef_[2])
904 feature_importance = 100.0 * (feature_importance /
905     feature_importance.max())
905 top_features = pd.DataFrame({'feature_imp':
906     feature_importance,
907     'features': cols},
908     columns=['feature_imp', 'features'])
907 top_features = top_features.sort_values(by='
908     feature_imp', ascending=False).head(20)
908 plt.bar(top_features.features, top_features.
909     feature_imp)
909 plt.xlabel('Importance', fontsize=35, fontweight='
910     bold')
910 plt.ylabel('Feature', fontsize=35, fontweight='bold'
911     )
911 plt.xticks(fontsize=30, rotation = 90)
912 plt.yticks(fontsize=30)
913 plt.show()
914
915 # ROC-AUC Curve for all three categories
916 visualizer = ROCAUC(logit, classes=["0", "1", "2"],
917     macro=False, micro=False)
918
919 visualizer.fit(X_train, y_train)
920 visualizer.score(X_test, y_test)
921 visualizer.show()
922
923 # Polynomial Degree and Error Plots
924 c_range = [0.001, 0.1, 1, 10, 100, 1000]
925 degree_range = [2]
926
927 for i in degree_range:
928     trans = PolynomialFeatures(degree = i)
929     x_poly = trans.fit_transform(X)
930     x_train, x_test, y_train, y_test =
931     train_test_split(x_poly, y, test_size = 0.2,
932         random_state=1)
932     mean_error = []
933     std_error = []
934     for c in c_range:
935         log_reg = LogisticRegression(C = c,
936             random_state=0, solver='newton-cg', multi_class='
937             multinomial')
938         log_reg.fit(x_train, y_train)
939         y_pred = log_reg.predict(x_test)
940
941         cnf_mtx = metrics.confusion_matrix(y_test,
942             y_pred)
943         f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx
944             [1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])
945
946         scores = cross_val_score(log_reg, x_test,
947             y_test, cv=5, scoring='accuracy')
948         mean_error.append(np.array(scores).mean())
949         std_error.append(np.array(scores).std())
950
951     print(" Logistic Regression")
952     print(" For Degree = ", i)
953     print(" For C = ", c)
954     print(" Confusion Matrix - \n", cnf_mtx)
955     print(' Train accuracy score: ', log_reg.
956         score(x_train, y_train))
957     print(' Test accuracy score: ', log_reg.
958         score(x_test, y_test))
959     print(" F1 Score = ", f1_score)
960     print(" Classification Report\n",
961         classification_report(y_test, y_pred))
962     print("\n")
963
964     plt.errorbar(c_range, mean_error, yerr =
965         std_error, linewidth=3)
966     plt.xlabel('C', fontsize=25)
967     plt.ylabel('Accuracy Score', fontsize=25)
968     title_cv = f"Logistic Regression - Cross
969     Validation \nFor Degree = {i}"
970     plt.title(title_cv, fontsize=25)
971     plt.show()
972
973 # k-NN Classifier
974 nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
975 x_train_nn, x_test_nn, y_train_nn, y_test_nn =
976     train_test_split(X, y, test_size = 0.2,
977         random_state=1)
978     merr = []
979     serr = []
980
981     for nn in nn_range:
982         knn_model = KNeighborsClassifier(n_neighbors=nn,
983             weights='uniform')
984         knn_model.fit(x_train_nn, y_train_nn)
985         y_pred_nn = knn_model.predict(x_test_nn)
986         print("NN = ", nn)
987         print('Train accuracy score:', knn_model.score(
988             x_train_nn, y_train_nn))
989         print('Test accuracy score:', knn_model.score(
990             x_test_nn, y_test_nn))
991
992         scores_knn = cross_val_score(knn_model,
993             x_test_nn, y_test_nn, cv=5, scoring='accuracy')
994         merr.append(np.array(scores_knn).mean())
995         serr.append(np.array(scores_knn).std())
996
997     plt.errorbar(nn_range, merr, yerr = serr, linewidth
998         =3)

```

