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: Pallavit Aggarwal

Date: 4th January 2023

ML - Final Assignment 2022

Pallavit Aggarwal – Ms IS
ID: 22333721

Q1.

Data Preparation Phase

A quick review of the listings.csv and reviews.csv in excel, and visually parsing through the data reveals a lot about the kind of data we will be dealing with.

The reviews file has 6202 unique property ids, whereas the listings file has 7566 ids. But when the two files are merged, using pandas.merge, on the 'id' field using an inner join, just 5180 common and unique ids are obtained. The merged file, has 2,33,978 records with rows in duplicates of property ids and listing data but unique comments of reviews for each id.

Using, langdetect (https://pypi.org/project/langdetect/) which is Google's language detection library, the language of the review is detected. There are 43 languages detected out of which, 2,02,524 reviews are in English. The next biggest languages in reviews is French with 11,867 detected rows, and German and Spanish follow next with 5969 and 5066 records respectively. There is also a "0" which has been detected with 564 rows. This means that the library was unable to decipher a language. These could be emojis, or garbage values. For the sake of simplicity, I consider only "EN" language reviews.

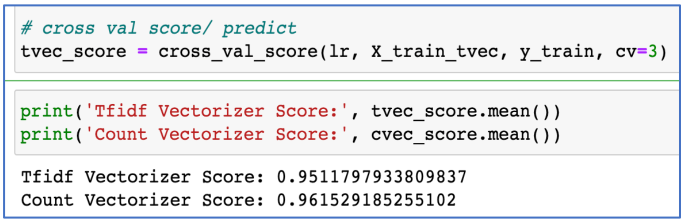
After selecting just "EN" reviews and rows, using regex, I remove anything that is not a number or a letter. I also remove <br> quotes and covert the reviews into lowercase. Next, apostrophe words are converted into whole words like "who's" to "who is", 'n't' is replaced with 'not' and using nltk.corpus stop words are removed.

Then NLTK library's VADER utility is used, which takes words and provides a sentiment score. I played around with *SentimentIntensityAnalyzer* to understand the scoring and the values. The compound scoring of the polarity scores is from -1 to +1 and is pretty consistent with the overall mood of the different reviews I selected at random and checked.

Feature Engineering & Data Insights

Once the above file is ready and saved as a csv file, the next phase for feature selection and feature engineering begins.

I divide the 'sentiment' column into bins of values 0,1,2 with values from -1 to 0, 0 to 0.5, and 0.5 to 1. Now using these sentiment bins as Y values, I generate the 'Comments' Feature vectors using CountVectorizer and TfidfVectorizer. Using logistic regression on the TFIDF and CountVectorizer output as feature columns and sentiment-bins are labels, I calculate the mean cross_val_score of both and CountVectorizer with 0.96 accuracy performs better.



```
# cross val score/ predict
tvec_score = cross_val_score(lr, X_train_tvec, y_train, cv=3)
```

```
print('Tfidf Vectorizer Score:', tvec_score.mean())
print('Count Vectorizer Score:', cvec_score.mean())
```

```
Tfidf Vectorizer Score: 0.9511797933809837
Count Vectorizer Score: 0.961529185255102
```

Fig. 1 Cross Val using LR on CountVectorizer features and sentiment labels

I run the above cross validation for different max_features and max_df values inside CountVectorizer to determine the best values.

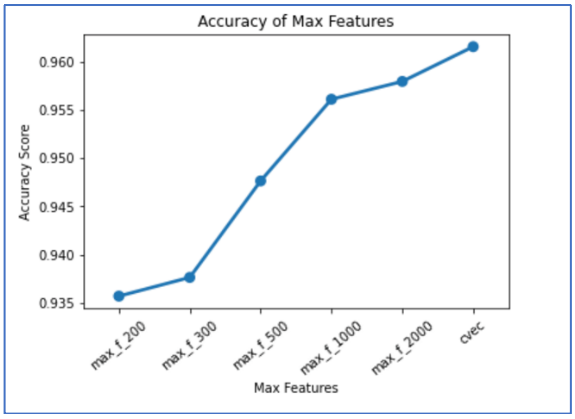


Fig 2. LR accuracy to choose Max Features

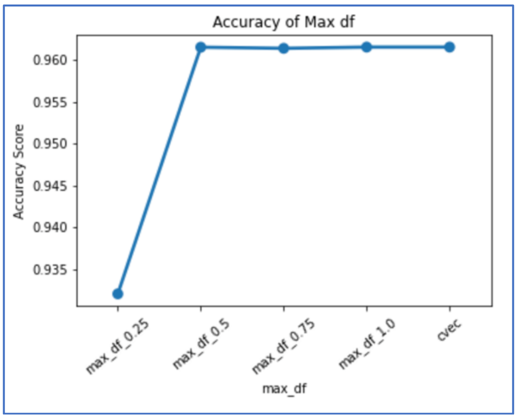
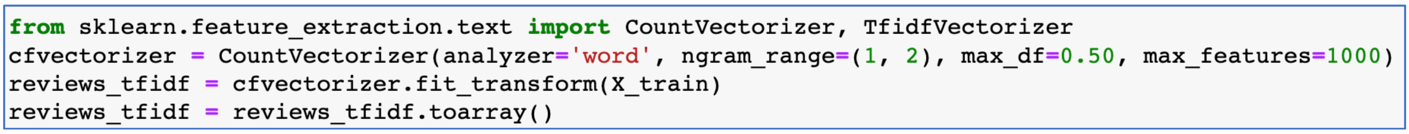


Fig 3.LR accuracy to choose Max DF value

The final CountVectorizer method has an Analyzer as "word", ngram_range as (1,2) to select two words distances as well in case of "not good" etc., max_df "0.5" as per above evaluation and max_features are 1000 so as not to overwhelm the model later on with a lot of features.



```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
cfvectorizer = CountVectorizer(analyzer='word', ngram_range=(1, 2), max_df=0.50, max_features=1000)
reviews_tfidf = cfvectorizer.fit_transform(X_train)
reviews_tfidf = reviews_tfidf.toarray()
```

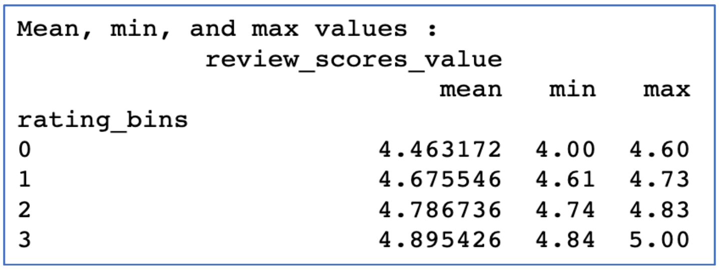
This gives us 1000 CountVector feature vectors generated from the comments.

