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**

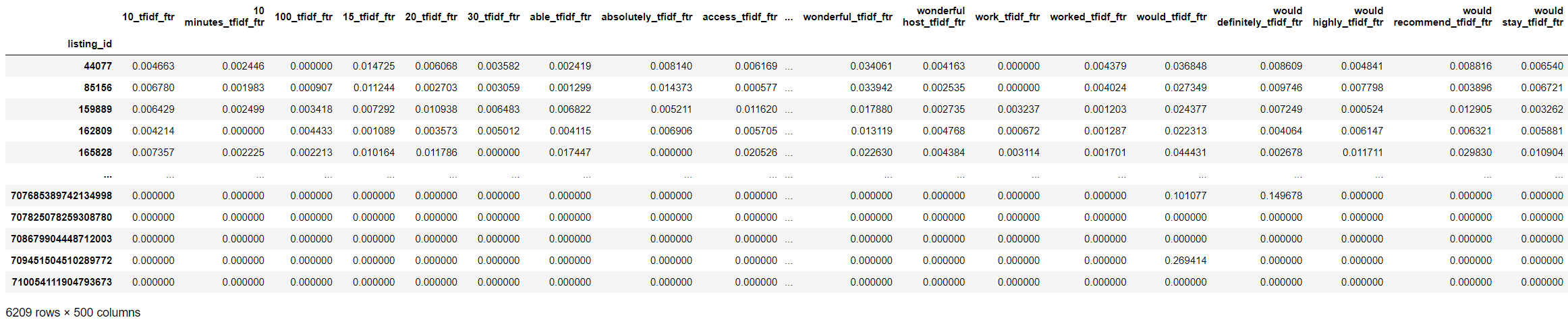
The reviews.csv file contained reviews in many languages and these reviews also had many special characters. To clean the reviews, I have first removed all the special tags from reviews data like **‘</br>’, ‘\n’, ‘\r’, ‘\t’**. Then, I removed all special characters from the data like emojis. To achieve 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** functionfrom **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**

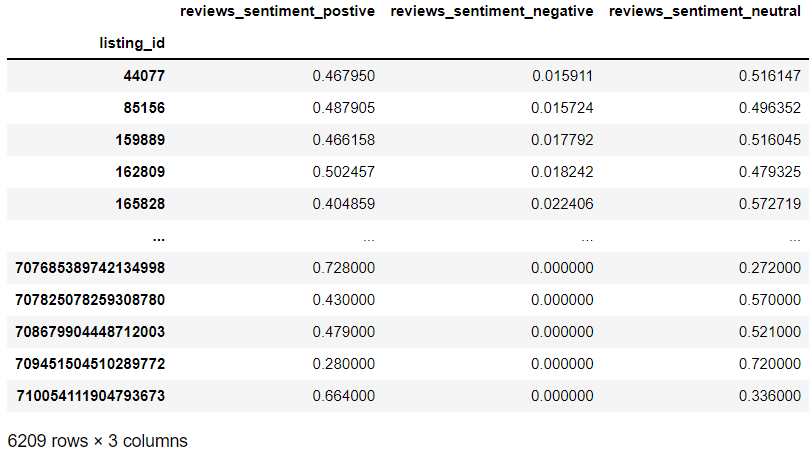
I loaded the translated comments into a data-frame and replaced all the null values with **none** text before extracting features. Furthermore, I have also removed all the common English Stop words like a, an, the, in, etc. from the comments. Primary reason for removing stop words is that stop words 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:



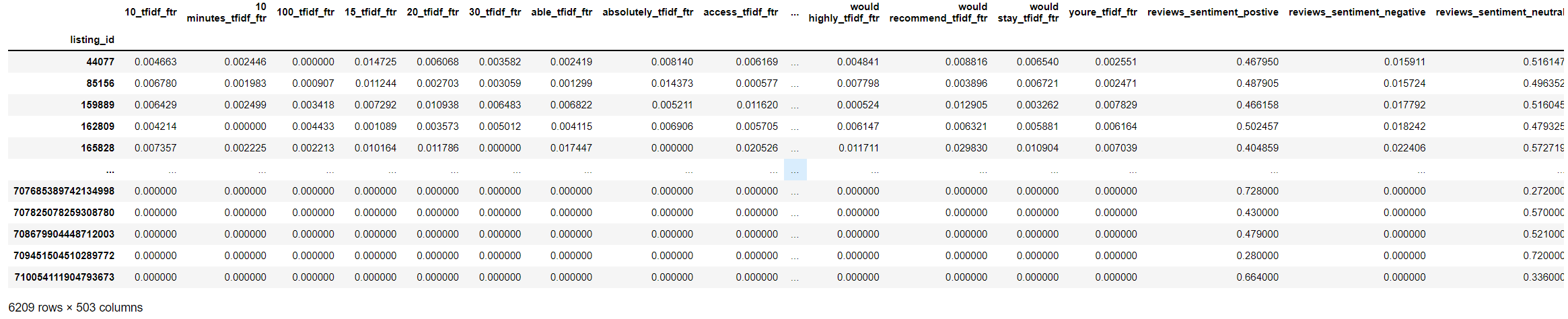
**Figure 1: TF-IDF Features extracted for each listing based on user reviews**

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.



**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**.



**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 |

1. **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 |

1. **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.

1. **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

1. **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.

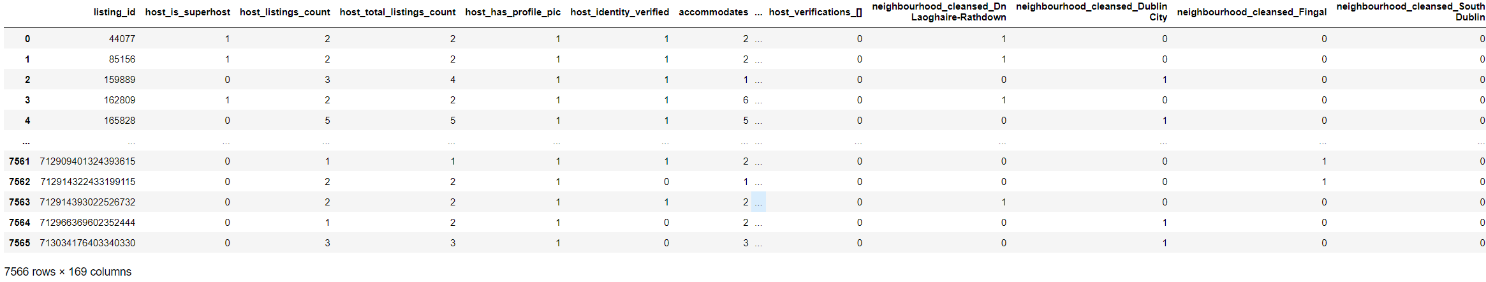
1. **One-Hot encoding on categorical columns**

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.

1. **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:

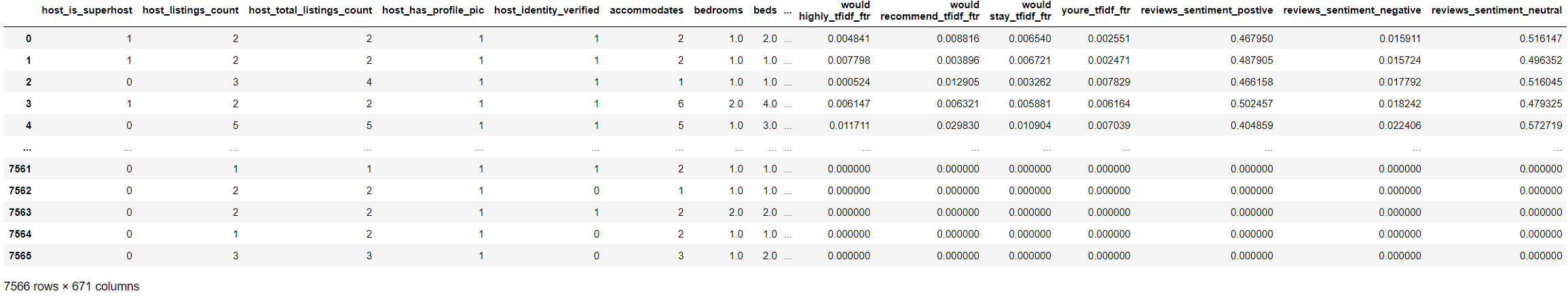


**Figure 4: Final Features Set for listings data**

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.



**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 featuresto 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.