```

984 plt.xlabel('NN', fontsize=25)
985 plt.ylabel('Accuracy Score', fontsize=25)
986 title_cv = f"k-NN - Cross Validation"
987 plt.title(title_cv, fontsize=25)
988 plt.show()
989
990 # ROC-AUC Curve
991 visualizer = ROCAUC(knn_model, classes=["0", "1", "2",
992     ""], macro=False, micro=False)
993
994 visualizer.fit(x_train_nn, y_train_nn)
995 visualizer.score(x_test_nn, y_test_nn)
996 visualizer.show()
997
998 # Review Scores Communication
999 X = listings[
1000     ['host_response_time', '
1001     host_response_rate', 'host_acceptance_rate',
1002     'bedrooms', 'beds', '
1003     neighbourhood_cleaned',
1004     'host_is_superhost', '
1005     host_listings_count', 'host_total_listings_count',
1006     '
1007     room_type', 'host_identity_verified', '
1008     accommodates', 'price', '
1009     minimum_nights', 'maximum_nights',
1010     'bath-products', 'electric-system',
1011     'food-services', 'house-furniture',
1012     'house-rules',
1013     'kitchen-appliances', 'parking',
1014     'recreation', 'safety',
1015     'host_email', 'host_work_email'] +
1016     list(reviews.columns[2:])]
1017
1018 y = listings[['review_scores_communication']]
1019 y = (y/y.max())*100
1020
1021 y = y.assign(
1022     rating_bin_ep = pd.qcut(
1023         y['review_scores_communication'],
1024         q=2,
1025         duplicates='drop',
1026         labels=[0,1]
1027     )
1028 )
1029
1030 # Min Max of Each Bin
1031 y.groupby('rating_bin_ep').min()
1032 y.groupby('rating_bin_ep').max()
1033
1034 # Splitting Data in 75-25 Ratio
1035 y = y[['rating_bin_ep']]
1036 X_train, X_test, y_train, y_test = train_test_split(
1037     X, y, test_size=0.25)
1038
1039 # Number of Records in Each Bin
1040 cnt_plt = sns.countplot(y)
1041 cnt_plt.bar_label(cnt_plt.containers[0])
1042 plt.show()
1043
1044 # Logistic Regression - Varied C Range, using '
1045     newton-cg' solver and multi_class='multinomial'
1046 c_range = [0.001, 0.1, 1, 10, 100, 1000]
1047 mean_error = []
1048 std_error = []
1049 for c in sorted(c_range):
1050
1051     logit = LogisticRegression(C=c, random_state=0,
1052         solver='newton-cg', multi_class='multinomial')
1053     logit.fit(X_train, y_train)
1054     y_pred = logit.predict(X_test)
1055     print("C = ", c)
1056     print('Train accuracy score:', logit.score(
1057         X_train, y_train))
1058     print('Test accuracy score:', logit.score(X_test,
1059         y_test))
1060     print("Mean Squared Error: ", mean_squared_error(
1061         y_test, y_pred))
1062     scores = cross_val_score(logit, X_test, y_test,
1063         cv=5, scoring='accuracy')
1064     mean_error.append(np.array(scores).mean())
1065     std_error.append(np.array(scores).std())
1066     plt.errorbar(c_range, mean_error, yerr = std_error,
1067         linewidth=3)
1068     plt.xlabel('C', fontsize=25)
1069     plt.ylabel('Accuracy Score', fontsize=25)
1070     title_cv = "Logistic Regression - Cross Validation \
1071         nFor Degree = 1"
1072     plt.title(title_cv, fontsize=25)
1073     plt.show()
1074
1075 # Feature Importance
1076 cols = X.columns
1077 cols = np.asarray(cols)
1078
1079 plt.figure(figsize=(30,15))
1080 feature_importance = abs(logit.coef_[0])
1081 feature_importance = 100.0 * (feature_importance /
1082     feature_importance.max())
1083 top_features = pd.DataFrame({'feature_imp':
1084     feature_importance,
1085     'features': cols},
1086     columns=['feature_imp', 'features'])
1087 top_features = top_features.sort_values(by='
1088     feature_imp', ascending=False).head(20)
1089 plt.bar(top_features.features, top_features.
1090     feature_imp)
1091 plt.xlabel('Importance', fontsize=35, fontweight='
1092     bold')
1093 plt.ylabel('Feature', fontsize=35, fontweight='bold')
1094
1095 plt.xticks(fontsize=30, rotation = 90)
1096 plt.yticks(fontsize=30)
1097 plt.show()
1098
1099 # Polynomial Degree and Error Plots
1100 c_range = [0.001, 0.1, 1, 10, 100, 1000]
1101 degree_range = [2]
1102
1103 for i in degree_range:
1104     trans = PolynomialFeatures(degree = i)
1105     x_poly = trans.fit_transform(X)
1106     x_train, x_test, y_train, y_test =
1107         train_test_split(x_poly, y, test_size = 0.2,
1108             random_state=(1))
1109     mean_error = []
1110     std_error = []
1111     for c in c_range:
1112         log_reg = LogisticRegression(C = c,
1113             random_state=0, solver='newton-cg', multi_class='
1114             multinomial')
1115         log_reg.fit(x_train, y_train)
1116         y_pred = log_reg.predict(x_test)
1117
1118         cnf_mtx = metrics.confusion_matrix(y_test,
1119             y_pred)
1120         f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx
1121             [1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])

```



