Final Assignment: AIRBNB Review Prediction Ketan Patil (22303876)

Introduction:

As part of our final assignment, we were asked to predict score for Airbnb listings in Dublin City under various categories like cleanliness, check-in, location, among others. I have downloaded data from Airbnb website (listings.csv and reviews.csv). I have used various Feature Engineering techniques to get meaningful features from the large dataset. Then, I performed Feature Selection on the data to get most impactful features and then used these selected features to train models. Details of each of these steps are mentioned in the following paragraphs.

Features Engineering on Reviews Data:

To perform feature engineering on the reviews data, I have followed below steps.

Step 1: Cleaning the comments and translating the comments to English

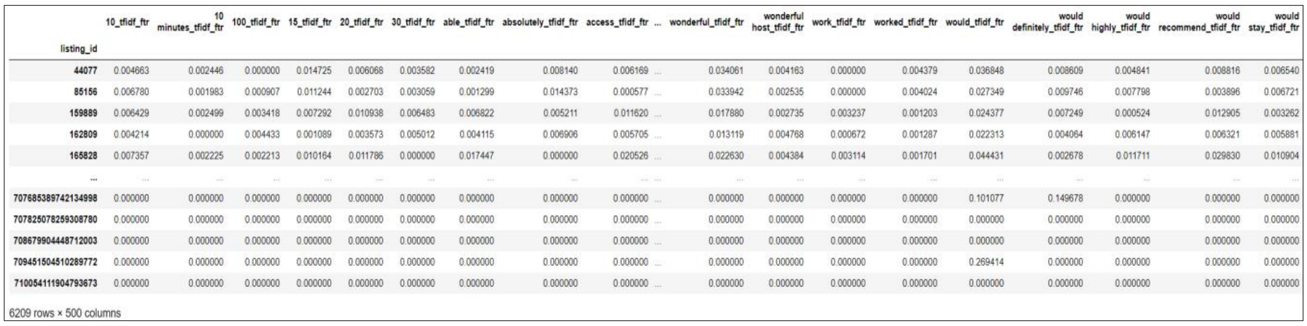
The reviews.csv file contained reviews in many languages and these reviews also had many special characters. To clean the reviews, I have first removed all the special tags from reviews data like **'</br>', '\n', '\r', '\t'**. Then, I removed all special characters from the data like emojis. To achieve this, I have used **clean-text** library and used **clean** function of this library with **no-emoji** parameter set to **True** to remove any special character from dataset. I created 2 lambda expressions and applied them on comments column and saved the result in a new column **clean_comments.**

Now, to translate the clean comments to English language, I have used **GoogleTranslator** function from **deep-translator** library and set it parameters like **source** to **auto** to autodetect the language of the review and **target** to **en** to translate the reviews to English. The results were stored in a new column **translated_comments** and final dataset was stored in **reviews_translated.csv** for further processing.

Step 2: Extracting features from translated 3reviews using TF-IDF approach and Sentiment Analysis

I loaded the translated comments into a data-frame and replaced all the null values with **none** text before extracting features. Furthermore, I have also removed all the common English Stop words like a, an, the, in, etc. from the comments. Primary reason for removing stop words is that stop words occur very frequently in the sentences and therefore, TF_IDF approach, which is based on the frequency of words, can assign higher weights to these words rather than other significant words. This can result in stop words having higher influence on the overall outcome which negatively impacts performance. The cleaned reviews were stored in **tf_idf_reviews** column.

Then, I performed **TF-IDF analysis** on the data and extracted 500 most significant features from the comments. The primary reason for choosing TF-IDF approach for feature extraction is that it normalises the words count taking into consideration the number of documents it appears in. This helps to reduce the undue significance of words that occur frequently in one document but not in other. Secondly, it also makes the rare words occurring in the reviews more significant than the common words, which is very important for extracting significant features. I have extracted both **unigram and bigram features** from the reviews to get a diverse feature space. Once, features were extracted for each review, I have then grouped the reviews based on **listing_id** and taken mean of the scores for the feature extracted for each listing. Below is the feature set:

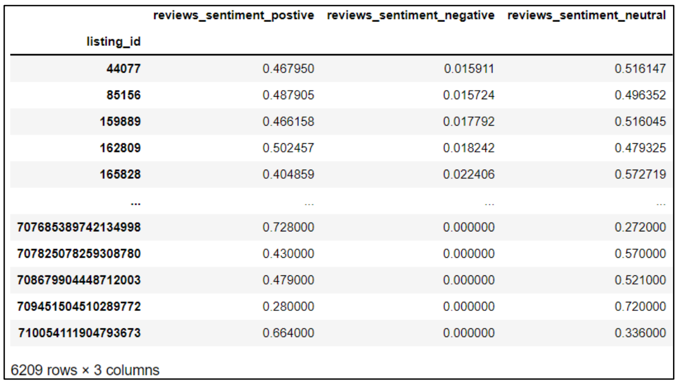


listing_id	10_tfidf_ftr	10 minutes_tfidf_ftr	100_tfidf_ftr	15_tfidf_ftr	20_tfidf_ftr	30_tfidf_ftr	able_tfidf_ftr	absolutely_tfidf_ftr	access_tfidf_ftr	...	wonderful_tfidf_ftr	wonderful host_tfidf_ftr	work_tfidf_ftr	worked_tfidf_ftr	would_tfidf_ftr	would definitely_tfidf_ftr	would highly_tfidf_ftr	would recommend_tfidf_ftr	would stay_tfidf_ftr
44077	0.004663	0.002446	0.000000	0.014725	0.006068	0.003582	0.002419	0.008140	0.006169	...	0.034061	0.004163	0.000000	0.004379	0.036848	0.008609	0.004841	0.008816	0.006540
85156	0.006780	0.001983	0.000907	0.011244	0.002703	0.003059	0.001299	0.014373	0.000577	...	0.033942	0.002535	0.000000	0.004024	0.027349	0.009746	0.007798	0.003896	0.006721
159889	0.006429	0.002499	0.003418	0.007292	0.010938	0.006483	0.006822	0.005211	0.011620	...	0.017880	0.002735	0.003237	0.001203	0.024377	0.007249	0.000524	0.012905	0.003262
162809	0.004214	0.000000	0.004433	0.001089	0.003573	0.005012	0.004115	0.006906	0.005705	...	0.013119	0.004768	0.000672	0.001287	0.022313	0.004064	0.006147	0.006321	0.005881
165828	0.007357	0.002225	0.002213	0.010164	0.011786	0.000000	0.017447	0.000000	0.020526	...	0.022630	0.004384	0.003114	0.001701	0.044431	0.002678	0.011711	0.029830	0.010904
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
707685389742134998	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.101077	0.149678	0.000000	0.000000	0.000000
707825078259308780	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
708679904448712003	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
709451504510289772	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.269414	0.000000	0.000000	0.000000	0.000000
710054111904793673	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

6209 rows × 500 columns

Figure 1: TF-IDF Features extracted for each listing based on user reviews

Then, I performed **Sentiment Analysis** using Vader library on the reviews and extracted the Positive, Negative and Neutral sentiment scores as features for each review. The primary reason for performing sentiment analysis is that we will be able to check if the sentiment of the review has any impact on overall score prediction. Once sentiment features were extracted for each review, I grouped them on the basis of **listing_id** and took mean of scores for each listing.

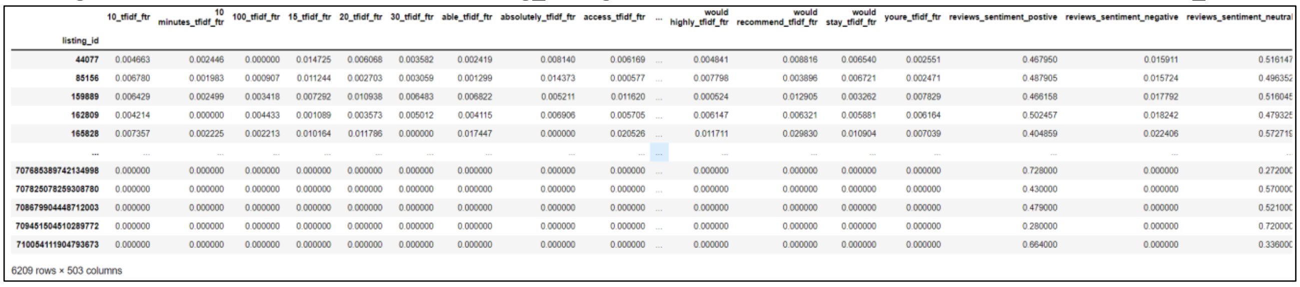


listing_id	reviews_sentiment_postive	reviews_sentiment_negative	reviews_sentiment_neutral
44077	0.467950	0.015911	0.516147
85156	0.487905	0.015724	0.496352
159889	0.466158	0.017792	0.516045
162809	0.502457	0.018242	0.479325
165828	0.404859	0.022406	0.572719
...	...	...	...
707685389742134998	0.728000	0.000000	0.272000
707825078259308780	0.430000	0.000000	0.570000
708679904448712003	0.479000	0.000000	0.521000
709451504510289772	0.280000	0.000000	0.720000
710054111904793673	0.664000	0.000000	0.336000

6209 rows × 3 columns

Figure 2: Sentiment Features extracted for each listing based on user reviews

Then, I merged these 2 datasets on the basis of listing_id to get final feature set for reviews and stored it in **review_features.csv**.



listing_id	10_tfidf_ftr	10 minutes_tfidf_ftr	100_tfidf_ftr	15_tfidf_ftr	20_tfidf_ftr	30_tfidf_ftr	able_tfidf_ftr	absolutely_tfidf_ftr	access_tfidf_ftr
44077	0.004663	0.002446	0.000000	0.014725	0.006068	0.003582	0.002419	0.008140	0.006169
85156	0.006780	0.001983	0.000907	0.011244	0.002703	0.003059	0.001299	0.014373	0.000577
159889	0.006429	0.002499	0.003418	0.007292	0.010938	0.006483	0.006822	0.005211	0.011620
162809	0.004214	0.000000	0.004433	0.001089	0.003573	0.005012	0.004115	0.006906	0.005705
165828	0.007357	0.002225	0.002213	0.010164	0.011786	0.000000	0.017447	0.000000	0.020526
...	...	...	...	...	...	...	...	...	...
707685389742134998	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
707825078259308780	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
708679904448712003	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
709451504510289772	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
710054111904793673	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

listing_id	...	would highly_tfidf_ftr	would recommend_tfidf_ftr	would stay_tfidf_ftr	youre_tfidf_ftr	reviews_sentiment_postive	reviews_sentiment_negative	reviews_sentiment_neutral
44077	...	0.004841	0.008816	0.006540	0.002551	0.467950	0.015911	0.516147
85156	...	0.007798	0.003896	0.006721	0.002471	0.487905	0.015724	0.49635
159889	...	0.000524	0.012905	0.003262	0.007829	0.466158	0.017792	0.51604
162809	...	0.006147	0.006321	0.005881	0.006164	0.502457	0.018242	0.47932
165828	...	0.011711	0.029830	0.010904	0.007039	0.404859	0.022406	0.57271
...	...	...	...	...	...	...	...	...
707685389742134998	...	0.000000	0.000000	0.000000	0.000000	0.728000	0.000000	0.27200
707825078259308780	...	0.000000	0.000000	0.000000	0.000000	0.430000	0.000000	0.57000
708679904448712003	...	0.000000	0.000000	0.000000	0.000000	0.479000	0.000000	0.52100
709451504510289772	...	0.000000	0.000000	0.000000	0.000000	0.280000	0.000000	0.72000
710054111904793673	...	0.000000	0.000000	0.000000	0.000000	0.664000	0.000000	0.33600

6209 rows × 503 columns

Figure 3: Final Features Set for reviews data

Features Engineering on Listings Data:

I have used various feature engineering techniques on the Listings dataset. These are based on type of data and Each of these are mentioned below:

1. Dropping Columns

Column Names	Reason for dropping
listing_url, picture_url, host_url, host_thumbnail_url, host_picture_url	These columns contain URLs to various Airbnb webpages related to the listing; therefore these are nor relevant features for the review scores.
scrape_id, source	These columns contain text which is not relevant to review score and no further features could be extracted from these fields.
host_id, host_name, host_location, host_location, host_neighbourhood, host_since	These columns contain information related to Host which are insignificant features for overall review scores and no further analysis like sentiment analysis etc. could be performed on these columns.
last_scraped, first_review, last_review, calendar_last_scraped	These columns contained various dates which I have not included in final feature set as no time-series related analysis could be performed using these fields.
calendar_updated, license, neighbourhood_group_cleansed, bathrooms	These are blank columns and therefore, I have removed these from final feature set
latitude, longitude	This positional information could not be used further for extracting any feature and therefore dropped
neighbourhood	I have dropped this column as another column neighbourhood_cleansed was provided in the data and it was included in the final feature set

2. Sentiment analysis on Text columns

Column Names	Reason for Sentiment Analysi
name, description	These columns contain information about the property listed on Airbnb. Therefore, I have performed Sentiment Analysis on these columns to extract sentiment related features from these columns. We will further check if these have any impact on overall score prediction
neighborhood_overview	This column contains information about the listed property's surroundings. I have used sentiment analysis to extract features to check if neighbourhood related sentiment scores have impact on overall predictions
host_about	This column contains information about the host of listed property. Again, I have used Sentiment analysis to check if scores related to description of host have any impact on overall predictions

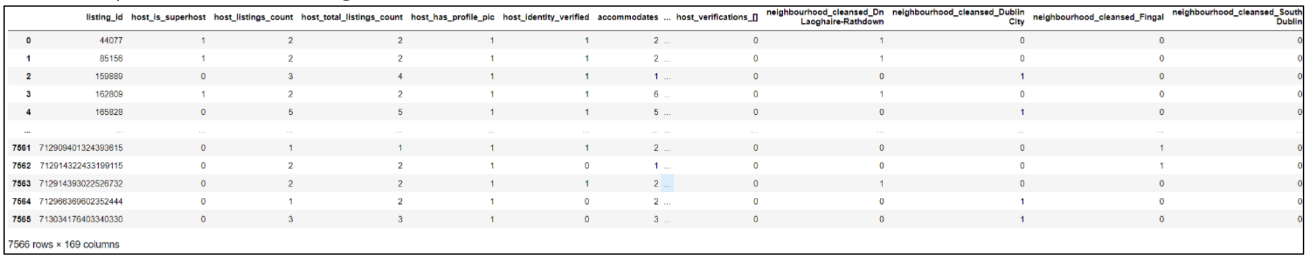
3. Converting columns to numeric columns

I have converted these columns to float values: host_response_rate, host_acceptance_rate and price. I have first removed text and special character like $, %, etc. from these columns and then converted it to float columns host_response_rate_float, host_acceptance_rate_float, price_float.

4. **Encoding T/F values to 1/0 value**
   host_is_superhost, host_has_profile_pic, host_identity_verified, has_availability ,instant_bookable columns were represented as t/f values. I have converted these to integer columns represented by 1/0 values
5. **Replacing NA values in numeric columns with mean value of that column**
   For the below mentioned columns, I have replaced the N/A values with the mean of the values in that column. These were already numeric columns and therefore conversion was not required.
   host_listings_count, host_total_listings_count, accommodates, bedrooms, beds, minimum_nights, maximum_nights, minimum_minimum_nights, maximum_minimum_nights, minimum_maximum_nights, maximum_maximum_nights, minimum_nights_avg_ntm, maximum_nights_avg_ntm, availability_30, availability_60, availability_90, availability_365, number_of_reviews, number_of_reviews_ltm, number_of_reviews_l30d, calculated_host_listings_count, calculated_host_listings_count_entire_homes, calculated_host_listings_count_private_rooms, calculated_host_listings_count_shared_rooms, reviews_per_month.