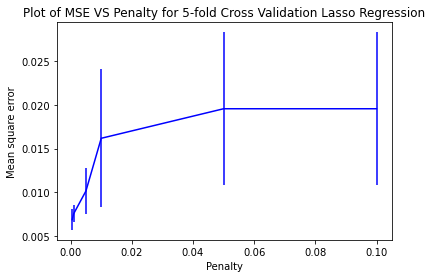
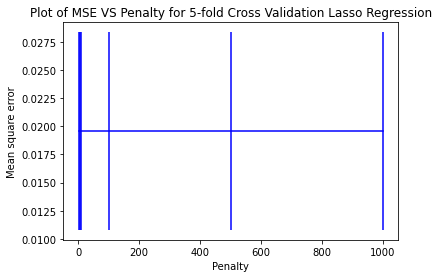
**Metric used for Performance Comparison: 5-fold cross validation with Mean Square Error**

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 strategyas it explains the closeness the predicted values to the actual value. Therefore, it is an ideal strategy for regression problem.

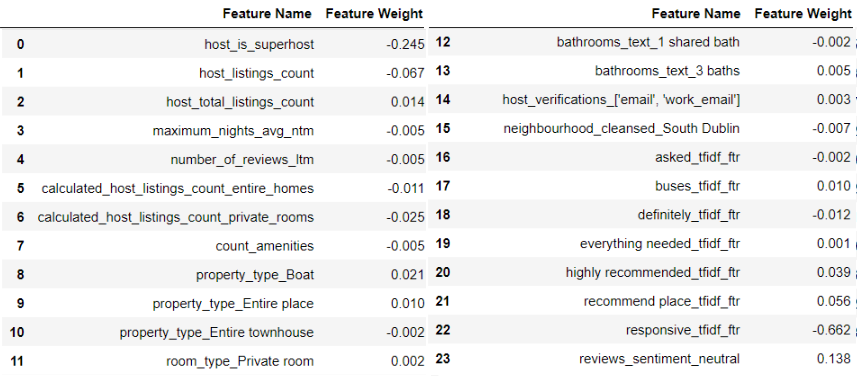
Features Selections and Models Tuning for Review Score Rating**:**

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:

****

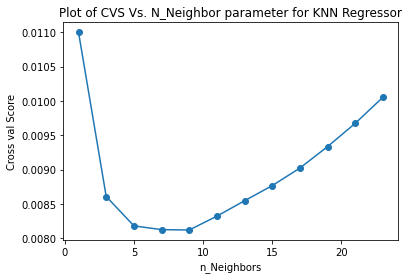
**Figure 7: Significant parameters identified using Lasso Regression**

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

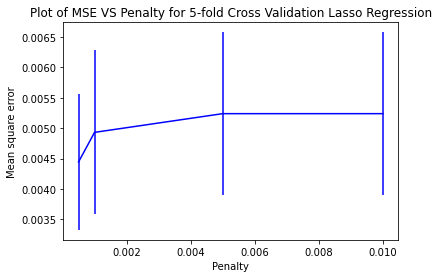
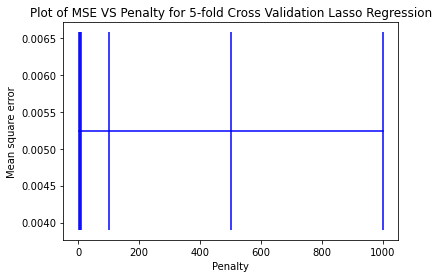


I can identify that for **n=9** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0081.**

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.

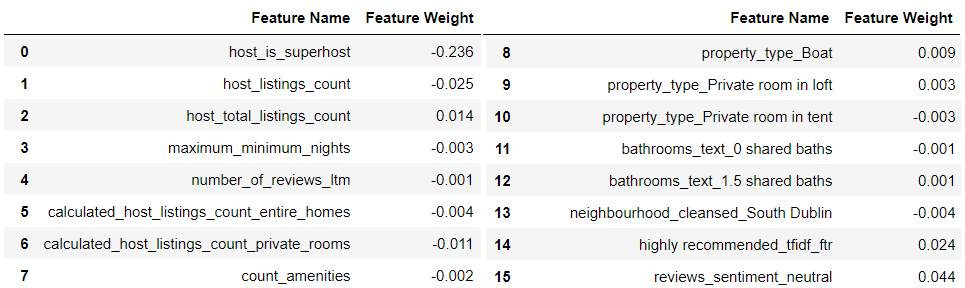
Features Selections and Models Tuning for Review Score Accuracy**:**

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.