```

1100     scores = cross_val_score(log_reg, x_test,
1101                             y_test, cv=5, scoring='accuracy')
1102     mean_error.append(np.array(scores).mean())
1103     std_error.append(np.array(scores).std())
1104
1105     print(" Logistic Regression")
1106     print(" For Degree = ", i)
1107     print(" For C = ", c)
1108     print(" Confusion Matrix - \n", cnf_mtx)
1109     print(' Train accuracy score: ', log_reg.
1110           score(x_train, y_train))
1111     print(' Test accuracy score: ', log_reg.
1112           score(x_test, y_test))
1113     print(" F1 Score = ", f1_score)
1114     print(" Classification Report\n",
1115           classification_report(y_test, y_pred))
1116     print("\n")
1117
1118     plt.errorbar(c_range, mean_error, yerr =
1119                 std_error, linewidth=3)
1120     plt.xlabel('C', fontsize=25)
1121     plt.ylabel('Accuracy Score', fontsize=25)
1122     title_cv = f"Logistic Regression - Cross
1123               Validation \nFor Degree = {i}"
1124     plt.title(title_cv, fontsize=25)
1125     plt.show()
1126
1127 # k-NN Classifier
1128 nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
1129 x_train_nn, x_test_nn, y_train_nn, y_test_nn =
1130     train_test_split(X, y, test_size = 0.2,
1131                     random_state=1)
1132 merr = []
1133 serr = []
1134
1135 for nn in nn_range:
1136     knn_model = KNeighborsClassifier(n_neighbors=nn,
1137                                     weights='uniform')
1138     knn_model.fit(x_train_nn, y_train_nn)
1139     y_pred_nn = knn_model.predict(x_test_nn)
1140     print("NN = ", nn)
1141     print('Train accuracy score:', knn_model.score(
1142         x_train_nn, y_train_nn))
1143     print('Test accuracy score:', knn_model.score(
1144         x_test_nn, y_test_nn))
1145
1146     scores_knn = cross_val_score(knn_model,
1147                                 x_test_nn, y_test_nn, cv=5, scoring='accuracy')
1148     merr.append(np.array(scores_knn).mean())
1149     serr.append(np.array(scores_knn).std())
1150
1151 plt.errorbar(nn_range, merr, yerr = serr, linewidth
1152             =3)
1153 plt.xlabel('NN', fontsize=25)
1154 plt.ylabel('Accuracy Score', fontsize=25)
1155 title_cv = f"k-NN - Cross Validation"
1156 plt.title(title_cv, fontsize=25)
1157 plt.show()
1158
1159 # Review Scores Location
1160 X = listings[
1161     ['host_response_time', '
1162     host_response_rate', 'host_acceptance_rate',
1163     'bedrooms', 'beds', '
1164     neighbourhood_cleansed',
1165     'host_is_superhost', '
1166     host_listings_count', 'host_total_listings_count',
1167     '
1168     'host_identity_verified', '
1169     room_type',
1170     'accommodates', 'price', '
1171     minimum_nights', 'maximum_nights',
1172     'bath-products', 'electric-system',
1173     'food-services', 'house-furniture', '
1174     house-rules',
1175     'kitchen-appliances', 'parking', '
1176     recreation', 'safety',
1177     'host_email', 'host_work_email'] +
1178     list(reviews.columns[2:])]
1179
1180 y = listings[['review_scores_location']]
1181 y = (y/y.max())*100
1182
1183 y = y.assign(
1184     rating_bin_ep = pd.qcut(
1185         y[['review_scores_location']],
1186         q=3,
1187         duplicates='drop',
1188         labels=[0,1,2]
1189     )
1190 )
1191
1192 # Min Max of Each Bin
1193 y.groupby('rating_bin_ep').min()
1194 y.groupby('rating_bin_ep').max()
1195
1196 # Splitting Data in 75-25 Ratio
1197 y = y[['rating_bin_ep']]
1198 X_train, X_test, y_train, y_test = train_test_split(
1199     X, y, test_size=0.25)
1200
1201 # Number of Records in Each Bin
1202 cnt_plt = sns.countplot(y)
1203 cnt_plt.bar_label(cnt_plt.containers[0])
1204 plt.show()
1205
1206 # Logistic Regression - Varied C Range, using '
1207 newton-cg' solver and multi_class='multinomial'
1208 c_range = [0.001, 0.1, 1, 10, 100, 1000]
1209 mean_error = []
1210 std_error = []
1211
1212 for c in sorted(c_range):
1213     logit = LogisticRegression(C=c, random_state=0,
1214                               solver='newton-cg', multi_class='multinomial')
1215     logit.fit(X_train, y_train)
1216     y_pred = logit.predict(X_test)
1217     print("C = ", c)
1218     print('Train accuracy score:', logit.score(
1219         X_train, y_train))
1220     print('Test accuracy score:', logit.score(X_test,
1221         y_test))
1222     print("Mean Squared Error: ", mean_squared_error(
1223         y_test, y_pred))
1224     scores = cross_val_score(logit, X_test, y_test,
1225                             cv=5, scoring='accuracy')
1226     mean_error.append(np.array(scores).mean())
1227     std_error.append(np.array(scores).std())
1228
1229 plt.errorbar(c_range, mean_error, yerr = std_error,
1230             linewidth=3)
1231 plt.xlabel('C', fontsize=25)
1232 plt.ylabel('Accuracy Score', fontsize=25)
1233 title_cv = "Logistic Regression - Cross Validation \n
1234           For Degree = 1"
1235 plt.title(title_cv, fontsize=25)
1236 plt.show()
1237
1238 # Feature Importance
1239 cols = X.columns
1240 cols = np.asarray(cols)

```

