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:

$$\hat{y} = \partial_0 + \partial_1 x_1 + \partial_2 x_2 + \partial_3 x_3 + \cdots + \partial_{(n-1)} x_{(n-1)} + \partial_n x_n$$

Wherein,

$\hat{y}$: dependent variable
∂_0: y-intercept
∂_1: slope of coefficient 1
x_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:

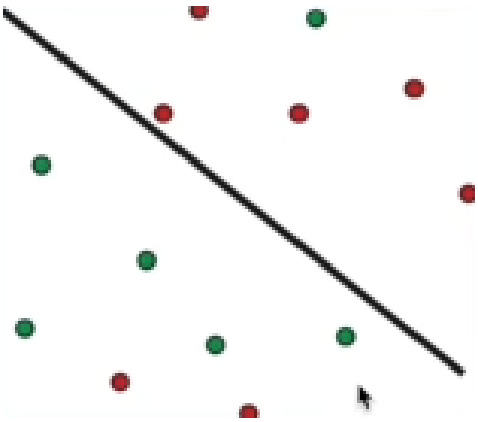
Don't just Google for the answers to the questions below and do not use jargon you don't fully understand. Read the lecture notes, think about your answers, and make sure to explain them in your own words.

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]

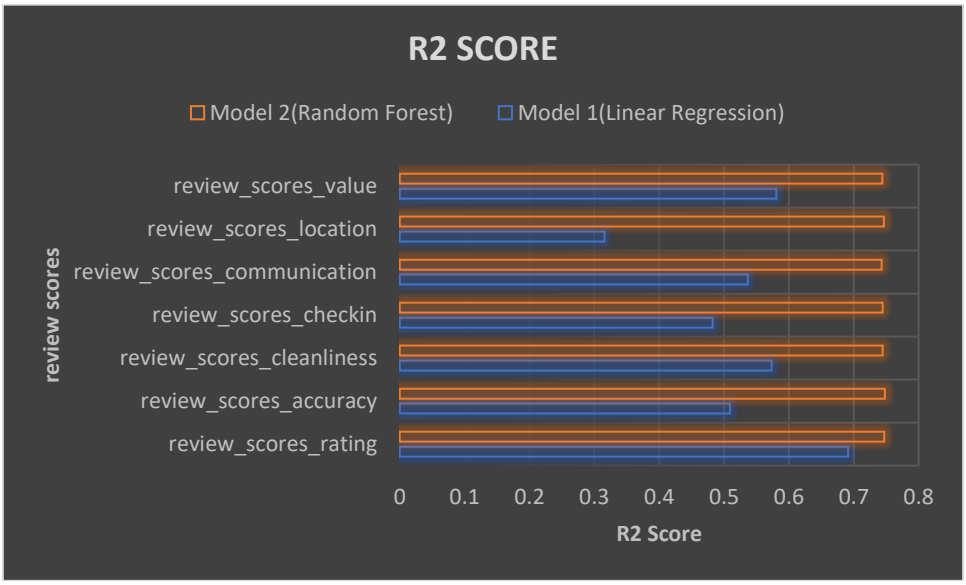
The dataset for a time series would only include recorded values and time values. Any machine learning approach cannot exploit this. Datasets may be feature engineered in a variety of ways, such as by adding X features and a Y output variable that can be utilized in a forecasting model. One such method 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

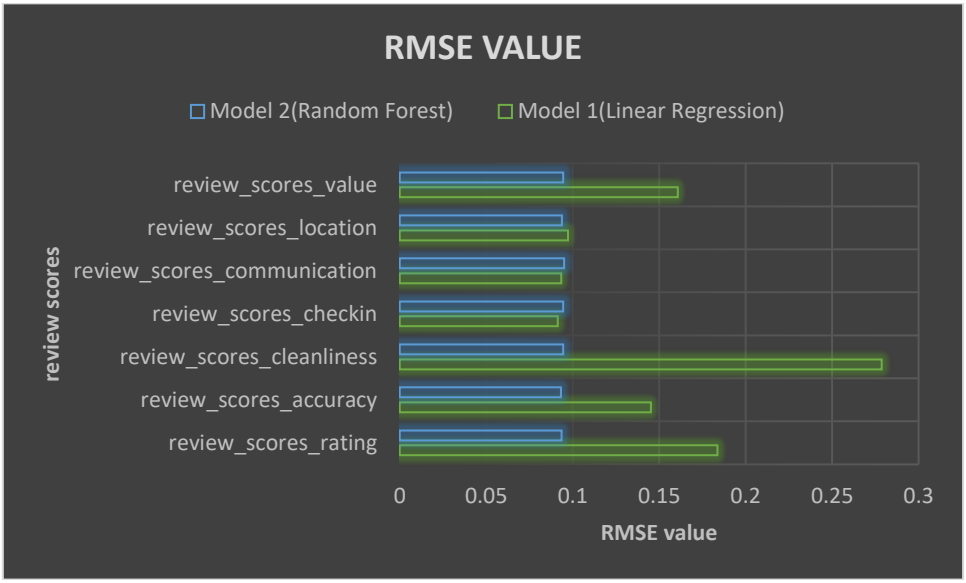
Table 1: R2 Score Values for both the models



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_sco
res_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_sc
ores_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_sc
ores_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_sc
ores_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('----------------------------------------')
```