

**CE/CZ4045 - NATURAL LANGUAGE PROCESSING**

**PROJECT REPORT**

**GROUP 31**

AKSHAT SHARMA (U1923701F)

ANGELIN GRACE WIJAYA (U2023625L)

BHUTRA DIVYANSH (U2023744B)

GUPTA SUHANA (U1923230B)

SINGH AISHWARYA (U1923952C)

Table of Contents

[**Introduction**](#_5rmdg4chlkeb) **4**

[**Section 1: Crawling**](#_73jqdgc7k18e) **4**

[1.1 How we crawled the corpus and stored it](#_e2glpsew8ys5) 4

[1.2 Patterns or insights derived from crawled corpus](#_pstqsvrroba6) 6

[1.3 The numbers of records, words, and types (i.e., unique words) in the corpus](#_h4c5h6exfq9) 8

[**Section 2: Classification**](#_t8o07be5ytlg) **9**

[2.1 Preprocessing Data](#_94ti183h7qy4) 9

[2.2 Motivation for Classification Approach](#_ls3ecz3wew0m) 13

[2.2.1 Machine Learning models](#_ixugh8m7xury) 13

[A. Gaussian Naive Bayes](#_20gkck62ojvr) 13

[B. Random Forest](#_98u4r0ruxgnd) 14

[2.2.2 Lexicon-based Sentiment Analysis](#_7byc4p40ipnz) 15

[A. VADER (Valence Aware Dictionary for Sentiment Reasoning)](#_657oyr59ajfe) 15

[B. TextBlob](#_c4h9zb2vv8sf) 16

[2.2.3 Deep Learning models](#_clq2b5fv440a) 18

[A. Bi-Long Short-Term Memory (LSTM)+Attention](#_cv4huk2chkrd) 18

[B. RoBERTa Embeddings](#_84419a79q5ma) 23

[2.3 Building Evaluation Dataset by Manually Labeling](#_4xsdoetzdchr) 26

[2.4 Evaluation metrics and discussion of results](#_axraxb394s5e) 27

[2.5 Discussion of performance metrics](#_q6zmv987sk74) 27

[2.5.1 Average Tweet Length](#_192c3vxdz7uv) 27

[2.5.2 Average Record Classification Metrics](#_os3pylslxf08) 27

[2.6 A simple UI for visualizing classified data](#_3qnilg6s4j3t) 27

[**Section 3: Explore some innovations for enhancing classification**](#_z8ewzwm5h6f4) **32**

[3.1 Potential Reasons for Misclassification](#_1kx4rsrew2tv) 32

[3.2 Innovations during Preprocessing](#_v7xv1flynjv6) 32

[3.2 Named Entity Recognition](#_jcy4hrceg96j) 33

[3.3 Principal Component Analysis](#_qxh4laji112j) 33

[3.4 Transfer Learning and Model Fine-Tuning](#_y2l0gpwx7kud) 36

[**Conclusion**](#_czv5k6dw9h9p) **37**

[**Bibliography**](#_rrc8fhvujhh8) **38**

[**Work Distribution**](#_sr3p6xcs63rc) **38**

## 

## Introduction

Football (aka Soccer) is by far the most popular sport in the world with an estimate of 3.5 billion fans worldwide. This sport is played by over 250 million players and there are 209 countries recognized by FIFA (International Federation of Association Football). Such a popular sport has become part of the fabric of society and particularly so in England where the English Premier League takes place. 20 teams compete for the Premier League title, it is the most competitive league on the planet. The influence of this league is tremendous such that every day millions of fans of various teams would tweet about the matches, transfers, discuss, criticize and applaud performances throughout the year.

Taking inspiration from this, we were curious to know how the fanbase felt about the teams they support (or hate) and how their opinions/sentiments change over a period of time, specifically for the big 6 teams with the largest fanbase. Thus, we decided to perform sentiment analysis on tweets regarding these teams which gives an indication as to what the majority of the fanbase honestly think about their team and whether any decision made by the team is welcomed or not. All the procedures and steps carried out to perform this sentiment analysis are discussed in the upcoming sections.