```

1217 plt.figure(figsize=(30,15))
1218 feature_importance = abs(logit.coef_[0])
1219 feature_importance = 100.0 * (feature_importance /
1220     feature_importance.max())
1221 top_features = pd.DataFrame({'feature_imp':
1222     feature_importance,
1223     'features': cols},
1224     columns=['feature_imp', 'features'])
1225 top_features = top_features.sort_values(by='
1226     feature_imp', ascending=False).head(20)
1227 plt.bar(top_features.features, top_features.
1228     feature_imp)
1229 plt.xlabel('Importance', fontsize=35, fontweight='
1230     bold')
1231 plt.ylabel('Feature', fontsize=35, fontweight='bold'
1232     )
1233 plt.xticks(fontsize=30, rotation = 90)
1234 plt.yticks(fontsize=30)
1235 plt.show()
1236
1237 plt.figure(figsize=(30,15))
1238 feature_importance = abs(logit.coef_[1])
1239 feature_importance = 100.0 * (feature_importance /
1240     feature_importance.max())
1241 top_features = pd.DataFrame({'feature_imp':
1242     feature_importance,
1243     'features': cols},
1244     columns=['feature_imp', 'features'])
1245 top_features = top_features.sort_values(by='
1246     feature_imp', ascending=False).head(20)
1247 plt.bar(top_features.features, top_features.
1248     feature_imp)
1249 plt.xlabel('Importance', fontsize=35, fontweight='
1250     bold')
1251 plt.ylabel('Feature', fontsize=35, fontweight='bold'
1252     )
1253 plt.xticks(fontsize=30, rotation = 90)
1254 plt.yticks(fontsize=30)
1255 plt.show()
1256
1257
1258 # ROC-AUC Curve for all three categories
1259 visualizer = ROCAUC(logit, classes=["0", "1", "2"],
1260     macro=False, micro=False)
1261 visualizer.fit(X_train, y_train)
1262 visualizer.score(X_test, y_test)
1263 visualizer.show()
1264
1265 # Polynomial Degree and Error Plots
1266 c_range = [0.001, 0.1, 1, 10, 100, 1000]
1267 degree_range = [2]
1268 for i in degree_range:
1269     trans = PolynomialFeatures(degree = i)
1270     x_poly = trans.fit_transform(X)
1271     x_train, x_test, y_train, y_test =
1272     train_test_split(x_poly, y, test_size = 0.2,
1273         random_state=1)
1274     mean_error = []
1275     std_error = []
1276     for c in c_range:
1277         log_reg = LogisticRegression(C = c,
1278             random_state=0, solver='newton-cg',multi_class='
1279             multinomial')
1280         log_reg.fit(x_train, y_train)
1281         y_pred = log_reg.predict(x_test)
1282
1283         cnf_mtx = metrics.confusion_matrix(y_test,
1284             y_pred)
1285         f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx
1286             [1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])
1287
1288         scores = cross_val_score(log_reg, x_test,
1289             y_test, cv=5, scoring='accuracy')
1290         mean_error.append(np.array(scores).mean())
1291         std_error.append(np.array(scores).std())
1292
1293         print(" Logistic Regression")
1294         print(" For Degree = ", i)
1295         print(" For C = ", c)
1296         print(" Confusion Matrix - \n", cnf_mtx)
1297         print(' Train accuracy score: ', log_reg.
1298             score(x_train, y_train))
1299         print(' Test accuracy score: ', log_reg.
1300             score(x_test, y_test))
1301         print(" F1 Score = ", f1_score)
1302         print(" Classification Report\n",
1303             classification_report(y_test, y_pred))
1304         print("\n")
1305
1306     plt.errorbar(c_range, mean_error, yerr =
1307         std_error, linewidth=3)
1308     plt.xlabel('C', fontsize=25)
1309     plt.ylabel('Accuracy Score', fontsize=25)
1310     title_cv = f"Logistic Regression - Cross
1311         Validation \nFor Degree = {i}"
1312     plt.title(title_cv, fontsize=25)
1313     plt.show()
1314
1315 # k-NN Classifier
1316 nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
1317 x_train_nn, x_test_nn, y_train_nn, y_test_nn =
1318     train_test_split(X, y, test_size = 0.2,
1319         random_state=1)
1320 merr = []
1321 serr = []
1322
1323 for nn in nn_range:
1324     knn_model = KNeighborsClassifier(n_neighbors=nn,
1325         weights='uniform')
1326     knn_model.fit(x_train_nn, y_train_nn)
1327     y_pred_nn = knn_model.predict(x_test_nn)
1328     print("NN = ", nn)
1329     print('Train accuracy score:',knn_model.score(
1330         x_train_nn, y_train_nn))
1331     print('Test accuracy score:',knn_model.score(
1332         x_test_nn, y_test_nn))
1333
1334     scores_knn = cross_val_score(knn_model,
1335         x_test_nn, y_test_nn, cv=5, scoring='accuracy')
1336     merr.append(np.array(scores_knn).mean())
1337     serr.append(np.array(scores_knn).std())
1338
1339 plt.errorbar(nn_range, merr, yerr = serr, linewidth
1340     =3)

```