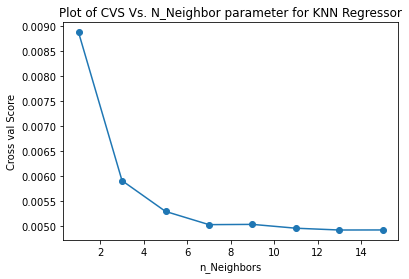
**Figure 9: Significant parameters identified using Lasso Regression**

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

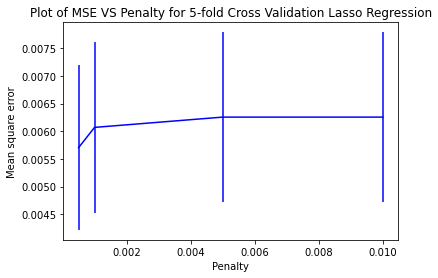
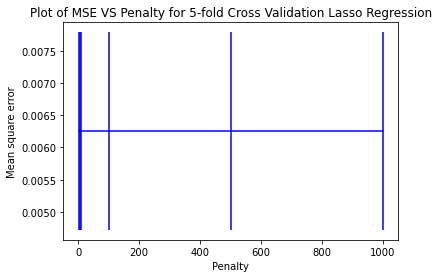


I can identify that for **n=7** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0050.**

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.

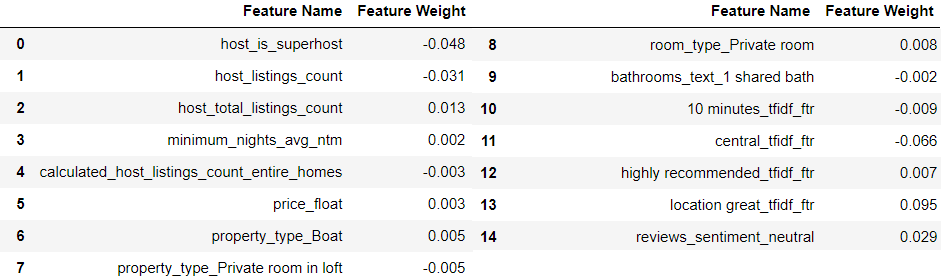
Features Selections and Models Tuning for Review Score Location**:**

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.



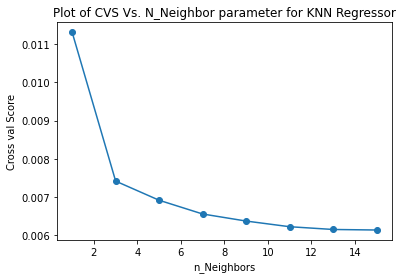
**Figure 11: Significant parameters identified using Lasso Regression**

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

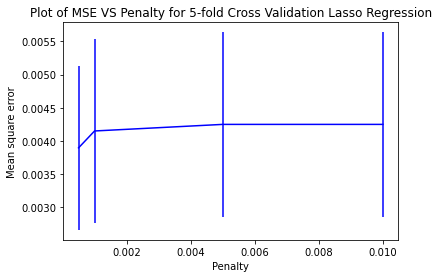
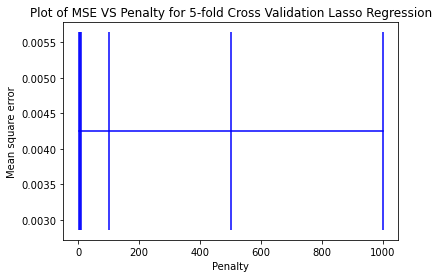


I can identify that for **n=13** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0061.**

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.

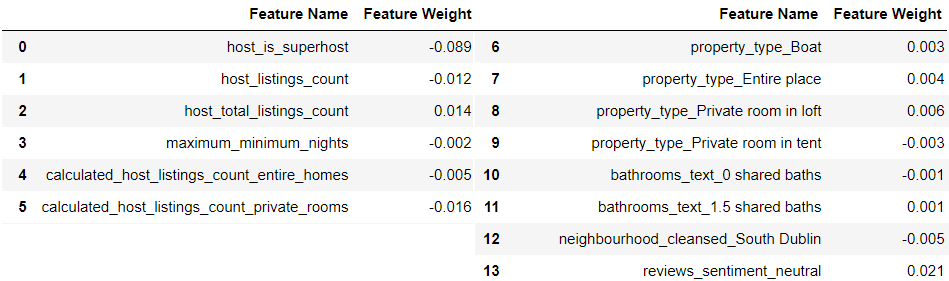
Features Selections and Models Tuning for Review Score Check-in**:**

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.



