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: Masanari Doi

Date: 4th/Jan/2023

Machine Learning Final Assignment

Masanari Doi - 19313167

Question 1

1. Dataset overview and cleanup, and feature selections

The datasets we will handle are listings.csv and reviews.csv. The former one includes the information about Airbnb listings in Dublin, such as name of listings, location or rating. The latter one has for example listing_id and review comments. In this assignment, utilising these two datasets, it is asked to evaluate the feasibility of predicting for a listing the individual ratings for accuracy, cleanliness, checkin, communication, location and value and also the overall review rating
Firstly, it is needed to preprocess the dataset to achieve a meaningful score.

Reviews

In the reviews.csv it has reviews in a column called comments. Some rows are empty and some words are not English so it is needed to remove it because countvectorizer will be used later that only accepts English. In addition, for a machine learning training model, it is essential to delete trivial words, such as punctuations and stop words. The code below does the text featuring.

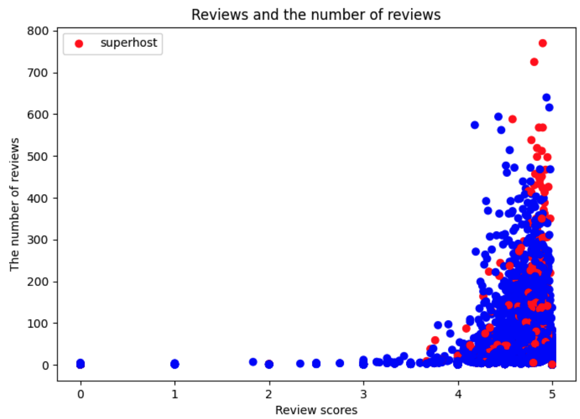
```
def cleanup_texts(data, column):
 # print(data)
  data[column] = data[column].astype(str)
  def is_nonEnglish(text):
   # Check if the comments contain any non-English characters or emoji
(non-ascii characters)
   if re.search(r'[^\x00-\x7F]', text):
     return True                           # True if the text contains any
non-English characters
   return False
  # REmove non-english texts
 filters = data[column].apply(is_nonEnglish)
 data = data[filters == False]

 # Stop words list in English
 stop_words = stopwords.words('english')

 def manipulate_texts(text):
   text = "".join([word for word in text if word not in
string.punctuation])
   text = text.lower()
   text = " ".join([word for word in text.split() if word not in
stop_words])
   return text
# Remove stop words and punctuations from the comments and make text
lower case
 data[column] = data[column].apply(lambda x: manipulate_texts(x))
 # data.dropna(axis=0, inplace=True)
 return data
```

`if re.search(r'[^\x00-\x7F]', text):` is used to find non-english words and make a filter. Then, `data = data[filters == False]` is used to remove non-english words. In order to remove punctuations and stop words, `text = "".join([word for word in text if word not in string.punctuation])` and `text = " ".join([word for word in text.split() if word not in stop_words])` is used. In addition, for more features, `text = text.lower()` is used to turn text into lower case because the weights for words should not be changed by whether lowercase or uppercase. Then, it is not possible to use the rows where there is no text, the code above removes these rows.

```
reviewData = reviewData[reviewData["comments"].notnull()]
reviewData = cleanup_texts(reviewData, "comments")
```

Listings

As a first approach, the columns in the data that have all nulls are removed because it can not affect the training, such as calendar_updated and license. Then, the data that does not look important is removed from the file, such as urls and ids because machine learning models can not go to urls and see the links content, and ids look irrelevant to the ratings. Furthermore, some data are also removed by experiments.



The figure above illustrates how being a superhost affects the number of reviews and review scores. It can be seen that whether being a superhost or not does not seem to make a big difference so host_is_superhost is removed.

Next, host_response_time, host_response_rate and host_acceptance_rate look relevant to communication, so there is a possibility that they affect review_scores_communication. It is found that among these three data, the rows of N/A are all the same so they are relevant to each other. For the experiment, host_response_rate is used because it can measure how much the hosts are eager to communicate with guests.

```
avg_rate = 0
# Get average
for row in range(len(listingData["review_scores_communication"])):
  if row in listingData["review_scores_communication"]:
    avg_rate = avg_rate + 
listingData["review_scores_communication"][row]

avg_rate = avg_rate / len(listingData["review_scores_communication"])
print(avg_rate)
```

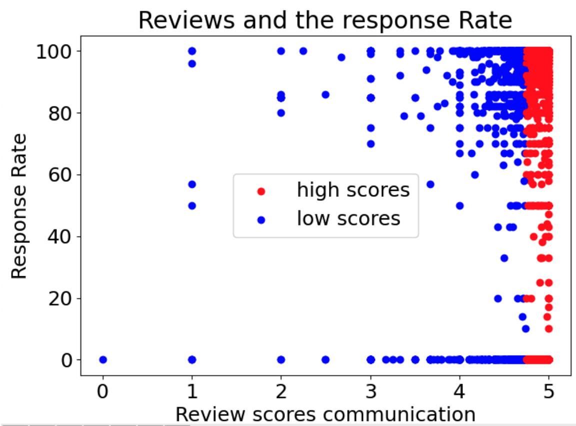
First of all, the code above is used to calculate the mean of review_scores_communication.

```
def to_float(x):
   if pd.isnull(x):
       return 0
   else:
       return float(x.replace("%", ""))
```

Then, the code removes “%” and makes the data to float.

