**QUESTION 1:**

**Objective:** Write a short report evaluating the feasibility of predicting for a listing the individual  
ratings for accuracy, cleanliness, checkin, communication, location, and value, and also  
the overall review rating.

**Datasets: (AirBnB: Dublin, Leinster, Ireland)**

**reviews.csv:** 6 columns and 243183 rows

**listings.csv:** 75 columns and 7566 rows

**Feature Engineering:**

**reviews.csv:**

* The columns that were not related to reviews were dropped (date, reviewer\_name)
* I convert the comments to lowercase, as it makes handling the data easier.
* The comments are in different languages, I then detected the language of every comment using a Python library: langdetect. Then, the comments that were in languages other than English were dropped from the data.
* Comments were in text format. So, I needed to find the sentiment of the comments as it gives a numerical score to the comment, which will simplify the model a bit. I chose not to perform tokenizing as there are already 75 columns in the listings dataset, and tokenizing comments would increase the number of features in the reviews dataset. Thereby, increasing the number of features to be considered will in turn increase the complexity of the model.
* I used the Python library vaderSentiment to determine the sentiment of the comment.
* Then, the cleaned data was sorted by listing ID, as it is the primary key. The values of sentiment are stored using vanilla Python, so I need the data frame to be sorted to prevent the wrong values from being stored in the wrong places.
* The comments column was dropped as it was in text format and I no longer needed it because I already had its sentiment.
* In case of any unforeseen circumstances, the final review dataset was stored as a CSV for future reference.

**listings.csv:**

* The columns which were not related to reviews were dropped (listing\_url, scrape\_id, last\_scraped, source, name, picture\_url, host\_id, host\_name, host\_url, host\_thumbnail\_url, host\_picture\_url, neighbourhood\_group\_cleansed, bathrooms, license, host\_location, host\_since, first\_review, last\_review, neighbourhood, neighbourhood\_cleansed, calendar\_updated, calendar\_last\_scraped, minimum\_minimum\_nights, maximum\_minimum\_nights, minimum\_maximum\_nights, maximum\_maximum\_nights, minimum\_nights\_avg\_ntm, maximum\_nights\_avg\_ntm, neighborhood\_overview, host\_about,host\_response\_time, host\_acceptance\_rate, host\_neighbourhood)
* Due to the high number of columns, I applied K-means clustering on the location (latitude and longitude) of the listing. This helped me cluster the locations and also reduce the number of columns by 1, as both latitude and longitude were dropped and only the location column was added.
* The host\_response\_rate was in percentage, so I removed the "%" symbol and converted it to a float.
* Since there were NaN values in a few columns, I used Imputer to fill in the NaN values.
* I compared Linear Imputer and kNN Imputer. The results of the comparison are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Labels** | **Actual Mean** | **Linear Imputer Mean** | **kNN Imputer Mean** |
| bedrooms | 1.524 | 1.516 | 1.513 |
| beds | 1.833 | 1.829 | 1.831 |
| host\_response\_rate | 94.378 | 92.578 | 86.094 |
| review\_scores\_rating | 4.603 | 4.596 | 4.629 |
| review\_scores\_accuracy | 4.777 | 4.71 | 4.731 |
| review\_scores\_cleanliness | 4.646 | 4.549 | 4.607 |
| review\_scores\_checkin | 4.829 | 4.779 | 4.793 |
| review\_scores\_communication | 4.844 | 4.793 | 4.808 |
| review\_scores\_location | 4.732 | 4.693 | 4.704 |
| review\_scores\_value | 4.615 | 4.537 | 4.578 |
| reviews\_per\_month | 1.319 | 1.308 | 1.406 |

I chose the Linear Imputer as its mean and the actual mean are very close whereas the kNN Imputer has distant mean values when compared with the Linear Imputer.

* The column "bathrooms\_text" was in text format; I needed it to be a numerical value to be able to compare it while performing feature selection. For this, I mapped the possible values of the data to a particular numerical value. Then, the "bathrooms\_text" column was dropped and a "bathrooms\_map" column was added, which had the mapped numeric values.
* The "price" column had the "$" symbol, so I removed the symbol and converted it to float for ease of usage.
* Few columns (host\_is\_superhost, host\_has\_profile\_pic, host\_identity\_verified, has\_availability, instant\_bookable) had values of "t" or "f," which signified "true" or "false". Since these were in text format, I mapped "t" to 1 and "f" to 0 and stored them in the data.
* Few columns (such as "host\_verifications" and "amenities") had data stored in text format. So, I removed the special characters except for the comma (,) and then split it with the delimiter to be a comma (,). The number of verifications required and the amenities available were stored in the data as "host\_verifications\_count" and "amenities\_count", respectively.
* The column "room\_type" was in text format; I needed it to be a numerical value to be able to compare it while performing feature selection. For this, I mapped the possible values of the data to a particular numerical value. Then, the "room\_type" column values were replaced with the map values.
* The "property\_type" column was not required, so I dropped it as well.

**Feature Selection:**

I selected the best features for the respective review scores using sklearn package’s SelectKBest method. It uses f\_regression as the scoring function. f\_regression gives f\_statistics and pvalue which are used by the SelectKBest method. I set the "k" parameter to 10 as I chose the 10 best features for each review score.

bestfeatures = SelectKBest(score\_func=f\_regression, k=10)

The SelectKBest method gives the feature label and its corresponding correlation score of "K" for the best features. In my case, it gives the 10 best features for each review score.