Now, some data pre-processing is done on the columns in listings.csv. host_is_superhost is converted from t/f to 0/1. Numerical values from bathrooms_text and price columns are extracted. "Entire home/apt" in host_is_superhost is converted to 1, and rows with value "Within a day" in id column are completely removed.

The different values in property_type are replaced with broader categories like "House", "Room" and "Other" and mapped to 0,1,2 categorical values. Same with room type where, values are mapped as follows "Private room": 0, "Entire home/apt": 1, "Shared room" : 2, "Hotel room" : 3.

Also, total amenities listed by an owner are counted, and the number of unique amenities is replaced by the list array of strings of amenitites.

Now, only review_score_values that are between >=4 and <=5 are selected. Now, rating_bin as a column is created which divides the review_score_values into 4 equally probabilistic bins.



```
Mean, min, and max values :
             review_scores_value
                            mean   min   max
rating_bins
0                       4.463172  4.00  4.60
1                       4.675546  4.61  4.73
2                       4.786736  4.74  4.83
3                       4.895426  4.84  5.00
```

Fig 4. Mean, Min and Max of the 4 bins for review_scores

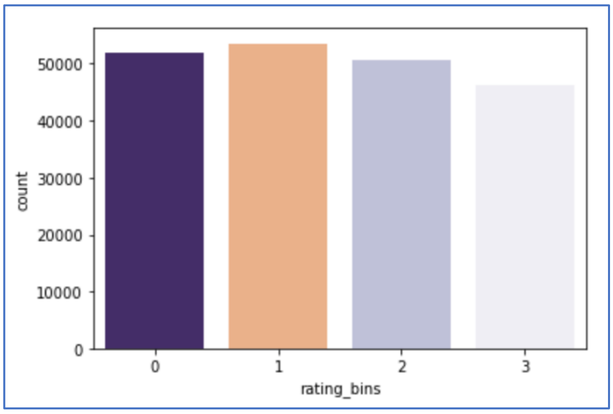


Fig 5. Count of the reviews in 4 bins, are almost equal

Now comparing, different features in the listings.csv with review_scores_value to figure out if there's any pattern we can find out.

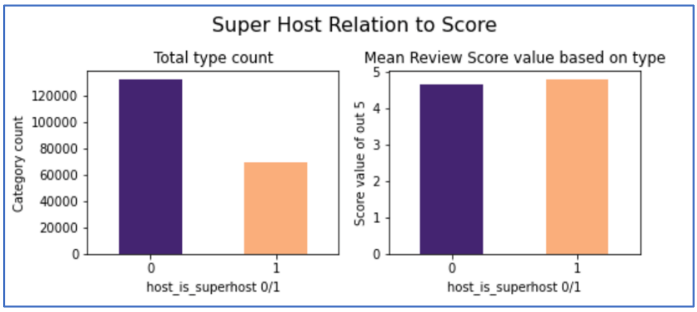


Fig 6. superhost values and review scores

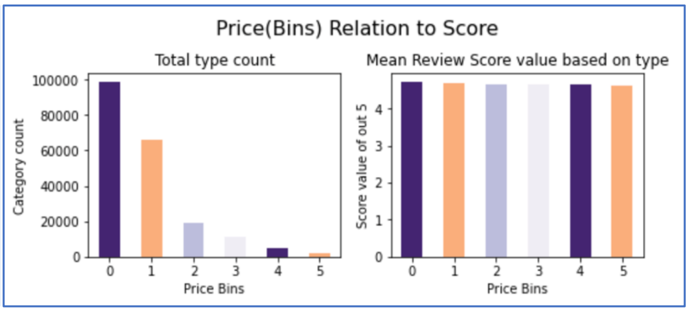


Fig 7. Price values in 6 different bins and review scores

Plotting host_is_superhost with the review scores value gives us the approximate same mean values for the review score, with a slightly higher mean for superhost. The price values are binned between the following 0, 100, 200, 300, 400, 600, 1000 and review scores plotted.

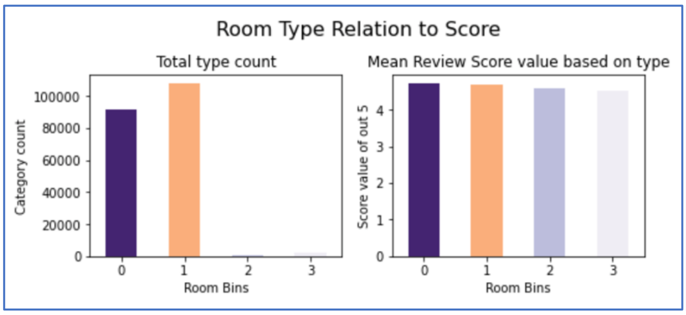


Fig 8. Room type bins vs review scores

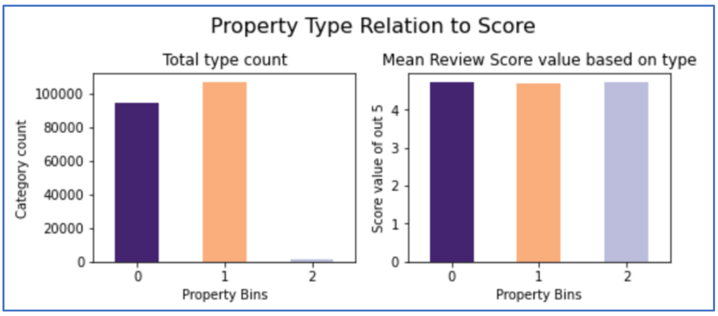


Fig 9. Property type bins vs review scores

In all the cases, the mean values of review scores seem extremely close to each other against changing features.

As far as features are considered, we have 1000 tfidf features which can predict the correct sentiment class with a 95% accuracy using Logistic regression, but sentiment is not satisfactory. Sentiment can play a huge role in defining the final review_scores value, but more features would need to be extracted or be present in the model to improve its accuracy.