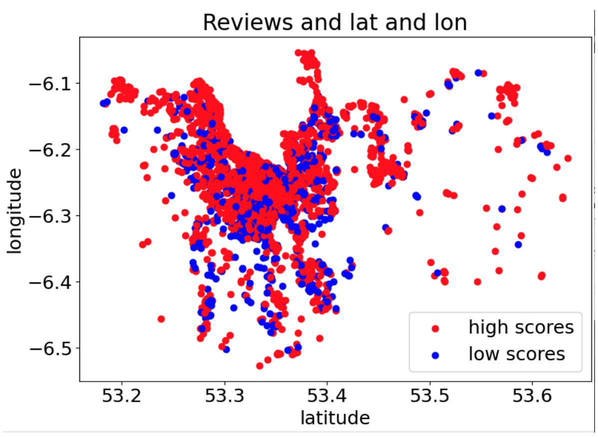
```
# If the rate > average_rate, plot red colour, otherwise blue
for row in range(len(listingData["review_scores_communication"])):
 if row in listingData["review_scores_communication"]:
   # If the rate > average_rate, plot red colour, otherwise blue
   if listingData["review_scores_communication"][row] > avg_rate:
           plt.scatter(listingData["review_scores_communication"][row],
listingData['host_response_rate'][row], color="red")
     if legend_one == 0:
           plt.scatter(listingData["review_scores_communication"][row],
listingData['host_response_rate'][row],  color="red",  label  =  "high
scores")
       plt.legend()
       legend_one = 1
   elif listingData["review_scores_communication"][row] <= avg_rate:
           plt.scatter(listingData["review_scores_communication"][row],
listingData['host_response_rate'][row], color="blue")
     if legend_two == 0:
           plt.scatter(listingData["review_scores_communication"][row],
listingData['host_response_rate'][row],  color="blue",  label  =  "low
scores")
       plt.legend()
       legend_two = 1
```

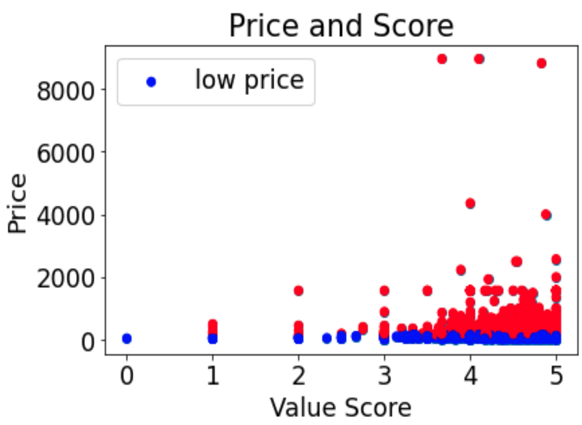
Finally, the code above is used to plot the data. In the for-loop it detects whether the review scores are larger than the average scores. If so they are plotted in red colour otherwise blue.



As above figure, it can clearly be seen that the response rate does not affect review_scores_communication because both high and low response rate exist in both high and low review_scores_communicaton. Therefore, host_response_time, host_response_rate and host_acceptance_rate are removed.

Moreover, in the same way of the above two figures, the figures below are obtained.





The figure on the left is plotting latitude and longitude, and review_scores_location in high scores with red and low scores with blue. It is illustrated that they are plotted at the almost same location so latitude and longitude do not look important as well and are removed.
The figure on the right side is plotting prices of listings and review_scores_value in the same colours as the left figure. It can also be seen that all prices are plotted regardless of scores so prices are also irrelevant. It is removed.

```
# Delete the columns that does not look important to the training.
listingData = listingData.drop(["listing_url", "scrape_id",
"last_scraped", "picture_url","host_id" ,"host_url", "host_name",
"host_neighbourhood", "host_thumbnail_url", "host_picture_url",
"host_verifications", "neighbourhood", "neighbourhood_cleansed",
"neighbourhood_group_cleansed", "bathrooms", "calendar_last_scraped",
"first_review","last_review","license", "calendar_updated",
"calculated_host_listings_count_shared_rooms", "first_review",
"last_review", "host_location","host_response_time",
"price","host_is_superhost", "host_is_superhost",
"host_response_rate","host_acceptance_rate", "host_since", "source",
"longitude", "latitude"], axis = 1)
```

These are the columns that are removed for the time being.
After the experiments above,

```
listingData['host_identity_verified'] =
listingData['host_identity_verified'].map({"t": 1, "f": 0})

listingData['has_availability'] =
listingData['has_availability'].map({"t": 1, "f": 0})

listingData['instant_bookable'] =
listingData['instant_bookable'].map({"t": 1, "f": 0})

listingData['host_has_profile_pic'] =
listingData['host_has_profile_pic'].map({"t": 1, "f": 0})
```

These codes above are conducted for the columns that have texts t or f. Only texts t or f can not be a significant feature because of its low vocabulary. Then, the text featuring is applied to the columns `["name", "description","neighborhood_overview", "host_about"]` as review.csv did the text featuring in `def cleanup_texts(data, column):`.

