- **APPROACH**

I pre-processed the data and applied KNN classification.

Pre-processing

At first the Train Data provided was loaded into dataframe using pandas library, where the delimiter was set to '#EOF'. I dropped the non-essential columns and had to split the string containing Rating and reviews combined to columns in the dataframe.

Further I used libraries like re, nltk for cleaning the Review column data by converting them to lower case and dropping commas, hyphen etc and further stemming the words and applied stopwords. Since the stopwords were also excluding word like 'not', I used a for loop so that the program doesn't drop 'not' from reviews. We saved all the cleaned reviews in **'corpus'** list.

Now I used scikit_learn's TfidfVectorizer to convert all the clean review in corpus to term inverse document frequency matrix. Hence the program learned words in corpus(cleaned review) and I got a sparse matrix which had inverse document frequency weight assigned to each word in corpus it was assigned to '**X' a**nd in **'y'** i had the labels or ratings for these reviews.

KNN Classification

Before implementing the classification, I split the Train Data into Training set and Test set in 80:20 ratio using sklearn's model selection library. In test_r array I stored word text reviews for further use.

I defined a KNN function, in which I used sklearn's librabry for linear kernel to find the cosine similarity distance. I used **for** loop to iterate through all the reviews words in '**test_r'** array. The linear kernel returns an array of cosine similarity scores which were calculated between whole training tf-idf matrix and single tf-idf vector in test set. The array is sorted according to the highest K value's indices. Now I created a list of training labels with the indices that I got. And another list is created containing similarity score having sign same as that of label.

Now I took sum of all similarities in the list. The positive sum denotes that the review is positive and the negative sum denotes that the review is negative. At the end of the loop, we get a list which is converted as Pandas dataframe **'prediction'**.

- **METHODOLOGY OF CHOOSING THE APPROACH AND ASSOCIATED PARAMETERS**

The approach to pre-process the data was simple, while pre-processing I eliminated all the unnecessary elements like review like commas, hyphen etc. After applying stopwords I noticed that words like 'not' were also dropped which was crucial in deciding the review is positive or negative. I found a way to include 'not' again, but I did not look if there were some other such words which got dropped by mistake.

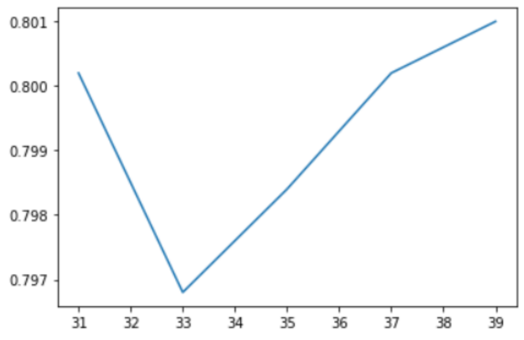
I used tf-idf representation, as this method was faster and less noisy than count vectorization for such a large dataset.

For calculating accuracy, I used f1 accuracy method instead of accuracy score since, it gives us better measure of incorrectly predicted cases as compared to accuracy score. Based on F-1 score, I did cross validation by checking through range of values to determine the best value for K in KNN model. I found out the model worked better in the range of odd numbers between 30 to 40.

```
In [208]: K_values
Out[208]:
```

	K	Accuracy_Score
0	31	0.8002
1	33	0.7968
2	35	0.7984
3	37	0.8002
4	39	0.8010

```
In [215]: plt.plot(K_values['K'], K_values['Accuracy_Score'])
Out[215]: [<matplotlib.lines.Line2D at 0x2e968f41640>]
```



So, doing the same pre-processing as done on training data. I did pre-processing to test data and using K=39 for implementation of KNN on test data and the results were saved in a file.

The runtime for code was about 30-35 minutes.