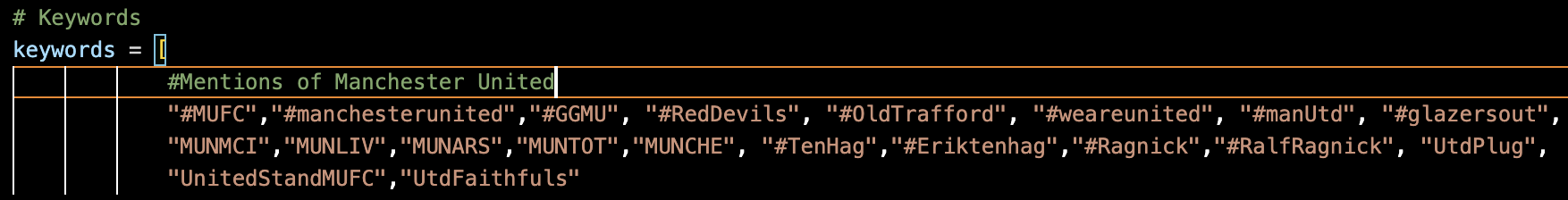
## Section 1: Crawling

### 1.1 How we crawled the corpus and stored it

Our group chose to perform sentiment analysis on tweets regarding the Premier League teams listed below:

1. Arsenal
2. Manchester United
3. Manchester City
4. Tottenham Hotspurs
5. Liverpool
6. Chelsea

Firstly, we gathered the related hashtags for each team (example hashtags of Tottenham Hotspurs: #coys #spurs #tottenhamhotspurs). The code snippet is shown in Figure 1.1.1.



*Fig 1.1.1: All hashtags used for Manchester United*

Secondly, we opted to scrape tweets from 29 nations that are either geopolitically significant or have a high football fan following. The list of countries we used is as shown in Figure 1.1.2 below.



*Fig 1.1.2: All countries used for scraping*

We also set the desired time frame to be between 2021-08-01 until 2022-09-04. Then, we imported the TwitterSearchScraper function from the snscrape.modules.twitter library and gathered tweets for each country with a maximum of 500 tweets per hashtag. The results for each country were assigned to their respective dataframe and then appended into a single dataframe with columns as seen in Table 1.

| **Field Name** | **Description** |
| --- | --- |
| username | Contains the username of the user that posted the tweet |
| content | Contains the actual tweet |
| date | Contains the date on which the tweet was posted |
| country | Contains the country where the user lives in |
| replyCount | Contains the number of people that commented to the tweet |
| retweetCount | Contains the number of people that shared that particular tweet |
| likeCount | Contains the number of people that liked that particular tweet |
| followersCount | Contains the number of followers for a particular user |
| verified | Contains Boolean status indicating if user’s account is verified/not |

*Table 1: Dataframe columns and their definition*

### 1.2 Patterns or insights derived from crawled corpus

Upon crawling and exploring the data, we found that tweets about the teams were not restricted to performance. In fact, they involved a lot of discussion and debates over the type of players the club should recruit, the different tactics supporters of the club wants their team to try, certain players or coaches preferred and disliked by the fanbase, and so on.

Since the tweets we specifically crawled for Premier Leagues teams had a huge amount of information about other footballing aspects we noticed that the same dataset could be used for for following subtopics:

1. Player Transfers
2. Football tactics and formations
3. Individual players and managers public sentiment analysis
4. Different football leagues comparison through sentiment analysis
5. Effect of fanbase sentiments on performance of team



*Figure 1.2.1: WordCloud of corpus before preprocessing*

**

*Figure 1.2.2: WordCloud of corpus after preprocessing*

### 1.3 The numbers of records, words, and types (i.e., unique words) in the corpus

More than 95000 tweets were first gathered from Twitter using SNScrape. We did note, however, that the scraped tweets contained a significant number of tweets that were either unrelated or otherwise unsuitable for sentiment analysis. Some tweets, for example, had more hashtags than normal text, and others were duplicates from the same account. As a result, the scraped tweets were meticulously treated before being utilized for indexing, querying, and classification - see Section 2.1 of this report for additional information on the pre-processing of tweets (Data Pre-Processing).

There were 96616 tweets (records) in the corpus after data pre-processing, and these contained tweets that were either Positive Statement, Negative Statement, or Neutral.