For example:

|  |  |
| --- | --- |
| **Feature Label** | **Correlation Score** |
| Sentiment | 6773.682638 |
| host\_response\_rate | 6431.843258 |
| amenities\_count | 263.548784 |
| host\_is\_superhost | 149.157935 |
| reviews\_per\_month | 120.950215 |
| number\_of\_reviews\_l30d | 78.696163 |
| number\_of\_reviews | 74.882495 |
| host\_identity\_verified | 72.692301 |
| number\_of\_reviews\_ltm | 44.521364 |
| instant\_bookable | 42.940935 |

The above table shows the 10 best features and their corresponding correlation scores for predicting the "review\_scores\_rating" field. Similarly, the remaining features also have their own 10 best features and the corresponding correlation values.

**Machine Learning Methodology:**

**Assumptions:** The seven review rating columns (accuracy, cleanliness, checkin, communication, location, value, and overall review rating) are not available for making predictions.

In order to find the feasibility of predicting individual ratings for accuracy, cleanliness, checkin, communication, location, value, and also the overall review rating, I have attempted the following regression models: Multiple Linear Regression Model, Logistic Regression Model, Lasso Regression Model with Polynomial Features, Ridge Regression Model with Polynomial Features, and Random Forest Regression Model. As a baseline model, I have used Dummy Regressor Model. Out of the above-mentioned models, the two best-performing ones are the Multiple Linear Regression Model and Random Forest Regression Model. I will be documenting the two best-performing regression models and the baseline model. The R2 score and RMSE value are used to evaluate the models.

**Baseline Model:**

I have used the Dummy Regressor model with mean strategy from the sklearn package as my baseline model. This regressor model makes predictions based on simple rules. It forms a baseline for the other models when comparing their performance.

**MODEL 1: Multiple Linear Regression Model:**

A multiple Regression Model is used to estimate the relationship between one independent variable and multiple dependent variables.

The formula which multiple regression follows is:

Wherein,

: dependent variable

: y-intercept

: slope of coefficient 1

: independent variable 1

In my case, the dependent variable is the one we are predicting and the best features are the independent variables.

First, I get the data for each feature, then split the data into train and test sets using sklearn package’s train\_test\_split. I split the data in an 80/20 split. The model was trained using the training data and then used to predict the values on the testing data. Finally, the RMSE and R2 scores were calculated and noted.

**MODEL 2: Random Forest Regression Model:**

Random Forest Model is a form of an ensemble regression model. In a random forest regression model, n decision trees are given a row sample and a feature sample with replacement from the original train data. Then, the decision tree becomes an expert on that sample data. So, when provided with test data, each of the decision tree predictions makes a prediction, and then a majority vote is taken if the data is a binary classification problem to choose the prediction that has the most votes. If all the decision trees give a continuous value, then in the majority voting stage, the mean of all the values is taken to make the final prediction. Usually, decision trees have low bias and high variance, which means that the training error is very low and the testing error is very high. But, in a random forest, the majority voting stage converts the high variance in testing data to a low variance prediction.

The data is split in an 80/20 fashion, with 80% of the data being training data and 20% of the data being testing data. In my case, we run a loop to get predictions for all of the review scores. Inside the loop, I create the random grid and then initialize the model. I apply a cross-validation of 3 to the model. Then I supply the model with my training data, and once the training is complete, I make predictions on the testing data and calculate the respective evaluation scores.

**Evaluation:**

In the final dataset, there are 41 columns and 7566 rows. All of the columns containing text were dropped. The review column was used to find the sentiment, which turned out to be one of the most important features. The description and neighborhood\_text columns had almost the same data, so they were dropped.

**Evaluation Metrics:**

As mentioned above I have used R2 Score and the RMSE value to evaluate the models.

As observed from table1, table 2, plot 1, and plot 2 (See Annexure), we can infer the following:

As observed from table 1, table 2, plot 1, and plot 2 (see Annexure), we can infer the following:

* For review\_scores\_rating, I got the R2 score and RMSE value to be 0.69 and 0.18 for the linear regression model and 0.75 and 0.09 for the random forest regression model. Looking at these values, we can state that it is feasible to predict review\_scores\_rating using both of our models.
* For review\_scores\_accuracy, I got the R2 score and RMSE value to be 0.51 and 0.15 for the linear regression model and 0.75 and 0.09 for the random forest regression model. Looking at these values, we can state that it is feasible to predict review\_scores\_accuracy using a random forest model but that it is not feasible to predict review\_scores\_accuracy using a linear regression model.
* For review\_scores\_cleanliness, I got the R2 score and RMSE value to be 0.57 and 0.28 for the linear regression model and 0.74 and 0.09 for the random forest regression model. Based on these values, we can conclude that predicting review\_scores\_cleanliness with a random forest model is feasible, but predicting review\_scores\_cleanliness with a linear regression model is not feasible.
* For review\_scores\_checkin, I got the R2 score and RMSE value to be 0.48 and 0.09 for the linear regression model and 0.74 and 0.09 for the random forest regression model. Looking at these values, we can state that it is feasible to predict review\_scores\_checkin using a random forest model but not using a linear regression model.
* For review\_scores\_communication, I got the R2 score and RMSE value to be 0.54 and 0.09 for the linear regression model and 0.74 and 0.09 for the random forest regression model. Based on these results, we can conclude that it is possible to predict review\_scores\_communication through communication using a random forest model but not a linear regression model.
* For review\_scores\_location, I got the R2 score and RMSE value to be 0.32 and 0.09 for the linear regression model and 0.75 and 0.09 for the random forest regression model. Looking at these values, we can state that it is feasible to predict review\_scores\_location using a random forest model, but it is not feasible to predict review\_scores\_location using a linear regression model.
* For review\_scores\_value, I got the R2 score and RMSE value to be 0.58 and 0.16 for the linear regression model and 0.74 and 0.09 for the random forest regression model. Looking at these values, we can state that it is feasible to predict review\_scores\_value using a random forest model but not using a linear regression model.

