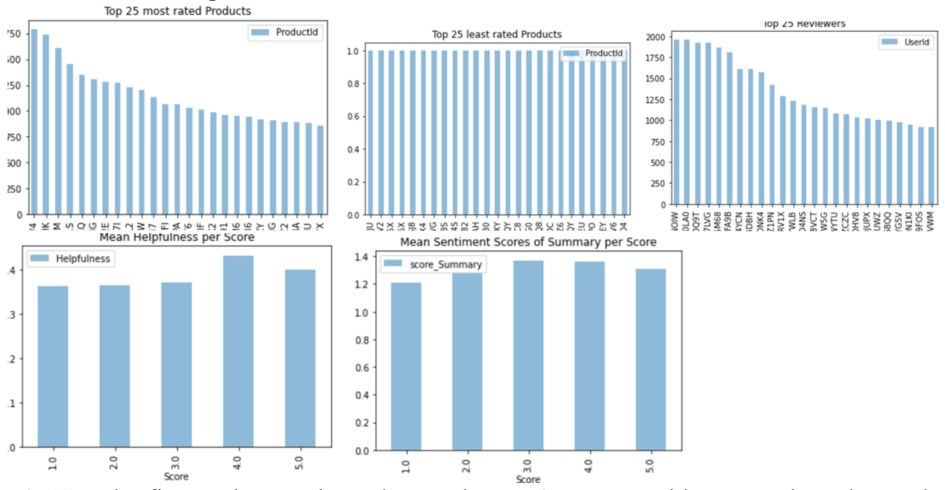
Kaggle Username: mkmk

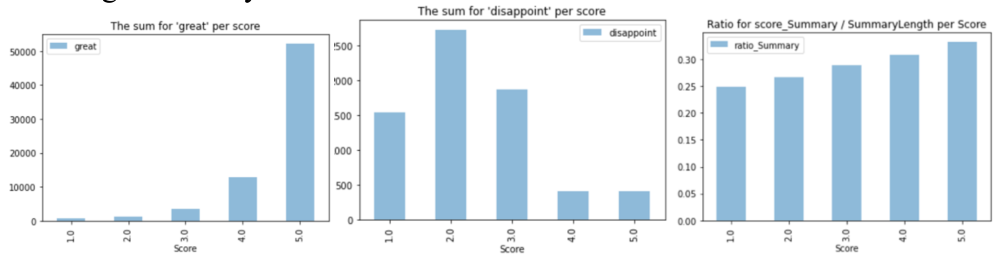
Preliminary Analysis / Exploration

For the preliminary exploration, I explore the top and least 25 most rated products, the top and least 25 reviewers, and the helpfulness for the reviews. I also explored the predict score of reviews' summaries by using 'SnowballStemmer 'and 'TfidfVectorizer'. Then I found that the reviews with highest score are from Score=3, and with the lowest score are from score=1. (Fig1)



Fig(1): The first and second graphs are the top 25 most and least rated products; the third graph is top 25 reviewers; the fourth graph is the score of helpfulness; the last graph is the mean scores of summaries for each Score.

By analyzing the most frequent words in per score, I found that some words will show out several times, so depending on the matrix from 'TfidfVectorizer', I analyzed the frequency of these words in different Scores. (Fig2) As the graphs show, the words like 'great' with a positive meaning are mostly in the reviews with higher score. The words like 'disappoint' with a negative meaning are mostly in the reviews with lower score.



Fig(2): The first and second graphs are the frequencies they show out in different Score; the last graph is the ratio foof 'score_Summary / SummaryLength' for per Score.

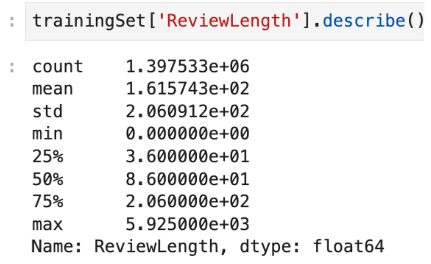
By analyzing the length of each summary, I found that the length is also related to score of summaries and the Score of products. Therefore, I analyzed them by using the value of 'score_Summary / SummaryLength' for per Score. (Fig(2))

Feature Extraction

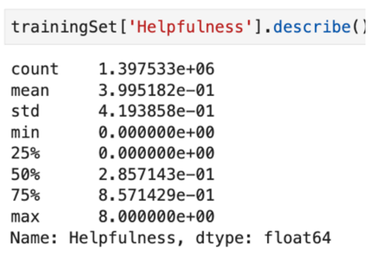
For the feature extraction part, I did several changes.

