Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

DECLARATION

I understand that this is an individual assessment and that collaboration is not permitted. I have not received any assistance with my work for this assessment. Where I have used the published work of others, I have indicated this with appropriate citation.

I have not and will not share any part of my work on this assessment, directly or indirectly, with any other student.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write."

I understand that by returning this declaration with my work, I am agreeing with the above statement. ☐

Name: Vipul Ghare

Date: 04th Janurary 2023

Question1

- We were given 2 CSV files, one of which had data about the Listings that are listed on Airbnb in Dublin, Ireland and the second dataset had information about the comments given by a reviewer.
- Our task is to find the scores of review_scores_rating, review_scores_ accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_ communication, review_scores_location, review_scores_value for each listing using machine learning techniques.
- To obtain this firstly, I started working on the listings.csv dataset.
- As we have to predict the scores, such as rating, accuracy etc. there are some features which won't play importance in deciding the rating of the listings, so these need to be handled.

Lisitings.csv

Feature Name	Count	Datatype	Action	Reason
id	7566	int64	nothing	Kept as it, as we need to merge review.csv on basis of listingID
listing_url	7566	object	dropped	Wont play any role in deciding the output feature
scrape_id	7566	int64	dropped	Wont play any role in deciding the output feature
last_scraped	7566	object	dropped	Wont play any role in deciding the output feature
source	7566	object	dropped	Wont play any role in deciding the output feature
name	7566	object	dropped	Wont play any role in deciding the output feature
description	7411	object	tfidf	Considered max_features as 500
neighborhood_overview	4194	object	tfidf	Considered max_features as 500
picture_url	7566	object	dropped	Wont play any role in deciding the output feature
host_id	7566	int64	dropped	Wont play any role in deciding the output feature
host_url	7566	object	dropped	Wont play any role in deciding the output feature
host_name	7566	object	dropped	Wont play any role in deciding the output feature
host_since	7566	object	converted to days	Subtracted the field value from current date, got the number of days
host_location	6232	object	dropped	Wont play any role in deciding the output feature
host_about	3556	object	dropped	Enough data not avaible
neighbourhood_group_cleansed	0	float64	dropped	Enough data not avaible
latitude	7566	float64	using smithield	Explainded below
longitude	7566	float64	using smithfield	Explained Below
property_type	7566	object	ohe	Directly performed One hot encoding
room_type	7566	object	ohe	Directly performed One hot encoding
accommodates	7566	int64	nothing	Only scaled this using Standard Scalar
bathrooms	0	float64	dropped	Enough data not avaible
bathrooms_text	7562	object	handled manually	truncated textual values and only considered numerical
bedrooms	7361	float64	standardscaler	Only scaled this using Standard Scalar
beds	7472	float64	standardscaler	Only scaled this using Standard Scalar
amenities	7566	object	countervectorizer	
price	7566	object	remove euros sign	Removed euros sign and the Standard Scalar
minimum_nights	7566	int64	standardscaler	Only scaled this using Standard Scalar
host_response_time	3932	object	dropped	Enough data not avaible
host_response_rate	3932	object	dropped	Enough data not avaible
host_acceptance_rate	4191	object	dropped	Wont play any role in deciding the output feature
host_is_superhost	7566	object	t/f	True = 0, False = 1
host_thumbnail_url	7566	object	dropped	Wont play any role in deciding the output feature
host_picture_url	7566	object	dropped	Wont play any role in deciding the output feature
host_neighbourhood	5234	object	dropped	Wont play any role in deciding the output feature
host_listings_count	7566	int64	nothing	Only scaled this using Standard Scalar
host_total_listings_count	7566	int64	nothing	Only scaled this using Standard Scalar
host_verifications	7566	object	multilabel binarizer	considered only unique values in it and then used one hot encoding
host_has_profile_pic	7566	object	dropped	Wont play any role in deciding the output feature
host_identity_verified	7566	object	t/f	True = 0, False = 1
neighbourhood	4194	object	dropped	Enough data not avaible
neighbourhood_cleansed	7566	object	ohe	Directly performed One hot encoding
maximum_maximum_nights	7566	int64	standardscaler	Only scaled this using Standard Scalar
minimum_nights_avg_ntm	7566	float64	standardscaler	Only scaled this using Standard Scalar
maximum_nights_avg_ntm	7566	float64	standardscaler	Only scaled this using Standard Scalar
calendar_updated	0	float64	dropped	Enough data not avaible
has_availability	7566	object	t/f	True = 0, False = 1
availability_30	7566	int64	standardscaler	Only scaled this using Standard Scalar
availability_60	7566	int64	standardscaler	Only scaled this using Standard Scalar
availability_90	7566	int64	standardscaler	Only scaled this using Standard Scalar
availability_365	7566	int64	standardscaler	Only scaled this using Standard Scalar
calendar_last_scraped	7566	object	dropped	Wont play any role in deciding the output feature
number_of_reviews	7566	int64	standardscaler	Only scaled this using Standard Scalar
number_of_reviews_ltm	7566	int64	standardscaler	Only scaled this using Standard Scalar
number_of_reviews_l30d	7566	int64	standardscaler	Only scaled this using Standard Scalar
first_review	6209	object	dateformat	Subtracted the field value from current date, got the number of days
last_review	6209	object	dateformat	Subtracted the field value from current date, got the number of days
review_scores_rating	6209	float64	took power of 4	Transformed to a higher value
review_scores_accuracy	6085	float64	took power of 4	Transformed to a higher value
review_scores_cleanliness	6086	float64	took power of 4	Transformed to a higher value
review_scores_checkin	6081	float64	took power of 4	Transformed to a higher value
review_scores_communication	6085	float64	took power of 4	Transformed to a higher value
review_scores_location	6081	float64	took power of 4	Transformed to a higher value
review_scores_value	6079	float64	took power of 4	Transformed to a higher value
license	0	float64	dropped	Enough data not avaible
instant_bookable	7566	object	t/f	True = 0, False = 1
calculated_host_listings_count	7566	int64	standardscaler	Only scaled this using Standard Scalar
calculated_host_listings_count_entire_homes	7566	int64	standardscaler	Only scaled this using Standard Scalar
calculated_host_listings_count_private_rooms	7566	int64	standardscaler	Only scaled this using Standard Scalar
calculated_host_listings_count_shared_rooms	7566	int64	standardscaler	Only scaled this using Standard Scalar
reviews_per_month	6209	float64	standardscaler	Only scaled this using Standard Scalar

- The above table shows every feature in the listings.csv, what action I have taken to handle this feature, and why I have this action.
- Some of the features were dropped as enough data was not present in the feature. E.g. Host_about had a 3556 data count which is less than 50% of the total count(7566), so such features were dropped.
- Textual features such as description/neighbourhood_overview were handled by converting them into numerical using TFIDF vectorizer.
- For features such as host_since, the date in the field was subtracted from the current date and I got the number of days count as the data.
- For Latitude and Longitude, I divided the Dublin City map into 4 zone quadrants such as A, B, C and D. I looked at the centre and I got (Smithfield 53.346978, -6.281294) as the centre. I considered the latitude and longitude of each listing and decided whether it lies in zone A, B, C or D using Smithfield latitude and longitude coordinates.
- I converted the true/false field into 1,0 such as host_is_superhost.
- Features such as amenities, and host_verifications needed some other computation. For this, I had to first split the elements from the field, then apply a count vectorizer on it and I considered only the top 200 amenities that a listing may have which have occurred frequently in the dataset.
- All the target features such as review_scores_ratings, and review_scores_checkin are very close values, so I took the power of 4 for these features so that the difference between the values of the target features increases and the model can understand the difference.

1.1 Treat missing values

- The missing values are always needed to be handled.
- If the data type of the feature is 'object', I replaced the missing value with the value which has mostly appeared.
- For the datatype as 'float/int'. I replaced the missing value with the value which is obtained by taking the mean of the feature.

1.2 Data Transformation.

There may be some feature values in the dataset which have a high magnitude and the model may tend to give high priority to these features e.g. a simple dataset having age and salary, Salary may be in 100k and age is in between 20-60. But both the features are important to us and the model here may tend to give more weightage to salary. So by the transformation of the features, we bring the data in between 0 to 1. Transformation doesn't change the meaning of the feature.

- Transformations methods such as min-max scalar and standard scalar can be used.
- I have used a standard scalar that takes numerical data features and scales them.

1.3 One hot encoding.

- At this point, all our numerical features are been scaled but the object data has to be converted to numerical data.
- So one hot coding is been done to obtain this.
- Features such as property_type, and room_type are been converted to numerical format using one hot encoding.
-

Reviews.csv

- Reviews.csv contain every comment on a listing by a reviewer.
- In this dataset, we only require two columns [listingid, comments]. Rest all columns can be dropped.
- To train our model we need our data in numerical format, and we need to convert comments into the numerical format.
- To obtain this we can use either the TFIDF vectorizer/bag of words/ Count vectorizer of these models.
- But before that, it was been observed that some of the comments had emojis and also some of the comments were non-English.
- I converted emojis into a textual format using the python library demojize.

Eg 😄 is converted to *beaming_face_with_smiling_eyes*

- Firstly, I check the language of the comments using the detect function of the *cld2 library.* And then I conditioned that if the language is not English then perform a translation of the comment.
- The comment was translated using Google Translator's translate function in python.
- The below code snippet explains how I handled emojis and converted the non-English comments to English.

```
for i in range(0,len(df),1):
  b = cld2.detect(df["comments"][i])
     langcode = b[2][0][1]
     if langcode != 'en':
        a = df.comments[i]
        b = GoogleTranslator(source='auto', target='en').translate(df.comments[i])
        b = emoji.demojize(b)
        b = b.replace(":"," ")
        b = ' '.join(b.split())
        Translatedcomments.append(b)
        print(i,len(b),"not in eng, so translated")
     else:
        b = df.comments[i]
        b = emoji.demojize(b)
        b = b.replace(":"," ")
        b = ' '.join(b.split())
        Translatedcomments.append(b)
        print(i,len(b),"english text")
```

- Now I have translated all the comments and also converted the emojis into text.
- Then, I merged all the comments into one row for each listing. E.g. If listing_id 141100 has 200 comments, there are 200 entries in review.csv. So I merged all these 200 comments into one cell. And there will be only one entry in review.csv for listing id 141100 with all the reviews in the [comments] column.
- The reason behind doing this is to get all the target variables, all the comments have to be calculated.
- At this stage, I merged the review.csv and listings.csv on the basis of listing_id.
- But, I found that not all the listing_id's from the listing.csv had data in the reviews.csv. So, I needed to handle this.
- I then wrote one code which would check if the listing_id in review.csv is there is present in listing_id of listing.csv. If not, then add that listing_id in review in review.csv with a comment as "review not available"
- Now we can merge both datasets on the basis of listing_id.

- We have the merged dataset which consists of the data of all the listings along with the reviews. The

comments column consists of all the reviews that have been listed for the particular listing.

- Now, we have to train our model, but for training, we cannot send textual data directly, we have to convert the textual data ie comments into numerical data.
- This can be achieved using either TFIDF or bag of words models.
- I have used TDIDF for this conversion as in the bag of words model, we consider a feature vector which gives the count of the number of occurrences of a word in our text. But, this approach doesn't consider the ordering of the sentence while TDIDF assigns weights for words.