```

1324 plt.xlabel('NN', fontsize=25)
1325 plt.ylabel('Accuracy Score', fontsize=25)
1326 title_cv = f"k-NN - Cross Validation"
1327 plt.title(title_cv, fontsize=25)
1328 plt.show()
1329
1330
1331 # ROC-AUC Curve
1332 visualizer = ROCAUC(knn_model, classes=["0", "1", "2"], macro=False, micro=False)
1333
1334 visualizer.fit(x_train_nn, y_train_nn)
1335 visualizer.score(x_test_nn, y_test_nn)
1336 visualizer.show()
1337
1338 # Review Scores Rating
1339 X = listings[
1340     ['host_response_time', '
1341     host_response_rate', 'host_acceptance_rate', '
1342     'bedrooms', 'beds', '
1343     neighbourhood_cleaned', '
1344     'host_is_superhost', '
1345     host_listings_count', 'host_total_listings_count',
1346     'room_type', 'host_identity_verified', '
1347     'accommodates', 'price', '
1348     minimum_nights', 'maximum_nights', '
1349     'bath-products', 'electric-system', '
1350     'food-services', 'house-furniture', '
1351     house-rules', 'kitchen-appliances', 'parking', '
1352     recreation', 'safety', '
1353     'host_email', 'host_work_email'] +
1354     list(reviews.columns[2:])]
1355
1356 y = listings[['review_scores_rating']]
1357 y = (y/y.max())*100
1358
1359 y = y.assign(
1360     rating_bin_ep = pd.qcut(
1361         y['review_scores_rating'],
1362         q=3,
1363         duplicates='drop',
1364         labels=[0,1,2]
1365     )
1366 )
1367
1368 # Min Max of Each Bin
1369 y.groupby('rating_bin_ep').min()
1370 y.groupby('rating_bin_ep').max()
1371
1372 # Splitting Data in 75-25 Ratio
1373 y = y[['rating_bin_ep']]
1374 X_train, X_test, y_train, y_test = train_test_split(
1375     X, y, test_size=0.25)
1376
1377 # Number of Records in Each Bin
1378 cnt_plt = sns.countplot(y)
1379 cnt_plt.bar_label(cnt_plt.containers[0])
1380 plt.show()
1381
1382 # Logistic Regression - Varied C Range, using '
1383     newton-cg' solver and multi_class='multinomial'
1384 c_range = [0.001, 0.1, 1, 10, 100, 1000]
1385 mean_error = []
1386 std_error = []
1387 for c in sorted(c_range):
1388     logit = LogisticRegression(C=c, random_state=0,
1389     solver='newton-cg', multi_class='multinomial')
1390     logit.fit(X_train, y_train)
1391     y_pred = logit.predict(X_test)
1392     print("C = ", c)
1393     print('Train accuracy score:', logit.score(
1394     X_train, y_train))
1395     print('Test accuracy score:', logit.score(X_test,
1396     y_test))
1397     print("Mean Squared Error: ", mean_squared_error(
1398     y_test, y_pred))
1399     scores = cross_val_score(logit, X_test, y_test,
1400     cv=5, scoring='accuracy')
1401     mean_error.append(np.array(scores).mean())
1402     std_error.append(np.array(scores).std())
1403     plt.errorbar(c_range, mean_error, yerr = std_error,
1404     linewidth=3)
1405     plt.xlabel('C', fontsize=25)
1406     plt.ylabel('Accuracy Score', fontsize=25)
1407     title_cv = "Logistic Regression - Cross Validation \
1408     nFor Degree = 1"
1409     plt.title(title_cv, fontsize=25)
1410     plt.show()
1411
1412 # Feature Importance
1413 cols = X.columns
1414 cols = np.asarray(cols)
1415
1416 plt.figure(figsize=(30,15))
1417 feature_importance = abs(logit.coef_[0])
1418 feature_importance = 100.0 * (feature_importance /
1419     feature_importance.max())
1420 top_features = pd.DataFrame({'feature_imp':
1421     feature_importance,
1422     'features': cols},
1423     columns=['feature_imp', 'features'])
1424 top_features = top_features.sort_values(by='
1425     feature_imp', ascending=False).head(20)
1426 plt.bar(top_features.features, top_features.
1427     feature_imp)
1428 plt.xlabel('Importance', fontsize=35, fontweight='
1429     bold')
1430 plt.ylabel('Feature', fontsize=35, fontweight='bold')
1431 plt.xticks(fontsize=30, rotation = 90)
1432 plt.yticks(fontsize=30)
1433 plt.show()
1434
1435 plt.figure(figsize=(30,15))
1436 feature_importance = abs(logit.coef_[1])
1437 feature_importance = 100.0 * (feature_importance /
1438     feature_importance.max())
1439 top_features = pd.DataFrame({'feature_imp':
1440     feature_importance,
1441     'features': cols},
1442     columns=['feature_imp', 'features'])
1443 top_features = top_features.sort_values(by='
1444     feature_imp', ascending=False).head(20)
1445 plt.bar(top_features.features, top_features.
1446     feature_imp)
1447 plt.xlabel('Importance', fontsize=35, fontweight='
1448     bold')
1449 plt.ylabel('Feature', fontsize=35, fontweight='bold')
1450 plt.xticks(fontsize=30, rotation = 90)
1451 plt.yticks(fontsize=30)
1452 plt.show()
1453
1454 plt.figure(figsize=(30,15))
1455 feature_importance = abs(logit.coef_[2])
1456 feature_importance = 100.0 * (feature_importance /
1457     feature_importance.max())
1458 top_features = pd.DataFrame({'feature_imp':

```

```

feature_importance,
1438     columns=['feature_imp', 'features': cols},
1439     top_features = top_features.sort_values(by='
feature_imp', ascending=False).head(20)
1440 plt.bar(top_features.features, top_features.
feature_imp)
1441 plt.xlabel('Importance', fontsize=35, fontweight='
bold')
1442 plt.ylabel('Feature', fontsize=35, fontweight='bold'
)
1443 plt.xticks(fontsize=30, rotation = 90)
1444 plt.yticks(fontsize=30)
1445 plt.show()
1446
1447 # ROC-AUC Curve for all three categories
1448 visualizer = ROCAUC(logit, classes=["0", "1", "2"],
1449 macro=False, micro=False)
1450
1451 visualizer.fit(X_train, y_train)
1452 visualizer.score(X_test, y_test)
1453 visualizer.show()
1454
1455 # Polynomial Degree and Error Plots
1456 c_range = [0.001, 0.1, 1, 10, 100, 1000]
1457 degree_range = [2]
1458
1459 for i in degree_range:
1460     trans = PolynomialFeatures(degree = i)
1461     x_poly = trans.fit_transform(X)
1462     x_train, x_test, y_train, y_test =
1463     train_test_split(x_poly, y, test_size = 0.2,
random_state=1)
1464     mean_error = []
1465     std_error = []
1466     for c in c_range:
1467         log_reg = LogisticRegression(C = c,
random_state=0, solver='newton-cg',multi_class='
multinomial')
1468         log_reg.fit(x_train, y_train)
1469         y_pred = log_reg.predict(x_test)
1470
1471         cnf_mtx = metrics.confusion_matrix(y_test,
y_pred)
1472         f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx
[1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])
1473
1474         scores = cross_val_score(log_reg, x_test,
y_test, cv=5, scoring='accuracy')
1475         mean_error.append(np.array(scores).mean())
1476         std_error.append(np.array(scores).std())
1477
1478         print(" Logistic Regression")
1479         print(" For Degree = ", i)
1480         print(" For C = ", c)
1481         print(" Confusion Matrix - \n", cnf_mtx)
1482         print(' Train accuracy score: ', log_reg.
score(x_train, y_train))
1483         print(' Test accuracy score: ', log_reg.
score(x_test, y_test))
1484         print(" F1 Score = ", f1_score)
1485         print(" Classification Report\n",
classification_report(y_test, y_pred))
1486         print("\n")
1487
1488     plt.errorbar(c_range, mean_error, yerr =
std_error, linewidth=3)
1489     plt.xlabel('C', fontsize=25)
1490     plt.ylabel('Accuracy Score', fontsize=25)
1491     title_cv = f"Logistic Regression - Cross
Validation \nFor Degree = {i}"
1492
1493     plt.title(title_cv, fontsize=25)
1494     plt.show()
1495
1496 # k-NN Classifier
1497 nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
1498 x_train_nn, x_test_nn, y_train_nn, y_test_nn =
1499 train_test_split(X, y, test_size = 0.2,
random_state=1)
1500 merr = []
1501 serr = []
1502
1503 for nn in nn_range:
1504     knn_model = KNeighborsClassifier(n_neighbors=nn,
weights='uniform')
1505     knn_model.fit(x_train_nn, y_train_nn)
1506     y_pred_nn = knn_model.predict(x_test_nn)
1507     print("NN = ", nn)
1508     print('Train accuracy score:',knn_model.score(
x_train_nn, y_train_nn))
1509     print('Test accuracy score:',knn_model.score(
x_test_nn, y_test_nn))
1510
1511     scores_knn = cross_val_score(knn_model,
x_test_nn, y_test_nn, cv=5, scoring='accuracy')
1512     merr.append(np.array(scores_knn).mean())
1513     serr.append(np.array(scores_knn).std())
1514
1515     plt.errorbar(nn_range, merr, yerr = serr, linewidth
=3)
1516     plt.xlabel('NN', fontsize=25)
1517     plt.ylabel('Accuracy Score', fontsize=25)
1518     title_cv = f"k-NN - Cross Validation"
1519     plt.title(title_cv, fontsize=25)
1520     plt.show()
1521
1522 # ROC-AUC Curve
1523 visualizer = ROCAUC(knn_model, classes=["0", "1", "2
"], macro=False, micro=False)
1524
1525 visualizer.fit(x_train_nn, y_train_nn)
1526 visualizer.score(x_test_nn, y_test_nn)
1527 visualizer.show()
1528
1529 # Review Scores Value
1530 X = listings[
1531     ['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'] +
1532     list(reviews.columns[2:])]
1533
1534 y = listings[['review_scores_value']]
1535 y = (y/y.max())*100
1536
1537 y = y.assign(
1538     rating_bin_ep = pd.qcut(
1539         y['review_scores_value'],

```