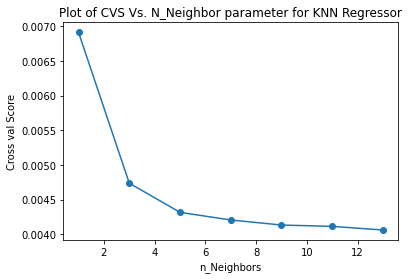
**Figure 13: Significant parameters identified using Lasso Regression**

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

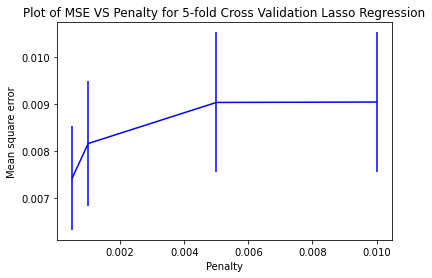
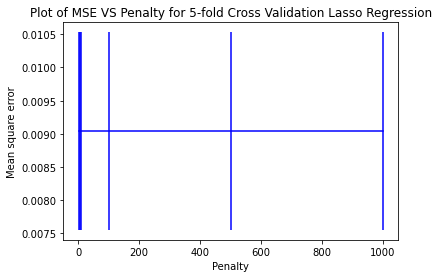


I can identify that for **n=9** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0041.**

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.

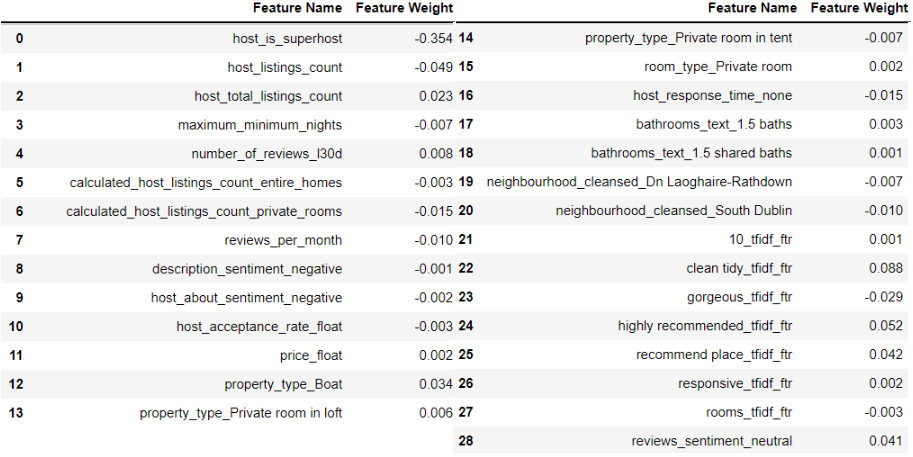
Features Selections and Models Tuning for Review Score Cleanliness**:**

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.



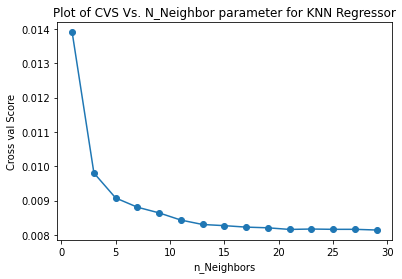
**Figure 15: Significant parameters identified using Lasso Regression**

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

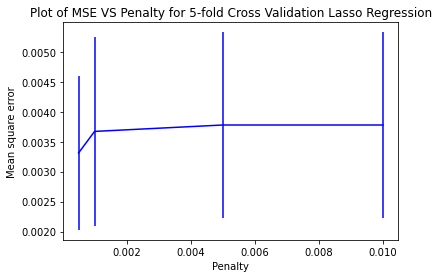
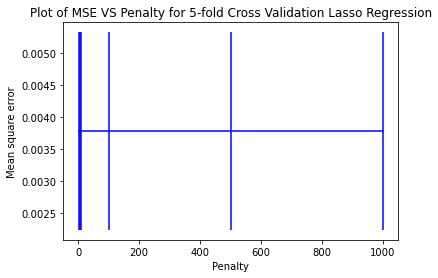


I can identify that for **n=21** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0081.**

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.

