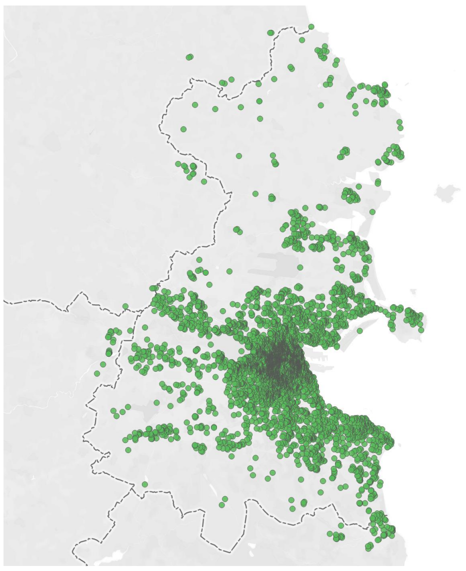
Machine Learning Final Assignment

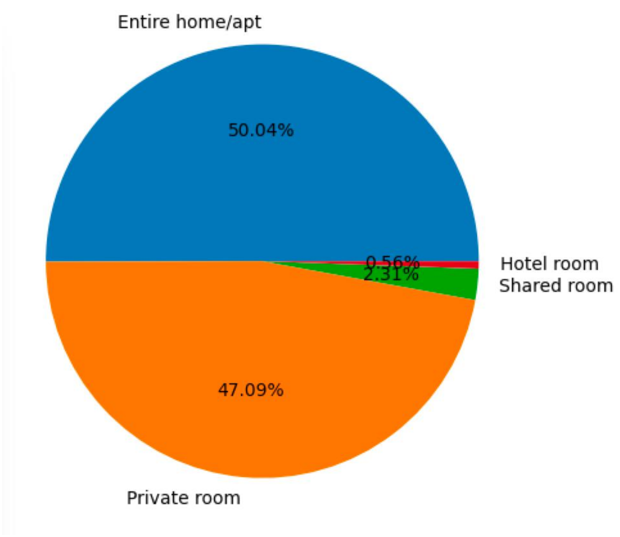
Jiaming Deng 22302794

i (a) Data Analytics and Visualization

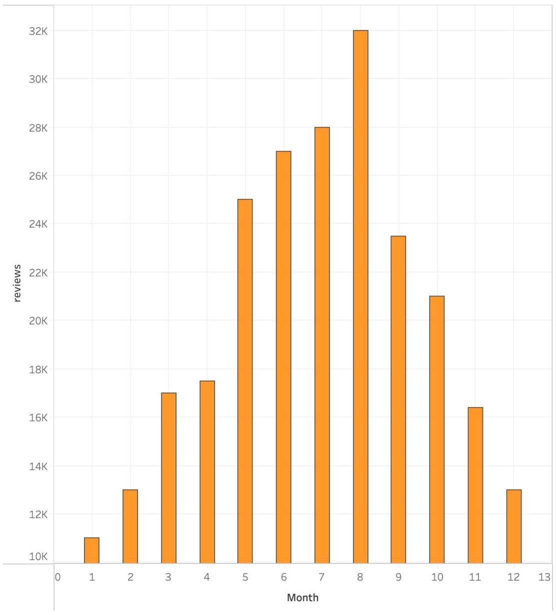
I have two data files reviews.csv and listings.csv, where reviews.csv has 6 columns of data and listings.csv has 75 columns of data. The reviews.csv contains mainly hotel ids, user ids, reviews and review times. The listings.csv contains mainly hotel ids, and hotel The listings.csv contains mainly the hotel id, as well as information about the hotel owner, neighbours, location, ratings, etc.

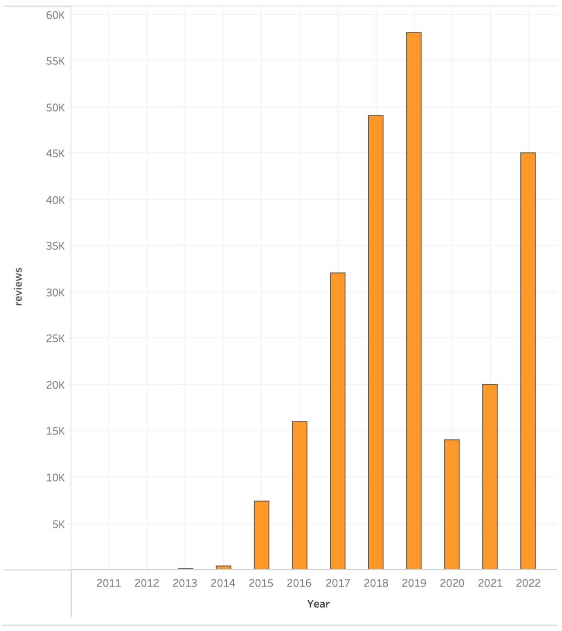
Firstly I show the location of the hostel on the map and display room types.





Secondly, I made a visualisation of the number of Airbnb room ids and reviews to help me understand the distribution of the number of reviews to subsequently extract features from them. The largest number of people travel from May to September, with the number of people commenting growing every year from 2015 until 2020 and recovering a lot by 2022.