In addition, it will be needed to evaluate prediction accuracy but many ratings seem to have over 4. Therefore, to make each rating look more unique, the code below is used to multiply every score by 100.

```
# multiply all numbers in y by 100
y = [x * 100 for x in y]
```

For the last part of preprocessing, the reviews are added to listings.csv in accordance with listing_id.

2. Vectorization and Training Models

```
vectors_array = []
for column in text_columns.columns:
 # Convert the text data into a matrix of token counts
 vectorizer = CountVectorizer()
 # print("__________________________________________")
 print(listingData[column])
 vector = vectorizer.fit_transform(listingData[column])
 vectors_array.append(vector)
```

Before training, we need to transform texts into vectors because the vectors can have the feature of texts.
It is decided to use Linear Regression and Decision Tree Regressor because they are easy to implement. Furthermore, since the data that will be trained contain a large number of texts and numerical values, these two models that can manipulate these data are suitable.

Linear Regression

When it uses multiple variables,

Model: $h_0(x) = \theta^T x$

Cost function: $J(\theta_1, \theta_2, , , , , , \theta_n) = J(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_0(x^{(i)}) - y^{(i)})$

I used the code below.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
LinearRegression,
model = LinearRegression()

# train data
model.fit(xTrain, yTrain)
# Make predictions on the test set
y_pred = model.predict(xTest)
# Compute the R^2 score
r2 = r2_score(yTest, y_pred)
print("R^2 score:", r2)
```

```
# get mean squared error
mse = mean_squared_error(yTest, y_pred)
print("Mean square error:", mse)
```

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score, f1_score

model = DecisionTreeRegressor()
# train data
model.fit(xTrain, yTrain)
# Make predictions on the test set
y_pred = model.predict(xTest)
# Compute the R^2 score
r2 = r2_score(yTest, y_pred)
print("R^2 score:", r2)
# get mean squared error
mse = mean_squared_error(yTest, y_pred)
```

3. Evaluation

Both of two models can be evaluated by

Mean Square error (MSE)

$$\frac{1}{m}\sum_{i=1}^{m}\left(\theta^T x^{(x)} - y^{(i)}\right)^2$$

When mean square error is low, the model is said to be fit.

and R^2

$$1 - \frac{\sum_{i=1}^{m}\left(\theta^T x^{(x)} - y^{(i)}\right)^2}{\sum_{i=1}^{m}\left(\theta^T x^{(x)} - \bar{y}\right)^2} \text{ where } \bar{y} = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}$$

Especially, $when\ R^2 = 1$, it means the model predicts perfectly and $when\ R^2 = 0$, prediction is not better than baseline.

```
R^2 score: -0.5194423987608836
Mean square error: 1198.6724789038235
```

This is the output from the linear regression model. $R^2 < 0$, so this means the output of linear regression is lower than the baseline model. MSE is also high.

```
R^2 score: 0.42386200869879886
Mean square error: 711.3335644937587
```

On the contrary, when using decision tree regressor, $R^2 > 0$, so it means it is better than the baseline. MSE is lower than that of linear regression model

Next, in order to increase the R^2 for linear regression, the text data only used for training this time is review. i.e . all other texts are removed, such as "name", "description", "neighborhood_overview" and "host_about". Then, the figure below was obtained.

```
R^2 score: 0.9951322171126459
Mean square error: 7.042317305855035
```

It can be seen that $R^2 > 0$ and it is close to 1. MSE is also low. This can be because when dropping other data, the number of rows of reviews is increased. In the first section, we used def cleanup_texts(data, column): that deletes the rows where there is non-English. Therefore, the actual number of reviews that the linear regression trains is increased, which gives this regression model higher output.

```
R^2 score: 0.6569385303742115
Mean square error: 475.14829356689825
```

On the contrary, when using decision tree regressor, although it is still $R^2 > 0$, R^2 decreased. This can be because the decision tree regressor is not good at manipulating a lot of data.

As above, it can be seen that when using an adequate model and adequate number of features, if the data has a mix of numbers and texts, the model can predict the outputs precisely. Therefore, there is the feasibility of predicting for a listing.

Question 2

(i)

1.
When not all the data is linearly separable, it causes an inaccurate prediction because it only draws the straight line.