```

1550         q=3,
1551         duplicates='drop',
1552         labels=[0,1,2]
1553     )
1554 )
1555
1556
1557 # Min Max of Each Bin
1558 y.groupby('rating_bin_ep').min()
1559 y.groupby('rating_bin_ep').max()
1560
1561
1562 # Splitting Data in 75-25 Ratio
1563 y = y['rating_bin_ep']
1564 X_train, X_test, y_train, y_test = train_test_split(
1565     X, y, test_size=0.25)
1566
1567
1568 # Number of Records in Each Bin
1569 cnt_plt = sns.countplot(y)
1570 cnt_plt.bar_label(cnt_plt.containers[0])
1571 plt.show()
1572
1573 # Logistic Regression - Varied C Range, using '
1574 newton-cg' solver and multi_class='multinomial'
1575 c_range = [0.001, 0.1, 1, 10, 100, 1000]
1576 mean_error = []
1577 std_error = []
1578 for c in sorted(c_range):
1579     logit = LogisticRegression(C=c, random_state=0,
1580     solver='newton-cg', multi_class='multinomial')
1581     logit.fit(X_train, y_train)
1582     y_pred = logit.predict(X_test)
1583     print("C = ", c)
1584     print('Train accuracy score:', logit.score(
1585     X_train, y_train))
1586     print('Test accuracy score:', logit.score(X_test,
1587     y_test))
1588     print("Mean Squared Error: ", mean_squared_error(
1589     y_test, y_pred))
1590     scores = cross_val_score(logit, X_test, y_test,
1591     cv=5, scoring='accuracy')
1592     mean_error.append(np.array(scores).mean())
1593     std_error.append(np.array(scores).std())
1594 plt.errorbar(c_range, mean_error, yerr = std_error,
1595     linewidth=3)
1596 plt.xlabel('C', fontsize=25)
1597 plt.ylabel('Accuracy Score', fontsize=25)
1598 title_cv = "Logistic Regression - Cross Validation \
1599 nFor Degree = 1"
1600 plt.title(title_cv, fontsize=25)
1601 plt.show()
1602
1603
1604 # Feature Importance
1605 cols = X.columns
1606 cols = np.asarray(cols)
1607
1608 plt.figure(figsize=(30,15))
1609 feature_importance = abs(logit.coef_[0])
1610 feature_importance = 100.0 * (feature_importance /
1611     feature_importance.max())
1612 top_features = pd.DataFrame({'feature_imp':
1613     feature_importance,
1614     'features': cols},
1615     columns=['feature_imp', 'features'])
1616 top_features = top_features.sort_values(by='
1617 feature_imp', ascending=False).head(20)
1618 plt.bar(top_features.features, top_features.
1619     feature_imp)
1620 plt.xlabel('Importance', fontsize=35, fontweight='
1621     bold')
1622 plt.ylabel('Feature', fontsize=35, fontweight='bold')
1623 plt.xticks(fontsize=30, rotation = 90)
1624 plt.yticks(fontsize=30)
1625 plt.show()
1626
1627 plt.figure(figsize=(30,15))
1628 feature_importance = abs(logit.coef_[2])
1629 feature_importance = 100.0 * (feature_importance /
1630     feature_importance.max())
1631 top_features = pd.DataFrame({'feature_imp':
1632     feature_importance,
1633     'features': cols},
1634     columns=['feature_imp', 'features'])
1635 top_features = top_features.sort_values(by='
1636 feature_imp', ascending=False).head(20)
1637 plt.bar(top_features.features, top_features.
1638     feature_imp)
1639 plt.xlabel('Importance', fontsize=35, fontweight='
1640     bold')
1641 plt.ylabel('Feature', fontsize=35, fontweight='bold')
1642 plt.xticks(fontsize=30, rotation = 90)
1643 plt.yticks(fontsize=30)
1644 plt.show()
1645
1646 # ROC-AUC Curve for all three categories
1647 visualizer = ROCAUC(logit, classes=["0", "1", "2"],
1648     macro=False, micro=False)
1649 visualizer.fit(X_train, y_train)
1650 visualizer.score(X_test, y_test)
1651 visualizer.show()
1652
1653 # Polynomial Degree and Error Plots
1654 c_range = [0.001, 0.1, 1, 10, 100, 1000]
1655 degree_range = [2]
1656
1657 for i in degree_range:
1658     trans = PolynomialFeatures(degree = i)
1659     x_poly = trans.fit_transform(X)
1660     x_train, x_test, y_train, y_test =
1661     train_test_split(x_poly, y, test_size = 0.2,
1662     random_state=1)
1663     mean_error = []
1664     std_error = []
1665     for c in c_range:
1666         log_reg = LogisticRegression(C = c,
1667         random_state=0, solver='newton-cg', multi_class='
1668         multinomial')
1669         log_reg.fit(x_train, y_train)
1670         y_pred = log_reg.predict(x_test)
1671
1672         cnf_mtx = metrics.confusion_matrix(y_test,
1673         y_pred)

```