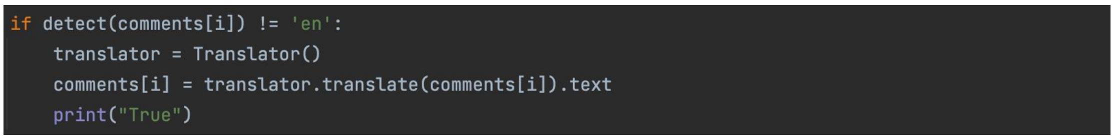




i (b) Feature Selection and Data Preprocessing

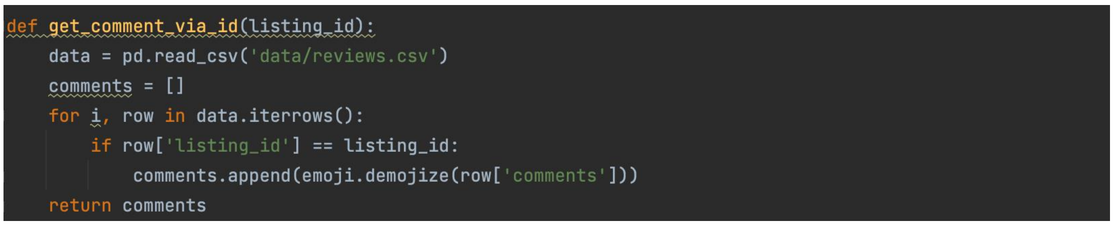
1) Firstly, guest comments have a great relevance to the various ratings of the hotel, so the first feature chosen is the guest's comments. Guest comments are in natural language, so we need to convert the natural language into features according to the following three steps.

1. The guest reviews are not all in English, but in Spanish, French, German, Korean etc., so they need to be translated into English. I used the api provided by Google Translate to do this.



```python
if detect(comments[i]) != 'en':
    translator = Translator()
    comments[i] = translator.translate(comments[i]).text
    print("True")
```

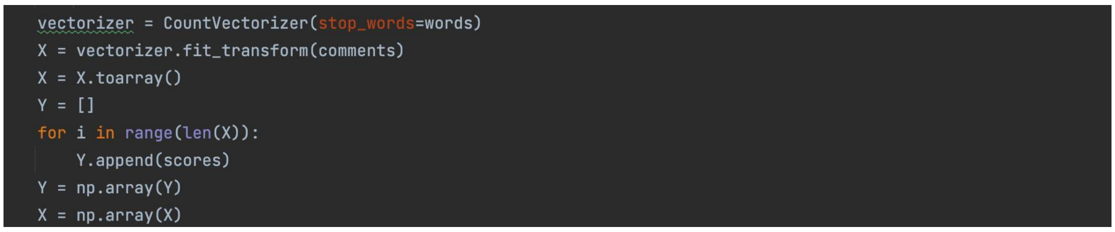
2. The comments contain emoji, so they need to be converted into natural language. I used the python package emoji to achieve this.



```python
def get_comment_via_id(listing_id):
    data = pd.read_csv('data/reviews.csv')
    comments = []
    for i, row in data.iterrows():
        if row['listing_id'] == listing_id:
            comments.append(emoji.demojize(row['comments']))
    return comments
```

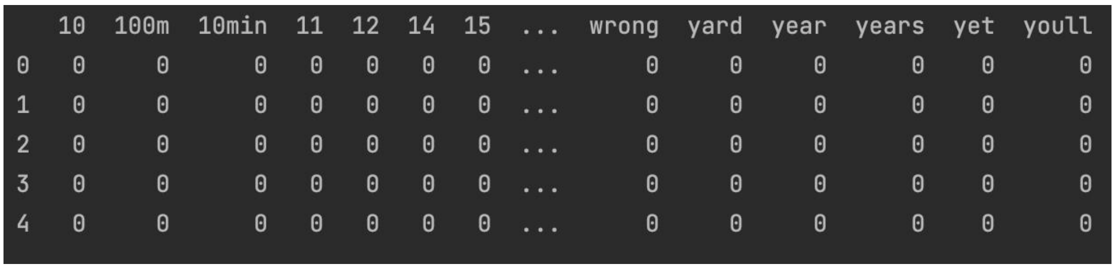
3. After the above two steps of data pre-processing, I used both one-hot encoding and word embedding to convert the natural language (English) into features.

The first is the one-hot encoding method, which converts words into feature vectors according to their frequency of occurrence. I read all the reviews of this hotel in the reviews.csv file according to the hotel id, used the CountVectorizer function and specified the stop words as the official stop words of nltk to convert the read reviews into a feature vector.



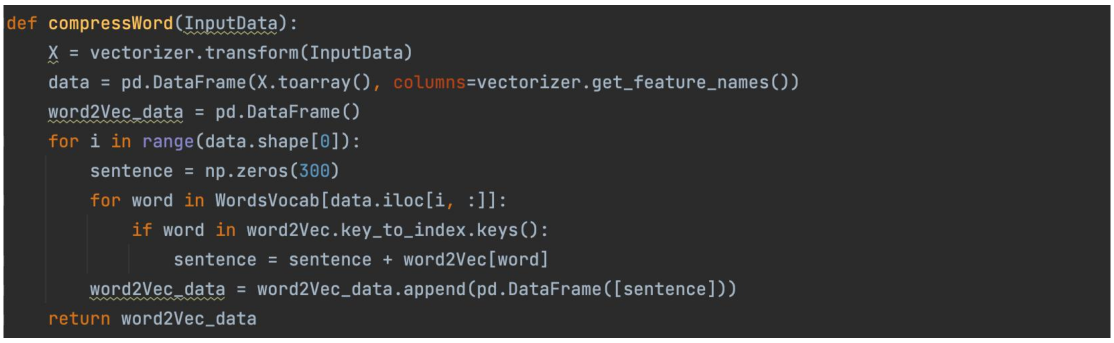
```python
vectorizer = CountVectorizer(stop_words=words)
X = vectorizer.fit_transform(comments)
X = X.toarray()
Y = []
for i in range(len(X)):
    Y.append(scores)
Y = np.array(Y)
X = np.array(X)
```

For example the final feature vector for hotel(id=44077) obtained is in the format 258x1558.