TFIDF (Term Frequency- Inverse Document Frequency)

- TFIDF is a widely used protocol and often works well and gives us decent results for the textual data. It results in letting us know which words are the most interesting topics in the document.
- TF – IDF consists of two terms 1) Term Frequency 2) Inverse Document Frequency (IDF)

- Term Frequency → Term frequency tries to measure how frequently a token has occurred in the given document.
- Document here can be considered as any sequence of words such as a sentence, a paragraph, a document file etc. (in our case it is 'reviews')
- TF can be calculated in many ways. One of the ways is by using the formula

$$tf = \frac{\text{Number of times token t appears in document d}}{\sum \text{Numberof times token t appears in document d}}$$

- The Term Frequency can be calculated by considering the number of times the token appeared in the document by the number of tokens in the document.
- The Term Frequency should be larger when the token t appears a lot in the document.
- Inverse Document Frequency → Document frequency(df) is the number of documents that contain the token. The Inverse Document is defined as

$$idf = 1 + \text{Log}\frac{1 + \text{Number of documents in D}}{1 + \text{df(t)}}$$

- If idf is large then df is small. If is small means the token appears in lots of the document
- TF – IDF is defined as

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

- This means that tfidf increases with increase in tf(t,d) means when token occurs more often in document and it increases when the token appears more rarely in overall documents.
- That means that the value of tfidf is large when a token appears a lot in the document but it rarely appears in the collection of documents.

We have to perform TFIDF on the comments column of our merged data frame to convert the textual format into numerical format and assign weights to it.

Text Pre-processing

i. Before TFIDF we need to split the data into tokens using one hot encoding methodology.
ii. Perform Stemming e.g. converting writing to write because it means the same. I have used Port stemmer to achieve stemming
iii. Then perform Lemmatisation that is likes, liked, liking words means the same 'like'. So these words are to be converted to 'like'.

- I also pre-processed the text by removing anything else than [A-Za-z0-9].
- I have checked if the word is in stopwords. Stop words are the joining words also which have not much importance in the text such as the, and at etc. I have also handled these words using the words function of nltk.corpus.
- The below code summarizes this.

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
REVIEWS = []
for i in range(len(df)):
    a = df.Translatedcomments[i]
    a = re.sub('[^a-zA-Z0-9 ]', '', a)
    tk = a.split(" ")
    b = ""
    for j in tk:
        ps = PorterStemmer()
        b = b + " " + ps.stem(j).upper()
    a = b.upper()
    a = a.split()
    for word in a:
        if word not in stop_words:
            line = " ".join(a)
    REVIEWS.append(line)
```

2. Machine learning methodology

- We have 7 different target values, but we can see that the difference between the two observations is very less for each of these seven target features.

So, before making predictions I decided to scale these variables, ie taking the power of 4 and then feeding it to our models.

4.78	>>	522.049383
4.79		526.431725
4.74		504.79305
4.84		548.758735
4.63		459.540682

eg

- Now, our data is ready for training. For training, I have selected to use Decision Tree Regressor and MLP Neural Network as two of the models.
- Decision Tree Regressor is a model that formulates the data into trees/chart wherein in some computation is done on every node based on the features.
- MLP Neural Network's Sequential Model which takes input as sequence of data, have multiple layers between the input and output and finally gives the output after performing some computation at every layer.

2.1 Decision Tree Regressor

- Before feeding the data in the model, we have to use TFIDF for the comments column
- The TfidfVectorizer uses max_ features as the parameter.
- Max_features is the parameter which considers the number of most frequently occurring words in a comment. For eg we consider max_features as 500, the model will consider the top 500 words that are most frequently occurred.
- But we have to decide the best value of max_features in TfidfVectorizer().
- To find this value, I have used a cross-validation technique and selected a wide range of max_features [1,5,10,25,50,75,100,250,500,1000].
- For the 7th iteration, the model will consider top 100 words and perform fit.transform to the consider vectorizer.
- Then these words are considered as features of the model we have to concatenate these features with the original dataset.
- We have to pass the newly formed dataset for training
- The decision Tree Regressor takes max_depth as a parameter.
- Max dept means how deep the tree should be, and we have to decide this value.
- To find the best value of max_dept, I have considered the range of 2 to 20.
- To find the best max_dept, I used cross validation technique of GridSearchCV.
- GridsearchCV is a cross validation method that intakes parameters to be tuned as input in form of a list.

eg tuningparamter = {"max_depth":range(2,20,1)}

- Then we have to build a cvmodel and pass our model, the tuning and as our problem is regression we have to consider scoring as "neg_mean_absolute_error".
- Upon fit operation on the cv model, this model has a function called *best_params_.get("max_depth")* which gives us the best value of our parameter which is in the range that we mentioned in the tuning parameter list.
- cvmodel.cv_results_ gives us a dataframe where all the results of the tuning gets saved.
- This dataframe contains *mean_test_score*, which is the mean score obtained during each iteration.
- Using this *test_mean_score*, I tried to plot the errorbar plot.
- The below code snippet tells the same

  - *tp = {"max_depth":range(2,20,1)}*
  - *dtr = DecisionTreeRegressor(random_state=21)*
  - *cv = GridSearchCV(dtr,tp,scoring="neg_mean_absolute_error")*
  - *cvmodel = cv.fit(xtrain,ytrain)*
  - *dataframe = pd.DataFrame(cvmodel.cv_results_)*
  - *maxdepth = cvmodel.best_params_.get("max_depth")*

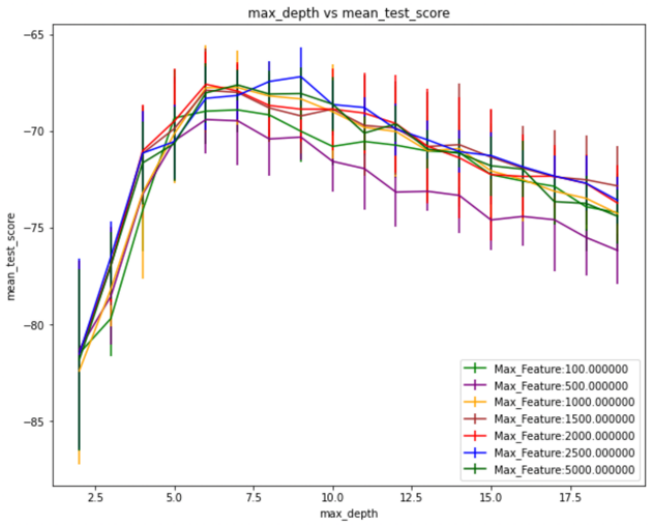


Figure 1.1 Mean_squared_error vs Max_depth(review_scores_rating)

- From the errorbar plot (Figure 1.1), we come to know that our model performs well on max_featurs = 2500 the max depth of 9. (Here the target variable is "review_scores_rating")
- So I decided to make predictions using max_features as 2500 and take max_depth as 9.
- After making prediction, I compared the mean square error with the baseline dummy regressor.

Similarly, using GridSearchCV and errorbar plots, max_features and max_depth parameters were found and these were used to make predictions for all other target variables.
The below plots depict the same.

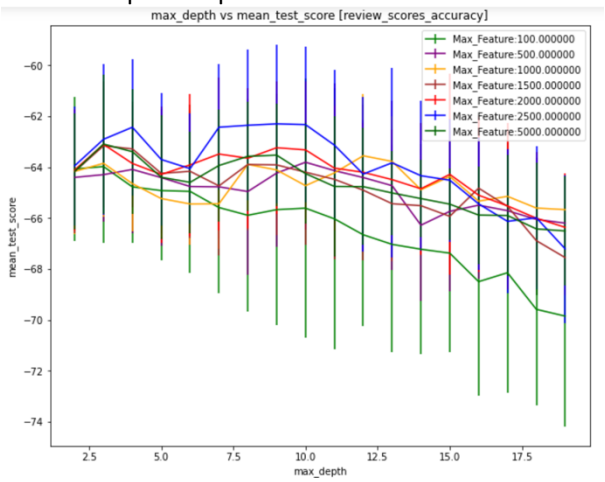
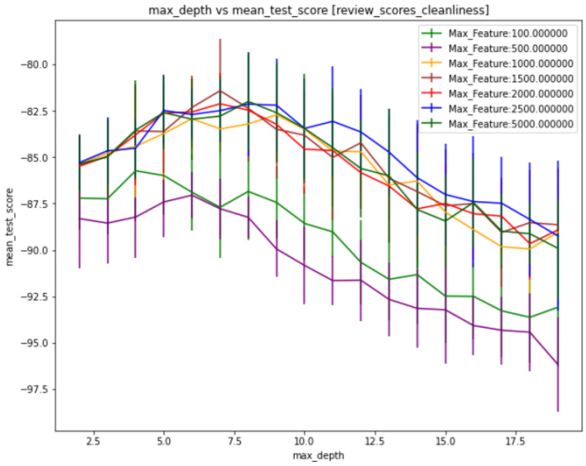


Figure 1.2 Mean_squared_error vs Max_depth (review_scores_accuracy)

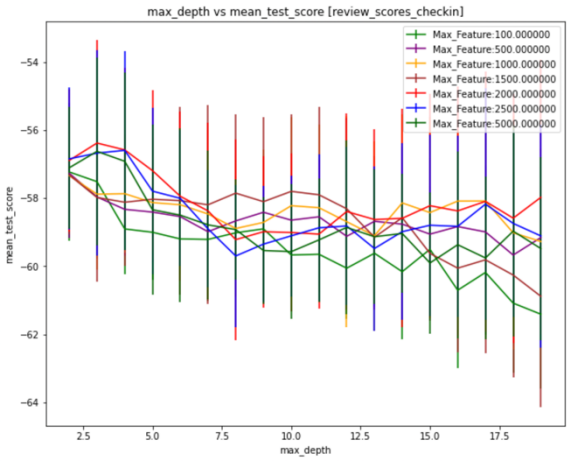
```
Model MSE is : 9964.415332
baseline is : 10736.957667
```




```
Our model is accepted
```

Figure 1.3 Mean_squared_error vs Max_depth (review_scores_cleanliness)

```
Model MSE is : 16095.694938
baseline is : 16820.044702
```




```
Our model is accepted
```

Figure 1.4 Mean_squared_error vs Max_depth (review_scores_checkin)

```
Model MSE is : 7876.665427
baseline is: 8995.261726
```

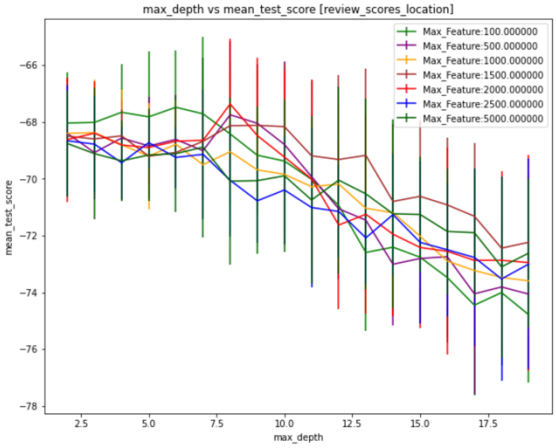


Figure 1.5 Mean_squared_error vs Max_depth (review_scores_location)