**Dummy Classifier:**

The R2 score and RMSE values for the Dummy Classifier as follows:

|  |  |  |
| --- | --- | --- |
| **Label** | **R2 Score** | **RMSE Value** |
| review\_scores\_rating | -0.0002461078555304752 | 0.6081281432450636 |
| review\_scores\_accuracy | -0.0002461078555304752 | 0.6081281432450636 |
| review\_scores\_cleanliness | -0.0002461078555304752 | 0.6081281432450636 |
| review\_scores\_checkin | -0.0002461078555304752 | 0.6081281432450636 |
| review\_scores\_communication | -0.0002461078555304752 | 0.6081281432450636 |
| review\_scores\_location | -0.0002461078555304752 | 0.6081281432450636 |
| review\_scores\_value | -0.0002461078555304752 | 0.6081281432450636 |

Table 3: R2 Score and RMSE Value of Dummy Classifier

It is clearly visible that Model 1 and Model 2 perform better than Dummy Classifier.

**QUESTION 2:**

**Don’t just Google for the answers to the questions below and do not use jargon you don’t fully understand. Read the lecture notes, think about your answers, and make sure to explain them in your own words.**

**i) Give two examples of situations when logistic regression would give inaccurate  
predictions. Explain your reasoning. [5 marks]**

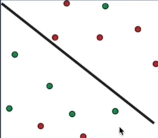
Since logistic regression estimates are based on the linear decision boundary, they will give inaccurate predictions when there is no linear correlation between the target labels and the features.

Assuming there are two circular distributions of labels, where one has a larger radius than the other, logistic regression cannot predict circular separation.

Also, when there are outliers and leverage points, the decision boundary can be impacted by them. Thus, decreasing the prediction accuracy.

When the data is not linearly separable, then logistic regression will give inaccurate results as the decision boundary will sway due to the outliers.

As we can observe from the below plot:



In the upper segment, the green dot will sway the decision boundary, and in the lower segment, the red dots will sway the decision boundary.

**ii) Discuss some advantages and disadvantages of a kNN classifier vs an MLP neural  
net classifier. Explain your reasoning. [5 marks]**

When compared to an MLP neural network classifier, kNN is fairly simple and only requires tuning of one hyperparameter (k), whereas neural networks require training of multiple hyperparameters.

For a kNN classifier, the k value should be wisely selected.

An MLP neural network classifier requires a large number of training data to achieve adequate accuracy when compared to a kNN classifier.

kNN takes less time to train the data when compared to an MLP neural network classifier. If you have a large number of data points, then the evaluation time of the kNN model is much larger than that of the MLP neural network model.

One does not need to train an MLP neural network classifier again and again before making predictions, whereas a kNN classifier needs to be trained before making predictions.

Once an MLP neural network classifier is trained on one task, its parameters can be used as a good initializer for another similar task. This is known as "transfer learning." Transfer learning cannot be achieved with kNN.

**iii) In k-fold cross-validation a dataset is resampled multiple times. What is the  
the idea behind this resampling i.e. why does resampling allow us to evaluate the  
generalization performance of a machine learning model. Why are k = 5 or  
k = 10 often suggested as good choices? [10 marks]**

In k-fold cross-validation, we divide the data into k equal parts and use one part as test data and the rest as training data. We repeat the process until all k parts are considered test data. We calculate the theta for each case. Then we obtain k estimates of J (theta) and use them to calculate the average and spread values.

The common choices of k are 5 or 10, as when k=5, it corresponds to an 80/20 split as 1 out of the 5 equal parts is considered to be the test data. When k = 10, it corresponds to a 90/10 split, as 1 out of 10 equal parts is considered to be the test data.

Each of the test sets has n/k points, where n is the total number of data points and k is the number of folds. We average over these sets to calculate our prediction accuracy. Averaging smoothes out the noise in the data if n/k is large enough, which means k is small enough.

We want to maximize the data used to train the model in order to learn the representative parameter values since fluctuations do not occur due to inadequate training. So, the k value should be large enough, as we want (k-1)/n to be large.

We also need to be mindful of the fact that as the k value increases, the computation time increases as well, since we have to fit the model k times.

Therefore, k = 5 or k = 10 is a reasonable compromise value, but sometimes we do need to use other values.

**iv) Discuss how lagged output values can be used to construct features for time  
series data. Illustrate with a small example. [5 marks]**

The dataset for a time series would only include recorded values and time values. Any machine learning approach cannot exploit this. Datasets may be feature engineered in a variety of ways, such as by adding X features and a Y output variable that can be utilized in a forecasting model. One such method of developing new features is the latency feature. Here, the values that were recorded are transferred to a future time. The program will let you choose the shift size. Therefore, the recorded values from the previous time step make up the new time value. This approach is predicated on the idea that historical values and the values we are attempting to anticipate will be somewhat correlated.

Example:The weather prediction is a key factor, along with other domain-specific variables, in my ability to predict whether team A will win the upcoming football game. In order to anticipate the weather on match day, I can also utilize weather data from prior matches that may be used with lag when creating the prediction model.