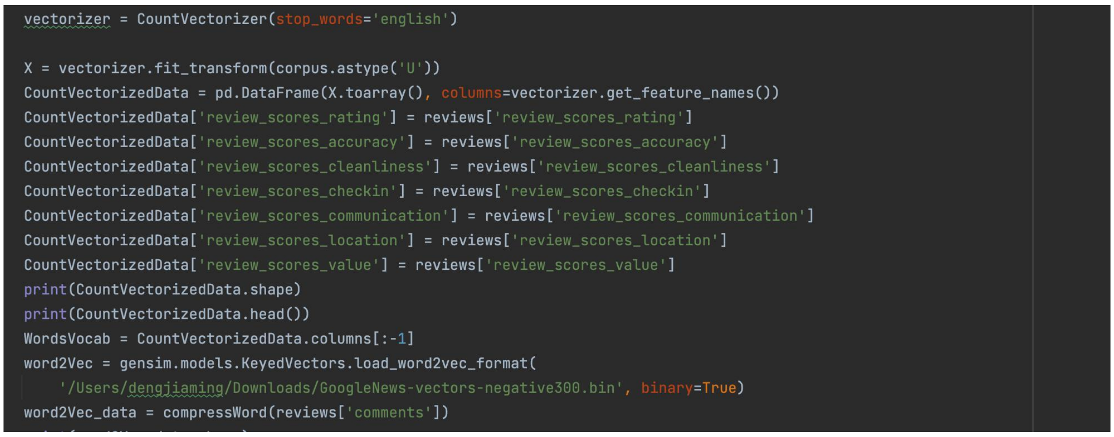


```
   10  100m  10min  11  12  14  15  ...  wrong  yard  year  years  yet  youll
0   0     0      0   0   0   0   0  ...      0     0     0      0    0      0
1   0     0      0   0   0   0   0  ...      0     0     0      0    0      0
2   0     0      0   0   0   0   0  ...      0     0     0      0    0      0
3   0     0      0   0   0   0   0  ...      0     0     0      0    0      0
4   0     0      0   0   0   0   0  ...      0     0     0      0    0      0
```

The second method is word embedding, which can be implemented in a number of ways, including TF-IDF, Word2Vec, FastText and others. In this assignment I used Word2Vec method.
As in my experiments, I found that the number of features obtained using one-hot encoding was very large (1000+). The word embedding process resulted in a feature vector that carried semantic information and was compressed for ease of computation, resulting in a higher prediction score. I found that the word embedding has two main functions:
1. Calculate similarity, e.g. man and woman are more alike than man and apple
Finding the distinctive one in a group of words, e.g. in a list of words such as [dog, cat, chicken, boy], using word vectors to identify boy as not being in the same category as the other three words.
2. Direct word operations, e.g. the classical: woman + king-man =queen. Since it carries semantic information, it is also possible to calculate the likelihood of a passage occurring, i.e. whether the passage is fluent or not.
Essentially, after word embedding, the individual words are combined into a set of vectors on a relatively low-dimensional space, and the proximity of these vectors is determined by the semantic relations between them. This largely optimises the feature vectors obtained by one-hot encoding.
The process implemented is to first convert the comment into the form of a hot-one code, convert the hot-one code into the form of a dataframe, read each row of the data, find out if the word in the comment is in Word2Vec's table, and if it is, put it into a new sentence. The model I used is a pre-trained word2Vec model from google. It contains word vectors for a vocabulary of 3 million words.



```
def compressWord(InputData):
    X = vectorizer.transform(InputData)
    data = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
    word2Vec_data = pd.DataFrame()
    for i in range(data.shape[0]):
        sentence = np.zeros(300)
        for word in WordsVocab[data.iloc[i, :]]:
            if word in word2Vec.key_to_index.keys():
                sentence = sentence + word2Vec[word]
        word2Vec_data = word2Vec_data.append(pd.DataFrame([sentence]))
    return word2Vec_data
```

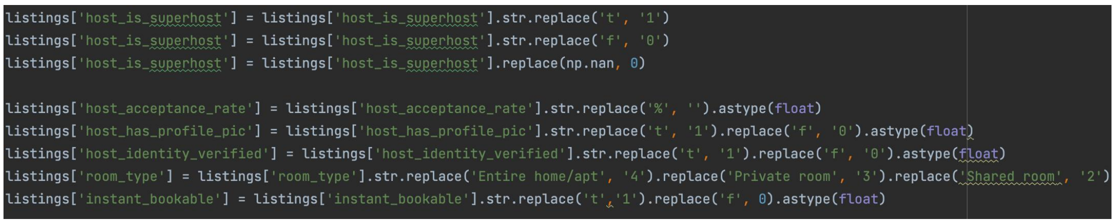


```
vectorizer = CountVectorizer(stop_words='english')

X = vectorizer.fit_transform(corpus.astype('U'))
CountVectorizedData = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
CountVectorizedData['review_scores_rating'] = reviews['review_scores_rating']
CountVectorizedData['review_scores_accuracy'] = reviews['review_scores_accuracy']
CountVectorizedData['review_scores_cleanliness'] = reviews['review_scores_cleanliness']
CountVectorizedData['review_scores_checkin'] = reviews['review_scores_checkin']
CountVectorizedData['review_scores_communication'] = reviews['review_scores_communication']
CountVectorizedData['review_scores_location'] = reviews['review_scores_location']
CountVectorizedData['review_scores_value'] = reviews['review_scores_value']
print(CountVectorizedData.shape)
print(CountVectorizedData.head())
WordsVocab = CountVectorizedData.columns[:-1]
word2Vec = gensim.models.KeyedVectors.load_word2vec_format(
    '/Users/dengjiaming/Downloads/GoogleNews-vectors-negative300.bin', binary=True)
word2Vec_data = compressWord(reviews['comments'])
```

2) Secondly, according to the data analysis, the listing.csv file contains information about the hotel. The guest's rating of the hotel is inextricably linked to the conditions of the hotel itself, the character and qualities of the owner, the character and qualities of the neighbours, the location and other information. This information can therefore be used to form a feature vector to predict the ratings. So finally I find 21 features including owner related features like host_identity_verified, host_has_profile_pic, hotel related like number_of_reviews, availability_30, bedrooms, accommodate and so on.