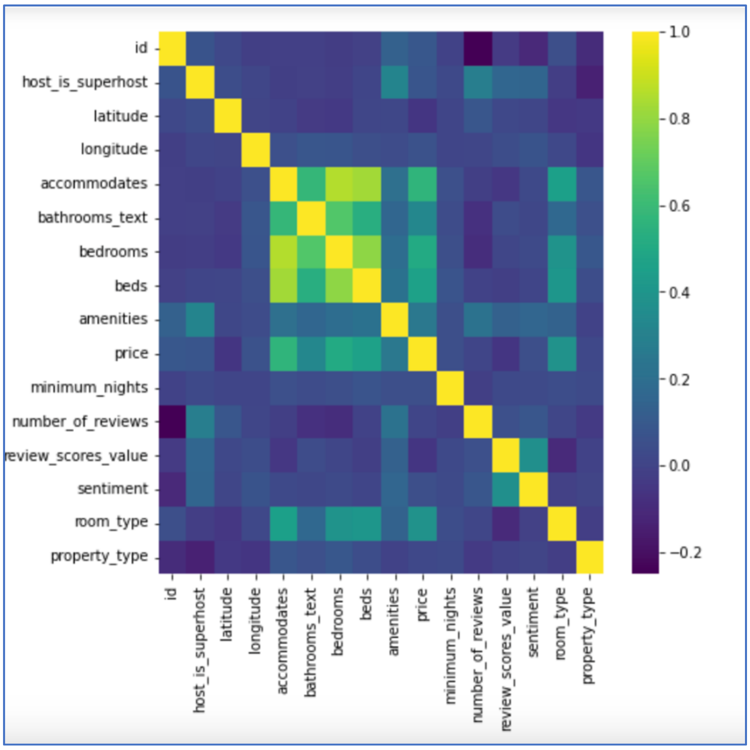


Fig 10. Correlation matrix between the listing columns and review_scores

I add more features, to improve the accuracy by combining multiple features in the listing column together. I discuss this in the next section.

Modelling & Evaluation

The problem statement at hand is to create a model that predicts the review scores. There are total 4949 rows, after cleaning and grouping the columns with column "id". The features are listing columns 'id','host_is_superhost','latitude','longitude','accommodates','bathrooms_text','bedrooms','beds','amenities','price','minimum_nights', 'number_of_reviews','review_scores_value','sentiment', 'room_type', 'property_type' and 1000 comment vectors.

I implemented multiple models, and for the sake of this report, I would like to approach this problem in two ways.

First would be regression, where in the model would predict the score values obtained by using different features. Second would be classification, where in the review scores are divided in bins, and classification model identifies the correct class based on the features.

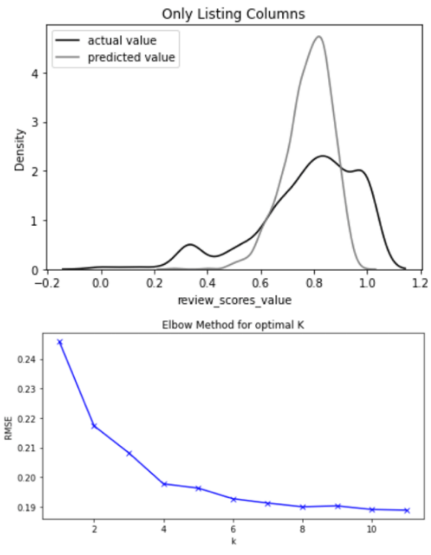
Regression:

In case of Regression, **KNeighborsRegressor** is used. The KNN algorithm uses 'feature similarity' to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set.

Since regularized models penalize variables, we need to standardize them so they are all penalized at the same rate. This is done by applying a minmax scaler on the data with feature range 0,1.

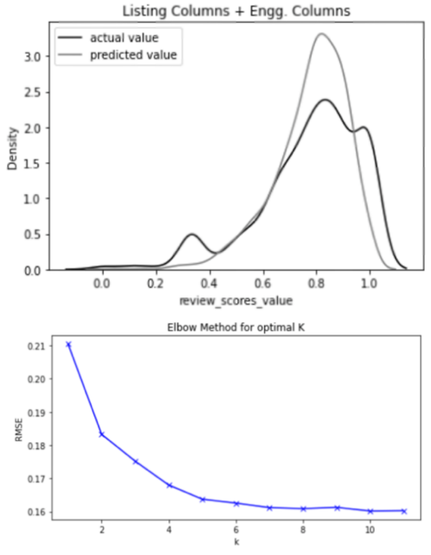
With KNN, I explored these 3 cases first –

1. With just listing columns
2. Listing Columns + Additional Review and Sentiment modifications which include 3 additional columns with the following values: **rvw_sentm** = number of reviews * sentiment ; **tot_price** = minimum nights * price and finally **sentm_rvw** = sentiment * review scores. To manage highly correlated fields, I remove 'number_of_reviews','minimum_nights', 'sentiment' from the dataset. review_scores will be removed as they will be prediction values.
3. Listing Columns + 1000 comment CountVectors.



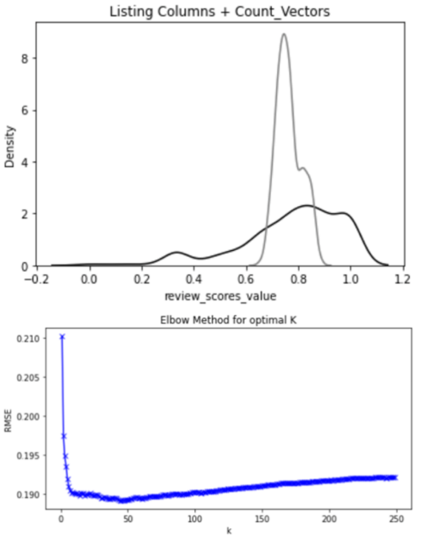
```
At k=10
Mean Squared Error =  0.03574272989915507
Root Mean Squared Error =  0.1890574777657712
R2 Score =  0.0784252971201519
```

Fig 11. Case 1



```
At k=10
Mean Squared Error =  0.025650100431720054
Root Mean Squared Error =  0.16015648732324286
R2 Score =  0.31813859324194804
```

Fig 12. Case 2



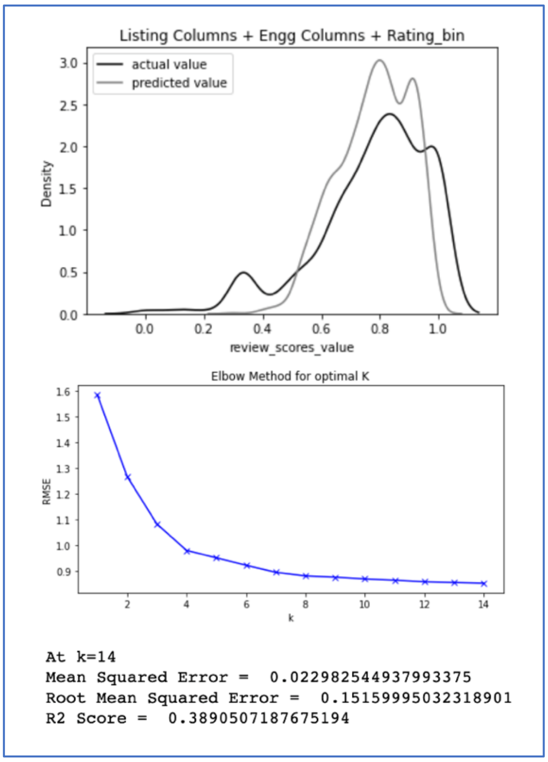
```
At k=43
Mean Squared Error =  0.03580322694496539
Root Mean Squared Error =  0.18921740655913608
R2 Score =  0.0768654681094676
```

Fig 12. Case 3

This is intriguing because engineered columns, have some data of review scores hidden in them. Thereby, they are providing a much better correlation and pattern for predicting the final review scores. Adding additional CountVec columns, doesn't help the KNN model as the Euclidian distance between points is not giving useful metrics with 1000+ features.