**Annexure:**

|  |  |  |
| --- | --- | --- |
| **Label** | **Model 1(Linear Regression)** | **Model 2(Random Forest)** |
| review\_scores\_rating | 0.691463021 | 0.747000757 |
| review\_scores\_accuracy | 0.509081549 | 0.747973203 |
| review\_scores\_cleanliness | 0.573358987 | 0.744565645 |
| review\_scores\_checkin | 0.482384728 | 0.744421076 |
| review\_scores\_communication | 0.53665603 | 0.743058509 |
| review\_scores\_location | 0.315557185 | 0.746459267 |
| review\_scores\_value | 0.580569656 | 0.744212421 |

**Table 1: R2 Score Values for both the models**

**Plot 1: Comparison of R2 Score of both models**

|  |  |  |
| --- | --- | --- |
| **Label** | **Model 1(Linear Regression)** | **Model 2(Random Forest)** |
| review\_scores\_rating | 0.183657773 | 0.093541118 |
| review\_scores\_accuracy | 0.145286575 | 0.093181577 |
| review\_scores\_cleanliness | 0.27869242 | 0.094441449 |
| review\_scores\_checkin | 0.091446886 | 0.094494901 |
| review\_scores\_communication | 0.093473282 | 0.094998681 |
| review\_scores\_location | 0.097345105 | 0.093741322 |
| review\_scores\_value | 0.160790656 | 0.094572046 |
|  |  |  |

**Table 2: RMSE Values for both models**

**Plot 2: Comparison of RMSE Score of both models**

**APPENDIX:**

**Feature Selection:**

# install libraries:

!pip install langdetect vaderSentiment miceforest missingpy

# Imports:

import pandas as pd

from langdetect import detect

import seaborn as sns

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

import seaborn as sns; sns.set()

import csv

## FEATURE ENGINEERING:

#### REVIEW DATA:

# Imports:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.svm import LinearSVC

from sklearn.model\_selection import train\_test\_split

import seaborn as sns

from sklearn.model\_selection import cross\_val\_predict

from sklearn.linear\_model import LogisticRegression

import matplotlib.patches as mpatches

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

import csv

# Reading the reviews and storing them in a dataframe

df\_reviews = pd.read\_csv('reviews.csv')

print(df\_reviews.info())

df\_clean\_reviews = df\_reviews.drop(["date", "reviewer\_name"], axis=1)

print(df\_clean\_reviews.info())

df\_clean\_reviews['comments'] = df\_clean\_reviews['comments'].str.lower()

print(df\_clean\_reviews.info())

# Get language of the reviews

def get\_language(x):

try:

return detect(x)

except:

return 'unknown'

df\_clean\_reviews['Language'] = df\_clean\_reviews['comments'].apply(get\_language)

df\_clean\_reviews.head()

# Dropping the reviews which are not in english and then dropping the column 'Language'

df\_clean\_reviews = df\_clean\_reviews[df\_clean\_reviews["Language"] == "en"].drop(["Language"], axis=1)

df\_clean\_reviews.info()

# saving the cleaned reviews for future use

df\_clean\_reviews.to\_csv('clean\_review.csv',index = False)

# Reading the cleaned reviews from csv

df\_clean\_reviews=pd.read\_csv('clean\_review.csv')

df\_clean\_reviews.info()

# Getting the comments to use for sentiment analysis

comments=df\_clean\_reviews['comments']

comments.head()

# Get sentiment of each comment

sentiment\_score=[]

sentimentAnalyzer=SentimentIntensityAnalyzer()

count=0

for comment in comments:

sentiment\_value\_from\_analyzer=sentimentAnalyzer.polarity\_scores(comment)

count+=1

sentiment\_score.append(sentiment\_value\_from\_analyzer['compound'])

print(f'The total number of sentiments acquired: {count}')

# Storing the sentiment acquired before into the dataframe

df\_clean\_reviews['sentiment\_score']=sentiment\_score

df\_clean\_reviews.info()

# saving the cleaned reviews with Sentiment for future use

df\_clean\_reviews.to\_csv('clean\_review.csv',index = False)

# Reading the cleaned reviews with Sentiment from csv

df\_clean\_reviews=pd.read\_csv('clean\_review.csv')

df\_clean\_reviews.info()

# Sorting the cleaned data by listing id as it is the primary key

# The values of sentiment will be stored using vanilla python, so

# we need the dataframe to be sorted to prevent wrong values at wrong places.

df\_clean\_reviews=df\_clean\_reviews.sort\_values('listing\_id')

df\_clean\_reviews.info()

# Storing the sentiment in a list to later add in the final dataframe for reviews

listing\_id=df\_clean\_reviews.loc[:,'listing\_id']

sentiment=df\_clean\_reviews.loc[:,'sentiment\_score']

new\_sentiment=[]

new\_id=[]

idx=0

i=0

while(i<listing\_id.size):

count=0

sum=0

id=listing\_id[i]

new\_id.append(id)

while(i<listing\_id.size and listing\_id[i]==id):

count+=1

sum+=sentiment[i]

i+=1

print(str(id)+" count: "+str(count))

new\_sentiment.append(round((sum/count),5))

print("ID: "+str(new\_id[idx])+" SENTIMENT: "+str(new\_sentiment[idx])+" SUM: "+str(sum))

print('-----------------------------------')

idx+=1

# Storing the comments in a list to later add in the final dataframe for reviews

listing\_id=df\_clean\_reviews.loc[:,'listing\_id']

comments=df\_clean\_reviews.loc[:,'comments']

new\_comments=[]

temp\_id=[]

idx=0

i=0

while(i<listing\_id.size):

count=0

all\_comments=""

id=listing\_id[i]

temp\_id.append(id)

while(i<listing\_id.size and listing\_id[i]==id):

count+=1

all\_comments+=comments[i]

i+=1

print(str(id)+" Count: "+str(count))

new\_comments.append(all\_comments)

print("ID: "+str(temp\_id[idx])+" LENGTH: "+str(len(all\_comments)))

print('-----------------------------------')

idx+=1

# Creating the final dataframe for reviews

print(len(new\_sentiment),len(new\_id),len(new\_comments))

df\_final\_reviews=pd.DataFrame({'id':new\_id, 'comments':new\_comments, 'sentiment':new\_sentiment})

df\_final\_reviews.shape

# Storing the final dataframe for reviews to csv

df\_final\_reviews.to\_csv('reviews\_final',index=False)

df\_final\_reviews=pd.read\_csv('reviews\_final')

df\_final\_reviews.shape

#### LISTING DATA:

# Imports:

from sklearn.experimental import enable\_iterative\_imputer

from sklearn.impute import IterativeImputer

from sklearn.linear\_model import LinearRegression

from sklearn.impute import KNNImputer

df\_listing = pd.read\_csv('listings.csv')

df\_listing.info()