6. **One-Hot encoding on categorical columns**
   property_type, room_type, host_response_time, bathrooms_text, host_verifications, neighbourhood_cleansed columns in the data had categorical data. Therefore, I have used One-Hot encoding technique to convert these columns to ordinal columns as we cannot use text data to train the models and this technique is used to convert these categorical text data to numerical data that can be used to train machine learning model.
7. **Amenities Column**
   This column contained list of amenities provided in the property. To check if amenities have impact on user's rating, I have converted this data into numeric column by counting the number of amenities provided in the property and storing this count in a new column **count_amenities.**

Below is the representation of listing data:



	listing_id	host_is_superhost	host_listings_count	host_total_listings_count	host_has_profile_pic	host_identity_verified	accommodates	...	host_verifications_[]	neighbourhood_cleansed_Dn Laoghaire-Rathdown	neighbourhood_cleansed_Dublin City	neighbourhood_cleansed_Fingal	neighbourhood_cleansed_South Dublin
0	44077	1	2	2	1	1	2	...	0	1	0	0	0
1	85156	1	2	2	1	1	2	...	0	1	0	0	0
2	159889	0	3	4	1	1	1	...	0	0	1	0	0
3	162809	1	2	2	1	1	6	...	0	1	0	0	0
4	165828	0	5	5	1	1	5	...	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
7561	712909401324393615	0	1	1	1	1	2	...	0	0	0	1	0
7562	712914322433199115	0	2	2	1	0	1	...	0	0	0	1	0
7563	712914393022526732	0	2	2	1	1	2	...	0	1	0	0	0
7564	712966369602352444	0	1	2	1	0	2	...	0	0	1	0	0
7565	713034176403340330	0	3	3	1	0	3	...	0	0	1	0	0

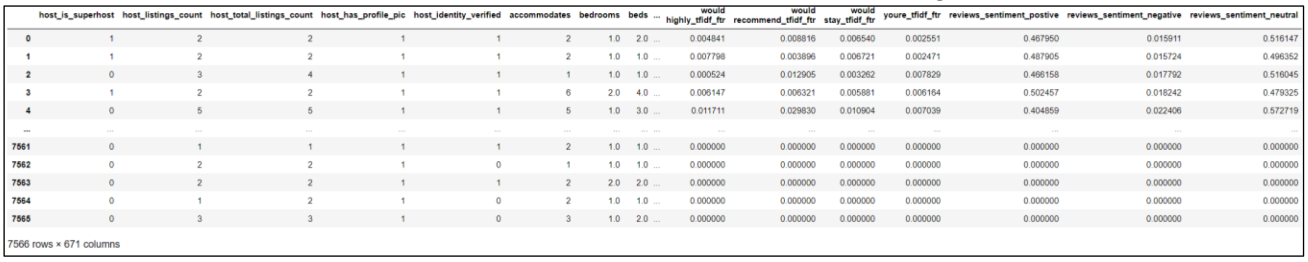
7566 rows × 169 columns

Figure 4: Final Features Set for listings data

I have also **renamed id** column to **listing_id** so that it can be utilised to merge data later. Finally, all these listing features were stored in the **listing_features.csv** file

Final Features Dataset:

Before performing feature selection, I have merged the data contained in **reviews_features.csv** and **listing_features.csv** based on **listing_id** column. Then, I dropped the **listing_id** column from the features set as this does not contain any meaningful information. Below is the final features dataset that will be used for feature selection and model training.



	host_is_superhost	host_listings_count	host_total_listings_count	host_has_profile_pic	host_identity_verified	accommodates	bedrooms	beds	...	would highly_tfidf_ftr	would recommend_tfidf_ftr	would stay_tfidf_ftr	youre_tfidf_ftr	reviews_sentiment_postive	reviews_sentiment_negative	reviews_sentiment_neutral
0	1	2	2	1	1	2	1.0	2.0	...	0.004841	0.008816	0.006540	0.002551	0.467950	0.015911	0.516147
1	1	2	2	1	1	2	1.0	1.0	...	0.007798	0.003896	0.006721	0.002471	0.487905	0.015724	0.496352
2	0	3	4	1	1	1	1.0	1.0	...	0.000524	0.012905	0.003262	0.007829	0.466158	0.017792	0.516045
3	1	2	2	1	1	6	2.0	4.0	...	0.006147	0.006321	0.005881	0.006164	0.502457	0.018242	0.479325
4	0	5	5	1	1	5	1.0	3.0	...	0.011711	0.029830	0.010904	0.007039	0.404859	0.022406	0.572719
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7561	0	1	1	1	1	2	1.0	1.0	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
7562	0	2	2	1	0	1	1.0	1.0	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
7563	0	2	2	1	1	2	2.0	2.0	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
7564	0	1	2	1	0	2	1.0	1.0	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
7565	0	3	3	1	0	3	1.0	2.0	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

7566 rows × 671 columns

Figure 5: Final Features Set containing listings features and reviews features

I have also used **MinMax feature scaling** technique to normalise the features and rating scores. I have used tis normalisation technique because it retains the relationship that existed between the variables even after the normalisation process. Secondly, due to normalisation, the standard deviations in the dataset also reduces which further reduces the impact of outliers in the data. This also lessens the biasness in the data.

Selected Baseline Model and 2 Other Models and Metric for Performance comparison:

Baseline Model: Linear Regression model

Linear regression is simple model that is used to identify and represent the relationship between the independent variable or features in the data with dependent variable or label in the data. This model is represented as a straight line drawn in the feature space in such a way that it minimises the difference between the predicted and actual values of the dependent variable.

I have chosen this as a baseline model because this model is very easy to train. It does not require any hyper-parameter tuning and it is very simple to interpret the results. Therefore, this makes it easy to compare other complex model's performance with this model and perform analysis of the results.

Model 1: Lasso Regressor

Lasso regressor is a very simple linear model that uses the Shrinkage technique to shrink the values of the coefficients in the equation towards a central value. This is done in order to achieve higher accuracy of the predicted values.

I have chosen this model because it provides a regularisation factor or penalty parameter called alpha, which can be used to tune the performance of this model. This parameter is used to eliminate the non-significant features which helps to improve the overall prediction accuracy. Secondly, this is a linear model and due to elimination, the model becomes very simple with very few features and results are easy to interpret.

Model 2: KNN Regressor

KNN Regressor uses features to in the vicinity to predict the value for a new datapoint. This means that it uses the features, also called neighbours, present near the new data point and calculated the mean of those features to the new datapoint. The number of neighbours to be used for calculating the new datapoint value can be specified.

I have chosen this model because this model can be used to capture the non-linearity in the features as it does not fit a straight line rather uses the average of nearby points to reduce the prediction error. Secondly, it also allows to tune the number of neighbours that are to be used, therefore allowing us to improve the overall model's accuracy.

Metric used for Performance Comparison: 5-fold cross validation with Mean Square Error

I have used 5-fold cross validation score as benchmark score for the evaluation of all these models for different types of prediction like value, cleanliness etc.

The primary reason for using this technique is that it uses every part of the input dataset to train as well as test the model's performance. Due to this, the effect of overfitting and underfitting of the data in the model is reduced significantly. It also results in better generalisation of the model as more data is used to train the model, therefore it adapts well to unseen data. I have used **Mean Square Error** as Scoring strategy as it explains the closeness the predicted values to the actual value. Therefore, it is an ideal strategy for regression problem.

Features Selections and Models Tuning for Review Score Rating:

I have trained Lasso regression model using 5-fold cross validation to eliminate the insignificant features from the dataset. I have tried multiple ranges for L1 regularisation parameter to tune the Lasso model and generated plots for MSE for each range of hyperparameter. Based on plots, I selected the parameter for which I got the least mean square error.

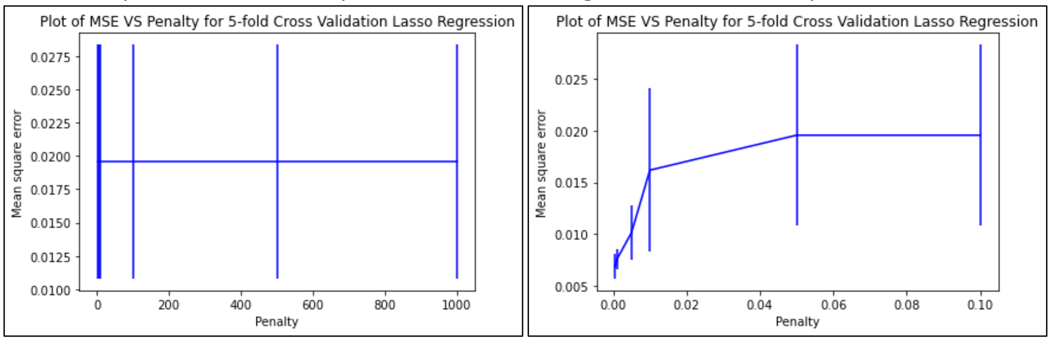
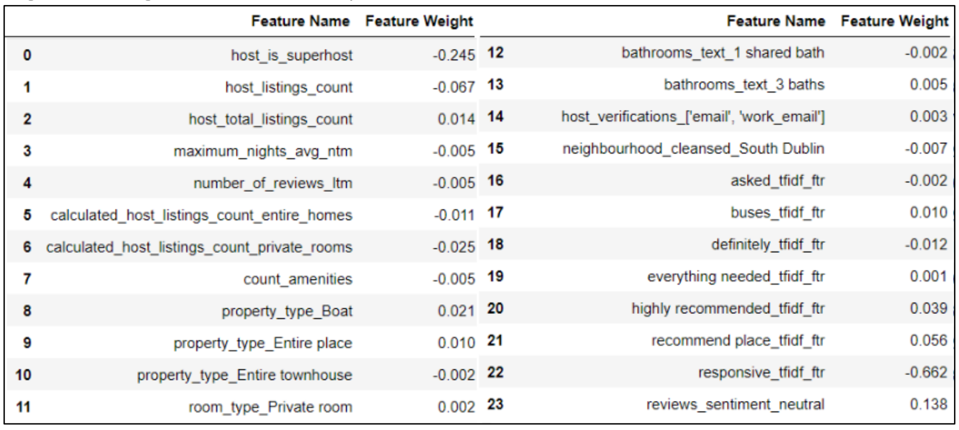


Figure 6: Plot of MSE vs C for C range [1, 5, 10, 100, 500, 1000] and [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]

Based on Figure 6, I can analyse the I got least MSE for alpha=0.0005. Therefore, this was used to identify the significant features for Review Score Rating. Following are the non-zero parameters identified:



	Feature Name	Feature Weight		Feature Name	Feature Weight
0	host_is_superhost	-0.245	12	bathrooms_text_1 shared bath	-0.002
1	host_listings_count	-0.067	13	bathrooms_text_3 baths	0.005
2	host_total_listings_count	0.014	14	host_verifications_['email', 'work_email']	0.003
3	maximum_nights_avg_ntm	-0.005	15	neighbourhood_cleansed_South Dublin	-0.007
4	number_of_reviews_ltm	-0.005	16	asked_tfidf_ftr	-0.002
5	calculated_host_listings_count_entire_homes	-0.011	17	buses_tfidf_ftr	0.010
6	calculated_host_listings_count_private_rooms	-0.025	18	definitely_tfidf_ftr	-0.012
7	count_amenities	-0.005	19	everything needed_tfidf_ftr	0.001
8	property_type_Boat	0.021	20	highly recommended_tfidf_ftr	0.039
9	property_type_Entire place	0.010	21	recommend place_tfidf_ftr	0.056
10	property_type_Entire townhouse	-0.002	22	responsive_tfidf_ftr	-0.662
11	room_type_Private room	0.002	23	reviews_sentiment_neutral	0.138

Figure 7: Significant parameters identified using Lasso Regression

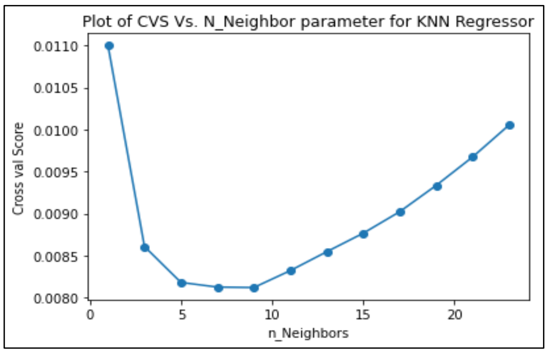
I can clearly identify that lasso has identified some very interesting features that impacts the overall rating of the property and these add meaning to how user rates a listing on Airbnb. For Instance, **host is super-host** is identified as significant along with

number of reviews in last 12 month and **number of amenities**. This plays a very important role when a user rates a listing. Furthermore, **type of property and type of rooms** are also significant. From the comments I can identify words like **responsive, highly recommend, everything needed, recommended place** are also significant which also impacts user's overall rating for property

I have used these features to first train the Baseline Linear Regression Model. **For baseline model, the Cross Validation Score (absolute of negative MSE) is: 0.01676**

Then, I trained Lasso regression model using the tuned hyperparameter 0.0005. **The Cross Validation Score (absolute of negative MSE) for Lasso Regression model is: 0. 01691**

Furthermore, the I trained KNN Regressor multiple time to tune the hyperparameter. The following is the plot of hyperparameter n_neighbors vs MSE



I can identify that for **n=9** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0081.**

Therefore, we can conclude KNN Regressor performs better than the baseline and Lasso Regressor therefore it can be used to predict the Review Score Rating for Airbnb Listing.

Features Selections and Models Tuning for Review Score Accuracy:

For Review Score Accuracy, I again used Lasso Regression to identify important parameters. I generated below plots for hyperparameter tuning.

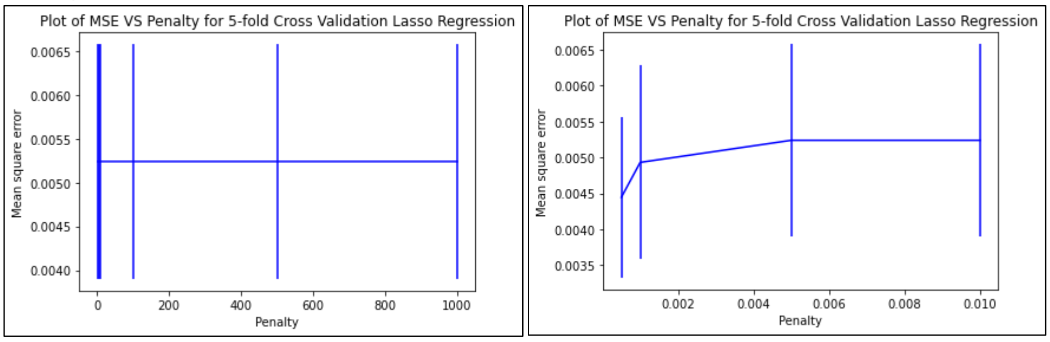
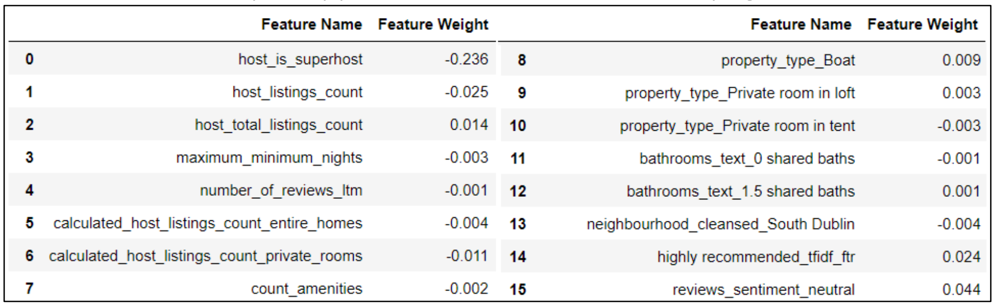


Figure 8: Plot of MSE vs C for C range [1, 5, 10, 100, 500, 1000] and [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]

From these plots I have chosen 0.0005 as penalty parameter and used this to identify significant features.



	Feature Name	Feature Weight
0	host_is_superhost	-0.236
1	host_listings_count	-0.025
2	host_total_listings_count	0.014
3	maximum_minimum_nights	-0.003
4	number_of_reviews_ltm	-0.001
5	calculated_host_listings_count_entire_homes	-0.004
6	calculated_host_listings_count_private_rooms	-0.011
7	count_amenities	-0.002
8	property_type_Boat	0.009
9	property_type_Private room in loft	0.003
10	property_type_Private room in tent	-0.003
11	bathrooms_text_0 shared baths	-0.001
12	bathrooms_text_1.5 shared baths	0.001
13	neighbourhood_cleansed_South Dublin	-0.004
14	highly recommended_tfidf_ftr	0.024
15	reviews_sentiment_neutral	0.044

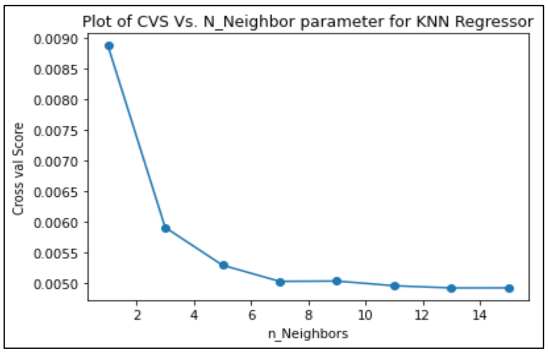
Figure 9: Significant parameters identified using Lasso Regression

For accuracy rating for listing on Airbnb, **host is super-host** is identified as significant along with **number of reviews in the last 12 month** and **number of amenities**. This plays a very important role when a user rates accuracy of a listing. Furthermore, **type of property and number of bathrooms** are also significant as user tends to give higher accuracy rating if these are accurately listed. Furthermore, neighbourhood of South Dublin is also significant for accuracy score.

For baseline model, the Cross Validation Score (absolute of negative MSE) is: 0.0074

For **Lasso regression** model using the tuned hyperparameter 0.0005 **the Cross Validation Score (absolute of negative MSE) for Lasso Regression model is: 0.0051**

For **KNN Regressor**, the following is the plot of hyperparameter n_neighbors vs MSE



I can identify that for **n=7** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0050.**

Therefore, we can conclude KNN Regressor performs better than the baseline and Lasso Regressor therefore it can be used to predict the Accuracy Score Rating for Airbnb Listing.

Features Selections and Models Tuning for Review Score Location:

For Review Score Location, I used Lasso Regression to identify important parameters. I generated below plots for hyperparameter tuning.