In the preprocessing data, first of all, I need to fill in the blank data with 0. Then replace the f and t of host_is_superhost and host_has_profile_pic, host_identity_verified with 0 and 1 and remove the percent sign in host_acceptance_rate. Finally, compare the four types of room_type according to the number of reservations, and replace them with 4, 3, 2, and 1 respectively to obtain the final feature vector.



```
listings['host_is_superhost'] = listings['host_is_superhost'].str.replace('t', '1')
listings['host_is_superhost'] = listings['host_is_superhost'].str.replace('f', '0')
listings['host_is_superhost'] = listings['host_is_superhost'].replace(np.nan, 0)

listings['host_acceptance_rate'] = listings['host_acceptance_rate'].str.replace('%', '').astype(float)
listings['host_has_profile_pic'] = listings['host_has_profile_pic'].str.replace('t', '1').replace('f', '0').astype(float)
listings['host_identity_verified'] = listings['host_identity_verified'].str.replace('t', '1').replace('f', '0').astype(float)
listings['room_type'] = listings['room_type'].str.replace('Entire home/apt', '4').replace('Private room', '3').replace('Shared room', '2')
listings['instant_bookable'] = listings['instant_bookable'].str.replace('t','1').replace('f', 0).astype(float)
```

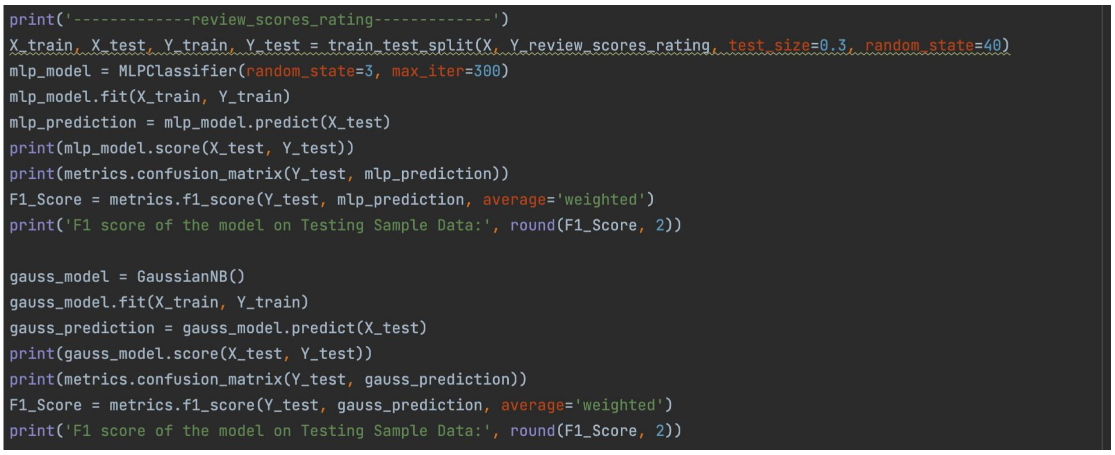
i (c) Mechine Learning Method Selection and Evaluation

1) For the machine learning method I chose the Gaussian Naive Bayes algorithm and the MLP algorithm. The reasons are metioned below.

A multilayer perceptron is a neural network consisting of fully connected layers containing at least one hidden layer, and the output of each hidden layer is transformed by an activation function, which is well suited to the many features in this problem. So I chose this algorithm.

Gaussian Naive Bayes algorithm is suitable for continuous variables, which assumes that each feature x_i is subject to a normal distribution under each category y, and the algorithm internally uses the probability density function of the normal distribution to calculate the probability. In this dataset, Gaussian Naive Bayes will not perform well and this can be compared with the well-performed MLP algorithm so I chose this algorithm.

The implementation idea is to first divide the data into a training set and a test set, with 70% of the training set and 30% of the test set, initialize the MLP and Gaussian Naive Bayes model, and use the training set to train the model. At the end, a combined report of the prediction score, confusion matrix and f1 score.



```
print('------------review_scores_rating------------')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_review_scores_rating, test_size=0.3, random_state=40)
mlp_model = MLPClassifier(random_state=3, max_iter=300)
mlp_model.fit(X_train, Y_train)
mlp_prediction = mlp_model.predict(X_test)
print(mlp_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, mlp_prediction))
F1_Score = metrics.f1_score(Y_test, mlp_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

gauss_model = GaussianNB()
gauss_model.fit(X_train, Y_train)
gauss_prediction = gauss_model.predict(X_test)
print(gauss_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, gauss_prediction))
F1_Score = metrics.f1_score(Y_test, gauss_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))
```

2) Evalution

1. prediction score

	scores_rating	scores_accuracy	scores_cleanliness	scores_checkin	scores_communicatio
MLP	0.39	0.45	0.38	0.50	0.53
Guass Bayes	0.04	0.04	0.03	0.04	0.03
	scores_location	scores_value			
MLP	0.38	0.34			
Guass Bayes	0.04	0.03			