2.
When the data points are too far from the decision boundary, it causes misclassification.

(ii)
One of the positive aspects of kNN is that it is easy to use because the parameter needed is only k. However, it can become expensive when there is a lot of data to train because it needs to search for all training data to k when prediction is conducted.

One of the disadvantages of using MLP is it is slow and difficult to train so the cost function is non-convex in the neural net weights or parameters. Moreover, it needs a larger number of weights and parameters to learn. On the contrary, one of the advantages of the MLP is that it can manipulate a lot of data and then produce a variety of functions that can map input x to output y.

(iii)

By means of k-fold cross-validation, it is possible to train and test multiple times. For example, if we split the data into 5 parts, we use the first part to test and the other to train, then we use the second part to test and the other to train. We keep doing it until the fifth part is used for the test.

As k increases, we can use more data to train, so we want k to be large. However, if k is too large, computation times increase because it is needed to fit the mode k times. We also do not want k to be large. Therefore, k = 5 or 10 are reasonable to balance the number of training and the computation time.

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

Lagged output values are used to help predict the future time stamp. For example, when we require the weather tomorrow, we use not only a lot of past data but also add the past few days' weather data because we can get a trend to help predict it.

APPENDIX

```
from cProfile import label
from re import X
# from statistics import LinearRegression
from tkinter import Y
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier

import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
import string


from sklearn.feature_extraction.text import TfidfVectorizer
from  sklearn.metrics  import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.metrics import f1_score
from sklearn.preprocessing import PolynomialFeatures
# from yellowbrick.classifier import ROCAUC
from langdetect import detect
import re
from sklearn.feature_extraction.text import CountVectorizer

from array import array


def cleanup_texts(data, column):

 # print(data)
  data[column] = data[column].astype(str)
  def is_nonEnglish(text):
```

```
   # Check if the comments contain any non-English characters or emoji (non-ascii
characters)
   if re.search(r'[^\x00-\x7F]', text):
     return True                       # True if the text contains any non-English
characters
   return False
  # REmove non-english texts
 filters = data[column].apply(is_nonEnglish)
 data = data[filters == False]

 # Stop words list in English
 stop_words = stopwords.words('english')

 def manipulate_texts(text):
   text = "".join([word for word in text if word not in string.punctuation])
   text = text.lower()
   text = " ".join([word for word in text.split() if word not in stop_words])
   return text

# Remove stop words and punctuations from the comments and make text lower case
 data[column] = data[column].apply(lambda x: manipulate_texts(x))
 # data.dropna(axis=0, inplace=True)
 return data

# Preprocessing in reviews.csv
reviewData = pd.read_csv('/Users/doimasanari/Documents/ML/final/code/reviews.csv')

# Sort the DataFrame by the values in column listing_id
# reviewData = reviewData.sort_values(by='listing_id')

# Remove the rows that have null
reviewData = reviewData[reviewData["comments"].notnull()]
reviewData = cleanup_texts(reviewData, "comments")

reviewData.to_csv('cleanedup_review.csv', index=False)

listingData =
pd.read_csv('/Users/doimasanari/Documents/ML/final/code/listings.csv')
# Delete the row where listingData is null
listingData = listingData[listingData["review_scores_rating"].notnull()]

listingData.to_csv('reviewScoresOk_listing.csv', index=False)

listingData = pd.read_csv('reviewScoresOk_listing.csv')
```

```
# Extract the x and y values from the data
x = listingData['review_scores_rating']
y = listingData['number_of_reviews']

# Extract the color values from the data
listingData = listingData[listingData["host_is_superhost"].notnull()]
colors = listingData['host_is_superhost']

# Store the color values
colors_list = []
label = []

# Iterate over the color values
for color in colors:
   # If the color value is "t", append "red" to the list
   if color == "t":
       colors_list.append("red")
   # If the color value is "f", append "blue" to the list
   elif color == "f":
       colors_list.append("blue")
   else: colors_list.append("blue")

colors_list.append("red")

plt.scatter(x, y, color=colors_list, label = "superhost")
plt.xlabel("Review scores")
plt.ylabel("The number of reviews")
plt.title("Reviews and the number of reviews")
plt.legend()
plt.show()

# feature selection response rate

listingData = pd.read_csv('reviewScoresOk_listing.csv')
listingData = listingData[listingData["review_scores_communication"].notnull()]

# Extract data from columns
# x = listingData["review_scores_communication"]
# y = listingData['host_response_rate']

avg_rate = 0