To grade for the reviews, I used 'nltk' and 'TfidfVectorizer' to clean the words in summaries and calculate the score of words. Then for each summary, I added the scores of words and stored the added scores into a new column 'sentiment'. Also, as the preliminary analysis shows, the length of reviews and summaries are also related to the scores, so I created the ‘ReviewLength’ and ‘SummaryLength’ features and filled the nan into 0. (Fig(3))

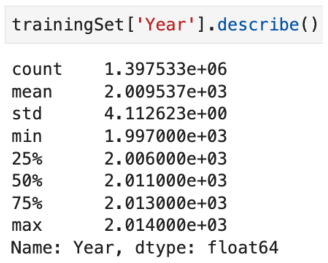
The feature ‘Time’ is transferred into the date data and stored into the new columns ‘Year’, ‘Hour’, ‘Date’ and ‘Month’ by using ‘to_datetime’ function. When applied for the ‘X_train’, the feature ‘Time’ and ‘Date’ were dropped, since other features have contained these data.



```
trainingSet['ReviewLength'].describe()

count    1.397533e+06
mean     1.615743e+02
std      2.060912e+02
min      0.000000e+00
25%      3.600000e+01
50%      8.600000e+01
75%      2.060000e+02
max      5.925000e+03
Name: ReviewLength, dtype: float64
```



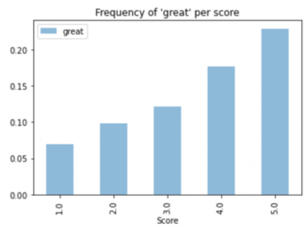
```
trainingSet['Helpfulness'].describe()

count    1.397533e+06
mean     3.995182e-01
std      4.193858e-01
min      0.000000e+00
25%      0.000000e+00
50%      2.857143e-01
75%      8.571429e-01
max      8.000000e+00
Name: Helpfulness, dtype: float64
```

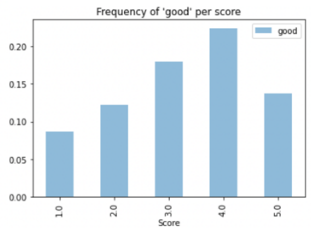


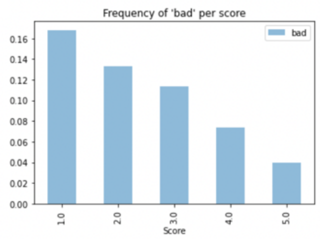
```
trainingSet['Year'].describe()

count    1.397533e+06
mean     2.009537e+03
std      4.112623e+00
min      1.997000e+03
25%      2.006000e+03
50%      2.011000e+03
75%      2.013000e+03
max      2.014000e+03
Name: Year, dtype: float64
```

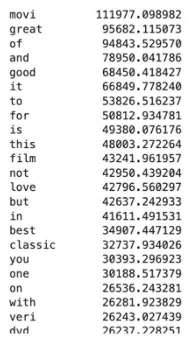
Fig(3): These three graphs are the description of ‘ReviewLength’, ‘Helpfulness’, and ‘Year’ respectively.

By using TfidVectorizer with a maximum of 0.5 and a minimum of 0.1 on the stemmed summary, the words are graded. By listing the top 50 words with highest scores, I chose the first six words that were clearly emotional and analyzed their frequencies in different product scores. The results showed that these words’ distribution is very different in different product score. Therefore, I added these words as features into the training data. (Fig(4))









```
movi       111977.098982
great       95682.115073
of          94843.529570
and         78950.041786
good        68450.418427
it          66849.778240
to          53826.516237
for         50812.934781
is          49380.076176
this        48003.272264
film        43241.961957
not         42950.439204
love        42796.560297
but         42637.242933
in          41611.491531
best        34907.447129
classic     32737.934026
you         30393.296923
one         30188.517379
on          26536.243281
with        26281.923829
veri        26243.027439
dvd         26237.228251
```

Fig(4): The first three graphs are respectively the frequency of ‘great’, ‘good’, and ‘bad’ in different product scores. The last graph is a part of the list of top 50 words with highest scores

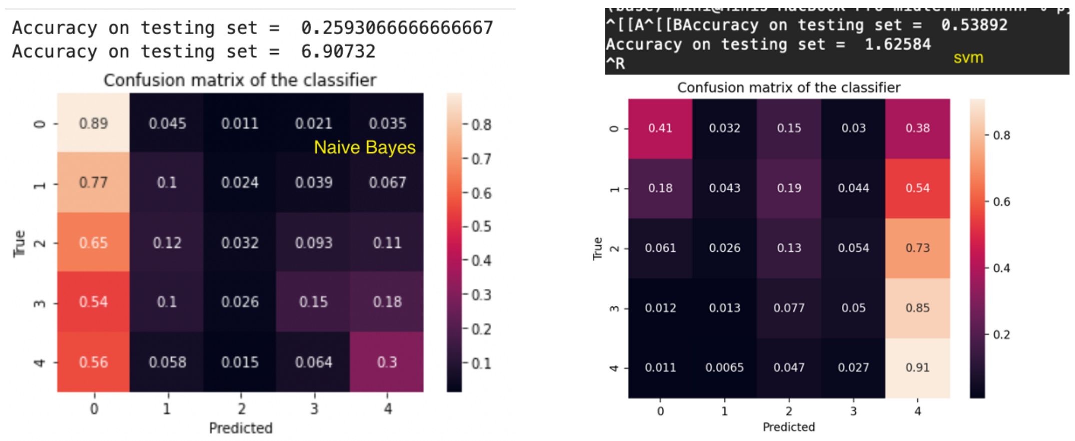
Workflow, Decisions, and Techniques Tried