```
Model MSE is : 9816.354576
baseline is : 10479.022780
Our model is accepted
```

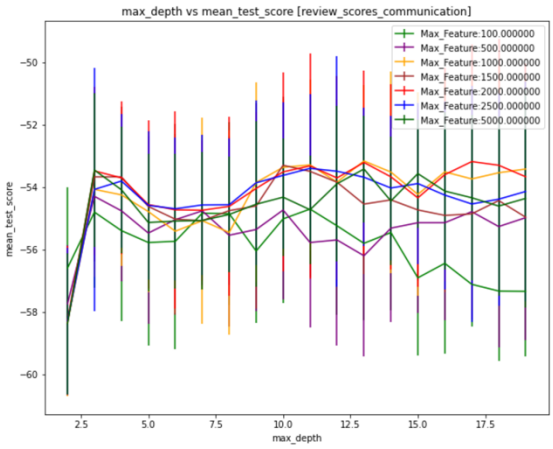


Figure 1.6 Mean_squared_error vs Max_depth (review_scores_communication)

```
Model MSE is : 868.332703
baseline is: 8280.603836
Our model is accepted
```

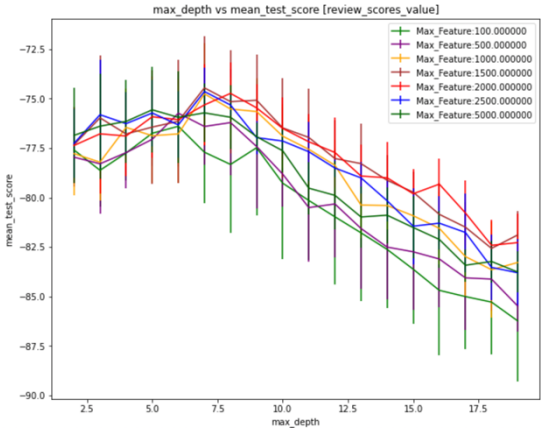


Figure 1.7 Mean_squared_error vs Max_depth (review_scores_value)

```
Model MSE is : 1640.647827
baseline is : 13632.679059
Our model is accepted
```

2.2 MLP Neural Network

- I used the dataset with max_features as 2000 and tried to train it with a Neural Network.
- The sequential Model of the Neural Network was used to train the dataset.
- While building a neural network we need to decide on various parameters such as dropout rate, momentum, hidden layer size etc.
- Dropout rate →It is a type of regularization in which we neglect some nodes in layers randomly while training the model. The value of dropout as 0.2 indicates that 20% of the neurons are neglected at random.
- Momentum→It is used to improve the speed of the training of the model by finding the global minima, which is the point where the loss in the model is the least
- Hidden Layer Size → It defines the number of neurons that should be present
- To get to know the best value of these parameters, I used keras.tuner which helps us to select these values.
- In Keras tuner we need to first define the model-building function in which the first line is building the neural network.
- We need to add the input layer and also give the input dimensions.
- Now I want to decide how many neurons to be used to get the best output so I gave a range which starts from 32 neurons to 257 neurons and it checks for every value ie a step of 32.
- Also, I wanted to select the dropout rate value, so I selected a sensible value list ie values=[0.1,0.15,0.2,0.25,0.3,0.35,0.4], so the tuner can check through all these values.
- The third parameter that I tuned is momentum. Again here we have to decide on a range of values. So I decided to select [0.01,0.02,0.03,0.04].
- Also, we need to compile the model. We have used Adam as the optimizer and evaluated the model using Mean Squared Error.
- The following code tells you how I model_builder function was defined

```
def model_builder(hp):
    model = Sequential()
model.add(Dense(64,input_dim = len(xtrain.columns)))

    DROPOUT_LAYER_RATE =
    hp.Choice("rate",values=[0.1,0.15,0.2,0.25,0.3,0.35,
    0.4])
    model.add(Dropout(rate=DROPOUT_LAYER_RATE))