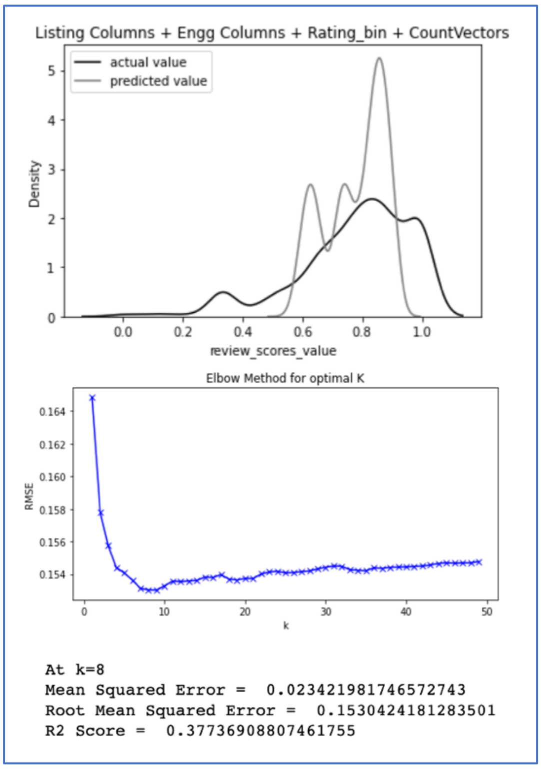
Since engineered columns provide extra information of the review score without revealing the review score, I add an additional column of **rating_bins** which divide **sentm_rvw** into equally distributed bins of 4 categories 0,1,2,4.

4. Listing Columns + Engg Columns + Rating_Bin
5. Listing Columns + Engg Columns + Rating_Bin + 1000 comment CountVectors



```
At k=14
Mean Squared Error =  0.022982544937993375
Root Mean Squared Error =  0.15159995032318901
R2 Score =  0.3890507187675194
```

Fig 13. Case 4



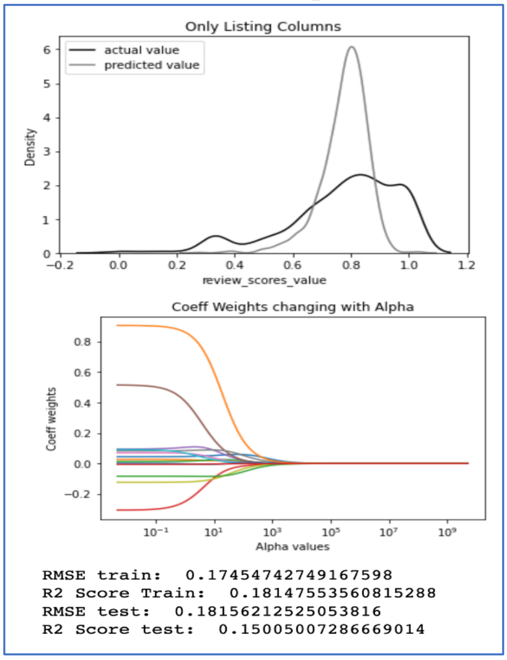
```
At k=8
Mean Squared Error =  0.023421981746572743
Root Mean Squared Error =  0.1530424181283501
R2 Score =  0.37736908807461755
```

Fig 14. Case 5

Now, **Ridge Regressor:**

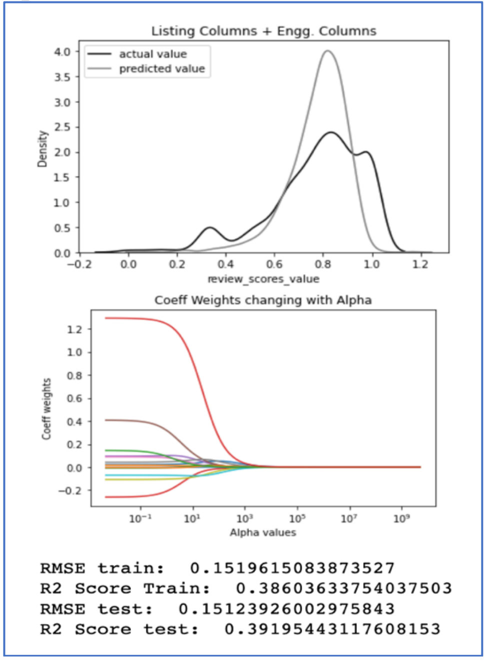
In feature selection, ridge regression avoids over-fitting by regularising the weights to keep them small, and model selection is straight forward as you only have to choose the value of a single regression parameter. Ridge penalizes coefficients by "shrinking" them and "smearing" their influence.

As per GridSearchCV the value for alpha=0.1 is the most optimal for case 1.



```
RMSE train:   0.17454742749167598
R2 Score Train:   0.18147553560815288
RMSE test:   0.18156212525053816
R2 Score test:   0.15005007286669014
```

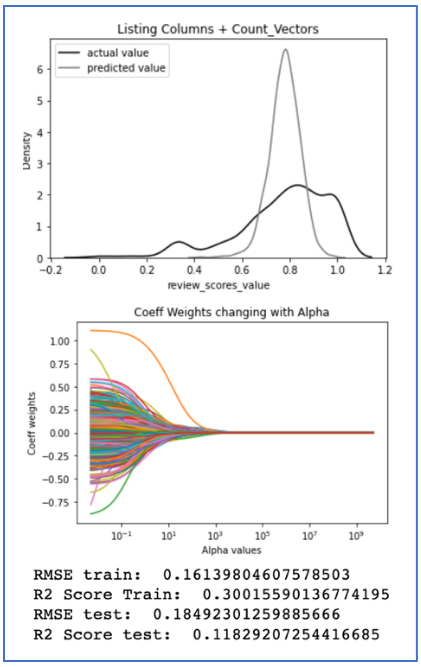
Fig 15. Case 1 alpha=0.1



```
RMSE train:   0.1519615083873527
R2 Score Train:   0.38603633754037503
RMSE test:   0.15123926002975843
R2 Score test:   0.39195443117608153
```

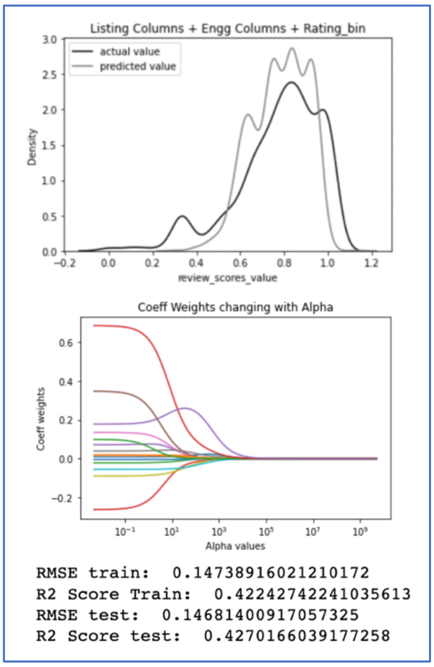
Fig 16. Case 2 alpha = 0.1

The R2 score in case of KNN was 0.07 and the R2 score for Ridge is 0.15, it is already performing better.