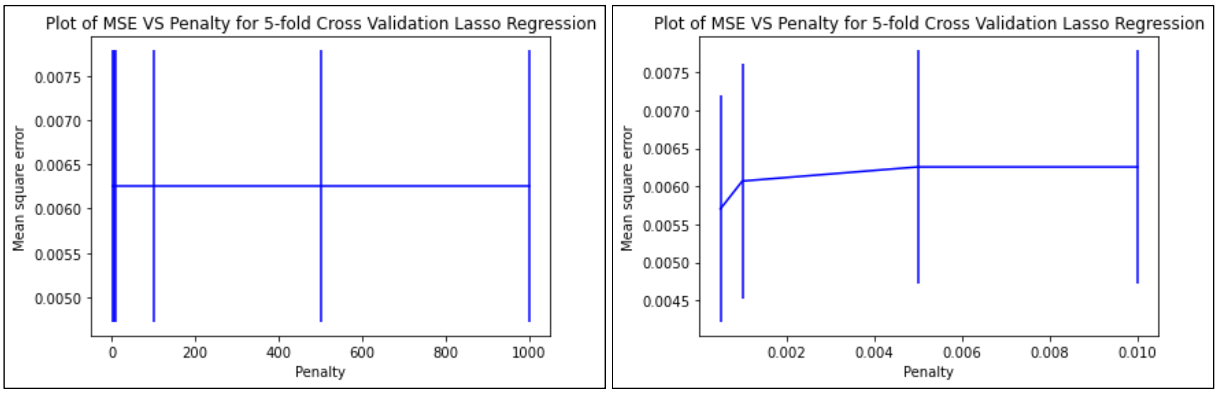
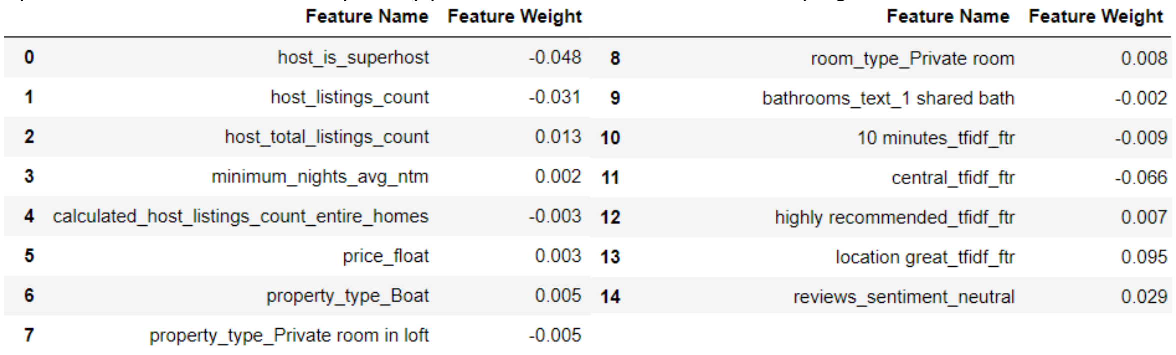


Figure 10: Plot of MSE vs C for C range [1, 5, 10, 100, 500, 1000] and [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]

From these plots I have chosen 0.0005 as penalty parameter and used this to identify significant features.



	Feature Name	Feature Weight		Feature Name	Feature Weight
0	host_is_superhost	-0.048	**8**	room_type_Private room	0.008
1	host_listings_count	-0.031	**9**	bathrooms_text_1 shared bath	-0.002
2	host_total_listings_count	0.013	**10**	10 minutes_tfidf_ftr	-0.009
3	minimum_nights_avg_ntm	0.002	**11**	central_tfidf_ftr	-0.066
4	calculated_host_listings_count_entire_homes	-0.003	**12**	highly recommended_tfidf_ftr	0.007
5	price_float	0.003	**13**	location great_tfidf_ftr	0.095
6	property_type_Boat	0.005	**14**	reviews_sentiment_neutral	0.029
7	property_type_Private room in loft	-0.005			

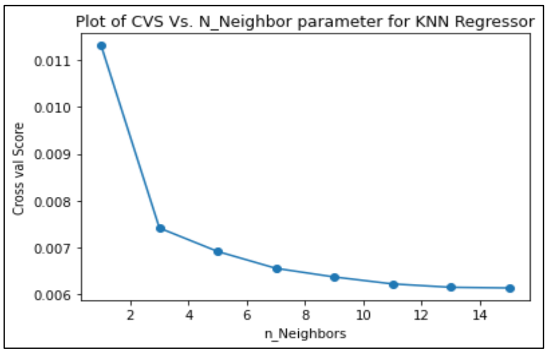
Figure 11: Significant parameters identified using Lasso Regression

For location rating, **host is super-host** is identified as significant along with price. From reviews we can see that keywords like **central, location great** were also considered significant as these describe the location of the listing.

For baseline model, the Cross Validation Score (absolute of negative MSE) is: 0.0081

For **Lasso regression** model using the tuned hyperparameter 0.0005 **the Cross Validation Score (absolute of negative MSE) for Lasso Regression model is: 0.0062**

For **KNN Regressor**, the following is the plot of hyperparameter n_neighbors vs MSE



I can identify that for **n=13** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0061.**

Therefore, we can conclude KNN Regressor performs better than the baseline and Lasso Regressor therefore it can be used to predict the Cleanliness Score Rating for Airbnb Listing.

Features Selections and Models Tuning for Review Score Check-in:

For Review Score Check-in, I generated below plots for hyperparameter tuning.

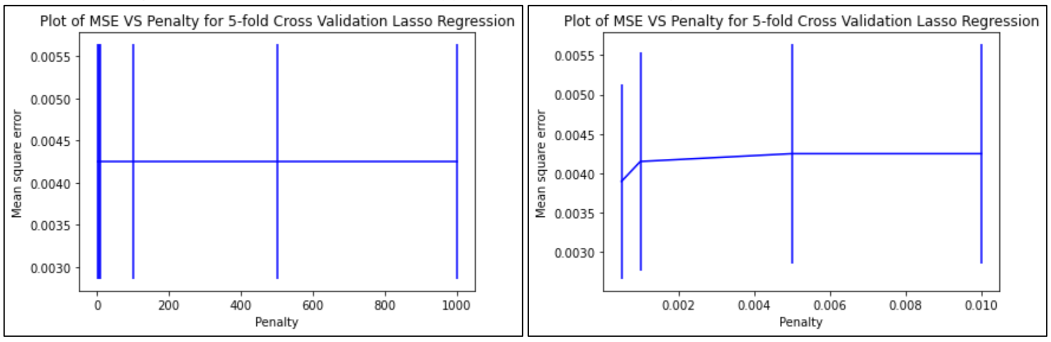
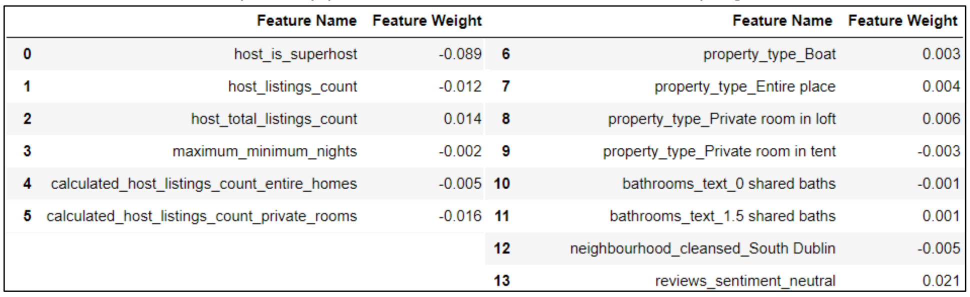


Figure 12: Plot of MSE vs C for C range [1, 5, 10, 100, 500, 1000] and [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]

From these plots I have chosen 0.0005 as penalty parameter and used this to identify significant features.



	Feature Name	Feature Weight		Feature Name	Feature Weight
0	host_is_superhost	-0.089	**6**	property_type_Boat	0.003
1	host_listings_count	-0.012	**7**	property_type_Entire place	0.004
2	host_total_listings_count	0.014	**8**	property_type_Private room in loft	0.006
3	maximum_minimum_nights	-0.002	**9**	property_type_Private room in tent	-0.003
4	calculated_host_listings_count_entire_homes	-0.005	**10**	bathrooms_text_0 shared baths	-0.001
5	calculated_host_listings_count_private_rooms	-0.016	**11**	bathrooms_text_1.5 shared baths	0.001
			12	neighbourhood_cleansed_South Dublin	-0.005
			13	reviews_sentiment_neutral	0.021

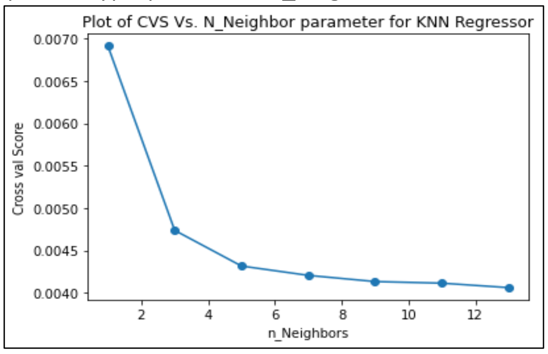
Figure 13: Significant parameters identified using Lasso Regression

For check-in rating, **host is super-host** is identified as significant as super-host ensure smooth check-in. Furthermore, **property type** is also considered significant.

For baseline model, the Cross Validation Score (absolute of negative MSE) is: 0.0056

For **Lasso regression** model using the tuned hyperparameter 0.0005 **the Cross Validation Score (absolute of negative MSE) for Lasso Regression model is: 0.0040**

For **KNN Regressor**, the following is the plot of hyperparameter n_neighbors vs MSE



I can identify that for **n=9** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0041.**

Therefore, we can conclude Lasso Regressor performs better than the baseline and KNN Regressor therefore it can be used to predict the Check-in Score Rating for Airbnb Listing.

Features Selections and Models Tuning for Review Score Cleanliness:

For Review Score Cleanliness, I generated below plots for hyperparameter tuning.

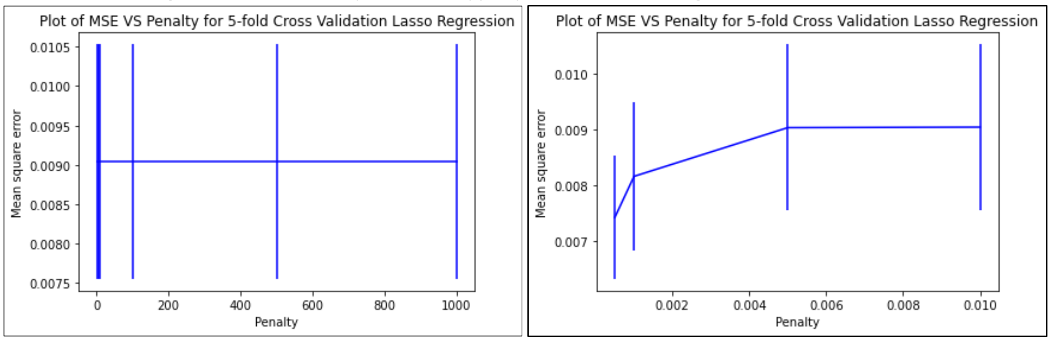
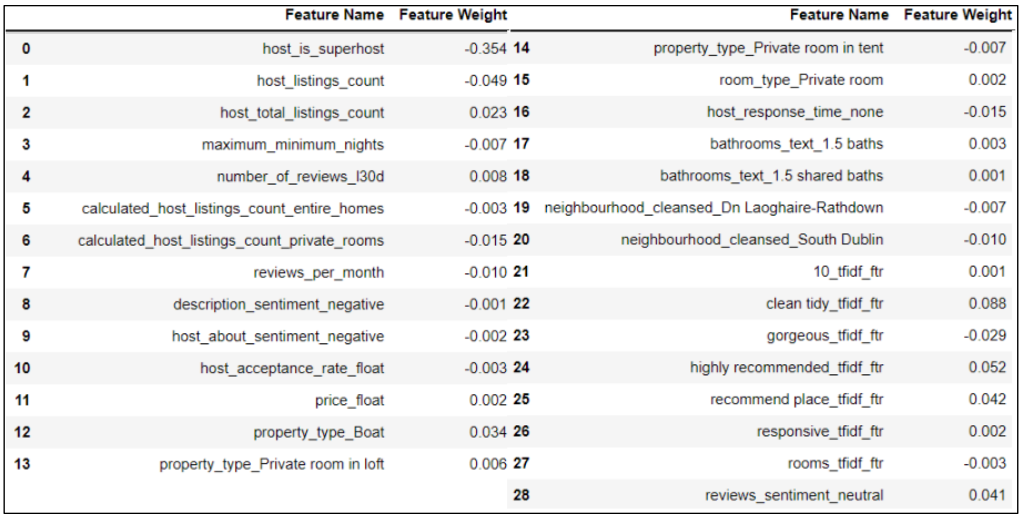


Figure 14: Plot of MSE vs C for C range [1, 5, 10, 100, 500, 1000] and [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]

From these plots I have chosen 0.0005 as penalty parameter and used this to identify significant features.



	Feature Name	Feature Weight		Feature Name	Feature Weight
0	host_is_superhost	-0.354	14	property_type_Private room in tent	-0.007
1	host_listings_count	-0.049	15	room_type_Private room	0.002
2	host_total_listings_count	0.023	16	host_response_time_none	-0.015
3	maximum_minimum_nights	-0.007	17	bathrooms_text_1.5 baths	0.003
4	number_of_reviews_l30d	0.008	18	bathrooms_text_1.5 shared baths	0.001
5	calculated_host_listings_count_entire_homes	-0.003	19	neighbourhood_cleansed_Dn Laoghaire-Rathdown	-0.007
6	calculated_host_listings_count_private_rooms	-0.015	20	neighbourhood_cleansed_South Dublin	-0.010
7	reviews_per_month	-0.010	21	10_tfidf_ftr	0.001
8	description_sentiment_negative	-0.001	22	clean tidy_tfidf_ftr	0.088
9	host_about_sentiment_negative	-0.002	23	gorgeous_tfidf_ftr	-0.029
10	host_acceptance_rate_float	-0.003	24	highly recommended_tfidf_ftr	0.052
11	price_float	0.002	25	recommend place_tfidf_ftr	0.042
12	property_type_Boat	0.034	26	responsive_tfidf_ftr	0.002
13	property_type_Private room in loft	0.006	27	rooms_tfidf_ftr	-0.003
			28	reviews_sentiment_neutral	0.041

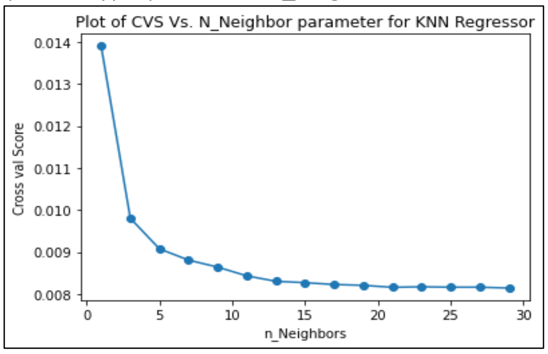
Figure 15: Significant parameters identified using Lasso Regression

For cleanliness rating, **host is super-host** is identified as significant as super-host ensure cleanliness for in their property. Furthermore, **number of reviews in the last 30 days** was considered more significant for cleanliness. I can also analyse that **description sentiment and host about sentiment** were also considered significant for cleanliness. From reviews we can see that keywords like **clean tidy, gorgeous** were also considered significant as these describe the cleanliness of the listing.

For baseline model, the Cross Validation Score (absolute of negative MSE) is: 0.0092

For **Lasso regression** model using the tuned hyperparameter 0.0005 **the Cross Validation Score (absolute of negative MSE) for Lasso Regression model is: 0.0084**

For **KNN Regressor**, the following is the plot of hyperparameter n_neighbors vs MSE



I can identify that for **n=21** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0081.**

Therefore, we can conclude KNN Regressor performs better than the baseline and Lasso Regressor therefore it can be used to predict the Cleanliness Score Rating for Airbnb Listing.

Features Selections and Models Tuning for Review Score Communication:

For Review Score communication, I generated below plots for hyperparameter tuning.

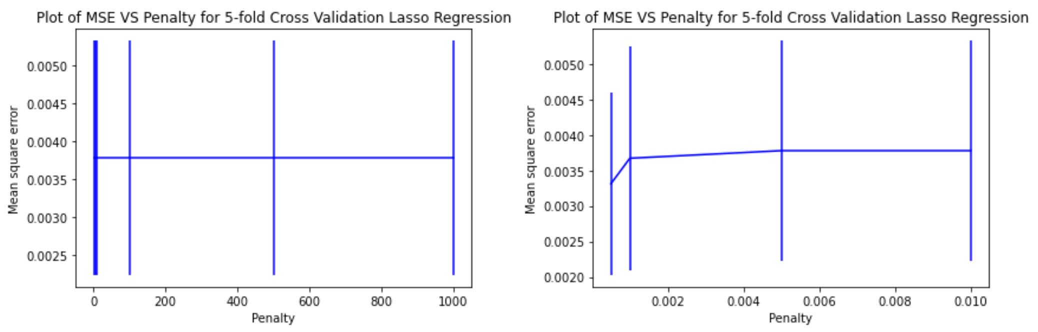
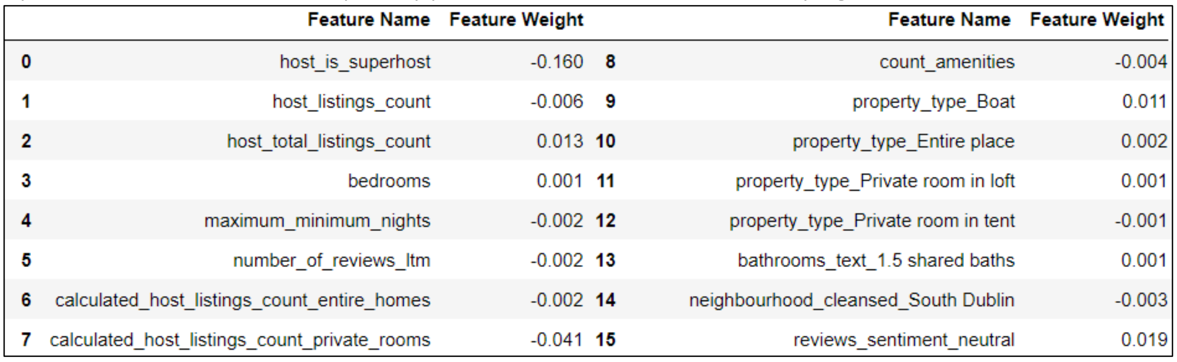


Figure 16: Plot of MSE vs C for C range [1, 5, 10, 100, 500, 1000] and [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]

From these plots I have chosen 0.0005 as penalty parameter and used this to identify significant features.