2. f1 score

	scores_rating	scores_accuracy	scores_cleanliness	scores_checkin
MLP	0.33	0.38	0.31	0.43
Guass Bayes	0.06	0.07	0.6	0.07
	scores_communication	scores_location	score_value	
MLP	0.46	0.33	0.28	
Guass Bayes	0.06	0.07	0.06	

3. analysis
In the model selection, I analyzed the reason why the performance of the Gaussian Bayesian algorithm is much worse than that of the MLP algorithm. Through the prediction score and f1 score, we can easily compare and conclude that the performance of the MLP algorithm is much better. The scores obtained by the MLP algorithm are often above 0.33, while those obtained by the Gaussian Bayesian algorithm are around 0.03. Because there are so many features, and the correlation between the features and the target score is not particularly large.

ii)
1. Example1: If the temperature is below 0 degrees, students will think it is cold and get up after 12 o'clock, otherwise students think it is hot and get up before 12 o'clock. Assuming that the indoor and outdoor temperatures are the same without air conditioning, if the air conditioning is on, the indoor temperature will increase by 10 degrees. I want to predict whether the students get up before 12 o’clock or not based on the following two features: outdoor temperature and whether the air conditioner is on.
The reason is that the target label has no linear correlation with the features. In such cases, logistic regression can’t predict targets with good accuracy.
Example2: Many modern companies and schools are using face recognition technology, but using logistic regression predictions are not accurate.
This is because face data is linearly indistinguishable and logistic regression requires the construction of decision boundaries to classify it.
2.
1) The MLP has a back-propagation feature. The error after the output is used to estimate the error in the direct leading layer of the output layer, and then this error is used to estimate the error in the layer further ahead, and so on back-propagating layer by layer, the error estimates for all the other layers are obtained. The MLP thus uses the output error to improve the prediction accuracy of the model, whereas the kNN does not have this property.
2) The kNN requires the choice of the value of k. The choice of k has a large impact on the prediction results and there is no theoretical choice of the optimal k-value size, often the optimal k-value choice is obtained in combination with k-fold cross-validation. However the MLP does not need to choose parameters, and therefore the performance of the MLP is more stable than that of the kNN.
3) The MLP is fully connected and therefore requires a large number of parameters such as network topology, initial values for weights and thresholds, long training time and good classification results. kNN is simple in principle, short training time, but not as good as MLP, especially when the sample is not balanced, the prediction bias is larger. For example, there are fewer samples in one category and more samples in other categories.
4) A common advantage of MLP and kNN is their robustness to noise and fault tolerance, with no requirement for linear correlation of the data.
3. K-fold cross validation randomly splits the dataset into K copies at a time, one

copy is kept for validating the model and the other K-1 copies are used for training. Because of the random feature of each selection, the data in a dataset is used repeatedly and each piece of data can be trained, which corresponds to obtaining as much valid information as possible from a limited amount of data and reduces the effect of noise in the data. When the amount of data is not large enough, it is easy to overfit the model if all the data is used to train the model. By dividing the data through K-fold cross validation and integrating the evaluation results, we can effectively reduce the variance in model selection. In other words, we expect the model to perform well on multiple subsets of the training set, rather than on the entire training data set alone. Therefore it evaluates the generalisation performance of a given algorithm after training on a given dataset and can reduce overfitting to a certain extent.
Let us first think of two edge cases.
1) The edge case where no cross-validation is used at all is when K = 1. At this point, all of the data is used for training, the model is prone to overfitting and therefore prone to being low bias and high variance.
2) The leave-one-out method is the other edge case of the K-fold, i.e. K = n. As the value of K increases, the variance in the evaluation of a single model gradually increases while the bias decreases. However, from the overall model perspective, the bias increases while the variance decreases.
So when the value of K wanders between 1 and n, it can be interpreted as a compromise between variance and bias. Taking K=10 as an example, at training time we have only 90% of the training data in the training set. In contrast to the case where no cross-validation is used, this makes the bias go up, but for the average of the results it reduces the model variance, and the final result is better or not depending on the degree of variation between the two. Therefore, as a rule of thumb, K = 5 or 10 performs well in most cases.
4. In time series analysis, the lagged values of a series can be used as features to help predict the future values of the series. For example, consider a time series of the number of people entering and leaving Trinity University Dublin's West Gate. To use this time series to predict future numbers, we can construct an eigenmatrix using the lagged values of the temperature series.
Assume that the sequence of the number of people entering and exiting the West Gate is as follows.

Day1	Day2	Day3	Day4	Day5	Day6	Day7
2013	2942	2502	2949	5021	3521	4452

We can create an eigenmatrix containing the lagged values of the sequence of people entering and leaving Westgate as follows.

Day	Number (n)	Number(n-1)	Number(n-2)
1	2013	NA	NA
2	2943	2013	NA
3	2502	2943	2013
4	2949	2502	2943
5	5021	2949	2502

6	3521	5021	2949
7	4452	3521	5021

In this example, the lagged values of the sequence of the number of people entering and leaving the West Gate are used as features to help predict the number of people entering and leaving the West Gate in the future. For example, the number of people entering and exiting the west gate on day 5 is 5021. the feature matrix shows that the number of people entering and exiting the west gate on the previous day was 2949 and the day before that was 2502. this information can be used to construct a model to predict the number of people entering and exiting the west gate on day 6.

Lagged values can be useful features for time series data as they capture some of the time dependence in the data. For example, in the series above, the number of people entering and exiting the west gate on day 5 may be influenced by the number of people entering and exiting the west gate on days 4 and 3. By using the lagged values as features, we can capture this time dependence in the model.