# Dropping useless data

df\_listing = df\_listing.drop(["listing\_url", "scrape\_id", "last\_scraped", "source", "name", "picture\_url", "host\_id", "host\_name", "host\_url",

"host\_thumbnail\_url", "host\_picture\_url", "neighbourhood\_group\_cleansed", "bathrooms", "license", "host\_location", "host\_since", "first\_review", "last\_review",'neighbourhood', 'neighbourhood\_cleansed',

'calendar\_updated', 'calendar\_last\_scraped','minimum\_minimum\_nights', 'maximum\_minimum\_nights', 'minimum\_maximum\_nights',

'maximum\_maximum\_nights', 'minimum\_nights\_avg\_ntm', 'maximum\_nights\_avg\_ntm','neighborhood\_overview', 'host\_about',

'host\_response\_time', 'host\_acceptance\_rate', 'host\_neighbourhood','description'], axis=1)

df\_listing.info()

df\_listing.columns.size

# Adding latitude and longitude columns to X

location=df\_listing.loc[:,['id','latitude','longitude']]

df\_listing=df\_listing.drop(['latitude','longitude'],axis=1)

location.head()

k\_means=KMeans(n\_clusters=7,init="k-means++")

# compute k-means clustering on X

k\_means.fit(location[location.columns[1:3]])

location['cluster\_labels']=k\_means.fit\_predict(location[location.columns[1:3]])

# Find the coordinates of the center of cluster

centers=k\_means.cluster\_centers\_

labels=k\_means.predict(location[location.columns[1:3]])

location.head(10)

location.plot.scatter(x = 'latitude', y = 'longitude', c=labels, s=50, cmap='viridis')

plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)

location=location.drop(['latitude','longitude'],axis=1)

df\_listing=df\_listing.merge(location,left\_on='id',right\_on='id')

print(df\_listing.head())

print(df\_listing.info())

# Making sure the host\_response\_rate column to have no symbols and is of type float

df\_listing['host\_response\_rate'] = df\_listing['host\_response\_rate'].str.replace('%', '')

df\_listing['host\_response\_rate'] = df\_listing['host\_response\_rate'].astype(float)

df\_temporary=df\_listing[['bedrooms','beds','host\_response\_rate','review\_scores\_rating','review\_scores\_accuracy','review\_scores\_cleanliness','review\_scores\_checkin','review\_scores\_communication','review\_scores\_location','review\_scores\_value','reviews\_per\_month']]

# To fix the mmissing values

linear\_regression = LinearRegression()

iterative\_imputer = IterativeImputer(estimator=linear\_regression,missing\_values=np.nan, max\_iter=100, verbose=2, imputation\_order='roman',random\_state=0)

imputed\_data\_iterative=iterative\_imputer.fit\_transform(df\_temporary)

imputed\_data\_iterative = pd.DataFrame(imputed\_data\_iterative)

knn\_imputer = KNNImputer(n\_neighbors=2)

imputed\_data\_knn = knn\_imputer.fit\_transform(df\_temporary)

imputed\_data\_knn = pd.DataFrame(imputed\_data\_knn)

listing\_data\_labels = ['bedrooms','beds','host\_response\_rate','review\_scores\_rating','review\_scores\_accuracy','review\_scores\_cleanliness','review\_scores\_checkin','review\_scores\_communication','review\_scores\_location','review\_scores\_value','reviews\_per\_month']

i=0

for listing\_data\_label in listing\_data\_labels:

mean\_value1=round(df\_temporary[listing\_data\_label].mean(),3)

mean\_value2=round(imputed\_data\_iterative[i].mean(),3)

mean\_value3=round(imputed\_data\_knn[i].mean(),3)

i+=1

print(mean\_value1,mean\_value2,mean\_value3)

fig, ax = plt.subplots(figsize=(10,6))

sns.distplot(imputed\_data\_iterative[7])

fig, ax = plt.subplots(figsize=(10,6))

sns.distplot(imputed\_data\_knn[7])

fig, ax = plt.subplots(figsize=(10,6))

sns.distplot(df\_temporary.review\_scores\_value)

# Store the iterative imputed data in df\_temporary

i=0

for listing\_data\_label in listing\_data\_labels:

df\_temporary[listing\_data\_label]=imputed\_data\_iterative[i]

i+=1

df\_temporary

# Storing df\_temporary in df\_listing

for listing\_data\_label in listing\_data\_labels:

df\_listing[listing\_data\_label]=df\_temporary[listing\_data\_label]

df\_listing.info()

# Converting number of bathrooms to number

bathrooms=df\_listing.loc[:,['id','bathrooms\_text']]

bathrooms\_text\_map = {'0 shared baths':1,

'0 baths': 1,

'Shared half-bath': 2,

'Half-bath':3,

'Private half-bath':4,

'1 shared bath':5,

'1 bath': 6,

'1 private bath': 7,

'1.5 baths': 9,

'1.5 shared baths': 8,

'2 shared baths':10,

'2 baths':11,

'2.5 shared baths': 12,

'2.5 baths': 13,

'3 shared baths': 14,

'3 baths': 15,

'3.5 shared baths': 16,

'3.5 baths': 17,

'4 shared baths': 18,

'4 baths': 19,

'4.5 baths': 20,

'5 baths': 21,

'5.5 baths': 22,

'6 shared baths': 23,

'6 baths': 24,

'6.5 baths': 25,

'7 baths': 26,

'7.5 baths': 27,

'8 baths': 28,

'8.5 baths': 29,

'9.5 baths': 30}

bathrooms['bathroom\_map'] = bathrooms.bathrooms\_text.map(bathrooms\_text\_map)

bathrooms.head()

df\_listing = df\_listing.drop(['bathrooms\_text'], axis=1).merge(bathrooms, left\_on='id', right\_on='id')

df\_listing['bathroom\_map'] = df\_listing['bathroom\_map'].fillna(6)

df\_listing = df\_listing.drop(['bathrooms\_text'], axis=1)

df\_listing.info()