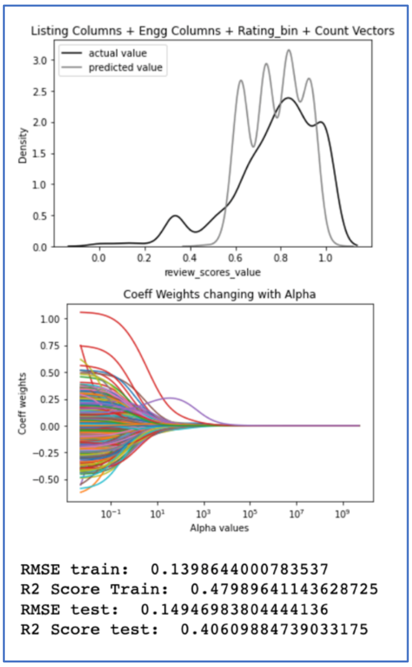
```
RMSE train:  0.16139804607578503
R2 Score Train:  0.30015590136774195
RMSE test:  0.18492301259885666
R2 Score test:  0.11829207254416685
```

Fig 17. Case 3 alpha = 15



```
RMSE train:  0.14738916021210172
R2 Score Train:  0.42242742241035613
RMSE test:  0.14681400917057325
R2 Score test:  0.4270166039177258
```

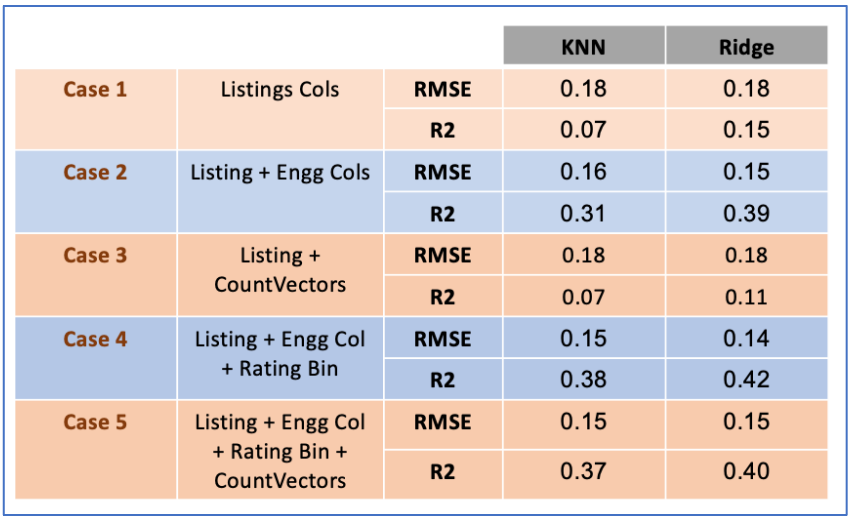
Fig 18. Case 4 alpha = 0.1



```
RMSE train:  0.1398644000783537
R2 Score Train:  0.47989641143628725
RMSE test:  0.14946983804444136
R2 Score test:  0.40609884739033175
```

Fig19. Case 5 alpha = 20

A comparison on R2 score shows that Ridge model performs better than KNN at predicting review scores values.



			KNN	Ridge
Case 1	Listings Cols	RMSE	0.18	0.18
		R2	0.07	0.15
Case 2	Listing + Engg Cols	RMSE	0.16	0.15
		R2	0.31	0.39
Case 3	Listing + CountVectors	RMSE	0.18	0.18
		R2	0.07	0.11
Case 4	Listing + Engg Col + Rating Bin	RMSE	0.15	0.14
		R2	0.38	0.42
Case 5	Listing + Engg Col + Rating Bin + CountVectors	RMSE	0.15	0.15
		R2	0.37	0.40

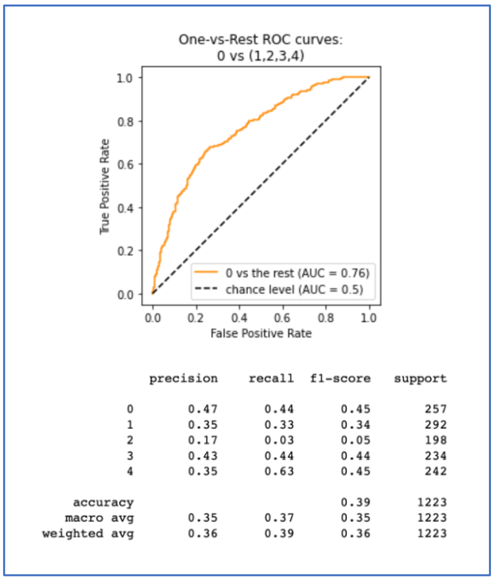
Fig 20. Comparison of KNN and Ridge based on RMSE and R(Squared)

Classification:

The idea behind thinking about this as a classification problem is that we could split the range of 4 & 5 into 10 classes like 4-4.1 , 4.1-4.2 and then run classification on top of it, to minimize the errors in predictions. This approach felt interesting to me. I apply classification using 5 classes, but it can be extended to above scenario.
In case of classification, I used LogisticRegression and KNeighbourClassifier. Instead of predicting review scores, which are in the range 4 to 5, we can divide them into 5 equally probabilistic bins or classes and predict the class based on the features.

The distinction between logistic regression classification and KNN would be that of distinguishing between linear and non-linear boundaries.

I run the same 5 cases with Logistic Regression first. Using GridSearchCV I find the best params for the different type cases. In all the cases the value of C is different, but the value of solver and penalty remain the same as "lbfgs" and "l2". Logistic Regression performs better than K Neighbour Classifier.