Appendix

```
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn import metrics
listings = pd.read_csv('data/listings.csv')
listings = listings.replace(np.nan, 0)
features = ['host_listings_count', 'host_total_listings_count', 'host_has_profile_pic', 'host_identity_verified', 'room_type', 'accommodates', 'bedrooms', 'beds', 'reviews_per_month', 'calculated_host_listings_count_shared_rooms', 'calculated_host_listings_count_private_rooms', 'calculated_host_listings_count_entire_homes', 'calculated_host_listings_count', 'instant_bookable', 'availability_30', 'availability_60', 'availability_90', 'availability_365', 'number_of_reviews', 'number_of_reviews_ltm', 'number_of_reviews_l30d']
print(len(features))
listings['host_is_superhost'] = listings['host_is_superhost'].str.replace('t', '1')
listings['host_is_superhost'] = listings['host_is_superhost'].str.replace('f', '0')
listings['host_is_superhost'] = listings['host_is_superhost'].replace(np.nan, 0)

listings['host_acceptance_rate'] = listings['host_acceptance_rate'].str.replace('%', '').astype(float)
listings['host_has_profile_pic'] = listings['host_has_profile_pic'].str.replace('t', '1').replace('f', '0').astype(float)
```

```
listings['host_identity_verified'] = listings['host_identity_verified'].str.replace('t', '1').replace('f', '0').astype(float)
listings['room_type'] = listings['room_type'].str.replace('Entire home/apt', '4').replace('Private room', '3').replace('Shared room', '2').replace('Hotel room', '1').astype(float)
listings['instant_bookable'] = listings['instant_bookable'].str.replace('t','1').replace('f', 0).astype(float)

X = listings[features].astype(int)

Y_review_scores_rating = listings['review_scores_rating'].astype(float)
Y_review_scores_rating = Y_review_scores_rating * 100
Y_review_scores_rating = Y_review_scores_rating.astype(int)

Y_review_scores_accuracy = listings['review_scores_accuracy'].astype(float)
Y_review_scores_accuracy = Y_review_scores_accuracy * 100
Y_review_scores_accuracy = Y_review_scores_accuracy.astype(int)

Y_review_scores_cleanliness = listings['review_scores_cleanliness'].astype(float)
Y_review_scores_cleanliness = Y_review_scores_cleanliness * 100
Y_review_scores_cleanliness = Y_review_scores_cleanliness.astype(int)

Y_review_scores_checkin = listings['review_scores_checkin'].astype(float)
Y_review_scores_checkin = Y_review_scores_checkin * 100
Y_review_scores_checkin = Y_review_scores_checkin.astype(int)

Y_review_scores_communication = listings['review_scores_communication'].astype(float)
Y_review_scores_communication = Y_review_scores_communication * 100
Y_review_scores_communication = Y_review_scores_communication.astype(int)

Y_review_scores_location = listings['review_scores_location'].astype(float)
Y_review_scores_location = Y_review_scores_location * 100
Y_review_scores_location = Y_review_scores_location.astype(int)

Y_review_scores_value = listings['review_scores_value'].astype(float)
Y_review_scores_value = Y_review_scores_value * 100
Y_review_scores_value = Y_review_scores_value.astype(int)

Y_review_scores_rating = np.array(Y_review_scores_rating)
Y_review_scores_accuracy = np.array(Y_review_scores_accuracy)
```

```python
Y_review_scores_cleanliness = np.array(Y_review_scores_cleanliness)
Y_review_scores_checkin = np.array(Y_review_scores_checkin)
Y_review_scores_communication = np.array(Y_review_scores_communication)
Y_review_scores_location = np.array(Y_review_scores_location)
Y_review_scores_value = np.array(Y_review_scores_value)

print('-------------review_scores_rating-------------')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_review_scores_rating, test_size=0.3, random_state=40)
mlp_model = MLPClassifier(random_state=3, max_iter=300)
mlp_model.fit(X_train, Y_train)
mlp_prediction = mlp_model.predict(X_test)
print(mlp_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, mlp_prediction))
F1_Score = metrics.f1_score(Y_test, mlp_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

gauss_model = GaussianNB()
gauss_model.fit(X_train, Y_train)
gauss_prediction = gauss_model.predict(X_test)
print(gauss_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, gauss_prediction))
F1_Score = metrics.f1_score(Y_test, gauss_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))


print('-------------review_scores_accuracy-------------')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_review_scores_accuracy, test_size=0.3, random_state=40)
mlp_model = MLPClassifier(random_state=3, max_iter=300)
mlp_model.fit(X_train, Y_train)
mlp_prediction = mlp_model.predict(X_test)
print(mlp_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, mlp_prediction))
F1_Score = metrics.f1_score(Y_test, mlp_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

gauss_model = GaussianNB()
```

```python
gauss_model.fit(X_train, Y_train)
gauss_prediction = gauss_model.predict(X_test)
print(gauss_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, gauss_prediction))
F1_Score = metrics.f1_score(Y_test, gauss_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))


print('-------------review_scores_cleanliness-------------')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_review_scores_cleanliness, test_size=0.3, random_state=40)
mlp_model = MLPClassifier(random_state=3, max_iter=300)
mlp_model.fit(X_train, Y_train)
mlp_prediction = mlp_model.predict(X_test)
print(mlp_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, mlp_prediction))
F1_Score = metrics.f1_score(Y_test, mlp_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

gauss_model = GaussianNB()
gauss_model.fit(X_train, Y_train)
gauss_prediction = gauss_model.predict(X_test)
print(gauss_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, gauss_prediction))
F1_Score = metrics.f1_score(Y_test, gauss_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

print('-------------review_scores_checkin-------------')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_review_scores_checkin, test_size=0.3, random_state=40)
mlp_model = MLPClassifier(random_state=3, max_iter=300)
mlp_model.fit(X_train, Y_train)
mlp_prediction = mlp_model.predict(X_test)
print(mlp_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, mlp_prediction))
F1_Score = metrics.f1_score(Y_test, mlp_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