	Feature Name	Feature Weight		Feature Name	Feature Weight
0	host_is_superhost	-0.160	**8**	count_amenities	-0.004
1	host_listings_count	-0.006	**9**	property_type_Boat	0.011
2	host_total_listings_count	0.013	**10**	property_type_Entire place	0.002
3	bedrooms	0.001	**11**	property_type_Private room in loft	0.001
4	maximum_minimum_nights	-0.002	**12**	property_type_Private room in tent	-0.001
5	number_of_reviews_ltm	-0.002	**13**	bathrooms_text_1.5 shared baths	0.001
6	calculated_host_listings_count_entire_homes	-0.002	**14**	neighbourhood_cleansed_South Dublin	-0.003
7	calculated_host_listings_count_private_rooms	-0.041	**15**	reviews_sentiment_neutral	0.019

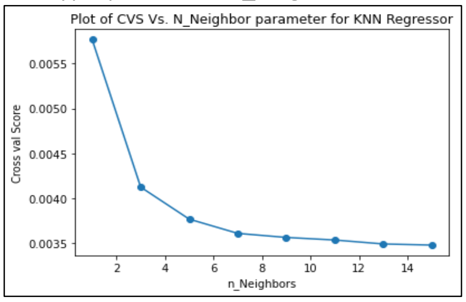
Figure 17: Significant parameters identified using Lasso Regression

For communication rating, **host is super-host** is identified as significant along with **number of reviews in last 12 months**.

For baseline model, the Cross Validation Score (absolute of negative MSE) is: 0.0067

For **Lasso regression** model using the tuned hyperparameter 0.0005 **the Cross Validation Score (absolute of negative MSE) for Lasso Regression model is: 0.0036**

For **KNN Regressor**, the following is the plot of hyperparameter n_neighbors vs MSE



I can identify that for **n=13** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0034.**

Therefore, we can conclude KNN Regressor performs better than the baseline and Lasso Regressor therefore it can be used to predict the Cleanliness Score Rating for Airbnb Listing.

Features Selections and Models Tuning for Review Score Value:

For Review Score value, I generated below plots for hyperparameter tuning.

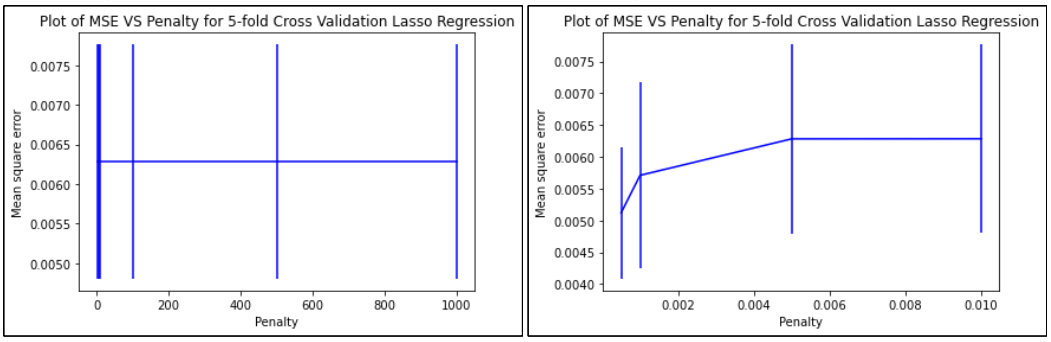
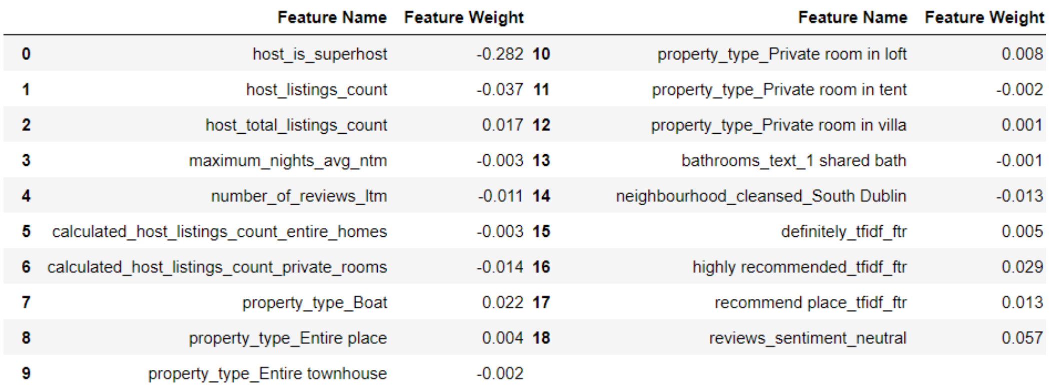


Figure 18: Plot of MSE vs C for C range [1, 5, 10, 100, 500, 1000] and [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]

From these plots I have chosen 0.0005 as penalty parameter and used this to identify significant features.



	Feature Name	Feature Weight
0	host_is_superhost	-0.282
1	host_listings_count	-0.037
2	host_total_listings_count	0.017
3	maximum_nights_avg_ntm	-0.003
4	number_of_reviews_ltm	-0.011
5	calculated_host_listings_count_entire_homes	-0.003
6	calculated_host_listings_count_private_rooms	-0.014
7	property_type_Boat	0.022
8	property_type_Entire place	0.004
9	property_type_Entire townhouse	-0.002
10	property_type_Private room in loft	0.008
11	property_type_Private room in tent	-0.002
12	property_type_Private room in villa	0.001
13	bathrooms_text_1 shared bath	-0.001
14	neighbourhood_cleansed_South Dublin	-0.013
15	definitely_tfidf_ftr	0.005
16	highly recommended_tfidf_ftr	0.029
17	recommend place_tfidf_ftr	0.013
18	reviews_sentiment_neutral	0.057

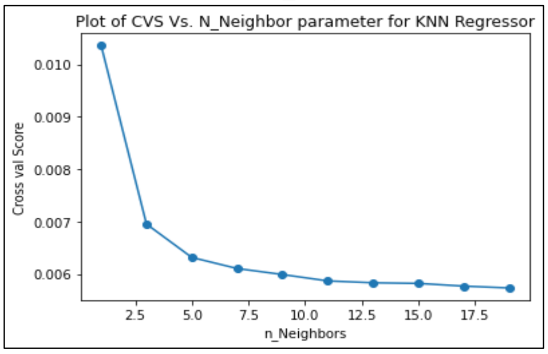
Figure 19: Significant parameters identified using Lasso Regression

For communication rating, **host is super-host** is identified as significant as along with **number of reviews in last 12 months**. **Property type** is significant as type of property is considered value or money. From reviews we can see that keywords like **highly recommend and recommend place** were also considered significant as these describe the overall value of the listing

For baseline model, the Cross Validation Score (absolute of negative MSE) is: 0.0083

For **Lasso regression** model using the tuned hyperparameter 0.0005 **the Cross Validation Score (absolute of negative MSE) for Lasso Regression model is: 0.0061**

For **KNN Regressor**, the following is the plot of hyperparameter n_neighbors vs MSE



I can identify that for **n=11** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0058.**

Therefore, we can conclude KNN Regressor performs better than the baseline and Lasso Regressor therefore it can be used to predict the Value Score Rating for Airbnb Listing.

Results discussion and Conclusion:

From the above results we can clearly see that KNN regressor performs better than the Lasso regressor and Baseline Linear Regression. The primary reason for this is that KNN regressor can capture the non-linearity in the data and adapt itself to this non-linearity. Due to this, the KNN model has generalised well which the other two linear models could not since these models use straight line to fit the model which is not capable enough to capture non-linearity in the data.

We were also able to select important features from the dataset. **Host is super-host was the most prominent feature** that was present for all review score ratings. This makes a lot of sense as super-hosts generally impact the over all user experience on their property and thus impact the overall rating. **Number of reviews** were also considered significant for most of the ratings as higher the reviews, the more it impacts the rating of the listing. **South Dublin Neighbourhood** was also considered significant which means it is popular among visitors for staying. **Property type** along with **number of bathrooms** were also considered significant. From the reviews data, we can clearly see that keywords identified by tf-idf for each of the review score rating were different and were significant for that particular type of rating. For instance, **highly recommended, responsive** for overall rating; **Clean tidy and gorgeous** for cleanliness; **location great and central** for location rating were considered significant which clearly explains that right set of tf-idf features were selected for model training.

The Mean Square error for each of these models is less than 0.01 for all the models which means that models have performed accurately and were able to adapt to the non-linearity in the data.

Question 2

1. **2 situations where logistic regression would five inaccurate predictions**
   Situation 1: The logistic model does not perform particularly well when weak correlation exists between features and labels. The primary reason is that it becomes difficult for Logistic regression model to establish relationship between the features and the labels because of weak correlation. The data points in feature space when plotted are scattered and therefore, it becomes difficult to draw a linear decision surface that can minimise the error in prediction by minimising the distance of points from the decision surface.
   Situation 2: The logistic model assumes that a liner relationship exists between the features and labels therefore, this model will also not perform well if there is non-linearity in data. A prime example of this type of data is time series data. The main reason for under performance is that Logistic regression model tries to fit a linear decision surface in the non-linear space. This will lead to errors in the model's prediction as model will not be able to adapt to the non-linear data and generalise well.

2. **KNN Classifier Vs. Neural Net classifier**
   Advantages of KNN Classifier: The main advantage of KNN Classifier is that it is an intuitive and simple algorithm to implement and it is very easy to interpret the results of this model. This model required very less training data and can adapt to non-linearity in the data and can generalise well to complex features in the dataset. Furthermore, it's sensitivity towards outliers in the data set is an added advantage as it can help to improve overall prediction accuracy.
   Disadvantages of KNN Classifier: The primary disadvantage is that this model becomes computationally very expensive if the dataset is large. The main reason for this is that in KNN model, for each new data point in the dataset, distance is calculated from all the existing points and if dataset is large, the number of calculations performed for each value of training set will increase exponentially. Also, it does not perform well if the dataset is imbalanced as it will predict the majority class most of the times. Furthermore, special care has to be taken to tune the model's hyperparameter as it can lead to overfitting of the training data.

   Advantages of Neural Net Classifier: The main advantage of Neural Net classifier is that it is very flexible model and it can easily adapt to complex non-linear decision boundaries. Since it can infer the hidden complex relationship between the features and labels, these models generalise well to the unseen relationships that exists in the unseen real-world data which improves its performance in real-world scenarios. It also aids this model to adapt to imbalanced as well non-linear data.
   Disadvantages of Neural Net Classifier: Like KNN, this model is also very computationally expensive. This will also depend on the hyperparameter selected for training the model. For instance, more the number of hidden layers in the Neural Net, more computationally expensive it will be. Secondly, this model requires large amount of training data and its performance is directly dependent on the amount of training data provided.

   Hence, we can conclude that both these models are computationally expensive and both are sensitive to the hyperparameter tuning but KNN requires less training data than Neural Net classifier. However, Neural net classifiers can learn the hidden complex relationships in the data and adapt to non-linearity in the data better than KNN classifiers.

3. **K-Fold Cross-validation**
   K-Fold cross validation is resampling technique in which the training data is split into k parts. Then, during each of the k iterations, k-1 parts are used to train the model and remaining one part of the k parts is used for validation. Due to this technique, every part of data was used to train the model, therefore model has more chance the generalise on the data. This will help model to adapt well to the unseen data. Furthermore, model's performance was also validated during each of the k iterations, which can be used to check the effects of overfitting and underfitting of the data in model. This will help us to analyse the performance of model on unseen data.

   K=5, or K=10 is considered as appropriate choice for this technique as these provide adequate amount of data for model to train on as well as test its performance. Too large or too small values of K can lead to underfitting and overfitting problems in the model. For instance, when k is small, for example k=2, this means that model will be trained on only 50% of the data. This will lead to overfitting as the model was trained on very less data, therefore model has very less chance to generalise to the unseen data, thereby decreasing its performance. On the other hand, if K is very large, for instance k=20, model will be trained on 95% of the data and this will lead to underfitting as model is trained on too much data and has very less data to test its performance. Therefore, model has generalised too much on training data and will not adopt well to unseen test data.

   Thus, we can conclude that K=5 or K=10 are ideal parameters to trade-off the impact of overfitting and underfitting of the model on data.

4. **Lagged output values to construct features**

Using Lagged output values as features means that using the output variables from the prior time steps to train the model. This means output of the model for t-1 time step will be used to train the model for t time step. Here we have used only 1 lagged feature, Similarly, we can use any number of time lagged output to construct our new features. For instance, we can use t-3, t-2, t-1 time step output to predict t time step output.

For example, to predict sales of a particular item, we can use the sales data of that item for the previous month. But we can also apply a time lag of 4 months 12 months to check the impact of seasonality on the sale of the product.

Quarterly Sales data for 2021:

Month	Sales
March, 2021	6
June, 2021	12
September, 2021	18
December, 2021	24

Quarterly Sales data for 2022,

Month	Sales
March, 2022	6
June, 2022	12
September, 2022	24
December, 2022	To be predicted

Then, we can time shift sales 2021 data and use it as feature to predict sales for December 2022

Month	Sales 2021	Sales 2022
March, 2022	NA	6
June, 2022	6	12
September, 2022	12	24
December, 2022	24	To be predicted

Based on above, we can clearly see that sales for December 2022 will be 48 units.

Appendix Code

1. TranslateReviewsToEnglish

```
# ## Before executing below code, please execute these commands in console
#
# #### pip install deep-translator
#
# #### pip install unidecode
#
# #### pip install clean-text

# ## Import Libraries
import pandas as pd
from deep_translator import GoogleTranslator
from cleantext import clean
from IPython.display import display

# ## Reading reviews.csv data and loading it in Dataframe
reviews_file = 'data/reviews.csv'
reviews_df = pd.read_csv(reviews_file)
display(reviews_df)
# ## Removing all the unnecessary tags and special characters like emojis
lambda_remove_tags = lambda review : review.replace("<br/>","").replace("\n", "").replace("\r", "").replace("\t", "").strip()
lambda_remove_emoji = lambda review : clean(review, no_emoji=True)
lambda_translate_sentence = lambda review : GoogleTranslator(source='auto', target='en').translate(review)
reviews_df['clean_comments'] = reviews_df['comments'].astype(str).apply(lambda_remove_tags).apply(lambda_remove_emoji)
display(reviews_df)
# ## Translating all comments to English
for i in range(len(reviews_df['clean_comments'])) :
    review = reviews_df['clean_comments'][i]
    try :
        reviews_df.at[i, 'translated_comments'] = lambda_translate_sentence(str(review))
    except:
        print(f'Error occured for i: {i}')
        print(f'Error review: {review} \n\n')
        reviews_df.at[i, 'translated_comments'] = review
display(reviews_df)
# ## Storing final features set in csv
reviews_df.to_csv('data/reviews_translated.csv')
```