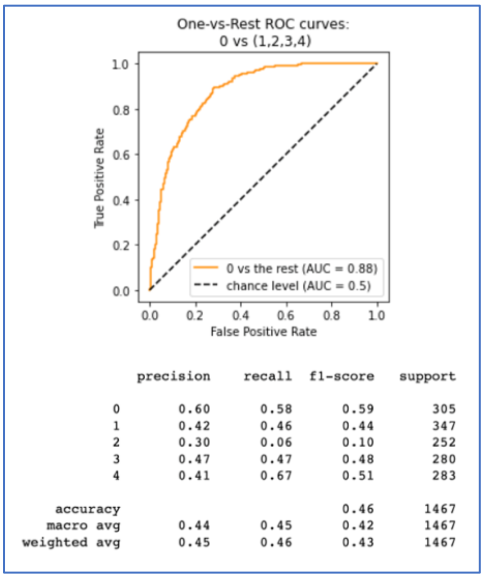


```
              precision    recall  f1-score   support

           0       0.47      0.44      0.45       257
           1       0.35      0.33      0.34       292
           2       0.17      0.03      0.05       198
           3       0.43      0.44      0.44       234
           4       0.35      0.63      0.45       242

    accuracy                           0.39      1223
   macro avg       0.35      0.37      0.35      1223
weighted avg       0.36      0.39      0.36      1223
```

Fig 21. LR Classifier on Case 1, C= 100

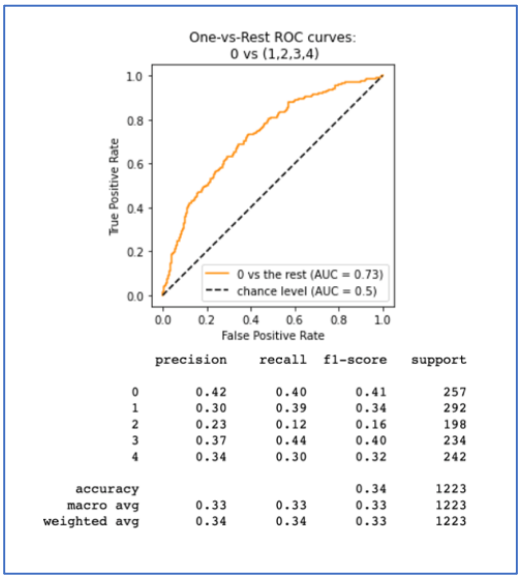


```
              precision    recall  f1-score   support

           0       0.60      0.58      0.59       305
           1       0.42      0.46      0.44       347
           2       0.30      0.06      0.10       252
           3       0.47      0.47      0.48       280
           4       0.41      0.67      0.51       283

    accuracy                           0.46      1467
   macro avg       0.44      0.45      0.42      1467
weighted avg       0.45      0.46      0.43      1467
```

Fig 22. LR Classifier on Case 2 , C=200

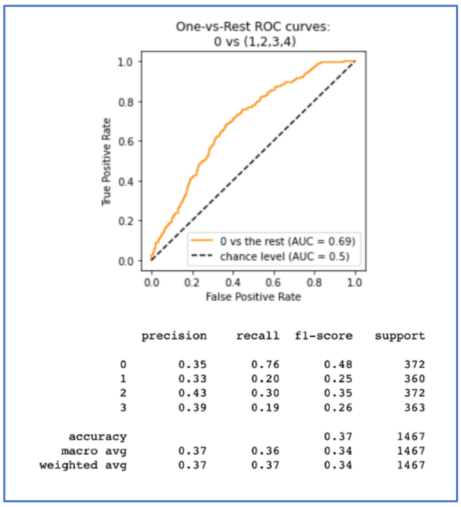


```
              precision    recall  f1-score   support

           0       0.42      0.40      0.41       257
           1       0.30      0.39      0.34       292
           2       0.23      0.12      0.16       198
           3       0.37      0.44      0.40       234
           4       0.34      0.30      0.32       242

    accuracy                           0.34      1223
   macro avg       0.33      0.33      0.33      1223
weighted avg       0.34      0.34      0.33      1223
```

Fig 23. LR Classifier on Case 3, C=1

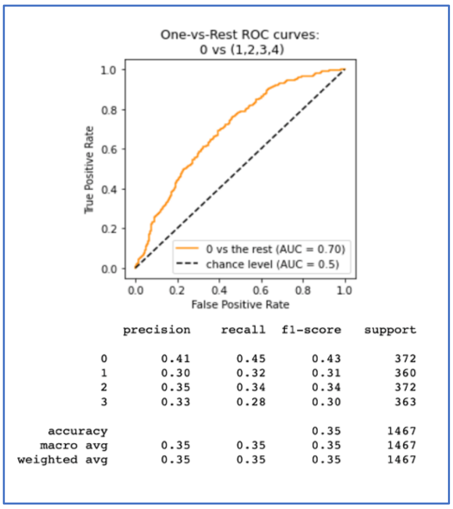


```
              precision    recall  f1-score   support

           0       0.35      0.76      0.48       372
           1       0.33      0.20      0.25       360
           2       0.43      0.30      0.35       372
           3       0.39      0.19      0.26       363

    accuracy                           0.37      1467
   macro avg       0.37      0.36      0.34      1467
weighted avg       0.37      0.37      0.34      1467
```

ig 24. LR Classifier on Case 4, C=100

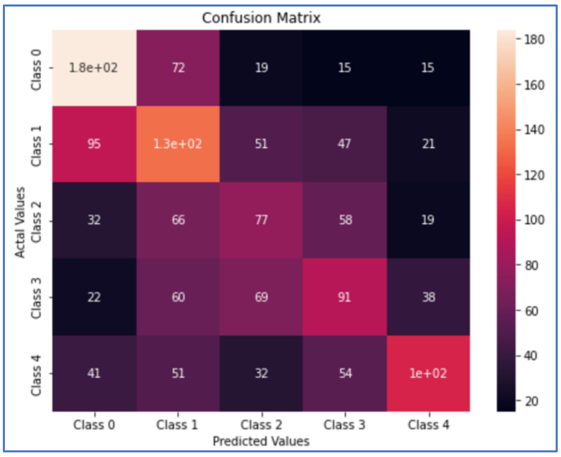


```
              precision    recall  f1-score   support

           0       0.41      0.45      0.43       372
           1       0.30      0.32      0.31       360
           2       0.35      0.34      0.34       372
           3       0.33      0.28      0.30       363

    accuracy                           0.35      1467
   macro avg       0.35      0.35      0.35      1467
weighted avg       0.35      0.35      0.35      1467
```

Fig 25. LR Classifier on Case 5, C=1



	Class 0	Class 1	Class 2	Class 3	Class 4
Class 0	1.8e+02	72	19	15	15
Class 1	95	1.3e+02	51	47	21
Class 2	32	66	77	58	19
Class 3	22	60	69	91	38
Class 4	41	51	32	54	1e+02

Fig 26. CM for Case 2 LR Classifier

The maximum accuracy is reported in Case 2, with engineered columns. The inclusion of CountVectors does not increase the accuracy meaningfully.