# Get average
for row in range(len(listingData["review_scores_communication"])):
 if row in listingData["review_scores_communication"]:
   avg_rate = avg_rate + listingData["review_scores_communication"][row]
```

```
avg_rate = avg_rate / len(listingData["review_scores_communication"])
print(avg_rate)

def to_float(x):
   if pd.isnull(x):
       return 0
   else:
       return float(x.replace("%", ""))

# Remove $
listingData['host_response_rate'] =
listingData['host_response_rate'].apply(to_float)

# Plot the data
plt.figure()
plt.rc('font', size=18)
plt.rcParams["figure.constrained_layout.use"] = True

# Parameter for legend
legend_one = 0
legend_two = 0

# If the rate > average_rate, plot red colour, otherwise blue
for row in range(len(listingData["review_scores_communication"])):
 if row in listingData["review_scores_communication"]:
   # If the rate > average_rate, plot red colour, otherwise blue
   if listingData["review_scores_communication"][row] > avg_rate:
     plt.scatter(listingData["review_scores_communication"][row],
listingData['host_response_rate'][row], color="red")
     if legend_one == 0:
       plt.scatter(listingData["review_scores_communication"][row],
listingData['host_response_rate'][row], color="red", label = "high scores")
       plt.legend()
       legend_one = 1
   elif listingData["review_scores_communication"][row] <= avg_rate:
     plt.scatter(listingData["review_scores_communication"][row],
listingData['host_response_rate'][row], color="blue")
     if legend_two == 0:
       plt.scatter(listingData["review_scores_communication"][row],
listingData['host_response_rate'][row], color="blue", label = "low scores")
       plt.legend()
       legend_two = 1
     # colors_listll.append("blue")
```

```python
plt.xlabel("Review scores communication")
plt.ylabel("Response Rate")
plt.title("Reviews and the response Rate")
# plt.legend()
plt.show()

# location analysis

listingData = pd.read_csv('reviewScoresOk_listing.csv')
listingData = listingData[listingData["review_scores_location"].notnull()]

print(listingData["review_scores_location"])

avg_rate = 0

# Get average
for row in range(len(listingData["review_scores_location"])):
 if row in listingData["review_scores_location"]:
   avg_rate = avg_rate + listingData["review_scores_location"][row]

avg_rate = avg_rate / len(listingData["review_scores_location"])
print(avg_rate)

# x = listingData["latitude"]
# y = listingData['longitude']
# z = listingData["review_scores_location"]

# Plot the data
plt.figure()
plt.rc('font', size=18)
plt.rcParams["figure.constrained_layout.use"] = True

# Parameter for legend
legend_one = 0
legend_two = 0

# If the rate > average_rate, plot red colour, otherwise blue
for row in range(len(listingData["review_scores_location"])):
 if row in listingData["review_scores_location"]:
   # If the rate > average_rate, plot red colour, otherwise blue
   if listingData["review_scores_location"][row] > avg_rate:
     plt.scatter(listingData["latitude"][row], listingData['longitude'][row],
color="red")
     if legend_one == 0:
```

```
       plt.scatter(listingData["latitude"][row], listingData['longitude'][row],
color="red", label = "high rate")
       plt.legend()
       legend_one = 1
   elif listingData["review_scores_location"][row] <= avg_rate:
     plt.scatter(listingData["latitude"][row], listingData['longitude'][row],
color="blue")
     if legend_two == 0:
       plt.scatter(listingData["latitude"][row], listingData['longitude'][row],
color="blue", label = "low rate")
       plt.legend()
       legend_two = 1
     # colors_listll.append("blue")

plt.xlabel("latitude")
plt.ylabel("longitude")
plt.title("Reviews and lat and lon")
# plt.legend()
plt.show()

# price analysis

import matplotlib.pyplot as plt

x = listingData["review_scores_value"]
y = listingData['price']

# Convert x data to numeric
y = [float(y.replace("$", "").replace(",", "")) for y in y]

av_price = 0

for row in range(len(y)):
   if y[row] > 75000:
     y[row] = 0
   av_price = av_price + y[row]

av_price = av_price / len(y)
print(av_price)

# Colur array
colors_list = []

for row in range(len(y)):
   # If price is higher than avg colour is red
```

```
    if y[row] > av_price:
        colors_list.append("red")
    # If price is lower than avg colour is blue
    else: colors_list.append("blue")

plt.scatter(x, y)
plt.scatter(x, y, color=colors_list, label = "low price")
plt.xlabel("Value Score")
plt.ylabel("Price")
plt.title("Price and Score")
plt.legend()
plt.show()

"""#### Preprocessing and feature selection in listings.scv """

listingData =
pd.read_csv('/Users/doimasanari/Documents/ML/final/code/listings.csv')
temp = pd.read_csv('/Users/doimasanari/Documents/ML/final/code/listings.csv')

print(listingData)

# Delete the columns that does not look important to the training.
listingData = listingData.drop(["listing_url", "scrape_id", "last_scraped",
"picture_url","host_id" ,"host_url", "host_name", "host_neighbourhood",
"host_thumbnail_url",
                                "host_picture_url", "host_verifications",
"neighbourhood", "neighbourhood_cleansed", "neighbourhood_group_cleansed",