2. Feature_Engineering_Reviews.py

```
# ## Import Libraries
import pandas as pd
from nltk.corpus import stopwords
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# ## Reading Translated reviews file reviews_translated.csv data and loading it in Dataframe
reviews_file = 'data/reviews_translated.csv'
transalated_reviews_df = pd.read_csv(reviews_file)
display(transalated_reviews_df)
# ## Removing Unnamed: 0 column from dataframe
transalated_reviews_df.drop('Unnamed: 0', axis=1, inplace=True)
display(transalated_reviews_df)
# ## Confirming if there any null values in any column
transalated_reviews_df.isnull().sum()
# ## Replacing null values with 'none' string
transalated_reviews_df.fillna('none', inplace=True)
# ## Confirming if there any null values in any column
transalated_reviews_df.isnull().sum()
# ## Removing all Punctuations and Stopwords from the reviews
english_stopwords = set(stopwords.words("english"))
def remove_punctuation_stopwords(review) :
    words = review.split()
    new_sentence = " "

    for word in words :
        new_word = word.translate(str.maketrans('', '', string.punctuation))
        if new_word not in english_stopwords:
            new_sentence += new_word + " "


    return new_sentence.strip().lower()
transalated_reviews_df['tf_idf_reviews'] = transalated_reviews_df['translated_comments'].astype(str).apply(remove_punctuation_stopwords)
display(transalated_reviews_df)
# ## Applying TF-IDF technique to get important features
tf_idf_reviews_list = list(transalated_reviews_df['tf_idf_reviews'])
tf_idf_vectorizer = TfidfVectorizer(analyzer='word', use_idf=True, max_features=500, ngram_range=(1, 2))
tf_idf_reviews = tf_idf_vectorizer.fit_transform(tf_idf_reviews_list)
tf_idf_reviews_array = tf_idf_reviews.toarray()
tf_idf_column_names = [feature_name + '_tfidf_ftr' for feature_name in tf_idf_vectorizer.get_feature_names()]
tf_idf_review_df = pd.DataFrame(data = tf_idf_reviews_array, columns = tf_idf_column_names)
tf_idf_review_df['listing_id'] = transalated_reviews_df['listing_id']
display(tf_idf_review_df)
# ## Grouping by Listing_ID and finding mean of TF-IDF features
tf_idf_features_df = tf_idf_review_df.groupby('listing_id').mean()
display(tf_idf_features_df)
# ## Applying Sentiment Analysis to get Sentiment related features for reviews
reviews_sentiment_df = pd.DataFrame()
reviews_sentiment_df['listing_id'] = transalated_reviews_df['listing_id']
sentiment_analyzer = SentimentIntensityAnalyzer()
reviews_sentiment_df['reviews_sentiment'] = transalated_reviews_df['tf_idf_reviews'].apply(lambda review:
sentiment_analyzer.polarity_scores(review))
reviews_sentiment_df['reviews_sentiment_postive'] = reviews_sentiment_df['reviews_sentiment'].apply(lambda review: review['pos'])
reviews_sentiment_df['reviews_sentiment_negative'] = reviews_sentiment_df['reviews_sentiment'].apply(lambda review: review['neg'])
reviews_sentiment_df['reviews_sentiment_neutral'] = reviews_sentiment_df['reviews_sentiment'].apply(lambda review: review['neu'])
reviews_sentiment_df.drop('reviews_sentiment', axis=1, inplace=True)
reviews_sentiment_df
# ## Grouping by Listing_ID and finding mean of Sentiment features
reviews_sentiment_features_df = reviews_sentiment_df.groupby('listing_id').mean()
display(reviews_sentiment_features_df)
# ## Merging Review Sentiment Features and TF-IDF Features
reviews_features_df = pd.merge(tf_idf_features_df, reviews_sentiment_features_df, on='listing_id', how='left')
display(reviews_features_df)
# ## Confirming Datatypes of the dataframe are all numeric
datatypes = pd.DataFrame(reviews_features_df.dtypes)
pd.set_option('display.max_rows', None)
display(datatypes)
# ## Confirming if there any null values in any column
reviews_features_df.isnull().sum()
# ## Storing final features set in csv
reviews_features_df.to_csv('data/review_features.csv')
```

3. Feature_Engineering_Listings.py

```
# ## Import Libraries
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import string
import numpy as np
import warnings
warnings.filterwarnings("ignore")
# ## Reading listings.csv data and loading it in Dataframe
listings_file = 'data/listings.csv'
df = pd.read_csv(listings_file)
listings_df = df
display(listings_df)
# ## Checking all Columns
list(listings_df.columns)
# ## Rename Columns
listings_df.rename(columns={'id':'listing_id'}, inplace=True)
# ## Dropping Columns that are not relevant for models
listings_df.drop('listing_url', axis=1, inplace=True)
listings_df.drop('scrape_id', axis=1, inplace=True)
listings_df.drop('last_scraped', axis=1, inplace=True)
listings_df.drop('source', axis=1, inplace=True)
listings_df.drop('picture_url', axis=1, inplace=True)
listings_df.drop('host_id', axis=1, inplace=True)
listings_df.drop('host_url', axis=1, inplace=True)
listings_df.drop('host_name', axis=1, inplace=True)
listings_df.drop('host_since', axis=1, inplace=True)
listings_df.drop('host_location', axis=1, inplace=True)
listings_df.drop('host_thumbnail_url', axis=1, inplace=True)
listings_df.drop('host_picture_url', axis=1, inplace=True)
listings_df.drop('host_neighbourhood', axis=1, inplace=True)
listings_df.drop('neighbourhood', axis=1, inplace=True)
listings_df.drop('neighbourhood_group_cleansed', axis=1, inplace=True)
listings_df.drop('latitude', axis=1, inplace=True)
listings_df.drop('longitude', axis=1, inplace=True)
listings_df.drop('bathrooms', axis=1, inplace=True)
listings_df.drop('calendar_updated', axis=1, inplace=True)
listings_df.drop('calendar_last_scraped', axis=1, inplace=True)
listings_df.drop('first_review', axis=1, inplace=True)
listings_df.drop('last_review', axis=1, inplace=True)
listings_df.drop('license', axis=1, inplace=True)
# ## Performing Sentiment Analysis on Long Text Columns
sentiment_analyzer = SentimentIntensityAnalyzer()
def perform_sentiment_analysis(data_frame, column_name) :
    data_frame[column_name].fillna('none', inplace=True)
    sentiment_column_name = column_name + '_sentiment'
    positive_sentiment_column_name = column_name + '_sentiment_positive'
    negative_sentiment_column_name = column_name + '_sentiment_negative'
    neutral_sentiment_column_name = column_name + '_sentiment_neutral'

    data_frame[sentiment_column_name] = data_frame[column_name].astype(str).apply(lambda col: sentiment_analyzer.polarity_scores(col))

    data_frame[positive_sentiment_column_name] = data_frame[sentiment_column_name].apply(lambda col: col['pos'])
    data_frame[negative_sentiment_column_name] = data_frame[sentiment_column_name].apply(lambda col: col['neg'])
    data_frame[neutral_sentiment_column_name] = data_frame[sentiment_column_name].apply(lambda col: col['neu'])

    data_frame.drop(sentiment_column_name, axis=1, inplace=True)
    data_frame.drop(column_name, axis=1, inplace=True)
    return data_frame
listings_df = perform_sentiment_analysis(listings_df, 'name')
listings_df = perform_sentiment_analysis(listings_df, 'description')
listings_df = perform_sentiment_analysis(listings_df, 'neighborhood_overview')
listings_df = perform_sentiment_analysis(listings_df, 'host_about')
# ## Converting to Float Columns containing numbers with special characters
aplha_specialChars = string.punctuation.replace('.','') + string.ascii_letters
convert_float_lambda = lambda col : float(col.translate(str.maketrans('', '', aplha_specialChars)).strip())
listings_df['host_response_rate'].fillna('0', inplace=True)
listings_df['host_response_rate_float'] = listings_df['host_response_rate'].astype(str).apply(convert_float_lambda)
listings_df.drop('host_response_rate', axis=1, inplace=True)
listings_df['host_acceptance_rate'].fillna('0', inplace=True)
listings_df['host_acceptance_rate_float'] = listings_df['host_acceptance_rate'].astype(str).apply(convert_float_lambda)
listings_df.drop('host_acceptance_rate', axis=1, inplace=True)
listings_df['price'].fillna('0', inplace=True)
listings_df['price_float'] = listings_df['price'].astype(str).apply(convert_float_lambda)
listings_df.drop('price', axis=1, inplace=True)
# ## Encoding columns that contains T/F Values to 1/0
def encode_t_f_values(data_frame, column_name) :
    data_frame[column_name].fillna('f', inplace=True)
    data_frame[column_name].loc[data_frame[column_name] == 't'] = 1
    data_frame[column_name].loc[data_frame[column_name] == 'f'] = 0

    return data_frame
listings_df = encode_t_f_values(listings_df, 'host_is_superhost')
listings_df = encode_t_f_values(listings_df, 'host_has_profile_pic')
listings_df = encode_t_f_values(listings_df, 'host_identity_verified')
listings_df = encode_t_f_values(listings_df, 'has_availability')
listings_df = encode_t_f_values(listings_df, 'instant_bookable')
listings_df = listings_df.astype({'host_is_superhost' :'int', 'host_has_profile_pic' :'int', 'host_identity_verified' :'int', 'has_availability' :'int',
'instant_bookable' :'int'})
# ## Repacing NAs in all numeric columns with mean values of that column
listings_df['host_listings_count'].fillna(listings_df['host_listings_count'].mean(), inplace=True)
listings_df['host_total_listings_count'].fillna(listings_df['host_total_listings_count'].mean(), inplace=True)
listings_df['accommodates'].fillna(listings_df['accommodates'].mean(), inplace=True)
listings_df['bedrooms'].fillna(listings_df['bedrooms'].mean(), inplace=True)
listings_df['beds'].fillna(listings_df['beds'].mean(), inplace=True)
listings_df['minimum_nights'].fillna(listings_df['minimum_nights'].mean(), inplace=True)
listings_df['maximum_nights'].fillna(listings_df['maximum_nights'].mean(), inplace=True)
listings_df['minimum_minimum_nights'].fillna(listings_df['minimum_minimum_nights'].mean(), inplace=True)
listings_df['maximum_minimum_nights'].fillna(listings_df['maximum_minimum_nights'].mean(), inplace=True)
listings_df['minimum_maximum_nights'].fillna(listings_df['minimum_maximum_nights'].mean(), inplace=True)
listings_df['maximum_maximum_nights'].fillna(listings_df['maximum_maximum_nights'].mean(), inplace=True)
listings_df['minimum_nights_avg_ntm'].fillna(listings_df['minimum_nights_avg_ntm'].mean(), inplace=True)
listings_df['maximum_nights_avg_ntm'].fillna(listings_df['maximum_nights_avg_ntm'].mean(), inplace=True)
listings_df['availability_30'].fillna(listings_df['availability_30'].mean(), inplace=True)
listings_df['availability_60'].fillna(listings_df['availability_60'].mean(), inplace=True)
listings_df['availability_90'].fillna(listings_df['availability_90'].mean(), inplace=True)
listings_df['availability_365'].fillna(listings_df['availability_365'].mean(), inplace=True)
listings_df['number_of_reviews'].fillna(listings_df['number_of_reviews'].mean(), inplace=True)
listings_df['number_of_reviews_ltm'].fillna(listings_df['number_of_reviews_ltm'].mean(), inplace=True)
listings_df['number_of_reviews_l30d'].fillna(listings_df['number_of_reviews_l30d'].mean(), inplace=True)
listings_df['calculated_host_listings_count'].fillna(listings_df['calculated_host_listings_count'].mean(), inplace=True)
listings_df['calculated_host_listings_count_entire_homes'].fillna(listings_df['calculated_host_listings_count_entire_homes'].mean(),
inplace=True)
listings_df['calculated_host_listings_count_private_rooms'].fillna(listings_df['calculated_host_listings_count_private_rooms'].mean(),
inplace=True)
listings_df['calculated_host_listings_count_shared_rooms'].fillna(listings_df['calculated_host_listings_count_shared_rooms'].mean(),
inplace=True)
listings_df['reviews_per_month'].fillna(listings_df['reviews_per_month'].mean(), inplace=True)
listings_df['review_scores_rating'].fillna(listings_df['review_scores_rating'].mean(), inplace=True)
listings_df['review_scores_accuracy'].fillna(listings_df['review_scores_accuracy'].mean(), inplace=True)
listings_df['review_scores_cleanliness'].fillna(listings_df['review_scores_cleanliness'].mean(), inplace=True)
listings_df['review_scores_checkin'].fillna(listings_df['review_scores_checkin'].mean(), inplace=True)
listings_df['review_scores_communication'].fillna(listings_df['review_scores_communication'].mean(), inplace=True)
listings_df['review_scores_location'].fillna(listings_df['review_scores_location'].mean(), inplace=True)
listings_df['review_scores_value'].fillna(listings_df['review_scores_value'].mean(), inplace=True)
# ## Encoding Amenities column by replacing it with the Count of amenities
listings_df['count_amenities'] = listings_df['amenities'].apply(lambda amenities_list : len(amenities_list.split(',')))
listings_df.drop('amenities', axis=1, inplace=True)
# ## One-Hot Encoding all the Categorical columns
listings_df['property_type'].fillna('none', inplace=True)
listings_df['room_type'].fillna('none', inplace=True)
listings_df['host_response_time'].fillna('none', inplace=True)
listings_df['bathrooms_text'].fillna('none', inplace=True)
listings_df['host_verifications'].fillna('none', inplace=True)
listings_df['neighbourhood_cleansed'].fillna('none', inplace=True)
listings_df = pd.get_dummies(listings_df, columns = ['property_type', 'room_type', 'host_response_time', 'bathrooms_text',
'host_verifications', 'neighbourhood_cleansed'])
display(listings_df)
# ## Confirming the datatypes of all columns
datatypes = pd.DataFrame(listings_df.dtypes)
pd.set_option('display.max_rows', None)
display(datatypes)
# ## Confirming if there any null values in any column
listings_df.isnull().sum()
# ## Storing final features set in csv
listings_df.to_csv('data/listing_features.csv')
```

4. Feature_Selection_Model_Tuning_ReviewScores_Accuracy

```
# ## Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso, LinearRegression
import warnings
warnings.filterwarnings("ignore")
# # Feature Selection
# ## Reading review features data and loading it in Dataframe
reviews_features_file = 'data/review_features.csv'
reviews_features_df = pd.read_csv(reviews_features_file)
display(reviews_features_df)
# ## Reading lisiting features data and loading it in Dataframe
listings_features_file = 'data/listing_features.csv'
listings_features_df = pd.read_csv(listings_features_file)
display(listings_features_df)
listings_features_df.drop('Unnamed: 0', axis=1, inplace=True)
display(listings_features_df)
# ## Merging review and listing features based on listing id
all_features_df = pd.merge(listings_features_df, reviews_features_df, on='listing_id', how='left')
display(all_features_df)
# ## Dropping listing id column from final feature set and replacing any null value with 0
all_features_df.drop('listing_id', axis=1, inplace=True)
all_features_df.fillna(0, inplace = True)
display(all_features_df)
# ## Confirming Datatypes for all features in dataframe
datatypes = pd.DataFrame(all_features_df.dtypes)
pd.set_option('display.max_rows', None)
display(datatypes)
# ## Confirming if there any null values in any column
all_features_df.isnull().sum()
# ## Separating labels from features
Y_review_scores_accuracy = pd.DataFrame(all_features_df['review_scores_accuracy'])
final_features_df = all_features_df.copy()
final_features_df.drop('review_scores_rating', axis=1, inplace=True)
final_features_df.drop('review_scores_accuracy', axis=1, inplace=True)
final_features_df.drop('review_scores_cleanliness', axis=1, inplace=True)
final_features_df.drop('review_scores_checkin', axis=1, inplace=True)
final_features_df.drop('review_scores_communication', axis=1, inplace=True)
final_features_df.drop('review_scores_location', axis=1, inplace=True)
final_features_df.drop('review_scores_value', axis=1, inplace=True)
pd.reset_option('^display.', silent=True)
display(final_features_df)
# ## Feature Selection using Lasso Regression
# ### Scaling the data using MinMax scaling
scaler_features = MinMaxScaler()
scaler_features.fit(final_features_df)
scaled_features = scaler_features.fit_transform(final_features_df)
scaled_features_df = pd.DataFrame(scaled_features, columns = final_features_df.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_accuracy)
scaled_label = scaler_label.fit_transform(Y_review_scores_accuracy)
scaled_Y_review_scores_accuracy = pd.DataFrame(scaled_label, columns = Y_review_scores_accuracy.columns)
display(scaled_features_df)
scaled_features_array = np.array(scaled_features_df)
scaled_label_array = np.array(scaled_Y_review_scores_accuracy)
def calculate_mse_stddev_penalty_lasso(penalty_parameters) :
    k_fold_split = 5

    k_fold_split_function =  KFold(n_splits = k_fold_split)

    mean_sqaure_error_penalty = []
    standard_deviation_penalty = []

    for penalty in penalty_parameters :
        lasso_model = Lasso(alpha = penalty)
        mean_sqaure_error_fold = []
        for train_index, test_index in k_fold_split_function.split(scaled_features_array):
            X_train, X_test = scaled_features_array[train_index], scaled_features_array[test_index]
            y_train, y_test = scaled_label_array[train_index], scaled_label_array[test_index]
            lasso_model.fit(X_train, y_train)
            predictions = lasso_model.predict(X_test)

            mean_sqaure_error_fold.append(mean_squared_error(y_test, predictions))

        mean_sqaure_error_penalty.append(np.array(mean_sqaure_error_fold).mean())
        standard_deviation_penalty.append(np.array(mean_sqaure_error_fold).std())

    return mean_sqaure_error_penalty, standard_deviation_penalty
penalty_parameters = [1, 5, 10, 100, 500, 1000]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
penalty_parameters =  [0.0005, 0.001, 0.005, 0.01]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
# ## Getting all Non-Zero features after Tuned Lasso Regression
feature_names = list(final_features_df.columns.values)
def get_lasso_parameters(penalty) :

    lasso_model_dictionary = {}
    X_Train,X_Test,y_Train,y_Test = train_test_split(scaled_features_array, scaled_label_array, test_size = 0.2, random_state=111, shuffle =
False)
    lasso_model_params_df = pd.DataFrame(columns = feature_names)

    lasso_model = Lasso(alpha = penalty)
    lasso_model.fit(X_Train, y_Train)

    model_dict = {}
    for i in range(len(final_features_df.columns)) :
        model_dict[feature_names[i]] = np.around(lasso_model.coef_[i-2], decimals = 3)

    lasso_model_params_df = lasso_model_params_df.append(model_dict, ignore_index = True)

    return lasso_model_params_df
penalty_parameter = 0.0005
lasso_df = get_lasso_parameters(penalty_parameter)
column_names = list()
lasso_param_dictionary = {}
for column_name in lasso_df.columns:
    column = lasso_df[column_name]
    count_of_non_zeros = (column != 0).sum()
    if count_of_non_zeros != 0 :
        column_names.append(column_name)
        lasso_param_dictionary[column_name] = column[0]
print(f'Total Non-zero Columns: {len(column_names)}')
print(f'Selected Non-zero Column Names: {column_names}')
lasso_non_zero_params_df = pd.DataFrame(columns = ['Feature Name', 'Feature Weight'])
for key in lasso_param_dictionary:
    lasso_non_zero_params_df.loc[len(lasso_non_zero_params_df.index)] = [key, lasso_param_dictionary[key]]
display(lasso_non_zero_params_df)

# # Models and Hyper-parameter tuning
# ## Normalising Selected Features Set
```

```
selected_features = final_features_df[column_names]
scaler_selected_features = MinMaxScaler()
scaler_selected_features.fit(selected_features)
scaled_selected_features = scaler_selected_features.fit_transform(selected_features)
scaled_selected_features_df = pd.DataFrame(scaled_selected_features, columns = selected_features.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_accuracy)
scaled_label = scaler_label.fit_transform(Y_review_scores_accuracy)
scaled_Y_review_scores_accuracy = pd.DataFrame(scaled_label, columns = Y_review_scores_accuracy.columns)
display(scaled_selected_features_df)
X_train, X_test, y_train, y_test = train_test_split(scaled_selected_features_df, scaled_Y_review_scores_accuracy, test_size = 0.2,
random_state = 111)
scaled_train_feature = np.array(X_train)
scaled_train_label = np.array(y_train)
scaled_test_feature = np.array(X_test)
scaled_test_label = np.array(y_test)
# ## Baseline Model: Linear Regression
Baseline_Linear_Model = LinearRegression()
val_score = cross_val_score(Baseline_Linear_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Baseline Linear Regression is: {abs(np.array(val_score).mean())}')
# ## Model 1: Tuned Lasso Regression
Lasso_Model = Lasso(alpha = 0.0005)
lasso_val_score = cross_val_score(Lasso_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Lasso Model is: {abs(np.array(lasso_val_score).mean())}')
# ## Model 2: KNN Regression
knn_score = []
neighbors = [x for x in range(1, (len(column_names) + 1), 2)]
for neighbor in neighbors:
    KNN_Model = KNeighborsRegressor(n_neighbors = neighbor)
    knn_val_score = cross_val_score(KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
    knn_score.append(abs(np.array(knn_val_score).mean()))
    print(f'Cross Validation Score for KNN Regressor is: {abs(np.array(knn_val_score).mean())}', " for n_neighbors = ", neighbor)
plt.plot(neighbors, knn_score, marker = 'o')
plt.xlabel("n_Neighbors")
plt.ylabel("Cross val Score")
plt.title("Plot of CVS Vs. N_Neighbor parameter for KNN Regressor")
plt.show()
# ### Tuned KNN Model
Tuned_KNN_Model = KNeighborsRegressor(n_neighbors = 7)
tuned_knn_val_score = cross_val_score(Tuned_KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring =
'neg_mean_squared_error')
print(f'Cross Validation Score for KNN Regression is: {abs(np.array(tuned_knn_val_score).mean())}')
```