    momentum_param =
    hp.Choice("momentum",values=[])
      opt =
    tf.keras.optimizers.SGD(momentum=momentum_p
    aram)
     HLS = hp.Int("units", min_value=32,
    max_value=257, step=32)
     model.add(Dense(units=HLS))
     model.add(Dense(1,activation="relu"))
      model.compile(
       return model
```

- After defining the function, we need to specify the objective. In our case it is to "val_mean_squared_error"
- Keras tuner also saves the weights of the model in a directory that we need to specify.
- Tuner.search() basically works like model.fit and using the tuner.get_best_hyperparameters we can get the best parameter value for the hyperparameters that we mentioned in the model_builder function.
- Using this information a neural network was build and predictions were made and stored in the history variable.
- I have plotted loss and the accuracy of the model with the increase of the epochs

- This process of tuning the hyperparameter, searching for the best parameter, and again building a neural network based on these parameters and Finally making the predictions, was performed on every target variable.

- The following plots were obtained

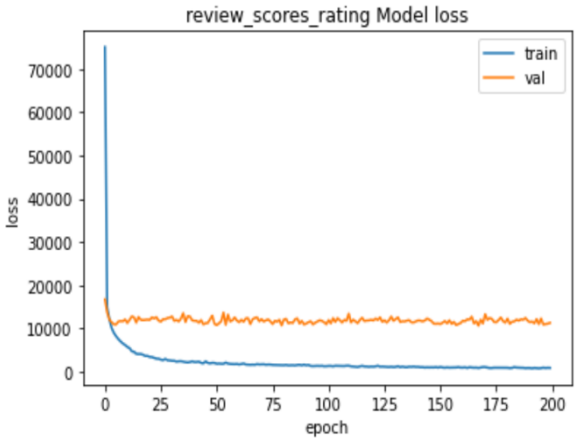
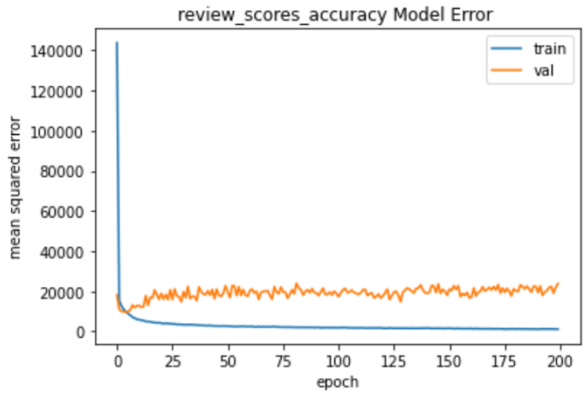


Figure 2.1 Mean squared Error and Loss vs Epoch (review_scores_rating)



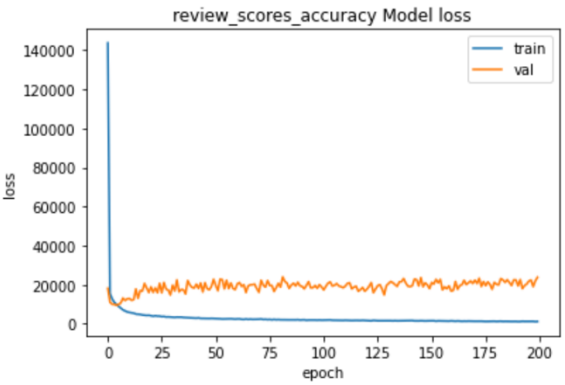


Figure 2.2 Mean squared Error and Loss vs Epoch(review_scores_accuracy)

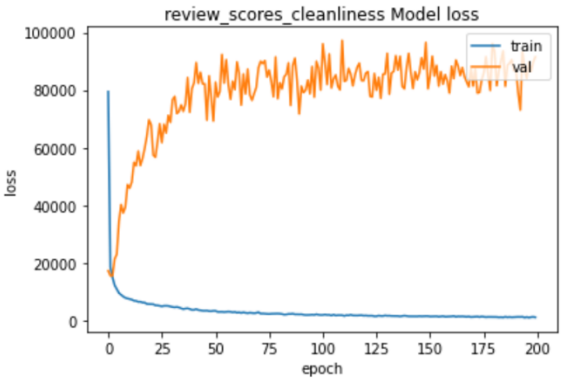


Figure 2.3 Mean squared Error and Loss vs Epoch (Review_Scores_Cleanliness)

Figure 2.4 Mean squared Error and Loss vs Epoch (Review_Scores_Checkin)

Figure 2.5 Mean squared Error and Loss vs Epoch (Review_Scores_Location)

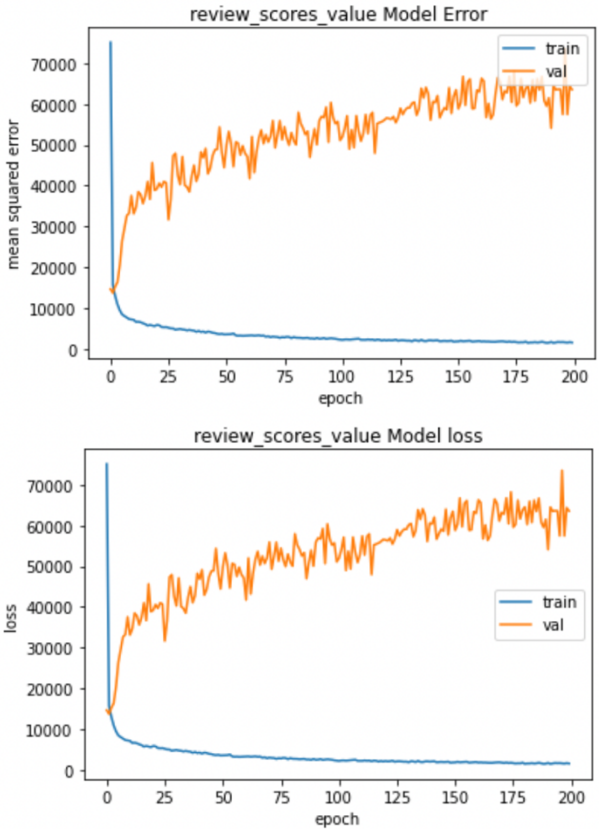
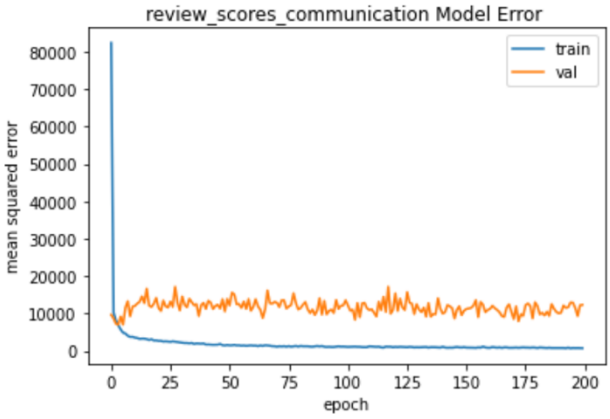


Figure 2.6 Mean squared Error and Loss vs Epoch (Review_Scores_value)



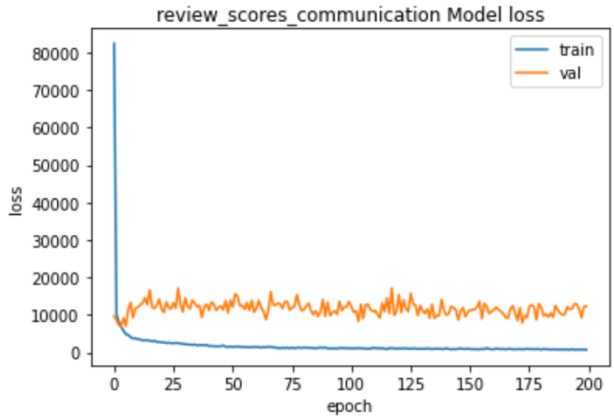


Figure 2.7 Mean squared Error and Loss vs Epoch (Review_Scores_Communication)

(iii) Summary

- I trained 2 models ie Decision Tree classifier and Multiple Layer Perceptron Neural Network for the listings and Reviews dataset that were obtained from Airbnb website
- I tuned the decision tree Regressor using a cross-validation method ie GridsearchCV, where we found the best value for max_features that are to be selected for TFIDF and also max_depth for a decision tree model that should consider.
- Also, I used the Keras tuner to tune the Neural Network, to find the best hyperparameters values such as momentum, number of hidden layers and dropout rate.
- Predictions were made for each target variable using these two models
- However, we compared the mean squared error of every model with the baseline model where dummy regressor was used to find the mean square error of the baseline model

Target Variable	Decision Tree Regressor MSE	MLP Neural Networks MSE	Baseline
Rating	12114.540	881.636	17418.529
Accuracy	9964.415	1130.824	10736.957
Cleanliness	16095.694	1209.700	16820.044
Check-in	7867.665	985.335	8995.261
Communication	7563.113	868.332	8280.603
Location	9816.354	1293.997	10479.022
Value	13212.442	1640.647	13632.679

Table 2

- Table 2 shows that our model Neural Network model gives a decent error than Decision Tree Regressor.
- The error obtained in the Decision tree Regressor Model is almost close to our baseline regressor.
- So, our neural network model was able to make good predictions.

Question 2

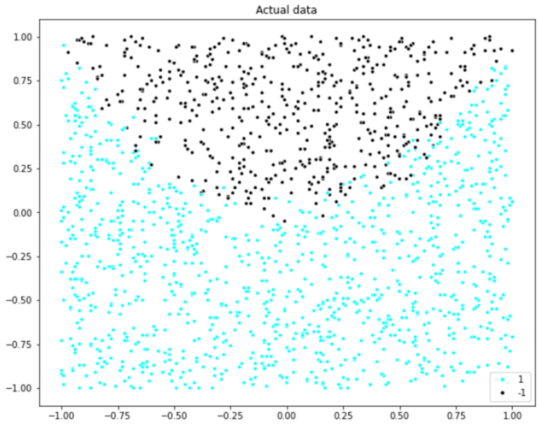
i)

- Logistic Regression – Logistic Regression is a machine Learning Algorithm that uses a linear model to predict the output of the target variable.
- The situations where logistic regression would give inaccurate predictions are:

a) <u>When the data is non-linear.</u>

- If the dataset cannot be separated by a straight line then the data is called non-linear data.

Eg.



- The above shows that the output points(-1/1) are not linearly separable they cannot be separated by passing a straight line between them.
- In such cases, if we try to use logistic regression, it will give a very inaccurate result.

b) <u>When there is a relationship between the input variables</u>

- Logistic Regression will give inaccurate results if there is a relationship between the independent variables of the model (input variables).

Emp ID	Name	Age	Years of Exp	Salary in euros(k)	Decision
846611	Vipul	23	1	55	Stayed
816233	Bob	27	5	80	Resigned

Attrition dataset.

- To predict the output variable decision, various input variables are considered shown in the above table.
- Here, Years of Experience and salary are independent variables that have a relationship we can say that, the more the number of years of experience more is the salary.
- In logistic regression, if the weight of one parameter changes, it will have an affect to the related parameter and the accuracy decreases.

ii)

- KNN(K nearest Neighbours) and Neural Networks both can be used for any type data ie Linear or Non Linear.

Advantages

- KNN is easy to use as it has only one parameter, ie 'K' the number of data points that are closest to the considered point.
  This is a hyperparameter and the best value of 'k' can be estimated using cross validation technique.
- MLP(Multiple Layer Perceptron) work better when we have a large dataset. They process whole into batches and are usually slow in training but give a better performance on huge data
- MLP classifier performs very well in high-level classification problems such as images, NLP(natural language processing), Audio, Video etc.

Disadvantage

- Both KNN and MLP take a huge amount of time to train. In KNN for a given point, we need to calculate the distance between the considered point and all the points and then sort them whereas we require a lot more time to train for Neural Network to train as it considers the complete data in a chunk such as batch size.
- MLP needs to consider many hyperparameters such as the number of hidden layers, early stopping, dropout regularization, hidden layer size, number of epochs, optimizer, batch size etc.
- MLP can be sometimes tricky, its cost function is non-convex and it is hard to interpret the real meaning of the weight associated with them.

We can say that KNN can work better for normal classification problems where there isn't any high-level computation and MLP can be used if we have tasks such as images, text, and videos.

iii)

- In real time, we have to make predictions on data that we haven't seen before or the data on which the model isn't trained.
- Generally, we split our data into 80:20 split, where 80% of the data is used to train the model and 20% of the data is used to test how the model on unseen data while training.
- To handle this, we divide the data into K-sized samples
- Use 1 sample for testing and samples for training.
- Then use 2nd sample to test and the remaining all to train the model.
- Suppose we choose 5 then, we will get a new estimate for the prediction error for every fold.
- Also, the accuracy of the model may tend to fluctuate

- But what value of 'k' should we select?
- Case 1) If we choose the value of very large ie around 90/100. Then we are going to estimate are accuracy on the small amount of datapoint. There we get fluctuation in the estimate the accuracy decreases just because of the noise and in this case the fluctuations can be very large as the test size is less.
- Case 2) If we choose the value of very small ie around 1/5. We get many data points in our test set. Then we estimate the error and because of noise in the data we can average the error in fluctuations in one fold to another
- From the above points, we want our testing data to be bigger to get high accuracy, but we also want our training data to be large so that we can get good model parameters.
- But as the K increases the time required to train the data also increases so we don't want k to be very large.
- So, there is a trade-off to choosing the perfect value of k, hence k=5 or 10 is roughly accepted to be good choice.

iv)

- Time series data is values present in a sequence with respect to a certain time interval and always has a timestamp involved in it.
- For example, consider a dataset having the closing value of the AIB Group plc stock price along with the date column.
- Here the closing value has corresponding data involved with it, such information is called to be time series data.
- Using such past data, predictions are made.
- Predictions can be the next day prediction, two-day prediction, a week later prediction or maybe the value of the stock price after a month or a year.
- Patterns can be found out in the time series dataset eg for the stock of AIB, we can find that usually closing of the stock is greater than opening as, all the transactions of the weekends are completed on Monday so bank makes a little more profit on Mondays than on other weekdays.
- But sometimes the dataset is large so we usually don't consider the full dataset, we only consider only certain previous values say 'n'.
- So, $[y_{n-1}, y_{k-n-1}, y_{k-n-2}, y_{k-n-3} ....., y_{k-1}]$ to predict our output.
- In our case, if we consider only last 5 outputs to predict our next day output, our input feature will look like.
- [123,121,122,123.5] and then predict the next number.
- Let's consider another example if we have the sales of each day for McDonald's at the O'Connell bridge, Dublin.
- We can use previous data to predict the sale of the next day, but what if we want to predict the sales after 3 days.
- In this case we take the feature vector of sales of past 'n' days then predict the next days' sale.

Days	Sales in euors
.	.
.	.
20th Dec	5.5k
21st Dec	6.5k
22nd Dec	5.6k
23rd Dec	?

Table 2.1

- So consider we have the above data, and we have to predict the sales on 23rd Dec for McDonald's at the O'Connell bridge, Dublin.
- We can take [.....,5.5,6.5,5.6] and then predict the value.
- But now if we have to predict the sales on 26th Dec, what can be done?

Days	Sales in euors
.	.
.	.
20th Dec	5.5k
21st Dec	6.5k
22nd Dec	5.6k
23rd Dec	Y1
24th Dec	Y2
25th Dec	Y3
26th Dec	?

- In this case, firstly Y1 was predicted as stated above in table 2.1 ie by considering input features as [.....,5.5,6.5,5.6].
- But for prediction of Y2, we then consider the input vector as [.....,5.5,6.5,5.6,Y1] and for Y3 it is considered as [.....,5.5,6.5,5.6,Y1,Y2], similarly sales for 26th Dec can be calculated by using [.....,5.5,6.5,5.6,Y1,Y2,Y3]
- This means that we made some predictions and that the predicted value was been used in the input feature to predict the next value.
- This type of technique is called as feedback model.

#Appendix

Listing.csv

```
import pandas as pd
import seaborn as sb
from datetime import date
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MultiLabelBinarizer

df = pd.read_csv("listings.csv",parse_dates=['host_since',
'last_scraped', 'first_review', 'last_review'])

df.corr()["review_scores_accuracy"]


cols =
["listing_url","last_scraped","scrape_id","source","name
","picture_url","host_id","host_url","host_name","host_
location","host_thumbnail_url","host_picture_url","hos
t_neighbourhood","neighbourhood_group_cleansed","b
athrooms","calendar_updated","calendar_last_scraped"
,"license","host_has_profile_pic"]

df = df.drop(labels=cols,axis=1)

df.info()

df.host_response_time.isna().sum()

df = df.drop(labels=["host_response_time"],axis=1)

df.host_response_rate.isna().sum()

df = df.drop(labels=["host_response_rate"],axis=1)

df.host_acceptance_rate.isna().sum()

df = df.drop(labels=["host_acceptance_rate"],axis=1)

df.bathrooms_text.value_counts()

df['price'].replace('[^0-9 .]', '', regex=True, inplace=True)

df.price.isna().sum()

todaysdate = date.today()
datecolumns = ['host_since', 'first_review', 'last_review']
for i in datecolumns:
   df[date] = pd.to_datetime(todaysdate) - df[i]

df.columns

df["first_review"].value_counts()

replacer = df["first_review"].mode()[0]
df["first_review"] = df["first_review"].fillna(replacer)

replacer = df["last_review"].mode()[0]
df["last_review"] = df["last_review"].fillna(replacer)

df.host_since.isna().sum()

df.first_review.isna().sum()

df.last_review.isna().sum()

true_false_column = [ 'host_is_superhost',
'host_identity_verified',
'has_availability','instant_bookable']

for i in true_false_column:
   df[i].replace({'t|T':1,'f|F':0})

analysis_col_list = ['property_type',
'neighbourhood_cleansed', "room_type"]

other_feature_dummies =
pd.get_dummies(df[analysis_col_list])

df = df.join(other_feature_dummies)

df = df.drop(labels=analysis_col_list,axis=1)

df.amenities

df.amenities = df.amenities.replace("[{[}]]", "")
df.amenities = df.amenities.replace("'", "")

count_vectorizer =  CountVectorizer(tokenizer=lambda
i: i.split(','),max_features=200)

amenities = vectorizer.fit_transform(df['amenities'])

vectorizer =  CountVectorizer(tokenizer=lambda i:
i.split(','),max_features=200)

amenities_df = vectorizer.fit_transform(df['amenities'])

amenities_df = amenities_df.toarray()

df_vectorized_amenities = pd.DataFrame(amenities_df,
columns=vectorizer.get_feature_names())

df['host_verifications'].replace(r'[^A-Za-z0-9 ]', '')

host_verification_df = df['host_verifications'].split(',')

mlb = MultiLabelBinarizer()
```

```
host_verification_df =
mlb.fit_transform(host_verification_df)

host_verification_main_df =
pd.DataFrame(host_verification_df,
columns=mlb.classes_)

df = df.join(host_verification_main_df)

df = df.join(df_vectorized_amenities)

df = df.drop(["amenities","host_verifications"],axis=1)

review_scores_columns = ['review_scores_rating',
'review_scores_accuracy',
'review_scores_cleanliness','review_scores_checkin',
'review_scores_communication',
'review_scores_location','review_scores_value']
for column in review_scores_columns:
   df[column].fillna(df[column].mean())

df['neighborhood_overview'].fillna('none', inplace=True)

df["neighborhood_overview"].isna().sum()

df.neighbourhood.value_counts()