			L R Classifier	KNN Classifier
Case 1	Listings Cols	Test Accuracy	0.38	0.36
		Precision	0.35	0.36
		Recall	0.37	0.36
Case 2	Listing + Engg Cols	Test Accuracy	0.46	0.40
		Precision	0.44	0.40
		Recall	0.45	0.40
Case 3	Listing + CountVectors	Test Accuracy	0.34	0.28
		Precision	0.33	0.24
		Recall	0.33	0.28
Case 4	Listing + Engg Col + Rating Bin	Test Accuracy	0.38	0.32
		Precision	0.38	0.32
		Recall	0.38	0.33
Case 5	Listing + Engg Col + Rating Bin + CountVectors	Test Accuracy	0.35	0.33
		Precision	0.35	0.39
		Recall	0.35	0.33

Fig 27. Comparison of KNN and LR classifiers to predict review classes

Conclusion & Take-Aways

In Regression the model is evaluated on MSE and R-Squared. The mean square error is the average of the square of the difference between the observed and predicted values of a variable, and R-squared is the proportion of the variance in the response variable that can be explained by the predictor variables in a regression model. A normalized MSE of 0.02 showed that the predicted values were very close to the true values, and an R-squared of 0.38 showed that the variation in outputs were 38% explainable from the feature sets. Ridge performed better than KNN during regression and the best values were obtained in case 4 where no CountVectors were used and engineered features were added to supplement some patterns with an RMSE score of 0.14 and R-Squared 0.42.

Regression was difficult to implement due to outliers in the data because of prices, amenities, sentiments, etc. Apart from this, the final review scores had to be normalized, and then predicted. The real world incoming data will have greater variations. A model, built on the current features would definitely require a sentiment score from the reviews, and for the engineering, a mean score of the current reviews in the database can be used to calculate the review_sent_bins.

Classification was important to do as looking at the problem in terms of bins or classes of reviews came organically during the implementation of the regression models. Classifiers Logistic and KNN could provide accuracy of 0.46 with engineered features. Again CountVectors did not supplement to the accuracy. Linear Model Classifier and KNN classifier show that the data can still perform better than other baseline models. Confusion matrix for multiclass

Take Aways: TF-IDF and CountVector features didn't play an integral role in predictions. Another model, that could handle a bigger feature-set and much minute difference between changing values could have been used to improve the accuracy of the problem statement modelled. MLP or CNN could have learned the variation in CountVector features and found out patterns to output the review_score. Decision Trees could have also been applied to check the accuracy (I implemented these as well, and they performed better than Ridge Regression).

There were very less patterns in the features, that could explain the final scores. Therefore, more feature augmentation and engineering could benefit the problem statement. Currently few features like neighbourhood were discarded but in future iterations of this problem, I'll be more conscious of such arbitrary features and their relations to the final scores.

Q2.

Q

Give two examples of situations when logistic regression would give inaccurate predictions. Explain your reasoning.

A

Logistic regression is a classification algorithm that predicts a binary outcome based on a series of independent variables. If there are more than two classes of the response variable, it's called multinomial logistic regression. Logistic regression works by measuring the relationship between the dependent variable (what we want to predict) and one or more independent variables (the features). It does this by estimating the probabilities with the help of its underlying logistic function.

Logistic Regression would give inaccurate predictions in the following scenarios:

1. Class imbalance and very little data : Model is essentially working off of a prior probability that disproportionately favours the positive class, and we didn't give the model enough data to escape this prior in the learning process. The logistic regression algorithm wants to minimize its cost function (cross-entropy) which can be defined in a really simple way as the distance between your points and the decision boundary.

2. Linearity and Collinearity : Logistic Regression is a linear model, so it may or may not work well on non-linear cases. When classes highly correlate or are highly nonlinear, the coefficients of logistic regression will not correctly predict the gain/loss from each individual feature. Another, major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables. Logistic Regression requires average or no multicollinearity between independent variables.

Q
Discuss some advantages and disadvantages of a kNN classifier vs an MLP neural net classifier. Explain your reasoning.

A
CNN utilizes the spatial information that other algorithms don't in order to reduce the number of parameters and overall complexity while learning similar information. For small problems, this is unnecessary and can make CNNs prohibitively expensive to train. For larger problems, the complexity of other algorithms actually grows faster, making CNNs more viable. In other words, CNNs scale better.

When we are solving a problem which directly focusses on finding similarity between observations, K-NN does better because of its inherent nature to optimize locally. This is also a flipside because, outliers can significantly kill the performance. Additionally, K-NN is most likely to overfit, and hence adjusting 'k' to maximise test set performance is the way to go. As the complexity of the space grows, the accuracy of K-NN comes down and we would need more data. However, there are many other questions that you have to answer before using this method. One important decision should be made on the ‘distance function’ like, should we standardize your data before feeding them to the distance function? etc.

Choosing a model in ML is largely about: Optimization (how we fit the data) and trade-offs between Precision and recall, Bias/Variance. Any deep neural networks have advantages over KNN because of nonlinearities over dense feature vectors in the data. Therefore, CNN might extract better predictions from the analysis of the features, but may be an overkill on otherwise easier problems or problems of a different nature that can be solved using the distance method approach of KNN.

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

A
K-fold Cross-Validation is when the dataset is split into a K number of folds and is used to evaluate the model's ability when given new data. K refers to the number of groups the data sample is split into. Each fold is used as a testing set at one point in the process.

If we only have a small dataset, splitting it into a training and test dataset at a 80:20 ratio respectively doesn’t seem to do much for us as there might be inherent patterns or bias that would be left hidden in the data.

However, when using K-fold cross validation, all parts of the data will be able to be used as part of the testing data. This way, all of our data from our small dataset can be used for both training and testing, allowing us to better evaluate the performance of our model.

The choice of k is usually 5 or 10, but there is no formal rule. As “k” gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller (i.e., the bias is smaller for k=10 than k= 5). In this context, the bias is the difference between the estimated and true values of performance.

Larger K means less bias towards overestimating the true expected error (as training folds will be closer to the total dataset) but higher variance and higher running time. Higher variance with large k means, all the K

training sets will have large data in common, so the trained K models will be somewhat correlated, resulting in K correlated test errors, so the mean of the test error will have higher variance.

We must ensure that the training set and testing set are drawn from the same distribution. And that both sets contain sufficient variation such that the underlining distribution is represented. People usually use 5-fold cross validation. This means that 20% of the data is used for testing, this is usually pretty accurate. However, if your dataset size increases dramatically, like if you have over 100,000 instances, it can be seen that a 10-fold cross validation would lead in folds of 10,000 instances. This should be sufficient to reliably test your model.