5. Feature_Selection_Model_Tuning_ReviewScores_Checkin

```
# ## Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso, LinearRegression
import warnings
warnings.filterwarnings("ignore")
# # Feature Selection
# ## Reading review features data and loading it in Dataframe
reviews_features_file = 'data/review_features.csv'
reviews_features_df = pd.read_csv(reviews_features_file)
display(reviews_features_df)
# ## Reading lisiting features data and loading it in Dataframe
listings_features_file = 'data/listing_features.csv'
listings_features_df = pd.read_csv(listings_features_file)
display(listings_features_df)
listings_features_df.drop('Unnamed: 0', axis=1, inplace=True)
```

```
display(listings_features_df)
# ## Merging review and listing features based on listing id
all_features_df = pd.merge(listings_features_df, reviews_features_df, on='listing_id', how='left')
display(all_features_df)
# ## Dropping listing id column from final feature set and replacing any null value with 0
all_features_df.drop('listing_id', axis=1, inplace=True)
all_features_df.fillna(0, inplace = True)
display(all_features_df)
# ## Confirming Datatypes for all features in dataframe
datatypes = pd.DataFrame(all_features_df.dtypes)
pd.set_option('display.max_rows', None)
display(datatypes)
# ## Confirming if there any null values in any column
all_features_df.isnull().sum()
# ## Separating labels from features
Y_review_scores_checkin = pd.DataFrame(all_features_df['review_scores_checkin'])
final_features_df = all_features_df.copy()
final_features_df.drop('review_scores_rating', axis=1, inplace=True)
final_features_df.drop('review_scores_accuracy', axis=1, inplace=True)
final_features_df.drop('review_scores_cleanliness', axis=1, inplace=True)
final_features_df.drop('review_scores_checkin', axis=1, inplace=True)
final_features_df.drop('review_scores_communication', axis=1, inplace=True)
final_features_df.drop('review_scores_location', axis=1, inplace=True)
final_features_df.drop('review_scores_value', axis=1, inplace=True)
pd.reset_option('^display.', silent=True)
display(final_features_df)
# ## Feature Selection using Lasso Regression
# ### Scaling the data using MinMax scaling
scaler_features = MinMaxScaler()
scaler_features.fit(final_features_df)
scaled_features = scaler_features.fit_transform(final_features_df)
scaled_features_df = pd.DataFrame(scaled_features, columns = final_features_df.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_checkin)
scaled_label = scaler_label.fit_transform(Y_review_scores_checkin)
scaled_Y_review_scores_checkin = pd.DataFrame(scaled_label, columns = Y_review_scores_checkin.columns)
display(scaled_features_df)
scaled_features_array = np.array(scaled_features_df)
scaled_label_array = np.array(scaled_Y_review_scores_checkin)
def calculate_mse_stddev_penalty_lasso(penalty_parameters) :
    k_fold_split = 5
    k_fold_split_function =  KFold(n_splits = k_fold_split)
    mean_sqaure_error_penalty = []
    standard_deviation_penalty = []
    for penalty in penalty_parameters :
        lasso_model = Lasso(alpha = penalty)
        mean_sqaure_error_fold = []
        for train_index, test_index in k_fold_split_function.split(scaled_features_array):
            X_train, X_test = scaled_features_array[train_index], scaled_features_array[test_index]
            y_train, y_test = scaled_label_array[train_index], scaled_label_array[test_index]
            lasso_model.fit(X_train, y_train)
            predictions = lasso_model.predict(X_test)
            mean_sqaure_error_fold.append(mean_squared_error(y_test, predictions))
        mean_sqaure_error_penalty.append(np.array(mean_sqaure_error_fold).mean())
        standard_deviation_penalty.append(np.array(mean_sqaure_error_fold).std())
    return mean_sqaure_error_penalty, standard_deviation_penalty
penalty_parameters = [1, 5, 10, 100, 500, 1000]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
penalty_parameters =  [0.0005, 0.001, 0.005, 0.01]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
# ## Getting all Non-Zero features after Tuned Lasso Regression
#
feature_names = list(final_features_df.columns.values)
def get_lasso_parameters(penalty) :
    lasso_model_dictionary = {}
    X_Train,X_Test,y_Train,y_Test = train_test_split(scaled_features_array, scaled_label_array, test_size = 0.2, random_state=111, shuffle = 
False)
    lasso_model_params_df = pd.DataFrame(columns = feature_names)
    lasso_model = Lasso(alpha = penalty)
    lasso_model.fit(X_Train, y_Train)
    model_dict = {}
    for i in range(len(final_features_df.columns)) :
        model_dict[feature_names[i]] = np.around(lasso_model.coef_[i-2], decimals = 3)
    lasso_model_params_df = lasso_model_params_df.append(model_dict, ignore_index = True)
    return lasso_model_params_df
penalty_parameter = 0.0005
lasso_df = get_lasso_parameters(penalty_parameter)
column_names = list()
lasso_param_dictionary = {}
for column_name in lasso_df.columns:
    column = lasso_df[column_name]
    count_of_non_zeros = (column != 0).sum()
    if count_of_non_zeros != 0 :
        column_names.append(column_name)
        lasso_param_dictionary[column_name] = column[0]
print(f'Total Non-zero Columns: {len(column_names)}')
print(f'Selected Non-zero Column Names: {column_names}')
lasso_non_zero_params_df = pd.DataFrame(columns = ['Feature Name', 'Feature Weight'])
for key in lasso_param_dictionary:
    lasso_non_zero_params_df.loc[len(lasso_non_zero_params_df.index)] = [key, lasso_param_dictionary[key]]
display(lasso_non_zero_params_df)
# # Models and Hyper-parameter tuning
# ## Normalising Selected Features Set
selected_features = final_features_df[column_names]
scaler_selected_features = MinMaxScaler()
scaler_selected_features.fit(selected_features)
scaled_selected_features = scaler_selected_features.fit_transform(selected_features)
scaled_selected_features_df = pd.DataFrame(scaled_selected_features, columns = selected_features.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_checkin)
scaled_label = scaler_label.fit_transform(Y_review_scores_checkin)
scaled_Y_review_scores_checkin = pd.DataFrame(scaled_label, columns = Y_review_scores_checkin.columns)
display(scaled_selected_features_df)
X_train, X_test, y_train, y_test = train_test_split(scaled_selected_features_df, scaled_Y_review_scores_checkin, test_size = 0.2, random_state 
= 111)
scaled_train_feature = np.array(X_train)
scaled_train_label = np.array(y_train)
scaled_test_feature = np.array(X_test)
scaled_test_label = np.array(y_test)
# ## Baseline Model: Linear Regression
Baseline_Linear_Model = LinearRegression()
val_score = cross_val_score(Baseline_Linear_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Baseline Linear Regression is: {abs(np.array(val_score).mean())}')
# ## Model 1: Tuned Lasso Regression
Lasso_Model = Lasso(alpha = 0.0005)
lasso_val_score = cross_val_score(Lasso_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Lasso Model is: {abs(np.array(lasso_val_score).mean())}')
knn_score = []
neighbors = [x for x in range(1, (len(column_names) + 1), 2)]
for neighbor in neighbors:
    KNN_Model = KNeighborsRegressor(n_neighbors = neighbor)
    knn_val_score = cross_val_score(KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
    knn_score.append(abs(np.array(knn_val_score).mean()))
    print(f'Cross Validation Score for KNN Regressor is: {abs(np.array(knn_val_score).mean())}', " for n_neighbors = ", neighbor)
plt.plot(neighbors, knn_score, marker = 'o')
plt.xlabel("n_Neighbors")
plt.ylabel("Cross val Score")
plt.title("Plot of CVS Vs. N_Neighbor parameter for KNN Regressor")
plt.show()
# ### Tuned KNN Model
Tuned_KNN_Model = KNeighborsRegressor(n_neighbors = 9)
tuned_knn_val_score = cross_val_score(Tuned_KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring =
'neg_mean_squared_error')
print(f'Cross Validation Score for KNN Regression is: {abs(np.array(tuned_knn_val_score).mean())}')
```

6. Feature_Selection_Model_Tuning_ReviewScores_Cleanliness

```
# ## Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso, LinearRegression
import warnings
warnings.filterwarnings("ignore")
# # Feature Selection
# ## Reading review features data and loading it in Dataframe
reviews_features_file = 'data/review_features.csv'
reviews_features_df = pd.read_csv(reviews_features_file)
display(reviews_features_df)
# ## Reading lisiting features data and loading it in Dataframe
listings_features_file = 'data/listing_features.csv'
listings_features_df = pd.read_csv(listings_features_file)
display(listings_features_df)
listings_features_df.drop('Unnamed: 0', axis=1, inplace=True)
display(listings_features_df)
# ## Merging review and listing features based on listing id
all_features_df = pd.merge(listings_features_df, reviews_features_df, on='listing_id', how='left')
display(all_features_df)
# ## Dropping listing id column from final feature set and replacing any null value with 0
all_features_df.drop('listing_id', axis=1, inplace=True)
all_features_df.fillna(0, inplace = True)
display(all_features_df)
# ## Confirming Datatypes for all features in dataframe
datatypes = pd.DataFrame(all_features_df.dtypes)
pd.set_option('display.max_rows', None)
display(datatypes)
# ## Confirming if there any null values in any column
all_features_df.isnull().sum()
# ## Separating labels from features
Y_review_scores_cleanliness = pd.DataFrame(all_features_df['review_scores_cleanliness'])
final_features_df = all_features_df.copy()
final_features_df.drop('review_scores_rating', axis=1, inplace=True)
final_features_df.drop('review_scores_accuracy', axis=1, inplace=True)
final_features_df.drop('review_scores_cleanliness', axis=1, inplace=True)
final_features_df.drop('review_scores_checkin', axis=1, inplace=True)
final_features_df.drop('review_scores_communication', axis=1, inplace=True)
final_features_df.drop('review_scores_location', axis=1, inplace=True)
final_features_df.drop('review_scores_value', axis=1, inplace=True)
pd.reset_option('^display.', silent=True)
display(final_features_df)
# ## Feature Selection using Lasso Regression
# ### Scaling the data using MinMax scaling
scaler_features = MinMaxScaler()
scaler_features.fit(final_features_df)
```

```
scaled_features = scaler_features.fit_transform(final_features_df)
scaled_features_df = pd.DataFrame(scaled_features, columns = final_features_df.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_cleanliness)
scaled_label = scaler_label.fit_transform(Y_review_scores_cleanliness)
scaled_Y_review_scores_cleanliness = pd.DataFrame(scaled_label, columns = Y_review_scores_cleanliness.columns)
display(scaled_features_df)
scaled_features_array = np.array(scaled_features_df)
scaled_label_array = np.array(scaled_Y_review_scores_cleanliness)
def calculate_mse_stddev_penalty_lasso(penalty_parameters) :
    k_fold_split = 5
    k_fold_split_function =  KFold(n_splits = k_fold_split)
    mean_sqaure_error_penalty = []
    standard_deviation_penalty = []
    for penalty in penalty_parameters :
        lasso_model = Lasso(alpha = penalty)
        mean_sqaure_error_fold = []
        for train_index, test_index in k_fold_split_function.split(scaled_features_array):
            X_train, X_test = scaled_features_array[train_index], scaled_features_array[test_index]
            y_train, y_test = scaled_label_array[train_index], scaled_label_array[test_index]
            lasso_model.fit(X_train, y_train)
            predictions = lasso_model.predict(X_test)
            mean_sqaure_error_fold.append(mean_squared_error(y_test, predictions))
        mean_sqaure_error_penalty.append(np.array(mean_sqaure_error_fold).mean())
        standard_deviation_penalty.append(np.array(mean_sqaure_error_fold).std())
    return mean_sqaure_error_penalty, standard_deviation_penalty
penalty_parameters = [1, 5, 10, 100, 500, 1000]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
penalty_parameters =  [0.0005, 0.001, 0.005, 0.01]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
# ## Getting all Non-Zero features after Tuned Lasso Regression
feature_names = list(final_features_df.columns.values)
def get_lasso_parameters(penalty) :
    lasso_model_dictionary = {}
    X_Train,X_Test,y_Train,y_Test = train_test_split(scaled_features_array, scaled_label_array, test_size = 0.2, random_state=111, shuffle = 
False)
    lasso_model_params_df = pd.DataFrame(columns = feature_names)
    lasso_model = Lasso(alpha = penalty)
    lasso_model.fit(X_Train, y_Train)
    model_dict = {}
    for i in range(len(final_features_df.columns)) :
        model_dict[feature_names[i]] = np.around(lasso_model.coef_[i-2], decimals = 3)
    lasso_model_params_df = lasso_model_params_df.append(model_dict, ignore_index = True)
    return lasso_model_params_df
penalty_parameter = 0.0005
lasso_df = get_lasso_parameters(penalty_parameter)
column_names = list()
lasso_param_dictionary = {}
for column_name in lasso_df.columns:
    column = lasso_df[column_name]
    count_of_non_zeros = (column != 0).sum()
    if count_of_non_zeros != 0 :
        column_names.append(column_name)
        lasso_param_dictionary[column_name] = column[0]
print(f'Total Non-zero Columns: {len(column_names)}')
print(f'Selected Non-zero Column Names: {column_names}')
lasso_non_zero_params_df = pd.DataFrame(columns = ['Feature Name', 'Feature Weight'])
for key in lasso_param_dictionary:
    lasso_non_zero_params_df.loc[len(lasso_non_zero_params_df.index)] = [key, lasso_param_dictionary[key]]
display(lasso_non_zero_params_df)
# # Models and Hyper-parameter tuning
# ## Normalising Selected Features Set
selected_features = final_features_df[column_names]
scaler_selected_features = MinMaxScaler()
scaler_selected_features.fit(selected_features)
scaled_selected_features = scaler_selected_features.fit_transform(selected_features)
scaled_selected_features_df = pd.DataFrame(scaled_selected_features, columns = selected_features.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_cleanliness)
scaled_label = scaler_label.fit_transform(Y_review_scores_cleanliness)
scaled_Y_review_scores_cleanliness = pd.DataFrame(scaled_label, columns = Y_review_scores_cleanliness.columns)
display(scaled_selected_features_df)
X_train, X_test, y_train, y_test = train_test_split(scaled_selected_features_df, scaled_Y_review_scores_cleanliness, test_size = 0.2,
random_state = 111)
scaled_train_feature = np.array(X_train)
scaled_train_label = np.array(y_train)
scaled_test_feature = np.array(X_test)
scaled_test_label = np.array(y_test)
# ## Baseline Model: Linear Regression
Baseline_Linear_Model = LinearRegression()
val_score = cross_val_score(Baseline_Linear_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Baseline Linear Regression is: {abs(np.array(val_score).mean())}')
# ## Model 1: Tuned Lasso Regression
Lasso_Model = Lasso(alpha = 0.0005)
lasso_val_score = cross_val_score(Lasso_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Lasso Model is: {abs(np.array(lasso_val_score).mean())}')
# ## Model 2: KNN Regression
knn_score = []
neighbors = [x for x in range(1, (len(column_names) + 1), 2)]
for neighbor in neighbors:
    KNN_Model = KNeighborsRegressor(n_neighbors = neighbor)
    knn_val_score = cross_val_score(KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
    knn_score.append(abs(np.array(knn_val_score).mean()))
    print(f'Cross Validation Score for KNN Regressor is: {abs(np.array(knn_val_score).mean())}', " for n_neighbors = ", neighbor)
plt.plot(neighbors, knn_score, marker = 'o')
plt.xlabel("n_Neighbors")
plt.ylabel("Cross val Score")
plt.title("Plot of CVS Vs. N_Neighbor parameter for KNN Regressor")
plt.show()
# ### Tuned KNN Model
Tuned_KNN_Model = KNeighborsRegressor(n_neighbors = 21)
tuned_knn_val_score = cross_val_score(Tuned_KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring =
'neg_mean_squared_error')
print(f'Cross Validation Score for KNN Regression is: {abs(np.array(tuned_knn_val_score).mean())}')
```