#FeatureEngineering


df[["host_since"]] = ss.fit_transform(df[["host_since"]])

df[["host_listings_count"]] =
ss.fit_transform(df[["host_listings_count"]])

df[["host_total_listings_count"]] =
ss.fit_transform(df[["host_total_listings_count"]])

df[["accommodates"]] =
ss.fit_transform(df[["accommodates"]])

df.bathrooms_text.isna().sum()

replacer = df["bathrooms_text"].mode()[0]
df["bathrooms_text"] =
df["bathrooms_text"].fillna(replacer)

df.bathrooms_text.isna().sum()

bathrooms=[]
for i in range(len(df)):
   a = df.bathrooms_text[i]
   c = a.split()
   bathrooms.append(c[0])

for i in range(len(df)):
   if bathrooms[i] == "Shared" or bathrooms[i] == "Half-
bath":
      bathrooms[i] = 0.5
   elif bathrooms[i] == "Private":
      bathrooms[i] = 1

df["bathrooms"] = bathrooms

df = df.drop(labels=["bathrooms_text"],axis=1)

df[["bathrooms"]] = ss.fit_transform(df[["bathrooms"]])

df[["price"]] = ss.fit_transform(df[["price"]])

df[["minimum_nights"]] =
ss.fit_transform(df[["minimum_nights"]])

df[["maximum_nights"]] =
ss.fit_transform(df[["maximum_nights"]])

df[["minimum_maximum_nights"]] =
ss.fit_transform(df[["minimum_maximum_nights"]])

df[["maximum_minimum_nights"]] =
ss.fit_transform(df[["maximum_minimum_nights"]])

df[["minimum_minimum_nights"]] =
ss.fit_transform(df[["minimum_minimum_nights"]])

df[["maximum_maximum_nights"]] =
ss.fit_transform(df[["maximum_maximum_nights"]])

df[["minimum_nights_avg_ntm"]] =
ss.fit_transform(df[["minimum_nights_avg_ntm"]])

df[["maximum_nights_avg_ntm"]] =
ss.fit_transform(df[["maximum_nights_avg_ntm"]])

df.has_availability.value_counts()

df.availability_30.value_counts()

df[["availability_30"]] =
ss.fit_transform(df[["availability_30"]])

df[["availability_365"]] =
ss.fit_transform(df[["availability_365"]])

df[["availability_90"]] =
ss.fit_transform(df[["availability_90"]])

df[["availability_60"]] =
ss.fit_transform(df[["availability_60"]])
```

```
df[["number_of_reviews"]] =
ss.fit_transform(df[["number_of_reviews"]])

df[["number_of_reviews_ltm"]] =
ss.fit_transform(df[["number_of_reviews_ltm"]])

df[["number_of_reviews_l30d"]] =
ss.fit_transform(df[["number_of_reviews_l30d"]])

df[["first_review"]] =
ss.fit_transform(df[["first_review"]])

df[["last_review"]] =
ss.fit_transform(df[["last_review"]])

df[["calculated_host_listings_count"]] =
ss.fit_transform(df[["calculated_host_listings_count"]])

df[["calculated_host_listings_count_private_rooms"]] =
ss.fit_transform(df[["calculated_host_listings_count_pri
vate_rooms"]])

df.reviews_per_month.fillna(df.reviews_per_month.me
an(), inplace = True)

df[["reviews_per_month"]] =
ss.fit_transform(df[["reviews_per_month"]])

df.longitude[0],df.latitude[0]

zone = []
for i in range(len(df)):
   if (df.latitude[i]>53.346978 and df.longitude[i]>(-
6.281294)):
      zone.append("A")
   elif (df.latitude[i]<53.346978 and df.longitude[i]<(-
6.281294)):
      zone.append("C")
   elif (df.latitude[i]>53.346978 and df.longitude[i]<(-
6.281294)):
      zone.append("B")
   else:
      zone.append("D")

df["zone"] = zone

df = df.join(pd.get_dummies(df["zone"]))

df = df.drop(["latitude","longitude"],axis=1)

df = df.drop(["zone"],axis=1)

df.rename(columns = {'A':'zoneA',
'B':'zoneB','C':'zoneC','D':'zoneD'}, inplace = True)

df[["bedrooms"]] = ss.fit_transform(df[["bedrooms"]])

df.description.isna().sum()

df.host_about.isna().sum()

df = df.drop(labels=["Unnamed:
0","host_about","neighbourhood"],axis=1)

df["description"].fillna("None",inplace=True)

description_tfidf =
vectorizer.fit_transform(df["description"])

description_tfidf = description_tfidf.toarray()

len(description_tfidf)

description_tfidf_df =
pd.DataFrame(data=description_tfidf)

description_tfidf_df.columns = ['description_tfidf' +
str(x) for x in range(500)]

df = df.join(description_tfidf_df)

#neighborhood_overview

df.neighborhood_overview.isna().sum()

neigh_over_tfidf =
vectorizer.fit_transform(df["neighborhood_overview"])

neigh_over_tfidf = neigh_over_tfidf.toarray()

len(neigh_over_tfidf)

neigh_over_tfidf_df =
pd.DataFrame(data=neigh_over_tfidf)

neigh_over_tfidf_df .columns = ['neighover_tfidf' +
str(x) for x in range(500)]

df = df.join(neigh_over_tfidf_df)

df =
df.drop(labels=["description","neighborhood_overview"
],axis=1)

for col in df.columns:
   print(col,df[col].isna().sum())


def replacer(A):
   import pandas as pd
```

```
    Q = pd.DataFrame(A.isna().sum(),columns=["CT"])
    missingcolumns = list(Q[Q.CT > 0].index)
    for i in missingcolumns:
        if(A[i].dtypes=="object"):
            replacer = A[i].mode()[0]
            A[i] = A[i].fillna(replacer)
        else:
            replacer = A[i].mean()
            A[i] = A[i].fillna(replacer)

replacer(df)

for col in df.columns:
    print(col,df[col].isna().sum())
```

2. Reviews

```
import pandas as pd
import pycld2 as cld2
from deep_translator import GoogleTranslator

df = pd.read_csv("reviews.csv")
df.comments.fillna("None")
df.comments.isna().sum()

df["comments"].fillna("None",inplace=True)
df.comments.isna().sum()

for i in range(0,len(df),1):
    if len(df["comments"][i]) > 5000:
        df = df.drop(index=i)

df.index = range(0,len(df),1)

for i in range(29310,len(df),1):
    if len(df["comments"][i])<4:
        Translatedcomments.append(df["comments"][i])
    else:
        b = cld2.detect(df["comments"][i])
        langcode = b[2][0][1]
        if langcode != 'en':
            a = df.comments[i]
            b = GoogleTranslator(source='auto',
target='en').translate(df.comments[i])
            b = emoji.demojize(b)
            b = b.replace(":"," ")
            b = ' '.join(b.split())
            Translatedcomments.append(b)
            print(i,len(b),"not in eng, so translated")
        else:
            b = df.comments[i]
            b = emoji.demojize(b)
            b = b.replace(":"," ")
            b = ' '.join(b.split())
            Translatedcomments.append(b)
            print(i,len(b),"english text")

df["Translatedcomments"] = Translatedcomments

df.to_csv("TranslatedCSV123.csv")
```

3. Combining both listing.csv and review.csv

```
import pandas as pd
from re import sub

df = pd.read_csv("MainReviews.csv")
list_listing_id = df.listing_id.unique()

A = []
for listingid in list_listing_id:
    tempdf = df[df.listing_id == listingid]
    tempdf.index = range(len(tempdf))
    Q=[]
    for i in range(len(tempdf)):
        r = sub("[^0-9a-zA-Z
]","",tempdf.Translatedcomments[i])
        Q.append(r)
    A.append(Q)

list_listing_id = list_listing_id.tolist()
CombinedReviews =
pd.DataFrame(list_listing_id,columns=["list_listing_id"])
CombinedReviews["Translatedcomments"] = A

ReviewsID = list_listing_id

df = pd.read_csv("STListing1.csv")

listing_id = df.id.unique()

listing_id

listing_id = listing_id.tolist()

for id in listing_id:
    if id not in ReviewsID:
        new_row = pd.Series(data={'list_listing_id':id,
'Translatedcomments':'Reviews not present'})
        CombinedReviews =
CombinedReviews.append(new_row,ignore_index=True
)

CombinedReviews

CombinedListing = pd.read_csv("STListing1.csv")

CombinedReviews.rename(columns =
{'list_listing_id':'id'}, inplace = True)
```

```
Maindf = pd.merge(CombinedListing,
CombinedReviews, on=['id'])

for col in Maindf.columns:
   print(col,Maindf[col].isna().sum())

Maindf.to_csv("Maindf29122022.csv")
```

4. Decision Tree Model

```
import pandas as pd
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
import re
from sklearn.feature_extraction.text import
TfidfVectorizer
from nltk.stem import PorterStemmer

df = pd.read_csv("Maindf29122022.csv")

df = df.drop(columns=["Unnamed: 0","Unnamed:
0.1","id"],axis=1)

REVIEWS = []
for i in range(len(df)):
   a = df.Translatedcomments[i]
   a = re.sub('[^a-zA-Z0-9 ]', '', a)
   tk = a.split(" ")
   b = ""
   for j in tk:
      ps = PorterStemmer()
      b = b + " " + ps.stem(j).upper()
   a = b.upper()
   a = a.split()
   for word in a:
      if word not in stop_words:
         line = " ".join(a)
   REVIEWS.append(line)

df["REVIEWS"] = REVIEWS

df = df.drop(columns=["Translatedcomments"],axis=1)

Model Training

from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

for i in df.columns:
   if df[i].isna().sum()>0:
      print(i,"True")

print(pd.isnull(df).any().any())