```

1662         f1_score = (2*cnf_mtx[1][1])/((2*cnf_mtx
1663         [1][1]) + cnf_mtx[0][1] + cnf_mtx[1][0])
1664
1665         scores = cross_val_score(log_reg, x_test,
1666         y_test, cv=5, scoring='accuracy')
1667         mean_error.append(np.array(scores).mean())
1668         std_error.append(np.array(scores).std())
1669
1670         print(" Logistic Regression")
1671         print(" For Degree = ", i)
1672         print(" For C = ", c)
1673         print(" Confusion Matrix - \n", cnf_mtx)
1674         print(' Train accuracy score: ', log_reg.
1675         score(x_train, y_train))
1676         print(' Test accuracy score: ', log_reg.
1677         score(x_test, y_test))
1678         print(" F1 Score = ", f1_score)
1679         print(" Classification Report\n",
1680         classification_report(y_test, y_pred))
1681         print("\n")
1682
1683         plt.errorbar(c_range, mean_error, yerr =
1684         std_error, linewidth=3)
1685         plt.xlabel('C', fontsize=25)
1686         plt.ylabel('Accuracy Score', fontsize=25)
1687         title_cv = f"Logistic Regression - Cross
1688         Validation \nFor Degree = {i}"
1689         plt.title(title_cv, fontsize=25)
1690         plt.show()
1691
1692     # k-NN Classifier
1693     nn_range = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
1694     x_train_nn, x_test_nn, y_train_nn, y_test_nn =
1695     train_test_split(X, y, test_size = 0.2,
1696     random_state=1)
1697     merr = []
1698     serr = []
1699
1700     for nn in nn_range:
1701         knn_model = KNeighborsClassifier(n_neighbors=nn,
1702         weights='uniform')
1703         knn_model.fit(x_train_nn, y_train_nn)
1704         y_pred_nn = knn_model.predict(x_test_nn)
1705         print("NN = ", nn)
1706         print('Train accuracy score:', knn_model.score(
1707         x_train_nn, y_train_nn))
1708         print('Test accuracy score:', knn_model.score(
1709         x_test_nn, y_test_nn))
1710
1711         scores_knn = cross_val_score(knn_model,
1712         x_test_nn, y_test_nn, cv=5, scoring='accuracy')
1713         merr.append(np.array(scores_knn).mean())
1714         serr.append(np.array(scores_knn).std())
1715
1716     plt.errorbar(nn_range, merr, yerr = serr, linewidth
1717     =3)
1718     plt.xlabel('NN', fontsize=25)
1719     plt.ylabel('Accuracy Score', fontsize=25)
1720     title_cv = f"k-NN - Cross Validation"
1721     plt.title(title_cv, fontsize=25)
1722     plt.show()
1723
1724     # ROC-AUC Curve
1725     visualizer = ROCAUC(knn_model, classes=["0", "1", "2
1726     "], macro=False, micro=False)
1727     visualizer.fit(x_train_nn, y_train_nn)
1728     visualizer.score(x_test_nn, y_test_nn)
1729     visualizer.show()
1730
1731     import random
1732     import numpy as np
1733
1734     import pandas as pd
1735     import matplotlib.pyplot as plt
1736     import seaborn as sns
1737
1738     from sklearn.linear_model import LogisticRegression
1739     from sklearn.model_selection import train_test_split
1740     from sklearn.dummy import DummyClassifier
1741     from sklearn.metrics import confusion_matrix
1742     from sklearn.metrics import classification_report
1743
1744     def binary_step(x, thresh=10):
1745         return np.where(x<thresh, 0, 1)
1746
1747     def decision_boundary(model):
1748         b = model.intercept_[0]
1749         w1, w2 = model.coef_[0]
1750
1751         c = -b / w2
1752         m = -w1 / w2
1753
1754         xd = np.linspace(X.x_1.min(), X.x_1.max())
1755         yd = m * xd + c
1756
1757         return xd, yd
1758
1759     size = 250
1760     # NO LINEAR CORELATION
1761     X = pd.DataFrame()
1762     X['x_1'] = np.random.rand(size, )
1763     X['x_2'] = np.random.randint(0, 60, size=size)
1764
1765     y = binary_step(10 + np.random.normal(0.0,1.0,size))
1766
1767     X_train, X_test, y_train, y_test = train_test_split(
1768     X, y, test_size=0.2,
1769
1770     random_state=42)
1771
1772     model_no_linear_corr = LogisticRegression().fit(
1773     X_train, y_train)
1774     ypred = model_no_linear_corr.predict(X)
1775
1776     print("Logistic Regression - No Colinearity")
1777     print(f"Model Score: {model_no_linear_corr.score(X,
1778     y)}")
1779     print(confusion_matrix(y, model_no_linear_corr.
1780     predict(X)))
1781     print(classification_report(y, model_no_linear_corr.
1782     predict(X)))
1783
1784     pred_actual_logi = pd.DataFrame({'x_1': X['x_1'],
1785     'x_2': X['x_2'],
1786     'Ground Truth': y,
1787     'Prediction': ypred})
1788
1789     # sns.scatterplot(x=X.x_2, y=X.x_1, style=y, hue=y,
1790     palette='deep')
1791
1792     xd, yd = decision_boundary(model_no_linear_corr)
1793     for idx, val in enumerate(yd):
1794         if val < 0:
1795             yd[idx] = 0
1796         elif val > 50:
1797             yd[idx] = 50
1798
1799     sns.scatterplot(x=X.x_1, y=X.x_2, style=ypred, hue=
1800     ypred, palette='deep')
1801     plt.plot(xd, yd)
1802
1803     # DATA IMBALANCE
1804     X = pd.DataFrame()

```