7. Feature_Selection_Model_Tuning_ReviewScores_Communication

```
# ## Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso, LinearRegression
import warnings
warnings.filterwarnings("ignore")
# # Feature Selection
# ## Reading review features data and loading it in Dataframe
reviews_features_file = 'data/review_features.csv'
```

```
reviews_features_df = pd.read_csv(reviews_features_file)
display(reviews_features_df)
# ## Reading lisiting features data and loading it in Dataframe
listings_features_file = 'data/listing_features.csv'
listings_features_df = pd.read_csv(listings_features_file)
display(listings_features_df)
listings_features_df.drop('Unnamed: 0', axis=1, inplace=True)
display(listings_features_df)
# ## Merging review and listing features based on listing id
all_features_df = pd.merge(listings_features_df, reviews_features_df, on='listing_id', how='left')
display(all_features_df)
# ## Dropping listing id column from final feature set and replacing any null value with 0
all_features_df.drop('listing_id', axis=1, inplace=True)
all_features_df.fillna(0, inplace = True)
display(all_features_df)
# ## Confirming Datatypes for all features in dataframe
datatypes = pd.DataFrame(all_features_df.dtypes)
pd.set_option('display.max_rows', None)
display(datatypes)
# ## Confirming if there any null values in any column
all_features_df.isnull().sum()
# ## Separating labels from features
Y_review_scores_communication = pd.DataFrame(all_features_df['review_scores_communication'])
final_features_df = all_features_df.copy()
final_features_df.drop('review_scores_rating', axis=1, inplace=True)
final_features_df.drop('review_scores_accuracy', axis=1, inplace=True)
final_features_df.drop('review_scores_cleanliness', axis=1, inplace=True)
final_features_df.drop('review_scores_checkin', axis=1, inplace=True)
final_features_df.drop('review_scores_communication', axis=1, inplace=True)
final_features_df.drop('review_scores_location', axis=1, inplace=True)
final_features_df.drop('review_scores_value', axis=1, inplace=True)
pd.reset_option('^display.', silent=True)
display(final_features_df)
# ## Feature Selection using Lasso Regression
# ### Scaling the data using MinMax scaling
scaler_features = MinMaxScaler()
scaler_features.fit(final_features_df)
scaled_features = scaler_features.fit_transform(final_features_df)
scaled_features_df = pd.DataFrame(scaled_features, columns = final_features_df.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_communication)
scaled_label = scaler_label.fit_transform(Y_review_scores_communication)
scaled_Y_review_scores_communication = pd.DataFrame(scaled_label, columns = Y_review_scores_communication.columns)
display(scaled_features_df)
scaled_features_array = np.array(scaled_features_df)
scaled_label_array = np.array(scaled_Y_review_scores_communication)
def calculate_mse_stddev_penalty_lasso(penalty_parameters) :
    k_fold_split = 5
    k_fold_split_function =  KFold(n_splits = k_fold_split)
    mean_sqaure_error_penalty = []
    standard_deviation_penalty = []
    for penalty in penalty_parameters :
        lasso_model = Lasso(alpha = penalty)
        mean_sqaure_error_fold = []
        for train_index, test_index in k_fold_split_function.split(scaled_features_array):
            X_train, X_test = scaled_features_array[train_index], scaled_features_array[test_index]
            y_train, y_test = scaled_label_array[train_index], scaled_label_array[test_index]
            lasso_model.fit(X_train, y_train)
            predictions = lasso_model.predict(X_test)
            mean_sqaure_error_fold.append(mean_squared_error(y_test, predictions))
        mean_sqaure_error_penalty.append(np.array(mean_sqaure_error_fold).mean())
        standard_deviation_penalty.append(np.array(mean_sqaure_error_fold).std())
    return mean_sqaure_error_penalty, standard_deviation_penalty
penalty_parameters = [1, 5, 10, 100, 500, 1000]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
penalty_parameters =  [0.0005, 0.001, 0.005, 0.01]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
# ## Getting all Non-Zero features after Tuned Lasso Regression
feature_names = list(final_features_df.columns.values)
def get_lasso_parameters(penalty) :
   lasso_model_dictionary = {}
   X_Train,X_Test,y_Train,y_Test = train_test_split(scaled_features_array, scaled_label_array, test_size = 0.2, random_state=111, shuffle = 
False)
   lasso_model_params_df = pd.DataFrame(columns = feature_names)
   lasso_model = Lasso(alpha = penalty)
   lasso_model.fit(X_Train, y_Train)
   model_dict = {}
   for i in range(len(final_features_df.columns)) :
      model_dict[feature_names[i]] = np.around(lasso_model.coef_[i-2], decimals = 3)
   lasso_model_params_df = lasso_model_params_df.append(model_dict, ignore_index = True)
   return lasso_model_params_df
penalty_parameter = 0.0005
lasso_df = get_lasso_parameters(penalty_parameter)
column_names = list()
lasso_param_dictionary = {}
for column_name in lasso_df.columns:
   column = lasso_df[column_name]
   count_of_non_zeros = (column != 0).sum()
   if count_of_non_zeros != 0 :
      column_names.append(column_name)
      lasso_param_dictionary[column_name] = column[0]
print(f'Total Non-zero Columns: {len(column_names)}')
print(f'Selected Non-zero Column Names: {column_names}')
lasso_non_zero_params_df = pd.DataFrame(columns = ['Feature Name', 'Feature Weight'])
for key in lasso_param_dictionary:
   lasso_non_zero_params_df.loc[len(lasso_non_zero_params_df.index)] = [key, lasso_param_dictionary[key]]
display(lasso_non_zero_params_df)
# # Models and Hyper-parameter tuning
# ## Normalising Selected Features Set
selected_features = final_features_df[column_names]
scaler_selected_features = MinMaxScaler()
scaler_selected_features.fit(selected_features)
scaled_selected_features = scaler_selected_features.fit_transform(selected_features)
scaled_selected_features_df = pd.DataFrame(scaled_selected_features, columns = selected_features.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_communication)
scaled_label = scaler_label.fit_transform(Y_review_scores_communication)
scaled_Y_review_scores_communication = pd.DataFrame(scaled_label, columns = Y_review_scores_communication.columns)
display(scaled_selected_features_df)
X_train, X_test, y_train, y_test = train_test_split(scaled_selected_features_df, scaled_Y_review_scores_communication, test_size = 0.2, 
random_state = 111)
scaled_train_feature = np.array(X_train)
scaled_train_label = np.array(y_train)
scaled_test_feature = np.array(X_test)
scaled_test_label = np.array(y_test)
# ## Baseline Model: Linear Regression
Baseline_Linear_Model = LinearRegression()
val_score = cross_val_score(Baseline_Linear_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Baseline Linear Regression is: {abs(np.array(val_score).mean())}')
# ## Model 1: Tuned Lasso Regression
Lasso_Model = Lasso(alpha = 0.0005)
lasso_val_score = cross_val_score(Lasso_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
```

```
print(f'Cross Validation Score for Lasso Model is: {abs(np.array(lasso_val_score).mean())}')
# ## Model 2: KNN Regression
knn_score = []
neighbors = [x for x in range(1, (len(column_names) + 1), 2)]
for neighbor in neighbors:
   KNN_Model = KNeighborsRegressor(n_neighbors = neighbor)
   knn_val_score = cross_val_score(KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
   knn_score.append(abs(np.array(knn_val_score).mean()))
   print(f'Cross Validation Score for KNN Regressor is: {abs(np.array(knn_val_score).mean())}', " for n_neighbors = ", neighbor)
plt.plot(neighbors, knn_score, marker = 'o')
plt.xlabel("n_Neighbors")
plt.ylabel("Cross val Score")
plt.title("Plot of CVS Vs. N_Neighbor parameter for KNN Regressor")
plt.show()
# ### Tuned KNN Model
Tuned_KNN_Model = KNeighborsRegressor(n_neighbors = 13)
tuned_knn_val_score = cross_val_score(Tuned_KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 
'neg_mean_squared_error')
print(f'Cross Validation Score for KNN Regression is: {abs(np.array(tuned_knn_val_score).mean())}')
```

8. Feature_Selection_Model_Tuning_ReviewScores_Location

```
# ## Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso, LinearRegression
import warnings
warnings.filterwarnings("ignore")
# # Feature Selection
# ## Reading review features data and loading it in Dataframe
reviews_features_file = 'data/review_features.csv'
reviews_features_df = pd.read_csv(reviews_features_file)
display(reviews_features_df)
# ## Reading lisiting features data and loading it in Dataframe
listings_features_file = 'data/listing_features.csv'
listings_features_df = pd.read_csv(listings_features_file)
display(listings_features_df)
listings_features_df.drop('Unnamed: 0', axis=1, inplace=True)
display(listings_features_df)
# ## Merging review and listing features based on listing id
all_features_df = pd.merge(listings_features_df, reviews_features_df, on='listing_id', how='left')
display(all_features_df)
# ## Dropping listing id column from final feature set and replacing any null value with 0
all_features_df.drop('listing_id', axis=1, inplace=True)
all_features_df.fillna(0, inplace = True)
display(all_features_df)
# ## Confirming Datatypes for all features in dataframe
datatypes = pd.DataFrame(all_features_df.dtypes)
pd.set_option('display.max_rows', None)
display(datatypes)
# ## Confirming if there any null values in any column
all_features_df.isnull().sum()
# ## Separating labels from features
Y_review_scores_location = pd.DataFrame(all_features_df['review_scores_location'])
final_features_df = all_features_df.copy()
final_features_df.drop('review_scores_rating', axis=1, inplace=True)
final_features_df.drop('review_scores_accuracy', axis=1, inplace=True)
final_features_df.drop('review_scores_cleanliness', axis=1, inplace=True)
final_features_df.drop('review_scores_checkin', axis=1, inplace=True)
final_features_df.drop('review_scores_communication', axis=1, inplace=True)
final_features_df.drop('review_scores_location', axis=1, inplace=True)
final_features_df.drop('review_scores_value', axis=1, inplace=True)
pd.reset_option('^display.', silent=True)
display(final_features_df)
# ## Feature Selection using Lasso Regression
# ### Scaling the data using MinMax scaling
scaler_features = MinMaxScaler()
scaler_features.fit(final_features_df)
scaled_features = scaler_features.fit_transform(final_features_df)
scaled_features_df = pd.DataFrame(scaled_features, columns = final_features_df.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_location)
scaled_label = scaler_label.fit_transform(Y_review_scores_location)
scaled_Y_review_scores_location = pd.DataFrame(scaled_label, columns = Y_review_scores_location.columns)
display(scaled_features_df)
scaled_features_array = np.array(scaled_features_df)
scaled_label_array = np.array(scaled_Y_review_scores_location)
def calculate_mse_stddev_penalty_lasso(penalty_parameters) :
   k_fold_split = 5
   k_fold_split_function =  KFold(n_splits = k_fold_split)
   mean_sqaure_error_penalty = []
   standard_deviation_penalty = []
   for penalty in penalty_parameters :
      lasso_model = Lasso(alpha = penalty)
      mean_sqaure_error_fold = []
      for train_index, test_index in k_fold_split_function.split(scaled_features_array):
         X_train, X_test = scaled_features_array[train_index], scaled_features_array[test_index]
         y_train, y_test = scaled_label_array[train_index], scaled_label_array[test_index]
         lasso_model.fit(X_train, y_train)
         predictions = lasso_model.predict(X_test)
         mean_sqaure_error_fold.append(mean_squared_error(y_test, predictions))
      mean_sqaure_error_penalty.append(np.array(mean_sqaure_error_fold).mean())
      standard_deviation_penalty.append(np.array(mean_sqaure_error_fold).std())
   return mean_sqaure_error_penalty, standard_deviation_penalty
penalty_parameters = [1, 5, 10, 100, 500, 1000]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
penalty_parameters =  [0.0005, 0.001, 0.005, 0.01]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
# ## Getting all Non-Zero features after Tuned Lasso Regression
feature_names = list(final_features_df.columns.values)
def get_lasso_parameters(penalty) :
   lasso_model_dictionary = {}
   X_Train,X_Test,y_Train,y_Test = train_test_split(scaled_features_array, scaled_label_array, test_size = 0.2, random_state=111, shuffle =
False)
   lasso_model_params_df = pd.DataFrame(columns = feature_names)
   lasso_model = Lasso(alpha = penalty)
   lasso_model.fit(X_Train, y_Train)
   model_dict = {}
   for i in range(len(final_features_df.columns)) :
      model_dict[feature_names[i]] = np.around(lasso_model.coef_[i-2], decimals = 3)
   lasso_model_params_df = lasso_model_params_df.append(model_dict, ignore_index = True)
   return lasso_model_params_df
penalty_parameter = 0.0005
lasso_df = get_lasso_parameters(penalty_parameter)
column_names = list()
lasso_param_dictionary = {}
```

```
for column_name in lasso_df.columns:
    column = lasso_df[column_name]
    count_of_non_zeros = (column != 0).sum()
    if count_of_non_zeros != 0 :
        column_names.append(column_name)
        lasso_param_dictionary[column_name] = column[0]
print(f'Total Non-zero Columns: {len(column_names)}')
print(f'Selected Non-zero Column Names: {column_names}')
lasso_non_zero_params_df = pd.DataFrame(columns = ['Feature Name', 'Feature Weight'])
for key in lasso_param_dictionary:
    lasso_non_zero_params_df.loc[len(lasso_non_zero_params_df.index)] = [key, lasso_param_dictionary[key]]
display(lasso_non_zero_params_df)
# # Models and Hyper-parameter tuning
# ## Normalising Selected Features Set
selected_features = final_features_df[column_names]
scaler_selected_features = MinMaxScaler()
scaler_selected_features.fit(selected_features)
scaled_selected_features = scaler_selected_features.fit_transform(selected_features)
scaled_selected_features_df = pd.DataFrame(scaled_selected_features, columns = selected_features.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_location)
scaled_label = scaler_label.fit_transform(Y_review_scores_location)
scaled_Y_review_scores_location = pd.DataFrame(scaled_label, columns = Y_review_scores_location.columns)
display(scaled_selected_features_df)
X_train, X_test, y_train, y_test = train_test_split(scaled_selected_features_df, scaled_Y_review_scores_location, test_size = 0.2, random_state
= 111)
scaled_train_feature = np.array(X_train)
scaled_train_label = np.array(y_train)
scaled_test_feature = np.array(X_test)
scaled_test_label = np.array(y_test)
# ## Baseline Model: Linear Regression
Baseline_Linear_Model = LinearRegression()
val_score = cross_val_score(Baseline_Linear_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Baseline Linear Regression is: {abs(np.array(val_score).mean())}')
# ## Model 1: Tuned Lasso Regression
Lasso_Model = Lasso(alpha = 0.0005)
lasso_val_score = cross_val_score(Lasso_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Lasso Model is: {abs(np.array(lasso_val_score).mean())}')
# ## Model 2: KNN Regression
knn_score = []
neighbors = [x for x in range(1, (len(column_names) + 1), 2)]
for neighbor in neighbors:
    KNN_Model = KNeighborsRegressor(n_neighbors = neighbor)
    knn_val_score = cross_val_score(KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
    knn_score.append(abs(np.array(knn_val_score).mean()))
    print(f'Cross Validation Score for KNN Regressor is: {abs(np.array(knn_val_score).mean())}', " for n_neighbors = ", neighbor)
plt.plot(neighbors, knn_score, marker = 'o')
plt.xlabel("n_Neighbors")
plt.ylabel("Cross val Score")
plt.title("Plot of CVS Vs. N_Neighbor parameter for KNN Regressor")
plt.show()
# ### Tuned KNN Model
Tuned_KNN_Model = KNeighborsRegressor(n_neighbors = 13)
tuned_knn_val_score = cross_val_score(Tuned_KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring =
'neg_mean_squared_error')
print(f'Cross Validation Score for KNN Regression is: {abs(np.array(tuned_knn_val_score).mean())}')
```