I chose jupyter notebook to visualize the data, especially for the preliminary analysis. As the code given by us, I firstly processed the train data by extracting features and other operations to the data, and then stored the data into the x_train.csv, which is used to train.

The classifier I firstly chose was KNN, since compared to other classifiers, I am more familiar with it. However, the accuracy score is very low and took a long time to calculate. Then I tried Naive Bayes, whose performance was even worse, with a 25% accuracy. (Fig(5))

For the decision tree classifier, I created some features ‘bin_helpfulness’ in binary format. For example, by determining whether the helpfulness is larger than the mean of it or smaller than it, I changed these data into 0 and. For the deature ‘Year’, ‘SummaryLength’, and ‘ReviewLength’, I used the Kmean clustering and label the data of these features respectively. However, the performance of decision tree classifier was still not good, with a 45% accuracy.

I also trained the data by using SVM classifier and Ensemble classifier, the performance of both classifiers improved about 5% accuracy. (Fig(5))



```
Accuracy on testing set =  0.2593066666666667
Accuracy on testing set =  6.90732
```

```
^[[A^[[BAccuracy on testing set =  0.53892
Accuracy on testing set =  1.62584
^R
```

Fig(5): The graph on the left is the accuracy score and confusion matrix of Naïve Bayes classifier; the graph on right is the accuracy score and confusion matrix of SVM.

Model Tuning/Testing

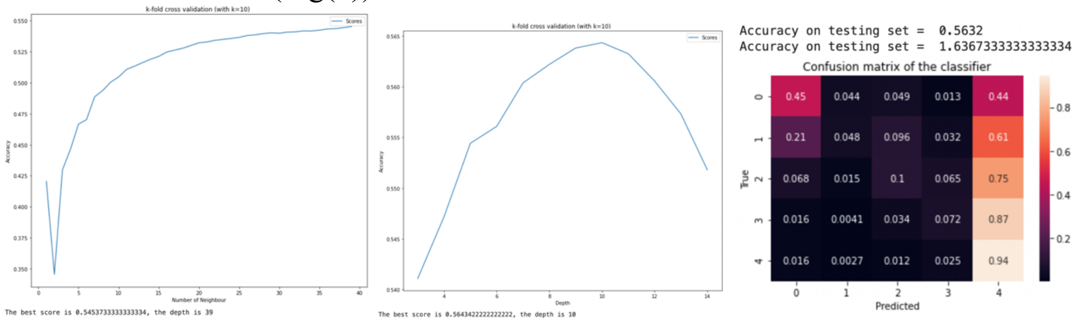
For training data, I used 'cross_val_score' with kfold=10 and calculated the mean of the results to find the best K.

For testing the model, I chose 'accuracy_score' and 'mean_squared_error' to check the accuracy of each model. The confusion matrix for each model was visualized by 'sns.heatmap'.

When training the model, I randomly chose 300000 datasets from the data used to train. Then I created a training and testing datasets from the dataset, with a 1/4 ratio, and 'random_state'=0.

When finding the optimal neighbors K of KNN model I chose a range of (35, 55), since after a few tests, I found that the optimal K is larger than 30. Fig(6))

As I said above, I created some features especially for the decision tree model but didn't work well. So, I dropped all these features, and process the model again. Luckily, the accuracy score increased about 5%. (Fig(6))



```
The best score is 0.5453733333333334, the depth is 39
```

```
The best score is 0.5643422222222222, the depth is 10
```

```
Accuracy on testing set =  0.5632
Accuracy on testing set =  1.6367333333333334
```

Fig(6): The first graph is the k-fold cross validation of KNN; the second and third graph are respectively the k-fold cross validation of Decision Trees and confusion matrix of Decision Tress model.

Creativity/Challenges/Effort

As I said above, the accuracy of the models was low at first, even though I have tried many times and created more related features. However, I found that many features that I created would make the accuracy become lower, since every time I created a new feature, I would delete the features that are contained in the new feature, which made the model performed worse. So, I deleted some of the new features and did some change. For example, I used df['SummaryLength'] instead of df['Summary'].str.len() / df['SummaryLength'] to process the data and train the model. Luckily, the accuracy of these models increased after I made the changes.