```

69 size = 2000
70 X['x_1'] = np.arange(0,2,0.05)
71
72 y = 10 * X.x_1 + np.random.normal(0.0,1.0,X.size)
73
74 for idx, val in enumerate(y):
75     if val < 2:
76         y[idx] = int(0)
77     else:
78         y[idx] = int(1)
79 y = y.astype('int')
80
81 model_data_imbalance = LogisticRegression().fit(X, y)
82
83 print("Logistic Regression - Data Imbalance")
84 print(f"Model Score: {model_data_imbalance.score(X,
85     y)}")
86
87 print(confusion_matrix(y, model_data_imbalance.
88     predict(X)))
89 print(classification_report(y, model_data_imbalance.
90     predict(X)))
91
92 sns.scatterplot(x=X.x_1, y=y, style=y, hue=y,
93     palette='deep')
94 plt.legend(loc='lower right')
95 plt.ylabel('Target')
96 plt.xlabel('Feature')
97
98 ypred = model_data_imbalance.predict(X)
99 sns.scatterplot(x=X.x_1, y=ypred, style=ypred, hue=
100     ypred, palette='deep')
101 plt.legend(loc='lower right')
102 ypred = model_data_imbalance.predict(X)
103 plt.ylabel('Predicted')
104 plt.xlabel('Feature')
105
106 pred_data_imbalance_logi = pd.DataFrame({'x_1': X['
107     x_1'],
108     'Ground Truth': y,
109     'Prediction': model_data_imbalance.
110     predict(X)})
111
112 # DUMMY CLASSIFIER
113 model_dummy = DummyClassifier(strategy='
114     most_frequent').fit(X, y)
115
116 print("Dummy Classifier - Data Imbalance")
117 print(f"Model Score: {model_dummy.score(X, y)}")
118 print(confusion_matrix(y, model_dummy.predict(X)))
119 print(classification_report(y, model_dummy.predict(X)
120     )))
121
122 pred_data_imbalance_dummy = pd.DataFrame({'x_1': X['
123     x_1'],
124     'Ground Truth': y,
125     'Prediction': model_dummy.predict(X)})
126
127 # HOLD OUT METHOD
128 import numpy as np
129 from sklearn.model_selection import train_test_split
130 from sklearn.linear_model import LinearRegression
131 from sklearn.metrics import mean_squared_error
132 import matplotlib.pyplot as plt
133 import pandas as pd
134
135 X = np.arange(0,1,0.05).reshape(-1, 1)
136 y = 10 * X + np.random.normal(0.0,1.0,X.size).
137     reshape(-1, 1)
138
139 intercept = []
140 slope = []
141 mean_error = []
142
143 for i in range(5):
144     Xtrain, Xtest, ytrain, ytest = train_test_split(
145         X,y,test_size=0.2)
146
147     model = LinearRegression().fit(Xtrain, ytrain)
148     ypred = model.predict(Xtest)
149
150     intercept.append(model.intercept_[0])
151     slope.append(model.coef_[0][0])
152     mean_error.append(mean_squared_error(ytest,
153         ypred))
154
155 print('Intercept: {:.2f}\nSlope: {:.2f}\nSquared
156     Error: {:.2f}'.format(model.intercept_[0],
157         model.coef_[0][0],
158         mean_squared_error(ytest, ypred)))
159
160 print('\n\n')
161
162 y_vals = model.intercept_ + X * model.coef_
163 plt.plot(X, y_vals, label='{:.2f}'.format(
164     mean_squared_error(ytest, ypred)))
165
166 vals = pd.DataFrame({
167     'intercept': intercept,
168     'slope': slope,
169     'mean_error': mean_error})
170
171 plt.scatter(X, y, c='black')
172 plt.xlabel('Input X')
173 plt.ylabel('Target y')
174 plt.legend(title="MSE", fancybox=True)
175 plt.show();
176
177 import numpy as np
178 arr = np.arange(3.0, 5.5, 0.05)
179
180 # LAGGED OUTPUT
181 import pandas as pd
182 data = pd.DataFrame({
183     'Date': ['2024-07-15',
184         '2024-08-15',
185         '2024-09-15',
186         '2024-10-15',
187         '2024-11-15',
188         '2024-12-15',
189         ],
190     'Product Usage': [ 1000,
191         2000,
192         4000,
193         8000,
194         16000,
195         32000
196         ],
197     'tag_1': [ None,
198         1000,
199         2000,
200         4000,
201         8000,
202         16000,
203         ],
204     'tag_2': [None,
205         None,
206         1000,
207         2000,
208         4000,
209         8000,
210         ],
211     })
212 print(data)

```