"bathrooms", "calendar_last_scraped",
                                "first_review","last_review","license",
"calendar_updated", "calculated_host_listings_count_shared_rooms", "first_review",
"last_review", "host_location",
                                "host_response_time", "price","host_is_superhost",
"host_is_superhost", "host_response_rate","host_acceptance_rate", "host_since",
"source", "longitude", "latitude"], axis = 1)

# print(listingData)

listingData['host_identity_verified'] =
listingData['host_identity_verified'].map({"t": 1, "f": 0})
listingData['has_availability'] = listingData['has_availability'].map({"t": 1, "f":
0})
listingData['instant_bookable'] = listingData['instant_bookable'].map({"t": 1, "f":
0})
listingData['host_has_profile_pic'] = listingData['host_has_profile_pic'].map({"t":
1, "f": 0})
```

```
print("____________________________________________________________")
print(listingData)

# Choose the column that has sentences
text_columns = ["name", "description","neighborhood_overview", "host_about" ]

# listingData[text_columns] = listingData[text_columns].astype(str)

# Remove stop words and punctuations from the commentsmake and make text lower case
for column in text_columns:
 listingData = cleanup_texts(listingData, column)
 # listingData[column] = listingData[column].apply(lambda x: cleanup_texts(x))

print("____________________________________________________________")
print(text_columns)
print(listingData)
print("____________________________________________________________")
# listingData = listingData[listingData["id"].notnull()]
# listingData.dropna(axis=0, inplace=True)
print(listingData)
# print(vectorised_texts)
# print(len(vectorised_texts))

listingData.to_csv('cleanedup_listing.csv', index=False)

# add reviews to listings.csv

reviewData = pd.read_csv('cleanedup_review.csv')
# print(reviewData)
reviewData["listing_id"] = reviewData["listing_id"].astype(int)

listingData = pd.read_csv('cleanedup_listing.csv')
listingData["id"] = listingData["id"].astype(int)
print(listingData)

for row in range(len(listingData)):
  temp_array = []
 # print(listingData["id"][row])
 if row in listingData["id"]:

print("_____________________________________________________________________________________________________________________________________________________________")
   # print(listingData["id"][row])
```

```
    # Look for for rows that contain the same listing_id
    filters = reviewData["listing_id"].isin([listingData["id"][row]])
    result = reviewData[filters]
    print(result)
    temp_array = result["comments"].tolist()

print("___________________________________________________________________________________________________________________________________________________________")
    # print(temp_array[119])
    # There is float so change it to strings
    temp_array = [str(x) for x in temp_array]
    temp_array = " ".join(temp_array)

    listingData.at[row, "reviews"] = temp_array

# print(listingData)

# listingData = listingData[listingData["reviews"].notnull()]
# listingData = listingData.dropna()
# listingData = listingData.drop(listingData.index[-1])
print("___________________________________________________________________________________________________________________________________________________________")
print(listingData)

# print(listingData)
listingData.to_csv('merged_listing.csv', index=False)

# featuring

listingData = pd.read_csv('merged_listing.csv')
print("___________________________________________________________")
print(listingData)

# listingData = listingData.drop(["id","review_scores_rating",
"review_scores_accuracy", "review_scores_cleanliness", "review_scores_checkin",
"review_scores_communication", "review_scores_location", "review_scores_value"  ],
axis = 1)
listingData = listingData.drop(["id"], axis = 1)

# listingData = listingData.drop(["id"], axis = 1)

# Drop rows with any NaN values
listingData.to_csv('dropped_rating_listing.csv', index=False)
listingData = pd.read_csv('dropped_rating_listing.csv')
```

```
print("___________________________________________________________")
# print(listingData)

# Drop the row if the column has NaN
for column in listingData.columns:
 print("___________________________________________________________")
 print(column)
 print("___________________________________________________________")
 listingData = listingData.dropna(subset=[column])
 print(listingData)

listingData.to_csv('clean_completed_listing.csv', index=False)
listingData = pd.read_csv('clean_completed_listing.csv')
print("___________________________________________________________")
print(listingData)


y = listingData.loc[:, "review_scores_rating":"review_scores_value"].values


# listingData = map(multiplyBy10, listingData["review_scores_rating"])
# listingData = map(multiplyBy10, listingData['review_scores_accuracy'])
# listingData = map(multiplyBy10, listingData['review_scores_cleanliness'])
# listingData = map(multiplyBy10, listingData['review_scores_checkin'])
# listingData = map(multiplyBy10, listingData['review_scores_communication'])
# listingData = map(multiplyBy10, listingData['review_scores_location'])
# listingData = map(multiplyBy10, listingData['review_scores_value'])

# y = map(multiplyBy10, y)

# multiply all numbers in y by 100
y = [x * 100 for x in y]