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

#Review Scores Rating

Y['review_scores_rating_Scaled']=np.power((Y['review_
scores_rating']),4)

Y = Y.drop(columns=["review_scores_rating"],axis=1)

mse_list = []
plt.figure(figsize=(10,8))
mf = [100,500,1000,1500,2000,2500,5000]
color = {100:"green",
        500:"purple",
       1000:"orange",
        1500:"brown",
        2000:"red",
        2500:"blue",
        5000:"darkgreen"}
for i in mf:
   vectorizer = TfidfVectorizer(max_features=i)
   reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
   reviewsTFIDF = reviewsTFIDF.toarray()
   df1 = pd.DataFrame(data=reviewsTFIDF)
   df1.columns = ['review' + str(x) for x in range(0,i,1)]
   H = df
   H = H.join(df1)
   H = H.drop(columns=["REVIEWS"],axis=1)
   X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
   Y = H[["review_scores_rating"]]

Y['review_scores_rating_Scaled']=np.power((Y['review_
scores_rating']),4)
   Y = Y.drop(columns=["review_scores_rating"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
   tp = {"max_depth":range(2,20,1)}
   dtr = DecisionTreeRegressor(random_state=21)
   cv =
GridSearchCV(dtr,tp,scoring="neg_mean_absolute_erro
r")
   cvmodel = cv.fit(xtrain,ytrain)
   dataframe = pd.DataFrame(cvmodel.cv_results_)
   maxdepth =
cvmodel.best_params_.get("max_depth")
   dtr =
DecisionTreeRegressor(criterion='mae',random_state=2
1,max_depth=maxdepth)
   model = dtr.fit(xtrain,ytrain)
```

```
    pred = model.predict(xtest)
    print("MSE for {} is
{}".format(i,mean_squared_error(pred,ytest)))
    mse_list.append(mean_squared_error(pred,ytest))

plt.errorbar(dataframe["param_max_depth"],dataframe
["mean_test_score"],yerr=dataframe["std_test_score"],
color=color[i],label="Max_Feature:%f"%i)
plt.xlabel('max_depth')
plt.ylabel('mean_test_score')
plt.title("max_depth vs mean_test_score")
plt.legend()

min(mse_list)

mf[mse_list.index(min(mse_list))]

from sklearn.dummy import DummyRegressor

vectorizer = TfidfVectorizer(max_features=2500)
reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
reviewsTFIDF = reviewsTFIDF.toarray()
df1 = pd.DataFrame(data=reviewsTFIDF)
df1.columns = ['review' + str(x) for x in range(0,2500,1)]
H = df
H = H.join(df1)
H = H.drop(columns=["REVIEWS"],axis=1)
X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
Y = H[["review_scores_rating"]]
Y['review_scores_rating_Scaled']=np.power((Y['review_
scores_rating']),4)
Y = Y.drop(columns=["review_scores_rating"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=33)
dtr =
DecisionTreeRegressor(random_state=21,max_depth=7)
model = dtr.fit(xtrain,ytrain)
pred = model.predict(xtest)
mse = mean_squared_error(ytest,pred)
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
    print("Our model is accepted")
else:
    print("Our model isnt accepted")

#Review Scores Accuracy
mse_list = []
plt.figure(figsize=(10,8))
mf = [100,500,1000,1500,2000,2500,5000]
color = {100:"green",
         500:"purple",
        1000:"orange",
         1500:"brown",
         2000:"red",
         2500:"blue",
         5000:"darkgreen"}
for i in mf:
    vectorizer = TfidfVectorizer(max_features=i)
    reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
    reviewsTFIDF = reviewsTFIDF.toarray()
    df1 = pd.DataFrame(data=reviewsTFIDF)
    df1.columns = ['review' + str(x) for x in range(0,i,1)]
    H = df
    H = H.join(df1)
    H = H.drop(columns=["REVIEWS"],axis=1)
    X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
    Y = H[["review_scores_accuracy"]]

Y['review_scores_accuracy_Scaled']=np.power((Y['revie
w_scores_accuracy']),4)
    Y =
Y.drop(columns=["review_scores_accuracy"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
    tp = {"max_depth":range(2,20,1)}
    dtr = DecisionTreeRegressor(random_state=21)
    cv =
GridSearchCV(dtr,tp,scoring="neg_mean_absolute_erro
r")
    cvmodel = cv.fit(xtrain,ytrain)
    dataframe = pd.DataFrame(cvmodel.cv_results_)
    maxdepth =
cvmodel.best_params_.get("max_depth")
    dtr =
DecisionTreeRegressor(criterion='mae',random_state=2
1,max_depth=maxdepth)
    model = dtr.fit(xtrain,ytrain)
    pred = model.predict(xtest)
    print("MSE for {} is
{}".format(i,mean_squared_error(pred,ytest)))
    mse_list.append(mean_squared_error(pred,ytest))

plt.errorbar(dataframe["param_max_depth"],dataframe
["mean_test_score"],yerr=dataframe["std_test_score"],
color=color[i],label="Max_Feature:%f"%i)
```

```
plt.xlabel('max_depth')
plt.ylabel('mean_test_score')
plt.title("max_depth vs mean_test_score
[review_scores_accuracy]")
plt.legend()
mf[mse_list.index(min(mse_list))]

vectorizer = TfidfVectorizer(max_features=2500)
reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
reviewsTFIDF = reviewsTFIDF.toarray()
df1 = pd.DataFrame(data=reviewsTFIDF)
df1.columns = ['review' + str(x) for x in range(0,2500,1)]
H = df
H = H.join(df1)
H = H.drop(columns=["REVIEWS"],axis=1)
X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
Y = H[["review_scores_accuracy"]]
Y['review_scores_accuracy_Scaled']=np.power((Y['revie
w_scores_accuracy']),4)
Y = Y.drop(columns=["review_scores_accuracy"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=33)
dtr =
DecisionTreeRegressor(random_state=21,max_depth=6)
model = dtr.fit(xtrain,ytrain)
pred = model.predict(xtest)
mse = mean_squared_error(ytest,pred)
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")


#Review Scores Cleanliness

 mse_list = []
plt.figure(figsize=(10,8))
mf = [100,500,1000,1500,2000,2500,5000]
color = {100:"green",
        500:"purple",
       1000:"orange",
        1500:"brown",
        2000:"red",
        2500:"blue",
        5000:"darkgreen"}
for i in mf:
   vectorizer = TfidfVectorizer(max_features=i)
   reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
   reviewsTFIDF = reviewsTFIDF.toarray()
   df1 = pd.DataFrame(data=reviewsTFIDF)
   df1.columns = ['review' + str(x) for x in range(0,i,1)]
   H = df
   H = H.join(df1)
   H = H.drop(columns=["REVIEWS"],axis=1)
   X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
   Y = H[["review_scores_cleanliness"]]

Y['review_scores_cleanliness_Scaled']=np.power((Y['revi
ew_scores_cleanliness']),4)
   Y =
Y.drop(columns=["review_scores_cleanliness"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
   tp = {"max_depth":range(2,20,1)}
   dtr = DecisionTreeRegressor(random_state=21)
   cv =
GridSearchCV(dtr,tp,scoring="neg_mean_absolute_erro
r")
   cvmodel = cv.fit(xtrain,ytrain)
   dataframe = pd.DataFrame(cvmodel.cv_results_)
   maxdepth =
cvmodel.best_params_.get("max_depth")
   dtr =
DecisionTreeRegressor(criterion='mae',random_state=2
1,max_depth=maxdepth)
   model = dtr.fit(xtrain,ytrain)
   pred = model.predict(xtest)
   print("MSE for {} is
{}".format(i,mean_squared_error(pred,ytest)))
   mse_list.append(mean_squared_error(pred,ytest))

plt.errorbar(dataframe["param_max_depth"],dataframe
["mean_test_score"],yerr=dataframe["std_test_score"],
color=color[i],label="Max_Feature:%f"%i)
plt.xlabel('max_depth')
plt.ylabel('mean_test_score')
plt.title("max_depth vs mean_test_score
[review_scores_cleanliness]")
plt.legend()


vectorizer = TfidfVectorizer(max_features=2000)
reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
reviewsTFIDF = reviewsTFIDF.toarray()
```

```
df1 = pd.DataFrame(data=reviewsTFIDF)
df1.columns = ['review' + str(x) for x in range(0,2000,1)]
H = df
H = H.join(df1)
H = H.drop(columns=["REVIEWS"],axis=1)
X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
Y = H[["review_scores_cleanliness"]]
Y['review_scores_cleanliness_Scaled']=np.power((Y['revi
ew_scores_cleanliness']),4)
Y =
Y.drop(columns=["review_scores_cleanliness"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
dtr =
DecisionTreeRegressor(random_state=21,max_depth=7)
model = dtr.fit(xtrain,ytrain)
pred = model.predict(xtest)
mse = mean_squared_error(ytest,pred)
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")

#Review Scores Checkin

mse_list = []
plt.figure(figsize=(10,8))
mf = [100,500,1000,1500,2000,2500,5000]
color = {100:"green",
        500:"purple",
       1000:"orange",
        1500:"brown",
        2000:"red",
        2500:"blue",
        5000:"darkgreen"}
for i in mf:
   vectorizer = TfidfVectorizer(max_features=i)
   reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
   reviewsTFIDF = reviewsTFIDF.toarray()
   df1 = pd.DataFrame(data=reviewsTFIDF)
   df1.columns = ['review' + str(x) for x in range(0,i,1)]
   H = df
   H = H.join(df1)
   H = H.drop(columns=["REVIEWS"],axis=1)
   X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
   Y = H[["review_scores_checkin"]]

Y['review_scores_checkin_Scaled']=np.power((Y['review
_scores_checkin']),4)
   Y =
Y.drop(columns=["review_scores_checkin"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
   tp = {"max_depth":range(2,20,1)}
   dtr = DecisionTreeRegressor(random_state=21)
   cv =
GridSearchCV(dtr,tp,scoring="neg_mean_absolute_erro
r")
   cvmodel = cv.fit(xtrain,ytrain)
   dataframe = pd.DataFrame(cvmodel.cv_results_)
   maxdepth =
cvmodel.best_params_.get("max_depth")
   dtr =
DecisionTreeRegressor(criterion='mae',random_state=2
1,max_depth=maxdepth)
   model = dtr.fit(xtrain,ytrain)
   pred = model.predict(xtest)
   print("MSE for {} is
{}".format(i,mean_squared_error(pred,ytest)))
   mse_list.append(mean_squared_error(pred,ytest))

plt.errorbar(dataframe["param_max_depth"],dataframe
["mean_test_score"],yerr=dataframe["std_test_score"],
color=color[i],label="Max_Feature:%f"%i)
plt.xlabel('max_depth')
plt.ylabel('mean_test_score')
plt.title("max_depth vs mean_test_score
[review_scores_checkin]")
plt.legend()

vectorizer = TfidfVectorizer(max_features=2000)
reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
reviewsTFIDF = reviewsTFIDF.toarray()
df1 = pd.DataFrame(data=reviewsTFIDF)
df1.columns = ['review' + str(x) for x in range(0,2000,1)]
H = df
H = H.join(df1)
H = H.drop(columns=["REVIEWS"],axis=1)
X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
Y = H[["review_scores_checkin"]]
```