9. Feature_Selection_Model_Tuning_ReviewScores_Rating

```
# ## Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso, LinearRegression
import warnings
warnings.filterwarnings("ignore")
# # Feature Selection
# ## Reading review features data and loading it in Dataframe
reviews_features_file = 'data/review_features.csv'
reviews_features_df = pd.read_csv(reviews_features_file)
display(reviews_features_df)
# ## Reading lisiting features data and loading it in Dataframe
listings_features_file = 'data/listing_features.csv'
listings_features_df = pd.read_csv(listings_features_file)
display(listings_features_df)
listings_features_df.drop('Unnamed: 0', axis=1, inplace=True)
display(listings_features_df)
# ## Merging review and listing features based on listing id
all_features_df = pd.merge(listings_features_df, reviews_features_df, on='listing_id', how='left')
display(all_features_df)
# ## Dropping listing id column from final feature set and replacing any null value with 0
all_features_df.drop('listing_id', axis=1, inplace=True)
all_features_df.fillna(0, inplace = True)
display(all_features_df)
# ## Confirming Datatypes for all features in dataframe
datatypes = pd.DataFrame(all_features_df.dtypes)
pd.set_option('display.max_rows', None)
display(datatypes)
# ## Confirming if there any null values in any column
all_features_df.isnull().sum()
# ## Separating labels from features
Y_review_scores_rating = pd.DataFrame(all_features_df['review_scores_rating'])
final_features_df = all_features_df.copy()
final_features_df.drop('review_scores_rating', axis=1, inplace=True)
final_features_df.drop('review_scores_accuracy', axis=1, inplace=True)
final_features_df.drop('review_scores_cleanliness', axis=1, inplace=True)
final_features_df.drop('review_scores_checkin', axis=1, inplace=True)
final_features_df.drop('review_scores_communication', axis=1, inplace=True)
final_features_df.drop('review_scores_location', axis=1, inplace=True)
final_features_df.drop('review_scores_value', axis=1, inplace=True)
pd.reset_option('^display.', silent=True)
display(final_features_df)
# ## Feature Selection using Lasso Regression
# ### Scaling the data using MinMax scaling
#
scaler_features = MinMaxScaler()
scaler_features.fit(final_features_df)
scaled_features = scaler_features.fit_transform(final_features_df)
scaled_features_df = pd.DataFrame(scaled_features, columns = final_features_df.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_rating)
scaled_label = scaler_label.fit_transform(Y_review_scores_rating)
scaled_Y_review_scores_rating = pd.DataFrame(scaled_label, columns = Y_review_scores_rating.columns)
display(scaled_features_df)
scaled_features_array = np.array(scaled_features_df)
scaled_label_array = np.array(scaled_Y_review_scores_rating)
def calculate_mse_stddev_penalty_lasso(penalty_parameters) :
    k_fold_split = 5
    k_fold_split_function =  KFold(n_splits = k_fold_split)
    mean_sqaure_error_penalty = []
    standard_deviation_penalty = []
    for penalty in penalty_parameters :
        lasso_model = Lasso(alpha = penalty)
        mean_sqaure_error_fold = []
        for train_index, test_index in k_fold_split_function.split(scaled_features_array):
            X_train, X_test = scaled_features_array[train_index], scaled_features_array[test_index]
            y_train, y_test = scaled_label_array[train_index], scaled_label_array[test_index]
            lasso_model.fit(X_train, y_train)
            predictions = lasso_model.predict(X_test)
            mean_sqaure_error_fold.append(mean_squared_error(y_test, predictions))
        mean_sqaure_error_penalty.append(np.array(mean_sqaure_error_fold).mean())
        standard_deviation_penalty.append(np.array(mean_sqaure_error_fold).std())
    return mean_sqaure_error_penalty, standard_deviation_penalty
penalty_parameters = [1, 5, 10, 100, 500, 1000]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
penalty_parameters =  [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
# ## Getting all Non-Zero features after Tuned Lasso Regression
feature_names = list(final_features_df.columns.values)
def get_lasso_parameters(penalty) :
    lasso_model_dictionary = {}
    X_Train,X_Test,y_Train,y_Test = train_test_split(scaled_features_array, scaled_label_array, test_size = 0.2, random_state=111, shuffle =
False)
    lasso_model_params_df = pd.DataFrame(columns = feature_names)
    lasso_model = Lasso(alpha = penalty)
    lasso_model.fit(X_Train, y_Train)
    model_dict = {}
    for i in range(len(final_features_df.columns)) :
        model_dict[feature_names[i]] = np.around(lasso_model.coef_[i-2], decimals = 3)
    lasso_model_params_df = lasso_model_params_df.append(model_dict, ignore_index = True)
    return lasso_model_params_df
penalty_parameter = 0.0005
lasso_df = get_lasso_parameters(penalty_parameter)
column_names = list()
lasso_param_dictionary = {}
for column_name in lasso_df.columns:
    column = lasso_df[column_name]
    count_of_non_zeros = (column != 0).sum()
    if count_of_non_zeros != 0 :
        column_names.append(column_name)
        lasso_param_dictionary[column_name] = column[0]
print(f'Total Non-zero Columns: {len(column_names)}')
print(f'Selected Non-zero Column Names: {column_names}')
lasso_non_zero_params_df = pd.DataFrame(columns = ['Feature Name', 'Feature Weight'])
for key in lasso_param_dictionary:
    lasso_non_zero_params_df.loc[len(lasso_non_zero_params_df.index)] = [key, lasso_param_dictionary[key]]
display(lasso_non_zero_params_df)
# # Models and Hyper-parameter tuning
# ## Normalising Selected Features Set
selected_features = final_features_df[column_names]
scaler_selected_features = MinMaxScaler()
scaler_selected_features.fit(selected_features)
scaled_selected_features = scaler_selected_features.fit_transform(selected_features)
scaled_selected_features_df = pd.DataFrame(scaled_selected_features, columns = selected_features.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_rating)
scaled_label = scaler_label.fit_transform(Y_review_scores_rating)
scaled_Y_review_scores_rating = pd.DataFrame(scaled_label, columns = Y_review_scores_rating.columns)
display(scaled_selected_features_df)
X_train, X_test, y_train, y_test = train_test_split(scaled_selected_features_df, scaled_Y_review_scores_rating, test_size = 0.2, random_state =
111)
scaled_train_feature = np.array(X_train)
scaled_train_label = np.array(y_train)
scaled_test_feature = np.array(X_test)
```

```
scaled_test_label = np.array(y_test)
# ## Baseline Model: Linear Regression
Baseline_Linear_Model = LinearRegression()
val_score = cross_val_score(Baseline_Linear_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Baseline Linear Regression is: {abs(np.array(val_score).mean())}')
# ## Model 1: Tuned Lasso Regression
Lasso_Model = Lasso(alpha = 0.0005)
lasso_val_score = cross_val_score(Lasso_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Lasso Model is: {abs(np.array(lasso_val_score).mean())}')
# ## Model 2: KNN Regression
knn_score = []
neighbors = [x for x in range(1, (len(column_names) + 1), 2)]
for neighbor in neighbors:
   KNN_Model = KNeighborsRegressor(n_neighbors = neighbor)
   knn_val_score = cross_val_score(KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
   knn_score.append(abs(np.array(knn_val_score).mean()))
   print(f'Cross Validation Score for KNN Regressor is: {abs(np.array(knn_val_score).mean())}', " for n_neighbors = ", neighbor)
plt.plot(neighbors, knn_score, marker = 'o')
plt.xlabel("n_Neighbors")
plt.ylabel("Cross val Score")
plt.title("Plot of CVS Vs. N_Neighbor parameter for KNN Regressor")
plt.show()
# ### Tuned KNN Model
Tuned_KNN_Model = KNeighborsRegressor(n_neighbors = 9)
tuned_knn_val_score = cross_val_score(Tuned_KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring =
'neg_mean_squared_error')
print(f'Cross Validation Score for KNN Regression is: {abs(np.array(tuned_knn_val_score).mean())}')
```

10. Feature_Selection_Model_Tuning_ReviewScores_Value

```
# ## Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso, LinearRegression
import warnings
warnings.filterwarnings("ignore")
# # Feature Selection
# ## Reading review features data and loading it in Dataframe
reviews_features_file = 'data/review_features.csv'
reviews_features_df = pd.read_csv(reviews_features_file)
display(reviews_features_df)
# ## Reading lisiting features data and loading it in Dataframe
listings_features_file = 'data/listing_features.csv'
listings_features_df = pd.read_csv(listings_features_file)
display(listings_features_df)
listings_features_df.drop('Unnamed: 0', axis=1, inplace=True)
display(listings_features_df)
# ## Merging review and listing features based on listing id
all_features_df = pd.merge(listings_features_df, reviews_features_df, on='listing_id', how='left')
display(all_features_df)
# ## Dropping listing id column from final feature set and replacing any null value with 0
all_features_df.drop('listing_id', axis=1, inplace=True)
all_features_df.fillna(0, inplace = True)
display(all_features_df)
# ## Confirming Datatypes for all features in dataframe
datatypes = pd.DataFrame(all_features_df.dtypes)
pd.set_option('display.max_rows', None)
display(datatypes)
# ## Confirming if there any null values in any column
all_features_df.isnull().sum()
# ## Separating labels from features
```

```
Y_review_scores_value = pd.DataFrame(all_features_df['review_scores_value'])
final_features_df = all_features_df.copy()
final_features_df.drop('review_scores_rating', axis=1, inplace=True)
final_features_df.drop('review_scores_accuracy', axis=1, inplace=True)
final_features_df.drop('review_scores_cleanliness', axis=1, inplace=True)
final_features_df.drop('review_scores_checkin', axis=1, inplace=True)
final_features_df.drop('review_scores_communication', axis=1, inplace=True)
final_features_df.drop('review_scores_location', axis=1, inplace=True)
final_features_df.drop('review_scores_value', axis=1, inplace=True)
pd.reset_option('^display.', silent=True)
display(final_features_df)
# ## Feature Selection using Lasso Regression
# ### Scaling the data using MinMax scaling
scaler_features = MinMaxScaler()
scaler_features.fit(final_features_df)
scaled_features = scaler_features.fit_transform(final_features_df)
scaled_features_df = pd.DataFrame(scaled_features, columns = final_features_df.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_value)
scaled_label = scaler_label.fit_transform(Y_review_scores_value)
scaled_Y_review_scores_value = pd.DataFrame(scaled_label, columns = Y_review_scores_value.columns)
display(scaled_features_df)
scaled_features_array = np.array(scaled_features_df)
scaled_label_array = np.array(scaled_Y_review_scores_value)
def calculate_mse_stddev_penalty_lasso(penalty_parameters) :
    k_fold_split = 5
    k_fold_split_function =  KFold(n_splits = k_fold_split)
    mean_sqaure_error_penalty = []
    standard_deviation_penalty = []
    for penalty in penalty_parameters :
        lasso_model = Lasso(alpha = penalty)
        mean_sqaure_error_fold = []
        for train_index, test_index in k_fold_split_function.split(scaled_features_array):
            X_train, X_test = scaled_features_array[train_index], scaled_features_array[test_index]
            y_train, y_test = scaled_label_array[train_index], scaled_label_array[test_index]
            lasso_model.fit(X_train, y_train)
            predictions = lasso_model.predict(X_test)
            mean_sqaure_error_fold.append(mean_squared_error(y_test, predictions))
        mean_sqaure_error_penalty.append(np.array(mean_sqaure_error_fold).mean())
        standard_deviation_penalty.append(np.array(mean_sqaure_error_fold).std())
    return mean_sqaure_error_penalty, standard_deviation_penalty
penalty_parameters = [1, 5, 10, 100, 500, 1000]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
penalty_parameters =  [0.0005, 0.001, 0.005, 0.01]
mean_sqaure_error, standard_deviation = calculate_mse_stddev_penalty_lasso(penalty_parameters)
plt.figure()
plt.errorbar(penalty_parameters, mean_sqaure_error, yerr = standard_deviation, color = 'blue')
plt.xlabel('Penalty')
plt.ylabel('Mean square error')
plt.title('Plot of MSE VS Penalty for 5-fold Cross Validation Lasso Regression')
plt.show()
# ## Getting all Non-Zero features after Tuned Lasso Regression
feature_names = list(final_features_df.columns.values)
def get_lasso_parameters(penalty) :
    lasso_model_dictionary = {}
    X_Train,X_Test,y_Train,y_Test = train_test_split(scaled_features_array, scaled_label_array, test_size = 0.2, random_state=111, shuffle =
False)
    lasso_model_params_df = pd.DataFrame(columns = feature_names)
    lasso_model = Lasso(alpha = penalty)
    lasso_model.fit(X_Train, y_Train)
    model_dict = {}
    for i in range(len(final_features_df.columns)) :
        model_dict[feature_names[i]] = np.around(lasso_model.coef_[i-2], decimals = 3)
    lasso_model_params_df = lasso_model_params_df.append(model_dict, ignore_index = True)
    return lasso_model_params_df
penalty_parameter = 0.0005
lasso_df = get_lasso_parameters(penalty_parameter)
column_names = list()
lasso_param_dictionary = {}
for column_name in lasso_df.columns:
    column = lasso_df[column_name]
    count_of_non_zeros = (column != 0).sum()
    if count_of_non_zeros != 0 :
        column_names.append(column_name)
        lasso_param_dictionary[column_name] = column[0]
print(f'Total Non-zero Columns: {len(column_names)}')
print(f'Selected Non-zero Column Names: {column_names}')
lasso_non_zero_params_df = pd.DataFrame(columns = ['Feature Name', 'Feature Weight'])
for key in lasso_param_dictionary:
    lasso_non_zero_params_df.loc[len(lasso_non_zero_params_df.index)] = [key, lasso_param_dictionary[key]]
display(lasso_non_zero_params_df)
# # Models and Hyper-parameter tuning
# ## Normalising Selected Features Set
selected_features = final_features_df[column_names]
scaler_selected_features = MinMaxScaler()
scaler_selected_features.fit(selected_features)
scaled_selected_features = scaler_selected_features.fit_transform(selected_features)
scaled_selected_features_df = pd.DataFrame(scaled_selected_features, columns = selected_features.columns)
scaler_label = MinMaxScaler()
scaler_label.fit(Y_review_scores_value)
scaled_label = scaler_label.fit_transform(Y_review_scores_value)
scaled_Y_review_scores_value = pd.DataFrame(scaled_label, columns = Y_review_scores_value.columns)
display(scaled_selected_features_df)
X_train, X_test, y_train, y_test = train_test_split(scaled_selected_features_df, scaled_Y_review_scores_value, test_size = 0.2, random_state =
111)
scaled_train_feature = np.array(X_train)
scaled_train_label = np.array(y_train)
scaled_test_feature = np.array(X_test)
scaled_test_label = np.array(y_test)
# ## Baseline Model: Linear Regression
Baseline_Linear_Model = LinearRegression()
val_score = cross_val_score(Baseline_Linear_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Baseline Linear Regression is: {abs(np.array(val_score).mean())}')
# ## Model 1: Tuned Lasso Regression
Lasso_Model = Lasso(alpha = 0.0005)
lasso_val_score = cross_val_score(Lasso_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
print(f'Cross Validation Score for Lasso Model is: {abs(np.array(lasso_val_score).mean())}')
# ## Model 2: KNN Regression
knn_score = []
neighbors = [x for x in range(1, (len(column_names) + 1), 2)]
for neighbor in neighbors:
    KNN_Model = KNeighborsRegressor(n_neighbors = neighbor)
    knn_val_score = cross_val_score(KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring = 'neg_mean_squared_error')
    knn_score.append(abs(np.array(knn_val_score).mean()))
    print(f'Cross Validation Score for KNN Regressor is: {abs(np.array(knn_val_score).mean())}', " for n_neighbors = ", neighbor)
plt.plot(neighbors, knn_score, marker = 'o')
plt.xlabel("n_Neighbors")
plt.ylabel("Cross val Score")
plt.title("Plot of CVS Vs. N_Neighbor parameter for KNN Regressor")
plt.show()
# ### Tuned KNN Model
Tuned_KNN_Model = KNeighborsRegressor(n_neighbors = 11)
tuned_knn_val_score = cross_val_score(Tuned_KNN_Model, scaled_train_feature, scaled_train_label, cv = 5,scoring =
'neg_mean_squared_error')
print(f'Cross Validation Score for KNN Regression is: {abs(np.array(tuned_knn_val_score).mean())}')
```