Q

Discuss how lagged output values can be used to construct features for time series data. Illustrate with a small example

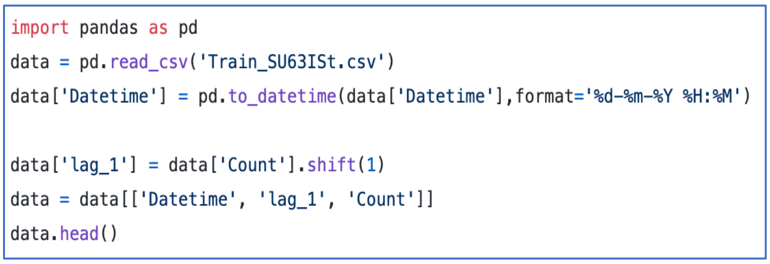
A

Lag features are target values from previous periods.

For example, if we want to forecast the sales of a retail outlet in period t you can use the sales of the previous month $t-1$ as a feature. That would be a lag of 1 and you could say it models some kind of momentum. But we could also apply a lag of 12 to model the sales of the same month a year ago

The Pandas library provides the shift() function to help create these shifted or lag features from a time series dataset. Shifting the dataset by 1 creates the t-1 column, adding a NaN (unknown) value for the first row. The time series dataset without a shift represents the t+1.

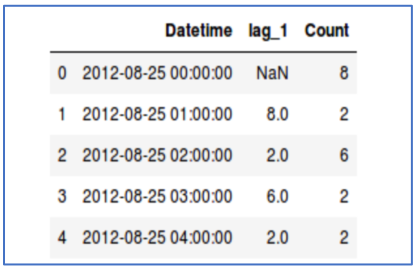
For example, if we are predicting the stock price for a company. So, the previous day's stock price is important to make a prediction, right? In other words, the value at time t is greatly affected by the value at time t-1. The past values are known as lags, so t-1 is lag 1, t-2 is lag 2, and so on.



```
import pandas as pd
data = pd.read_csv('Train_SU63ISt.csv')
data['Datetime'] = pd.to_datetime(data['Datetime'],format='%d-%m-%Y %H:%M')

data['lag_1'] = data['Count'].shift(1)
data['lag_1'] = data['Count'].shift(1)
data = data[['Datetime', 'lag_1', 'Count']]
data.head()
```

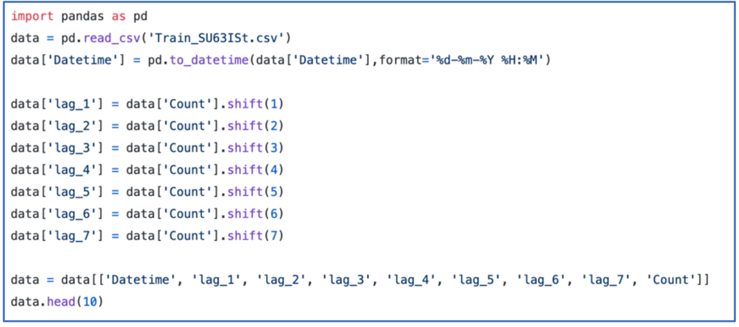
Fig 1 . Reading the dummy time-series and count data csv



	Datetime	lag_1	Count
0	2012-08-25 00:00:00	NaN	8
1	2012-08-25 01:00:00	8.0	2
2	2012-08-25 02:00:00	2.0	6
3	2012-08-25 03:00:00	6.0	2
4	2012-08-25 04:00:00	2.0	2

Fig.2 Head of the data

The lag value we choose will depend on the correlation of individual values with its past values. If the series has a weekly trend, which means the value last Monday can be used to predict the value for this Monday, the we should create lag features for seven days.

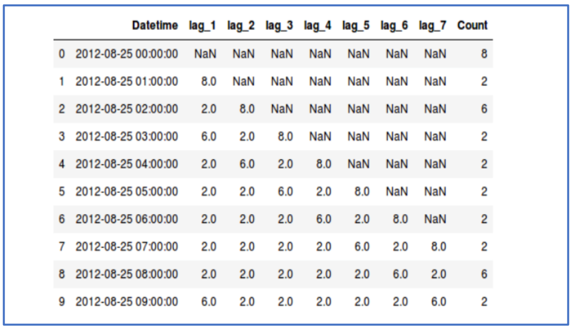


```
import pandas as pd
data = pd.read_csv('Train_SU63ISt.csv')
data['Datetime'] = pd.to_datetime(data['Datetime'],format='%d-%m-%Y %H:%M')

data['lag_1'] = data['Count'].shift(1)
data['lag_2'] = data['Count'].shift(2)
data['lag_3'] = data['Count'].shift(3)
data['lag_4'] = data['Count'].shift(4)
data['lag_5'] = data['Count'].shift(5)
data['lag_6'] = data['Count'].shift(6)
data['lag_7'] = data['Count'].shift(7)

data = data[['Datetime', 'lag_1', 'lag_2', 'lag_3', 'lag_4', 'lag_5', 'lag_6', 'lag_7', 'Count']]
data.head(10)
```

Fig.3 Creating lag features for the past 7 times



	Datetime	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7	Count
0	2012-08-25 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	8
1	2012-08-25 01:00:00	8.0	NaN	NaN	NaN	NaN	NaN	NaN	2
2	2012-08-25 02:00:00	2.0	8.0	NaN	NaN	NaN	NaN	NaN	6
3	2012-08-25 03:00:00	6.0	2.0	8.0	NaN	NaN	NaN	NaN	2
4	2012-08-25 04:00:00	2.0	6.0	2.0	8.0	NaN	NaN	NaN	2
5	2012-08-25 05:00:00	2.0	2.0	6.0	2.0	8.0	NaN	NaN	2
6	2012-08-25 06:00:00	2.0	2.0	2.0	6.0	2.0	8.0	NaN	2
7	2012-08-25 07:00:00	2.0	2.0	2.0	2.0	6.0	2.0	8.0	2
8	2012-08-25 08:00:00	2.0	2.0	2.0	2.0	2.0	6.0	2.0	6
9	2012-08-25 09:00:00	6.0	2.0	2.0	2.0	2.0	2.0	6.0	2

Fig.4 Head of the new dataset

When choosing lags to use as features, partial autocorrelation tells us the correlation of a lag, accounting for all of the previous lags -- the amount of "new" correlation the lag contributes, so to speak. Plotting the partial autocorrelation can help us choose which lag features to use.

-------------------------Thank you-------------------------

Appendix

https://github.com/aggarpa/ML_Final_Assignment_Code