```
Y['review_scores_checkin_Scaled']=np.power((Y['review
_scores_checkin']),4)
Y = Y.drop(columns=["review_scores_checkin"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
dtr =
DecisionTreeRegressor(random_state=21,max_depth=2)
model = dtr.fit(xtrain,ytrain)
pred = model.predict(xtest)
mse = mean_squared_error(ytest,pred)
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")

#Review Scores Communication

mse_list = []
plt.figure(figsize=(10,8))
mf = [100,500,1000,1500,2000,2500,5000]
color = {100:"green",
       500:"purple",
      1000:"orange",
       1500:"brown",
       2000:"red",
       2500:"blue",
       5000:"darkgreen"}
for i in mf:
   vectorizer = TfidfVectorizer(max_features=i)
   reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
   reviewsTFIDF = reviewsTFIDF.toarray()
   df1 = pd.DataFrame(data=reviewsTFIDF)
   df1.columns = ['review' + str(x) for x in range(0,i,1)]
   H = df
   H = H.join(df1)
   H = H.drop(columns=["REVIEWS"],axis=1)
   X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
   Y = H[["review_scores_communication"]]

Y['review_scores_communication_Scaled']=np.power((Y
['review_scores_communication']),4)
   Y =
Y.drop(columns=["review_scores_communication"],axis
=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
   tp = {"max_depth":range(2,20,1)}
   dtr = DecisionTreeRegressor(random_state=21)
   cv =
GridSearchCV(dtr,tp,scoring="neg_mean_absolute_erro
r")
   cvmodel = cv.fit(xtrain,ytrain)
   dataframe = pd.DataFrame(cvmodel.cv_results_)
   maxdepth =
cvmodel.best_params_.get("max_depth")
   dtr =
DecisionTreeRegressor(criterion='mae',random_state=2
1,max_depth=maxdepth)
   model = dtr.fit(xtrain,ytrain)
   pred = model.predict(xtest)
   print("MSE for {} is
{}".format(i,mean_squared_error(pred,ytest)))
   mse_list.append(mean_squared_error(pred,ytest))

plt.errorbar(dataframe["param_max_depth"],dataframe
["mean_test_score"],yerr=dataframe["std_test_score"],
color=color[i],label="Max_Feature:%f"%i)
plt.xlabel('max_depth')
plt.ylabel('mean_test_score')
plt.title("max_depth vs mean_test_score
[review_scores_communication]")
plt.legend()

vectorizer = TfidfVectorizer(max_features=2000)
reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
reviewsTFIDF = reviewsTFIDF.toarray()
df1 = pd.DataFrame(data=reviewsTFIDF)
df1.columns = ['review' + str(x) for x in range(0,2000,1)]
H = df
H = H.join(df1)
H = H.drop(columns=["REVIEWS"],axis=1)
X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
Y = H[["review_scores_communication"]]
Y['review_scores_communication_Scaled']=np.power((Y
['review_scores_communication']),4)
Y =
Y.drop(columns=["review_scores_communication"],axis
=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
dtr =
DecisionTreeRegressor(random_state=21,max_depth=1
3)
```

```
model = dtr.fit(xtrain,ytrain)
pred = model.predict(xtest)
mse = mean_squared_error(ytest,pred)
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")

# Review scores Location

mse_list = []
plt.figure(figsize=(10,8))
mf = [100,500,1000,1500,2000,2500,5000]
color = {100:"green",
        500:"purple",
       1000:"orange",
        1500:"brown",
        2000:"red",
        2500:"blue",
        5000:"darkgreen"}
for i in mf:
   vectorizer = TfidfVectorizer(max_features=i)
   reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
   reviewsTFIDF = reviewsTFIDF.toarray()
   df1 = pd.DataFrame(data=reviewsTFIDF)
   df1.columns = ['review' + str(x) for x in range(0,i,1)]
   H = df
   H = H.join(df1)
   H = H.drop(columns=["REVIEWS"],axis=1)
   X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
   Y = H[["review_scores_location"]]

Y['review_scores_location_Scaled']=np.power((Y['revie
w_scores_location']),4)
   Y =
Y.drop(columns=["review_scores_location"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
   tp = {"max_depth":range(2,20,1)}
   dtr = DecisionTreeRegressor(random_state=21)
   cv =
GridSearchCV(dtr,tp,scoring="neg_mean_absolute_erro
r")
   cvmodel = cv.fit(xtrain,ytrain)
   dataframe = pd.DataFrame(cvmodel.cv_results_)
   maxdepth =
cvmodel.best_params_.get("max_depth")
   dtr =
DecisionTreeRegressor(criterion='mae',random_state=2
1,max_depth=maxdepth)
   model = dtr.fit(xtrain,ytrain)
   pred = model.predict(xtest)
   print("MSE for {} is
{}".format(i,mean_squared_error(pred,ytest)))
   mse_list.append(mean_squared_error(pred,ytest))

plt.errorbar(dataframe["param_max_depth"],dataframe
["mean_test_score"],yerr=dataframe["std_test_score"],
color=color[i],label="Max_Feature:%f"%i)
plt.xlabel('max_depth')
plt.ylabel('mean_test_score')
plt.title("max_depth vs mean_test_score
[review_scores_location]")
plt.legend()

vectorizer = TfidfVectorizer(max_features=1000)
reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
reviewsTFIDF = reviewsTFIDF.toarray()
df1 = pd.DataFrame(data=reviewsTFIDF)
df1.columns = ['review' + str(x) for x in range(0,1000,1)]
H = df
H = H.join(df1)
H = H.drop(columns=["REVIEWS"],axis=1)
X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
Y = H[["review_scores_location"]]
Y['review_scores_location_Scaled']=np.power((Y['revie
w_scores_location']),4)
Y = Y.drop(columns=["review_scores_location"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
dtr =
DecisionTreeRegressor(random_state=21,max_depth=4)
model = dtr.fit(xtrain,ytrain)
pred = model.predict(xtest)
mse = mean_squared_error(ytest,pred)
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
```

```
    print("Our model isnt accepted")

#Review score Value
mse_list = []
plt.figure(figsize=(10,8))
mf = [100,500,1000,1500,2000,2500,5000]
color = {100:"green",
         500:"purple",
        1000:"orange",
         1500:"brown",
         2000:"red",
         2500:"blue",
         5000:"darkgreen"}
for i in mf:
    vectorizer = TfidfVectorizer(max_features=i)
    reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
    reviewsTFIDF = reviewsTFIDF.toarray()
    df1 = pd.DataFrame(data=reviewsTFIDF)
    df1.columns = ['review' + str(x) for x in range(0,i,1)]
    H = df
    H = H.join(df1)
    H = H.drop(columns=["REVIEWS"],axis=1)
    X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
    Y = H[["review_scores_value"]]

Y['review_scores_value_Scaled']=np.power((Y['review_s
cores_value']),4)
    Y = Y.drop(columns=["review_scores_value"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
    tp = {"max_depth":range(2,20,1)}
    dtr = DecisionTreeRegressor(random_state=21)
    cv =
GridSearchCV(dtr,tp,scoring="neg_mean_absolute_erro
r")
    cvmodel = cv.fit(xtrain,ytrain)
    dataframe = pd.DataFrame(cvmodel.cv_results_)
    maxdepth =
cvmodel.best_params_.get("max_depth")
    dtr =
DecisionTreeRegressor(criterion='mae',random_state=2
1,max_depth=maxdepth)
    model = dtr.fit(xtrain,ytrain)
    pred = model.predict(xtest)
    print("MSE for {} is
{}".format(i,mean_squared_error(pred,ytest)))
    mse_list.append(mean_squared_error(pred,ytest))

plt.errorbar(dataframe["param_max_depth"],dataframe
["mean_test_score"],yerr=dataframe["std_test_score"],
color=color[i],label="Max_Feature:%f"%i)
plt.xlabel('max_depth')
plt.ylabel('mean_test_score')
plt.title("max_depth vs mean_test_score
[review_scores_value]")
plt.legend()

mf[mse_list.index(min(mse_list))]

vectorizer = TfidfVectorizer(max_features=500)
reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
reviewsTFIDF = reviewsTFIDF.toarray()
df1 = pd.DataFrame(data=reviewsTFIDF)
df1.columns = ['review' + str(x) for x in range(0,500,1)]
H = df
H = H.join(df1)
H = H.drop(columns=["REVIEWS"],axis=1)
X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
Y = H[["review_scores_value"]]
Y['review_scores_value_Scaled']=np.power((Y['review_s
cores_value']),4)
Y = Y.drop(columns=["review_scores_value"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=21)
dtr =
DecisionTreeRegressor(random_state=21,max_depth=6)
model = dtr.fit(xtrain,ytrain)
pred = model.predict(xtest)
mse = mean_squared_error(ytest,pred)
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
    print("Our model is accepted")
else:
    print("Our model isnt accepted")

# MLP Neural Network

import pandas as pd
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
import re
from sklearn.feature_extraction.text import
TfidfVectorizer
from nltk.stem import PorterStemmer
```

```
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyRegressor
from keras.models import Sequential
from keras.layers import Dense,Dropout
from keras.callbacks import EarlyStopping
from sklearn.metrics import
mean_squared_error,mean_absolute_error

df = pd.read_csv("/Users/vipulghare/Desktop/Trinity
College Dublin/Sem 1/Machine
Learning/Assignments/Final
Assignment/Codes/Preprocessing/TFIDFmainDF/Finaldf.
csv")

df.head()

df = df.drop(columns=["Unnamed: 0"],axis=1)

df.head()

vectorizer = TfidfVectorizer(max_features=2000)

df.REVIEWS
reviewsTFIDF = vectorizer.fit_transform(df.REVIEWS)
reviewsTFIDF =  reviewsTFIDF.toarray()
df1.columns = ['review' + str(i) for i in range(0,2000,1)]
H = df
H = H.join(df1)
H = H.drop(columns=["REVIEWS"],axis=1)
X =
H.drop(columns=["review_scores_rating","review_score
s_accuracy","review_scores_cleanliness","review_score
s_checkin","review_scores_communication","review_sc
ores_location","review_scores_value"])
Y = H[["review_scores_rating"]]
Y['review_scores_rating_Scaled']=np.power((Y['review_
scores_rating']),4)
Y = Y.drop(columns=["review_scores_rating"],axis=1)

#Review Score Rating

import os
os.chdir("/Users/vipulghare/Desktop/Trinity College
Dublin/Sem 1/Machine Learning/Assignments/Final
Assignment/Codes/Preprocessing/dummywork")

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=33)

import numpy as np
import tensorflow as tf
import keras_tuner as kt

def model_builder(hp):
  model = Sequential()
  model.add(Dense(64,input_dim =
len(xtrain.columns)))

  DROPOUT_LAYER_RATE =
hp.Choice("rate",values=[0.1,0.15,0.2,0.25,0.3,0.35,0.4])
  model.add(Dropout(rate=DROPOUT_LAYER_RATE))

  momentum_param =
hp.Choice("momentum",values=[0.01,0.02,0.03,0.04])
  opt =
tf.keras.optimizers.SGD(momentum=momentum_para
m)