```

```python
gauss_model = GaussianNB()
gauss_model.fit(X_train, Y_train)
gauss_prediction = gauss_model.predict(X_test)
print(gauss_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, gauss_prediction))
F1_Score = metrics.f1_score(Y_test, gauss_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

print('-------------review_scores_communication-------------')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_review_scores_communication, test_size=0.3, random_state=40)
mlp_model = MLPClassifier(random_state=3, max_iter=300)
mlp_model.fit(X_train, Y_train)
mlp_prediction = mlp_model.predict(X_test)
print(mlp_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, mlp_prediction))
F1_Score = metrics.f1_score(Y_test, mlp_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

gauss_model = GaussianNB()
gauss_model.fit(X_train, Y_train)
gauss_prediction = gauss_model.predict(X_test)
print(gauss_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, gauss_prediction))
F1_Score = metrics.f1_score(Y_test, gauss_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

print('-------------review_scores_location-------------')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_review_scores_location, test_size=0.3, random_state=40)
mlp_model = MLPClassifier(random_state=3, max_iter=300)
mlp_model.fit(X_train, Y_train)
mlp_prediction = mlp_model.predict(X_test)
print(mlp_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, mlp_prediction))
F1_Score = metrics.f1_score(Y_test, mlp_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

```

```
gauss_model = GaussianNB()
gauss_model.fit(X_train, Y_train)
gauss_prediction = gauss_model.predict(X_test)
print(gauss_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, gauss_prediction))
F1_Score = metrics.f1_score(Y_test, gauss_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

print('-------------review_scores_value-------------')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_review_scores_value, test_size=0.3, random_state=40)
mlp_model = MLPClassifier(random_state=3, max_iter=300)
mlp_model.fit(X_train, Y_train)
mlp_prediction = mlp_model.predict(X_test)
print(mlp_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, mlp_prediction))
F1_Score = metrics.f1_score(Y_test, mlp_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

gauss_model = GaussianNB()
gauss_model.fit(X_train, Y_train)
gauss_prediction = gauss_model.predict(X_test)
print(gauss_model.score(X_test, Y_test))
print(metrics.confusion_matrix(Y_test, gauss_prediction))
F1_Score = metrics.f1_score(Y_test, gauss_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))
```

```
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import WhitespaceTokenizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer
import emoji
from googletrans import Translator
from langdetect import detect
```

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.neural_network import MLPClassifier
import warnings
warnings.filterwarnings('ignore')
# stop words
# emoji
# translate
# TODO listing features, wordVec2, logisticRegression, kNN

def get_comment_via_id(listing_id):
    data = pd.read_csv('data/reviews.csv')
    comments = []
    for i, row in data.iterrows():
        if row['listing_id'] == listing_id:
            comments.append(emoji.demojize(row['comments']))
    return comments


def get_scores_via_id(listing_id):
    data = pd.read_csv('data/listings.csv')
    scores = []
    for i, row in data.iterrows():
        if row['listing_id'] == listing_id:
            scores.append(row['review_scores_rating'])
            scores.append(row['review_scores_accuracy'])
            scores.append(row['review_scores_cleanliness'])
            scores.append(row['review_scores_checkin'])
            scores.append(row['review_scores_communication'])
            scores.append(row['review_scores_location'])
            scores.append(row['review_scores_value'])
            break
    return scores


if __name__ == '__main__':
    listings = pd.read_csv('data/listings.csv')
    ids = listings['listing_id'].unique()
```

```
words = stopwords.words('english')
for w in ['!', ',', '.', '?', '-s', '-ly', '</s>', 's', '<', '>', 'br/', 'in']:
    words.append(w)
words = set(words)
stemmer = SnowballStemmer('english')
translator = Translator()
for ID in ids:
    comments = get_comment_via_id(ID)
    scores = get_scores_via_id(ID)
    for i in range(len(comments)):
        try:
            if detect(comments[i]) != 'en':
                translator = Translator()
                comments[i] = translator.translate(comments[i]).text
                # print("True")
        except:
            print("False")
    vectorizer = CountVectorizer(stop_words=words)
    X = vectorizer.fit_transform(comments)
    X = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
    print(X.shape)
    print(X.head())
    X = X.toarray()
    Y = []
    for i in range(len(X)):
        Y.append(scores)
    Y = np.array(Y)
    X = np.array(X)
    print(X)
    print(X.shape)
    print(Y)
    print(Y.shape)
    print(vectorizer.get_feature_names())
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=40)
    model_overall = MLPClassifier(random_state=3, max_iter=300)
    model_accuracy = MLPClassifier(random_state=3, max_iter=300)
    model_cleanliness = MLPClassifier(random_state=3, max_iter=300)
    model_communication = MLPClassifier(random_state=3, max_iter=300)
    model_value = MLPClassifier(random_state=3, max_iter=300)
    model_checkin = MLPClassifier(random_state=3, max_iter=300)
    model_location = MLPClassifier(random_state=3, max_iter=300)

    model_overall.fit(X_train, Y_train[:, 0])
    print(model_overall.score(X_test, Y_test[:, 0]))

    model_accuracy.fit(X_train, Y_train[:, 1])
    print(model_accuracy.score(X_test, Y_test[:, 1]))

    model_cleanliness.fit(X_train, Y_train[:, 2])
    print(model_cleanliness.score(X_test, Y_test[:, 2]))

        model_checkin.fit(X_train, Y_train[:, 3])
        print(model_checkin.score(X_test, Y_test[:, 3]))