print(y)
print("______________________________________________")
# print(listingData)
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LinearRegression
from scipy.sparse import hstack

# Select the columns that contain text data
text_columns = listingData.select_dtypes(include=['object'])
print(text_columns.columns)

vectors_array = []
```

```
for column in text_columns.columns:
 # Convert the text data into a matrix of token counts
 vectorizer = CountVectorizer()
 # print("__________________________________")
 print(listingData[column])
 vector = vectorizer.fit_transform(listingData[column])
 vectors_array.append(vector)

numerical_columns = listingData.select_dtypes(exclude=['object'])

# print(numerical_columns)

# Convert the numeric data into dummy variables
numeric_dummies = pd.get_dummies(numerical_columns)

print("_____________________________________")
print(vectors_array)

# text_df = pd.DataFrame(text_matrix.todense())
# vectors_array = pd.to_numeric(vectors_array)

X = hstack(vectors_array)
X = hstack((X, numeric_dummies))
# print(text_matrix)

# Concatenate the dummy variables with the matrix of token counts
# X = pd.concat([text_df, numeric_dummies], axis=1)
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.2)
# print("______________________________")
# print(X)

# train a model

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
LinearRegression,
model = LinearRegression()

# train data
model.fit(xTrain, yTrain)
# Make predictions on the test set
y_pred = model.predict(xTest)
# Compute the R^2 score
r2 = r2_score(yTest, y_pred)
print("R^2 score:", r2)
# get mean squared error
```

```
mse = mean_squared_error(yTest, y_pred)
print("Mean square error:", mse)


# get a slope here
print("slope = ", model.coef_)
# get an intercept here
print("intercept = ", model.intercept_)


# from sklearn.dummy import DummyClassifier
# dummy = DummyClassifier().fit(xTrain, yTrain)

# ydummy = dummy.predict(xTrain)
# y_pred = np.argmax(ydummy, axis=1)
# y_train1 = np.argmax(yTrain, axis=1)
# print(classification_report(y_train1, y_pred))
# print(confusion_matrix(y_train1,y_pred))

# plt.plot([0, 1], [mse, mse], 'k--')
# plt.plot([0, 1], [rmse, rmse], 'k--')
# plt.plot([0, 1], [mae, mae], 'k--')
# plt.plot([0, 1], [r2, r2], 'k--')
# plt.legend(['MSE', 'RMSE', 'MAE', 'R²'])
# plt.show()

# print(confusion_matrix(yTest,y_pred))

from sklearn.tree import DecisionTreeRegressor
# from sklearn.tree import DecisionTreeClassifer
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,
recall_score, f1_score

model = DecisionTreeRegressor()
# train data
model.fit(xTrain, yTrain)
# Make predictions on the test set
y_pred = model.predict(xTest)
# Compute the R^2 score
r2 = r2_score(yTest, y_pred)
print("R^2 score:", r2)
# get mean squared error
mse = mean_squared_error(yTest, y_pred)

model_scores = cross_val_score(model, xTrain, yTrain, cv = 5)
```

```
print("mean cross validation score: {}".format(np.mean(model_scores)))
print("score without cv: {}".format(model.score(xTrain, yTrain)))

print("Score: ", model.score(xTest, yTest))
try:
   print("Intercept: ", model.intercept_)
   print("slope: ", model.coef_)
except AttributeError:
   print("DecisionTreeRegressor does not have an intercept or coefficients.")









# trash of a lots of efforts

# def is_nonEnglish(text):
#    # Check if the comments contain any non-English characters or emoji (non-ascii
characters)
#    if re.search(r'[^\x00-\x7F]', text):
#      return True                        # True if the text contains any non-English
characters
#    return False

# # Convert the values in the comments column to strings
# reviewData['comments'] = reviewData['comments'].astype(str)

# # Apply the is_notEnglish function to each row of the comments columns and delete
the row if it has non-english words
# filters = reviewData['comments'].apply(is_nonEnglish)
# reviewData = reviewData[filters == False]

# Remove the rows that have null

# print("----------------------------------------------------------------")
# print(reviewData)

# # Stop words list in English
# stop_words = stopwords.words('english')

# def cleanup_texts(text):
#     text = "".join([word for word in text if word not in string.punctuation])
#     text = text.lower()
```

```
#     text = " ".join([word for word in text.split() if word not in stop_words])
#     return text

# # Remove stop words and punctuations from the commentsmake and make text lower
case
# reviewData['comments'] = reviewData['comments'].apply(lambda x: cleanup_texts(x))



# print("----------------------------------------------------------------")
# for row in range(len(reviewData)):
#   if row in reviewData["listing_id"] and reviewData["listing_id"][row] == 44077:
#     print(reviewData["comments"][row])
# print(new_df)


# j = 0
# # listingData = listingData.assign(new_column=[])
# # listingData["new_column"] = []
# # print(listingData["new_column"])
# # listingData.dropna(inplace=True)

# review_array = [[] for _ in range(1)]
# for row in range(len(listingData)):
 #   temp_array = []
#   # print(listingData["id"][row])
#   if row in listingData["id"]:
#
print("____________________________________________________________________________