  HLS = hp.Int("units", min_value=32, max_value=257,
step=32)
  model.add(Dense(units=HLS))

  model.add(Dense(1,activation="relu"))

  model.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=0.00
1),
    loss=[tf.keras.losses.MeanSquaredError()],
    metrics=["mean_squared_error"])

  return model

tuner = kt.Hyperband(
  model_builder,
  objective= "val_mean_squared_error",
  max_epochs=10,
  directory='keras_tuner_dir30',
  project_name='keras_tuner_demo23'
)


tuner.search(X,Y,validation_split=0.2)

tuner.get_best_hyperparameters(num_trials=1)[0].get("
units")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
rate")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
momentum")

model = Sequential()
model.add(Dropout(0.2,input_dim=len(xtrain.columns))
)
model.add(Dense(224,activation="relu"))
model.add(Dropout(0.04))
model.add(Dense(224,activation="relu"))
```

```
model.add(Dropout(0.04))
model.add(Dense(224,activation="relu"))
model.add(Dropout(0.04))
model.add(Dense(224,activation="relu"))
model.add(Dense(1))

model.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=0.00
1),
    loss=[tf.keras.losses.MeanSquaredError()],
    metrics=["mean_squared_error"])

history = model.fit(xtrain,
ytrain,epochs=200,batch_size=64,validation_split=0.2)

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('review_scores_rating Model Error ')
plt.ylabel('mean squared error')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('review_scores_rating Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()


mse=history.history["mean_squared_error"][199]
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")

#Review Score Accuracy

X =
H.drop(columns=["review_scores_rating","review_score
s_rating","review_scores_cleanliness","review_scores_c
heckin","review_scores_communication","review_score
s_location","review_scores_value"])
Y = H[["review_scores_accuracy"]]
Y['review_scores_accuracy_scaled']=np.power((Y['revie
w_scores_accuracy']),4)
xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=33)
tuner.search(X,Y,validation_split=0.2)
tuner.get_best_hyperparameters(num_trials=1)[0].get("
units")
tuner.get_best_hyperparameters(num_trials=1)[0].get("
rate")
tuner.get_best_hyperparameters(num_trials=1)[0].get("
momentum")
model = Sequential()
model.add(Dropout(0.25,input_dim=len(xtrain.columns)
))
model.add(Dense(96,activation="relu"))
model.add(Dropout(0.03))
model.add(Dense(96,activation="relu"))
model.add(Dropout(0.03))
model.add(Dense(96,activation="relu"))
model.add(Dropout(0.03))
model.add(Dense(96,activation="relu"))
model.add(Dense(1))

model.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=0.00
1),
    loss=[tf.keras.losses.MeanSquaredError()],
    metrics=["mean_squared_error"])

history = model.fit(xtrain,
ytrain,epochs=200,batch_size=64,validation_split=0.2)

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('review_scores_accuracy Model Error ')
plt.ylabel('mean squared error')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('review_scores_accuracy Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

mse=history.history["mean_squared_error"][199]
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
```

```
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")

#Review score cleanliness

X =
H.drop(columns=["review_scores_rating","review_score
s_rating","review_scores_accuracy","review_scores_che
ckin","review_scores_communication","review_scores_l
ocation","review_scores_value"])
Y = H[["review_scores_cleanliness"]]

Y['review_scores_cleanliness_scaled']=np.power((Y['revi
ew_scores_cleanliness']),4)
Y =
Y.drop(columns=["review_scores_cleanliness"],axis=1)
xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=33)

tuner = kt.Hyperband(
   model_builder,
   objective= "val_mean_squared_error",
   max_epochs=10,
   directory='keras_tuner_dir41',
   project_name='keras_tuner_demo23'
)

tuner.search(X,Y,validation_split=0.2)

tuner.get_best_hyperparameters(num_trials=1)[0].get("
units")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
rate")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
momentum")

model = Sequential()
model.add(Dropout(0.4,input_dim=len(xtrain.columns))
)
model.add(Dense(224,activation="relu"))
model.add(Dropout(0.01))
model.add(Dense(224,activation="relu"))
model.add(Dropout(0.01))
model.add(Dense(224,activation="relu"))
model.add(Dropout(0.01))
model.add(Dense(224,activation="relu"))
model.add(Dense(1))

model.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=0.00
1),
    loss=[tf.keras.losses.MeanSquaredError()],
    metrics=["mean_squared_error"])

history = model.fit(xtrain,
ytrain,epochs=200,batch_size=64,validation_split=0.2)

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('review_scores_cleanliness Model Error ')
plt.ylabel('mean squared error')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('review_scores_cleanliness Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

mse=history.history["mean_squared_error"][199]
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")

#Review Scores Checkin

X =
H.drop(columns=["review_scores_rating","review_score
s_rating","review_scores_cleanliness","review_scores_a
ccuracy","review_scores_communication","review_scor
es_location","review_scores_value"])
Y = H[["review_scores_checkin"]]
Y['review_scores_checkin_scaled']=np.power((Y['review
_scores_checkin']),4)
Y = Y.drop(columns=["review_scores_checkin"],axis=1)
xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=33)
tuner = kt.Hyperband(
```

```
    model_builder,
    objective= "val_mean_squared_error",
    max_epochs=10,
    directory='keras_tuner_dir42',
    project_name='keras_tuner_demo23'
)
tuner.search(X,Y,validation_split=0.2)
tuner.get_best_hyperparameters(num_trials=1)[0].get("
rate")
tuner.get_best_hyperparameters(num_trials=1)[0].get("
units")
tuner.get_best_hyperparameters(num_trials=1)[0].get("
momentum")
model = Sequential()
model.add(Dropout(0.3,input_dim=len(xtrain.columns))
)
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.03))
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.03))
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.03))
model.add(Dense(256,activation="relu"))
model.add(Dense(1))

model.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=0.00
1),
    loss=[tf.keras.losses.MeanSquaredError()],
    metrics=["mean_squared_error"])

history = model.fit(xtrain,
ytrain,epochs=200,batch_size=64,validation_split=0.2)

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('review_scores_checkin Model Error ')
plt.ylabel('mean squared error')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('review_scores_checkin Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

mse=history.history["mean_squared_error"][199]
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")

# Review scores communication

X =
H.drop(columns=["review_scores_rating","review_score
s_rating","review_scores_cleanliness","review_scores_c
heckin","review_scores_accuracy","review_scores_locat
ion","review_scores_value"])
Y = H[["review_scores_communication"]]

Y['review_scores_communication_scaled']=np.power((Y
['review_scores_communication']),4)

Y =
Y.drop(columns=["review_scores_communication"],axis
=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=33)

tuner = kt.Hyperband(
   model_builder,
   objective= "val_mean_squared_error",
   max_epochs=10,
   directory='keras_tuner_dir52',
   project_name='keras_tuner_demo23'
)

tuner.search(X,Y,validation_split=0.2)

tuner.get_best_hyperparameters(num_trials=1)[0].get("
units")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
rate")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
momentum")

model = Sequential()
model.add(Dropout(0.2,input_dim=len(xtrain.columns))
)
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.04))
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.04))
```

```
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.04))
model.add(Dense(256,activation="relu"))
model.add(Dense(1))

model.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=0.00
1),
    loss=[tf.keras.losses.MeanSquaredError()],
    metrics=["mean_squared_error"])

history = model.fit(xtrain,
ytrain,epochs=200,batch_size=64,validation_split=0.2)

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('review_scores_communication Model Error ')
plt.ylabel('mean squared error')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('review_scores_communication Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

mse=history.history["mean_squared_error"][199]
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")

#Review Scores Location

X =
H.drop(columns=["review_scores_rating","review_score
s_rating","review_scores_cleanliness","review_scores_c
heckin","review_scores_communication","review_score
s_accuracy","review_scores_value"])
Y = H[["review_scores_location"]]
Y['review_scores_location_scaled']=np.power((Y['review
_scores_location']),4)

Y = Y.drop(columns=["review_scores_location"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=33)

tuner = kt.Hyperband(
   model_builder,
   objective= "val_mean_squared_error",
   max_epochs=10,
   directory='keras_tuner_dir57',
   project_name='keras_tuner_demo23'
)

tuner.search(X,Y,validation_split=0.2)

tuner.get_best_hyperparameters(num_trials=1)[0].get("
units")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
rate")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
momentum")

model = Sequential()
model.add(Dropout(0.4,input_dim=len(xtrain.columns))
)
model.add(Dense(224,activation="relu"))
model.add(Dropout(0.02))
model.add(Dense(224,activation="relu"))
model.add(Dropout(0.02))
model.add(Dense(224,activation="relu"))
model.add(Dropout(0.02))
model.add(Dense(224,activation="relu"))
model.add(Dense(1))

model.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=0.00
1),
    loss=[tf.keras.losses.MeanSquaredError()],
    metrics=["mean_squared_error"])

history = model.fit(xtrain,
ytrain,epochs=200,batch_size=64,validation_split=0.2)

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('review_scores_location Model Error ')
plt.ylabel('mean squared error')
```

```
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('review_scores_location Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

mse=history.history["mean_squared_error"][199]
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")

#REeview Score Value

X =
H.drop(columns=["review_scores_rating","review_score
s_rating","review_scores_cleanliness","review_scores_c
heckin","review_scores_communication","review_score
s_accuracy","review_scores_location"])
Y = H[["review_scores_value"]]

Y['review_scores_value_scaled']=np.power((Y['review_s
cores_value']),4)

Y = Y.drop(columns=["review_scores_value"],axis=1)

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.
2,random_state=33)


tuner = kt.Hyperband(
   model_builder,
   objective= "val_mean_squared_error",
   max_epochs=10,
   directory='keras_tuner_dir62',
   project_name='keras_tuner_demo23'
)

tuner.search(X,Y,validation_split=0.2)

tuner.get_best_hyperparameters(num_trials=1)[0].get("
units")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
momentum")

tuner.get_best_hyperparameters(num_trials=1)[0].get("
rate")

model = Sequential()
model.add(Dropout(0.4,input_dim=len(xtrain.columns))
)
model.add(Dense(192,activation="relu"))
model.add(Dropout(0.04))
model.add(Dense(192,activation="relu"))
model.add(Dropout(0.04))
model.add(Dense(192,activation="relu"))
model.add(Dropout(0.04))
model.add(Dense(192,activation="relu"))
model.add(Dense(1))

model.compile(

optimizer=tf.keras.optimizers.Adam(learning_rate=0.00
1),
    loss=[tf.keras.losses.MeanSquaredError()],
    metrics=["mean_squared_error"])

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('review_scores_value Model Error ')
plt.ylabel('mean squared error')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('review_scores_value Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

mse=history.history["mean_squared_error"][199]
print("Model MSE is : %f" %mse)
dummy = DummyRegressor(strategy="mean").fit(X, Y)
ydummy = dummy.predict(X)
mse_dummy = mean_squared_error(Y,ydummy)
print("baseline is : %f"
%mean_squared_error(Y,ydummy))
if(mse_dummy>mse):
   print("Our model is accepted")
else:
   print("Our model isnt accepted")
```