        model_communication.fit(X_train, Y_train[:, 4])
        print(model_communication.score(X_test, Y_test[:, 4]))

        model_location.fit(X_train, Y_train[:, 5])
        print(model_location.score(X_test, Y_test[:, 5]))

        model_value.fit(X_train, Y_train[:, 6])
        print(model_value.score(X_test, Y_test[:, 6]))
import gensim.models
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import WhitespaceTokenizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer
import emoji
from googletrans import Translator
from langdetect import detect
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
from sklearn.linear_model import LogisticRegression, LinearRegression
from gensim.models import Word2Vec
from sklearn.naive_bayes import GaussianNB

import matplotlib

matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings('ignore')


def get_comment_via_id(listing_id):
    data = pd.read_csv('data/reviews.csv')
    comments = []
    for i, row in data.iterrows():
        if row['listing_id'] == listing_id:
            comments.append(emoji.demojize(row['comments']))
    return comments


def get_scores_via_id(listing_id):
    data = pd.read_csv('data/listings.csv')
    scores = []
    for i, row in data.iterrows():
        if row['id'] == listing_id:
            scores.append(row['review_scores_rating'])
            scores.append(row['review_scores_accuracy'])
            scores.append(row['review_scores_cleanliness'])
            scores.append(row['review_scores_checkin'])
            scores.append(row['review_scores_communication'])
            scores.append(row['review_scores_location'])
            scores.append(row['review_scores_value'])
            break
    return scores


def get_all_comments():
    data = pd.read_csv('data/reviews.csv')
    comments = []
    for i, row in data.iterrows():
        comments.append(row['comments'])
    return comments


def get_word_vector_via_id(listing_id):
    comments = get_comment_via_id(listing_id)


def draw_first():
    reviews = pd.read_csv('data/reviews.csv')
    listings = pd.read_csv('data/listings.csv')
    print(reviews.groupby('listing_id').size())
    ax = reviews.groupby('listing_id').size().plot(kind='bar')
    ax.set(xlabel='lising_id', ylabel='nunber of comments')
    plt.show()


def compressWord(InputData):
    X = vectorizer.transform(InputData)
    data = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
    word2Vec_data = pd.DataFrame()
    for i in range(data.shape[0]):
        sentence = np.zeros(300)
        for word in WordsVocab[data.iloc[i, :]]:
            if word in word2Vec.key_to_index.keys():
                sentence = sentence + word2Vec[word]
        word2Vec_data = word2Vec_data.append(pd.DataFrame([sentence]))
    return word2Vec_data


if __name__ == '__main__':
```

```python
reviews = pd.read_csv('data/reviews.csv')
listings = pd.read_csv('data/listings.csv')
print(reviews.shape)
print(listings.shape)
reviews = pd.merge(reviews, listings)
print(reviews.shape)
corpus = reviews['comments'].values

vectorizer = CountVectorizer(stop_words='english')

    X = vectorizer.fit_transform(corpus.astype('U'))
    CountVectorizedData = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
    CountVectorizedData['review_scores_rating'] = reviews['review_scores_rating']
    CountVectorizedData['review_scores_accuracy'] = reviews['review_scores_accuracy']
    CountVectorizedData['review_scores_cleanliness'] = reviews['review_scores_cleanliness']
    CountVectorizedData['review_scores_checkin'] = reviews['review_scores_checkin']
    CountVectorizedData['review_scores_communication'] = reviews['review_scores_communication']
    CountVectorizedData['review_scores_location'] = reviews['review_scores_location']
    CountVectorizedData['review_scores_value'] = reviews['review_scores_value']
    print(CountVectorizedData.shape)
    print(CountVectorizedData.head())
    WordsVocab = CountVectorizedData.columns[:-1]
    word2Vec = gensim.models.KeyedVectors.load_word2vec_format(
        '/Users/dengjiaming/Downloads/GoogleNews-vectors-negative300.bin', binary=True)
    word2Vec_data = compressWord(reviews['comments'])
    print(word2Vec_data.shape)
    word2Vec_data.reset_index(inplace=True, drop=True)

    word2Vec_data['review_scores_rating'] = CountVectorizedData['review_scores_rating']
    word2Vec_data['review_scores_accuracy'] = CountVectorizedData['review_scores_accuracy']
    word2Vec_data['review_scores_cleanliness'] = CountVectorizedData['review_scores_cleanliness']
    word2Vec_data['review_scores_checkin'] = CountVectorizedData['review_scores_checkin']
    word2Vec_data['review_scores_communication'] = CountVectorizedData['review_scores_communication']
    word2Vec_data['review_scores_location'] = CountVectorizedData['review_scores_location']
    word2Vec_data['review_scores_value'] = CountVectorizedData['review_scores_value']
```

```

X = word2Vec_data[word2Vec_data.columns[:-1]].values
Y = word2Vec_data[word2Vec_data.columns[-1]].values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=428)

mlp_model = MLPClassifier(random_state=3, max_iter=300)
mlp_model.fit(X_train, Y_train)
mlp_prediction = mlp_model.predict(X_test)
print(metrics.classification_report(Y_test, mlp_prediction))
print(metrics.confusion_matrix(Y_test, mlp_prediction))
F1_Score = metrics.f1_score(Y_test, mlp_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))

gauss_model = GaussianNB()
gauss_model.fit(X_train, Y_train)
gauss_prediction = gauss_model.predict(X_test)
print(metrics.classification_report(Y_test, gauss_prediction))
print(metrics.confusion_matrix(Y_test, gauss_prediction))
F1_Score = metrics.f1_score(Y_test, gauss_prediction, average='weighted')
print('F1 score of the model on Testing Sample Data:', round(F1_Score, 2))
```