_____________________________________________________________________________")
#     print(listingData["id"][row])
#     # for row2 in range(len(reviewData)):
#     #   if row2 in reviewData["listing_id"] and listingData["id"][row] ==
reviewData["listing_id"][row2]:
#     #     # print(reviewData["comments"][row2])
#     #     temp_array.append(reviewData["comments"][row2])
#     #     reviewData.dropna(inplace=True)

#     # Look for for rows that contain the same listing_id
#     filters = reviewData["listing_id"].isin([listingData["id"][row]])
#     result = reviewData[filters]
#     # print(result)
#     temp_array = result["comments"].tolist()
#     # yyy = row
#     # listingData.loc[row, "comments"] = temp_array[0]
```

```
#     temp_array = " ".join(temp_array)
#     # listingData[[row, "comments"]] = listingData.apply(lambda r: temp_array,
axis=1, result_type="expand")
#     # listingData[["comments"]] = listingData.apply(lambda r: temp_array, axis=1,
result_type="expand")
#     # listingData.at[row, "comments"] = temp_array[0]
#     # twmp_array = array("u", temp_array)
#     # temp_array = temp_array.toarray()
#     print(temp_array)
#     # review_array.append(temp_array)
#     # listingData.at[row, "id"] = None
#     # listingData.at[row, "id"] = temp_array[0]
#     listingData.at[row, "new_column"] = temp_array
#     # listingData.loc[row, "id"] = temp_array
#     # listingData[row]["new_column"].append(review_array)
#     # print()
#         # j += 1

#   # print(reviewData["listing_id"][j])
# print(len(review_array))
# # print(yyy)
#   # print(review_array)

#   # listingData.loc[row, "new_column"] = review_array
# # listingData.loc["new_column"] = review_array
#   # print("jfiejidnrvgigitm")
#   # print(listingData["new_column"])

# print(listingData)

# import math
# listingData = pd.read_csv('reviewScoresOk_listing.csv')
# resCount = 0
# resRateCount = 0
# acceptanceCount = 0
# print(listingData["host_response_time"])
# # print(listingData["host_response_time"][15])
# print("____________________________")
# # Count the number of N/A and change text to number. if its N/A its 0 otherwise 1
# for row in range(len(listingData["host_response_time"])):
 #   if row in listingData["host_response_time"]:
#     resCount += 1
#     listingData["host_response_time"][row] = 1
#     # listingData["host_response_time"][row] = 0
#   else if listingData["host_response_time"].isna().any():
```

```
#     listingData["host_response_time"][row] == 0
#     print(listingData["host_response_time"][row])
#   if row in listingData["host_response_rate"] and
listingData["host_response_rate"][row] == "NaN":
listingData["host_response_rate"][row] = 0
#   else :
#     resRateCount += 1
#     listingData["host_response_rate"][row] = 1
#   if row in listingData["host_acceptance_rate"] and
listingData["host_acceptance_rate"][row] == "NaN":
listingData["host_acceptance_rate"][row] = 0
#   else :
#     acceptanceCount += 1
#     listingData["host_acceptance_rate"][row] = 1

# print("____________________________")
# print(resCount, resRateCount,acceptanceCount)

# def to_int(cell):
#     if pd.isna(cell):
#         return 0
#     return 1

# listingData["host_response_time"] =
listingData["host_response_time"].apply(to_int)


# print(listingData["host_response_time"][15])


# print(listingData["host_response_rate"][14])


# plt.figure()
# plt.rc('font', size=18)
# plt.rcParams["figure.constrained_layout.use"] = True
# plt.scatter(x1RealPlus, x2RealPlus, color="blue", label="actual +1")
# plt.scatter(x1RealMinus, x2RealMinus, color="green", label="actual -1")
# plt.scatter(x1PredPlus, x2PredPlus, color="red", marker="+", label = "predicted
+1")
# plt.scatter(x1PredMinus, x2PredMinus, color="yellow", marker="+", label =
"predicted -1")
# plt.xlabel("x_1")
# plt.ylabel("x_2")
# plt.legend(bbox_to_anchor=(1.04, 1), borderaxespad=0)
```

```
# plt.show()

# vectorarize

# why its not working??
# import pandas as pd
# from sklearn.feature_extraction.text import CountVectorizer
# from sklearn.linear_model import LinearRegression
# from scipy.sparse import hstack


# # Select the columns that contain text data
# text_columns = listingData.select_dtypes(include=['object'])
# print(text_columns.columns)

# # Initialize the CountVectorizer
# vectorizer = CountVectorizer()

# # Convert the text data to numerical vectors
# vectors = vectorizer.fit_transform(text_columns)

# vectors_array = []

# # Select the numerical columns from the input data
# numerical_columns = listingData.select_dtypes(exclude=['object'])

# print(pd.DataFrame(vectors.toarray()))

# X = hstack(pd.DataFrame(vectors.toarray()))

# X = hstack((X, numerical_columns))

# xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.2)

# vectorarizer

# numeric_columns = []
# for column in listingData.columns:
#     if column not in text_columns:
#         numeric_columns.append(column)
```