# Storing final listings as csv

df\_listing.to\_csv('final\_listing.csv',index = False)

final\_listing = pd.read\_csv('final\_listing.csv')

final\_listing.shape

df\_final\_listings = pd.read\_csv('final\_listing.csv')

# Making sure all the required columns are in their correct format.

df\_final\_listings['price'] = df\_final\_listings['price'].str.replace('$', '')

df\_final\_listings['price'] = df\_final\_listings['price'].str.replace(',', '')

df\_final\_listings['price'] = df\_final\_listings['price'].astype(float)

df\_final\_listings['host\_is\_superhost'] = df\_final\_listings['host\_is\_superhost'].map({'t': 1, 'f': 0})

df\_final\_listings['host\_has\_profile\_pic'] = df\_final\_listings['host\_has\_profile\_pic'].map({'t': 1, 'f': 0})

df\_final\_listings['host\_identity\_verified'] = df\_final\_listings['host\_identity\_verified'].map({'t': 1, 'f': 0})

df\_final\_listings['has\_availability'] = df\_final\_listings['has\_availability'].map({'t': 1, 'f': 0})

df\_final\_listings['instant\_bookable'] = df\_final\_listings['instant\_bookable'].map({'t': 1, 'f': 0})

# Merging the df\_final\_listings and df\_final\_reviews

df\_final = pd.merge(df\_final\_listings,df\_final\_reviews,left\_on='id',right\_on='id',how='left')

print(df\_final.shape)

print(df\_final.info())

def number\_of\_items(content):

return len(content.split(','))

def punctuation\_processing(content):

return str(content).translate(str.maketrans('', '', '!"#$%&\'()\*+-./:;<=>?@[\]^\_`{|}~\\'))

df\_temp=df\_final\_listings.loc[:,['id','host\_verifications']]

df\_temp['host\_verifications'] = df\_temp['host\_verifications'].apply(lambda content: punctuation\_processing(content=content))

df\_temp['host\_verifications\_count'] = df\_temp['host\_verifications'].apply(lambda content: number\_of\_items(content=content))

df\_temp = df\_temp.drop(['host\_verifications'], axis=1)

df\_final = df\_final.merge(df\_temp, left\_on='id', right\_on='id')

df\_final = df\_final.drop(['host\_verifications'], axis=1)

df\_final.info()

df\_temp=df\_final\_listings.loc[:,['id','amenities']]

df\_temp['amenities'] = df\_temp['amenities'].apply(lambda content: punctuation\_processing(content=content))

df\_temp['amenities\_count'] = df\_temp['amenities'].apply(lambda content: number\_of\_items(content=content))

df\_temp = df\_temp.drop(['amenities'], axis=1)

df\_final = df\_final.merge(df\_temp, left\_on='id', right\_on='id',how='left')

df\_final = df\_final.drop(['amenities'], axis=1)

df\_final.info()

# df\_final=df\_final.drop(['description'],axis=1)

df\_final['room\_type'] = df\_final['room\_type'].map({'Shared room': 1, 'Hotel room': 2, 'Private room': 3, 'Entire home/apt': 4})

df\_final = df\_final.drop(['property\_type'],axis=1)

df\_final.to\_csv('final\_dataset.csv')

df\_final.info()

## FEATURE SELECTION:

df\_final=pd.read\_csv('FinalDataset.csv')

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import f\_regression

bestfeatures = SelectKBest(score\_func=f\_regression, k=10)

Y=[]

X=df\_final

y\_features=['review\_scores\_rating','review\_scores\_accuracy','review\_scores\_cleanliness','review\_scores\_checkin','review\_scores\_communication','review\_scores\_location','review\_scores\_value']

for f in y\_features:

Y.append(pd.DataFrame(df\_final[f]))

X=X.drop(f,axis=1)

topFeatures={}

for y in Y:

print(y.columns[0])

fit = bestfeatures.fit(X,y)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(X.columns)

#concat two dataframes for better visualization

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

featureScores.columns = ['Specs','Score'] #naming the dataframe columns

labels=(featureScores.nlargest(10,'Score')) #print 10 best features

topFeatures[y.columns[0]]=labels['Specs'].array

print(featureScores.nlargest(10,'Score'))

print('-----------------------------------')

topFeatures

**Models:**

import numpy as np

import pandas as pd

df\_final=pd.read\_csv('FinalDataset.csv')

df\_final.head()

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import f\_regression

#apply SelectKBest class to extract top 10 best features

bestfeatures = SelectKBest(score\_func=f\_regression, k=10)

Y=[]

X=df\_final

y\_features=['review\_scores\_rating','review\_scores\_accuracy','review\_scores\_cleanliness','review\_scores\_checkin','review\_scores\_communication','review\_scores\_location','review\_scores\_value']

# y\_features=['review\_scores\_value']

for f in y\_features:

Y.append(pd.DataFrame(df\_final[f]))

X=X.drop(f,axis=1)

topFeatures={}

for y in Y:

# X=X.drop('review\_scores\_location',axis=1)

# y = pd.DataFrame(final\_df['review\_scores\_location'])

# X = final\_df.iloc[:,0:19] #independent columns

# y = final\_df.iloc[:,20]

fit = bestfeatures.fit(X,y)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(X.columns)

#concat two dataframes for better visualization

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

featureScores.columns = ['Specs','Score'] #naming the dataframe columns

labels=(featureScores.nlargest(10,'Score')) #print 10 best features

topFeatures[y.columns[0]]=labels['Specs'].array

print(featureScores.nlargest(10,'Score'))

## MODELS:

- Linear Regression

- Logistic Regression

- SVM

- Baseline

- kNN

- Decision Trees

- Neural nets

- ConvNets

#### Multiple Linear Regression:

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import accuracy\_score

from sklearn.metrics import r2\_score

from numpy import mean

from numpy import absolute

from numpy import sqrt

import pandas as pd

# for y in Y:

# features=topFeatures[y.columns[0]]

# fX=X.copy()

# fX=fX[[c for c in X.columns if c in features]]

# # cv = KFold(n\_splits=10, random\_state=1, shuffle=True)

# kf = KFold(n\_splits=5)

# print(y.columns[0])

# kf=KFold(n\_splits=5)

# r2\_mean=[]

# x\_train,x\_test,y\_train,y\_test = train\_test\_split(fX,y,test\_size=0.2)

# for train,test in kf.split(fX):

# score=[]

# model = LinearRegression().fit(pd.DataFrame(fX).iloc[train],pd.DataFrame(y).iloc[train])

# ypred = model.predict(pd.DataFrame(fX).iloc[test])

# score.append(r2\_score(pd.DataFrame(y).iloc[test],ypred))

# print(f'TEST R2: {mean(score)}')

# # print(f'Accuracy: {accuracy\_score(y\_test.astype(int),ypred.astype(int))}')

# # print(f'RMSE: {mean\_squared\_error(y\_test,ypred)}')

# # print(f'R2: {r2\_score(y\_test,ypred)}')

# print('-----------------------')

for y in Y:

features=topFeatures[y.columns[0]]

fX=X.copy()

fX=fX[[c for c in X.columns if c in features]]

# cv = KFold(n\_splits=10, random\_state=1, shuffle=True)

# kf = KFold(n\_splits=5)

print(y.columns[0])

x\_train,x\_test,y\_train,y\_test = train\_test\_split(fX,y,test\_size=0.2)

model = LinearRegression().fit(x\_train, y\_train)

ypred = model.predict(x\_test)

print(f'RMSE: {mean\_squared\_error(y\_test,ypred)}')

print(f'R2 Score: {r2\_score(y\_test,ypred)}')

print('-----------------------')

#### Logistic Regression: NOT ACCOUNTED FOR

# ['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga']

import numpy as np

from sklearn.linear\_model import LogisticRegression

from sklearn import preprocessing

for y in Y:

# print(y.shape)

features=topFeatures[y.columns[0]]

fX=X.copy()

fX=fX[[c for c in X.columns if c in features]]

# cv = KFold(n\_splits=10, random\_state=1, shuffle=True)

print(y.columns[0])

model = LogisticRegression(penalty='none',solver='sag')

lab = preprocessing.LabelEncoder()

y\_transformed\_train = lab.fit\_transform(y\_train)

y\_transformed\_test = lab.fit\_transform(y\_test)

model.fit(x\_train, y\_transformed\_train.ravel())

ypred\_LR=model.predict(x\_test)

print('mean %f'%(mean\_squared\_error(y\_transformed\_test,ypred\_LR)))

print('-----------------------')

#### Lasso Regression:

#Imports:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.preprocessing import PolynomialFeatures

from sklearn import model\_selection

from sklearn.linear\_model import Ridge

from sklearn.linear\_model import Lasso

import matplotlib as mtplt

from sklearn.model\_selection import KFold

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from math import sqrt

from sklearn.metrics import mean\_absolute\_error

def kfcv(x,y,c\_range,name\_of\_model,range\_of\_poly):

for poly\_degree in range\_of\_poly:

mean\_error=[];std\_error=[];

x\_poly=PolynomialFeatures(poly\_degree).fit\_transform(x)

# x\_poly\_test =PolynomialFeatures(poly\_degree).fit\_transform(x\_test)

for c in c\_range:

if(name\_of\_model=='Lasso'):

model=Lasso(alpha=1/(2\*c))

elif(name\_of\_model=='Ridge'):

model=Ridge(alpha=1/(2\*c))

mean\_square\_error\_temp=[]

kf=KFold(n\_splits=5)

for train,test in kf.split(x):

model.fit(pd.DataFrame(x\_poly).iloc[train],pd.DataFrame(y).iloc[train])

predictions=model.predict(pd.DataFrame(x\_poly).iloc[test])

mean\_square\_error\_temp.append(mean\_squared\_error(pd.DataFrame(y).iloc[test],predictions))

mean\_error.append(np.array(mean\_square\_error\_temp).mean())

std\_error.append(np.array(mean\_square\_error\_temp).std())

plt.errorbar(c\_range,mean\_error,yerr=std\_error)

plt.xlabel('C'); plt.ylabel('Mean square error')

plt.title(f'K fold CV - Choice of C in {name\_of\_model} regression for Feature: {y.columns[0]} for Polynomial Feature Degree: {poly\_degree}')

plt.xscale('log')

plt.show()

# print('R2 score:',r2\_score(pd.DataFrame(y).iloc[test],predictions))

# print('MAE:',mean\_absolute\_error(pd.DataFrame(y).iloc[test],predictions))

# print('MSE:',mean\_squared\_error(pd.DataFrame(y).iloc[test],predictions))

# print('RMSE:',mean\_squared\_error(pd.DataFrame(y).iloc[test],predictions,squared=False))

c\_vals=[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

range\_of\_poly = [1,2,3,4,5]

model='Lasso'

for y in Y:

# print(y.shape)

features=topFeatures[y.columns[0]]

fX=X.copy()

fX=fX[[c for c in X.columns if c in features]]

# poly\_feature=5

print(y.columns[0])

kfcv(fX,y,c\_vals,model,range\_of\_poly)

print('-------------------------------------')

# print('----------------------------------------------------')

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_absolute\_error

# mean\_error=[];std\_error=[];

for y in Y:

# print(y.columns[0])

label\_value=y.columns[0]

c=0

poly\_degree=0

if(label\_value=='review\_scores\_rating'):

poly\_degree=4

c=0.0001

elif(label\_value=='review\_scores\_accuracy'):

poly\_degree=5

c=0.0001

elif(label\_value=='review\_scores\_cleanliness'):

poly\_degree=3

c=0.001

elif(label\_value=='review\_scores\_checkin'):

poly\_degree=4

c=0.001

elif(label\_value=='review\_scores\_communication'):

poly\_degree=5

c=0.0001

elif(label\_value=='review\_scores\_location'):

poly\_degree=5

c=0.001

elif(label\_value=='review\_scores\_value'):

poly\_degree=5

c=0.001

Lasso\_x\_poly = PolynomialFeatures(poly\_degree).fit\_transform(fX)

# mean\_square\_error\_temp=[]

kf=KFold(n\_splits=5)

for train,test in kf.split(fX):

model=Lasso(alpha=1/(2\*c))

model.fit(pd.DataFrame(Lasso\_x\_poly).iloc[train],pd.DataFrame(y).iloc[train])

predictions=model.predict(pd.DataFrame(Lasso\_x\_poly).iloc[test])

# mean\_square\_error\_temp.append(mean\_squared\_error(pd.DataFrame(y).iloc[test],predictions))

# mean\_error.append(np.array(mean\_square\_error\_temp).mean())

# std\_error.append(np.array(mean\_square\_error\_temp).std())

print('R2 score:',r2\_score(pd.DataFrame(y).iloc[test],predictions))

print('MAE:',mean\_absolute\_error(pd.DataFrame(y).iloc[test],predictions))

print('MSE:',mean\_squared\_error(pd.DataFrame(y).iloc[test],predictions))

print('RMSE:',mean\_squared\_error(pd.DataFrame(y).iloc[test],predictions,squared=False))

#### Ridge Classifier:

c\_vals=[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

range\_of\_poly = [1,2,3,4,5]

model='Ridge'

for y in Y:

features=topFeatures[y.columns[0]]

fX=X.copy()

fX=fX[[c for c in X.columns if c in features]]

print(y.columns[0])

kfcv(fX,y,c\_vals,model,range\_of\_poly)

print('-------------------------------------')

#### Random Forest:

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import RandomizedSearchCV

for y in Y :

print(y.columns[0])

feats = topFeatures[y.columns[0]]

fX = X.copy()

fX = fX[[c for c in X.columns if c in feats]]

# Number of trees in random forest

number\_of\_trees = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

# Number of features to consider at every split

max\_features = [1.0, 'sqrt']

# Maximum number of levels in tree

max\_levels = [int(x) for x in np.linspace(10, 110, num = 11)]

max\_levels.append(None)

# Minimum number of samples required to split a node

min\_samples\_to\_split = [2, 5, 10]

# Minimum number of samples required at each leaf node

min\_samples\_on\_leaf = [1, 2, 4]

# Method of selecting samples for training each tree

method\_to\_use = [True, False]

# Create the random grid

random\_grid = {'n\_estimators': number\_of\_trees,

'max\_features': max\_features,

'max\_depth': max\_features,

'min\_samples\_split': min\_samples\_to\_split,

'min\_samples\_leaf': min\_samples\_on\_leaf,

'bootstrap': method\_to\_use}

# Use the random grid to search for best hyperparameters

# First create the base model to tune

random\_forest\_model = RandomForestRegressor()

# Random search of parameters, using 3 fold cross validation,

# search across 100 different combinations, and use all available cores

random\_rf = RandomizedSearchCV(estimator = random\_forest\_model, param\_distributions = random\_grid, n\_iter = 100, cv = 3, verbose=2, random\_state=42, n\_jobs = -1)

# Fit the random search model

random\_rf.fit( x\_train, y\_train.values.ravel() )

base\_model = RandomForestRegressor(n\_estimators = 10, random\_state = 42)

base\_model.fit( x\_train, y\_train.values.ravel() )

best\_random = random\_rf.best\_estimator\_

predictions = best\_random.predict( x\_test )

print(f'Acc: {accuracy\_score(y\_test.astype(int),predictions.astype(int))}')

print(f'RMSE: {mean\_squared\_error(y\_test,predictions)}')

print(f'R2: {r2\_score(y\_test,predictions)}')

#### Dummy Classifier:

from sklearn.dummy import DummyRegressor

for y in Y:

features=topFeatures[y.columns[0]]

fX=X.copy()

fX=fX[[c for c in X.columns if c in features]]

print(y.columns[0])

dummy\_classifier = DummyRegressor(strategy="mean")

dummy\_classifier.fit(x\_train, y\_train)

ypred=dummy\_classifier.predict(x\_test)

print(f'R2 Score: {r2\_score(y\_test,ypred)}')

print(f'MSE: {mean\_squared\_error(y\_test,ypred)}')

print(f'RMSE: {mean\_squared\_error(y\_test,ypred,squared=False)}')

print(f'Accuracy: {accuracy\_score(y\_test.astype(int),ypred.astype(int))}')

print('------------------------------------------')