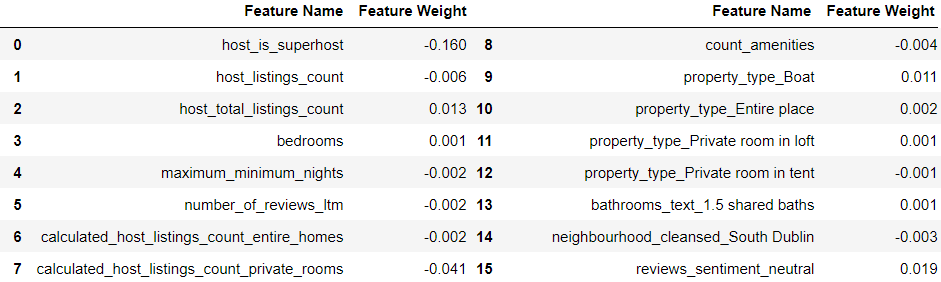
Features Selections and Models Tuning for Review Score Communication**:**

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.



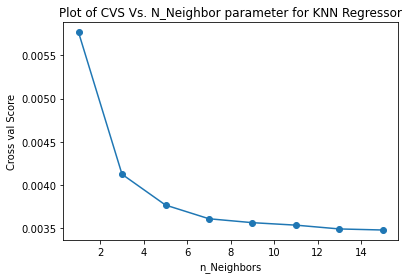
**Figure 17: Significant parameters identified using Lasso Regression**

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

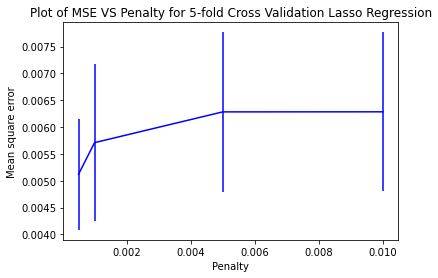
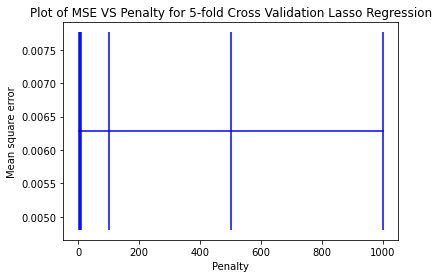


I can identify that for **n=13** we are getting the best MSE and using this **the Cross Validation Score (absolute of negative MSE) for KNN Regression is: 0.0034.**

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.

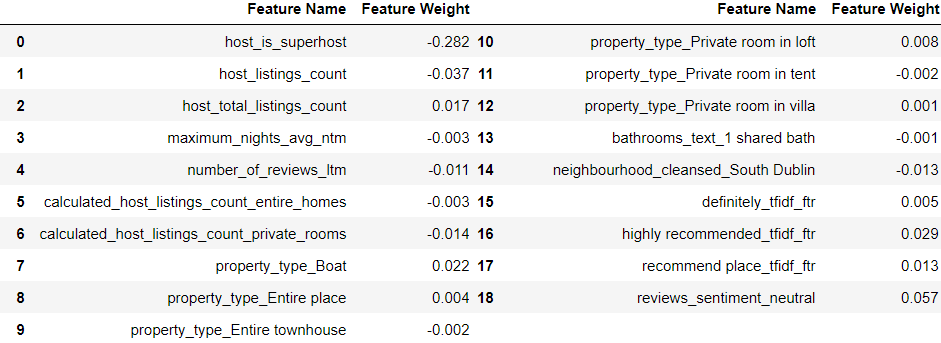
Features Selections and Models Tuning for Review Score Value**:**

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.



**Figure 19: Significant parameters identified using Lasso Regression**

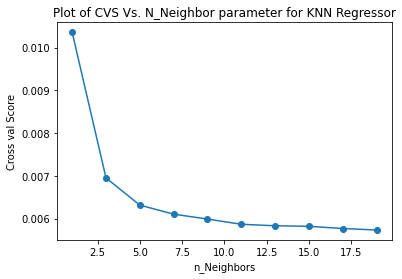
For communication rating, **host is super-host** is identified as significant as along with **number of reviews in last 12 months**.

**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.

1. **KNN Classifier Vs. Neural Net classifier**

**Advantages of KNN Classifier:** Themain 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.

1. **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.

1. **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())}')