The exact number of records, words, and unique words in the final corpus is provided in the table below:

| **Number of Records in Corpus** | 96616 |
| --- | --- |
| **Number of Words in Corpus** | 1915338 |
| **Number of Unique Words in Corpus** | 153073 |
| **Number of Premier League Teams for which tweets were extracted** | 6 |

*Table 2: Number of records, words, and unique words in the final corpus*

**1.4 Dataset and Labeling**

We chose to classify our scraped tweets on premier league teams into 3 sentiments with respect to public opinion/sentiment. The sentiments and their corresponding labels are:

1. Positive: 1
2. Negative: -1
3. Neutral: 0

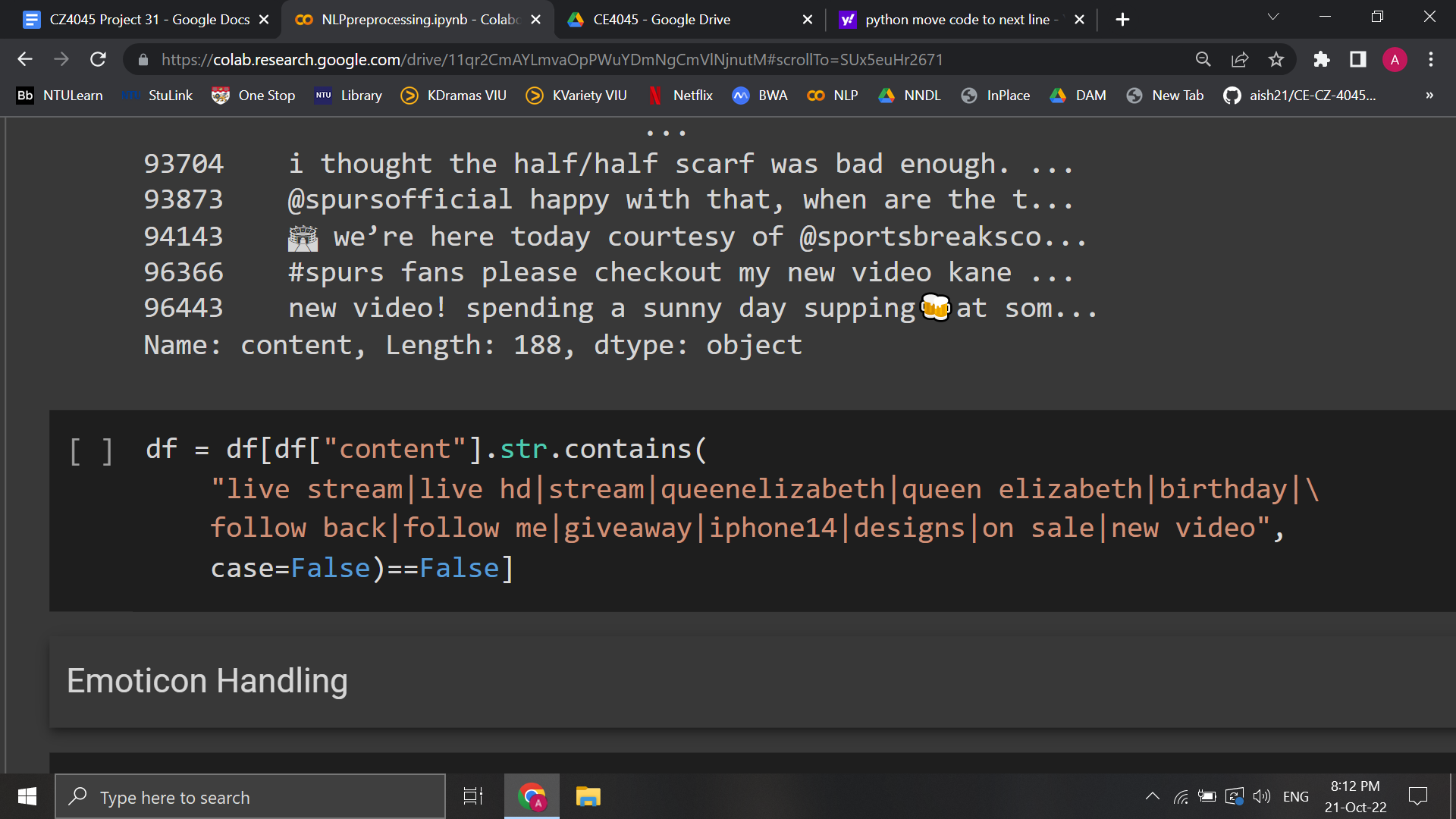
## Section 2: Classification

### 2.1 Preprocessing Data

We had to preprocess the data in order to reduce noise for the classification model in the next section. Below is the list of preprocessing steps we performed:

1. **Removal of irrelevant tweets containing certain keywords**

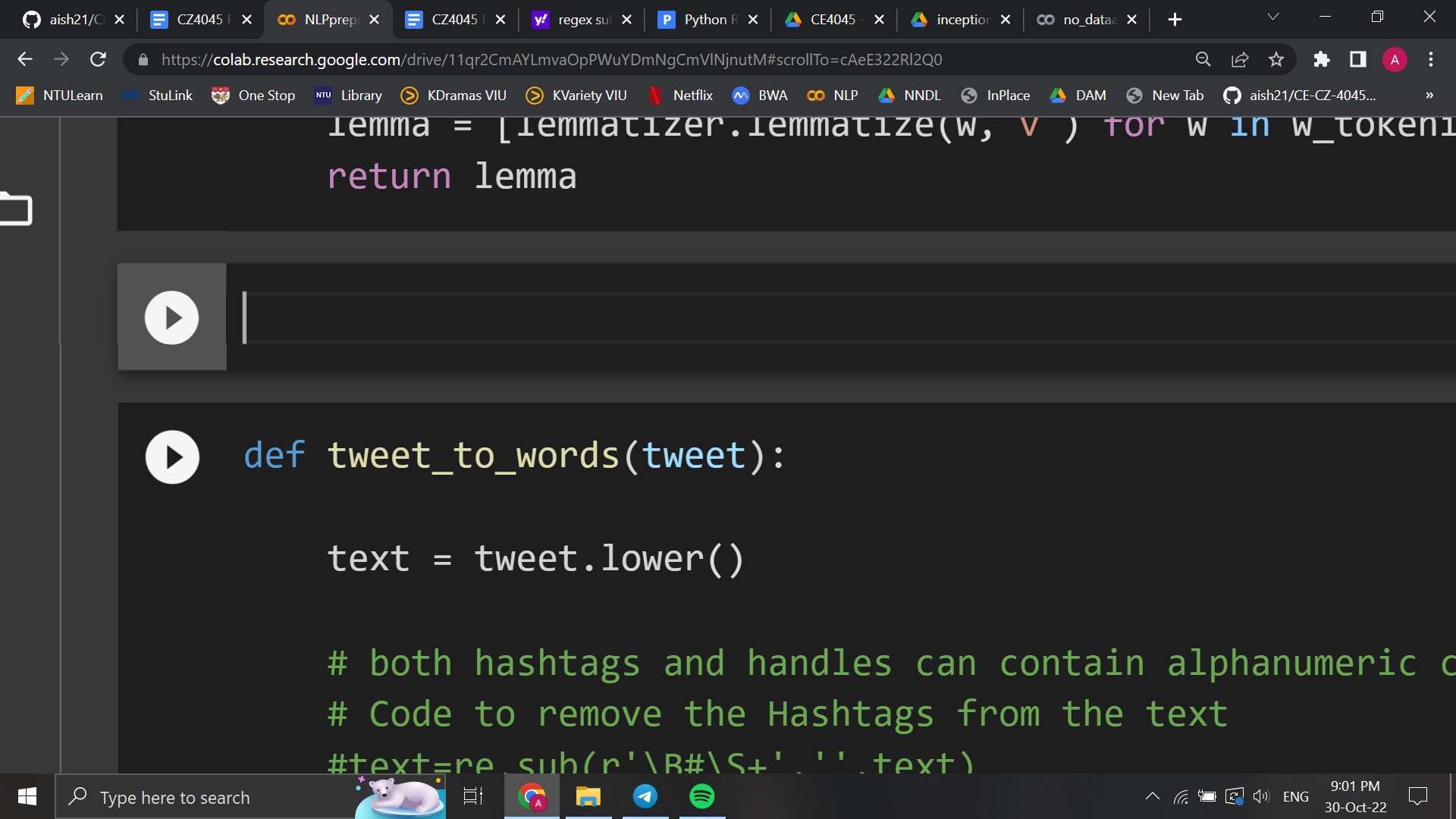
As we went through the tweets, we found that there were quite a number of irrelevant topics, such as advertisements or live streaming reminders. Therefore we manually selected keywords of such topics and used the code snippet in Figure 2.2.1 to remove them from the dataframe.



*Fig 2.1.1: Code snippet for removing irrelevant tweets*

1. **Lowering of the text**

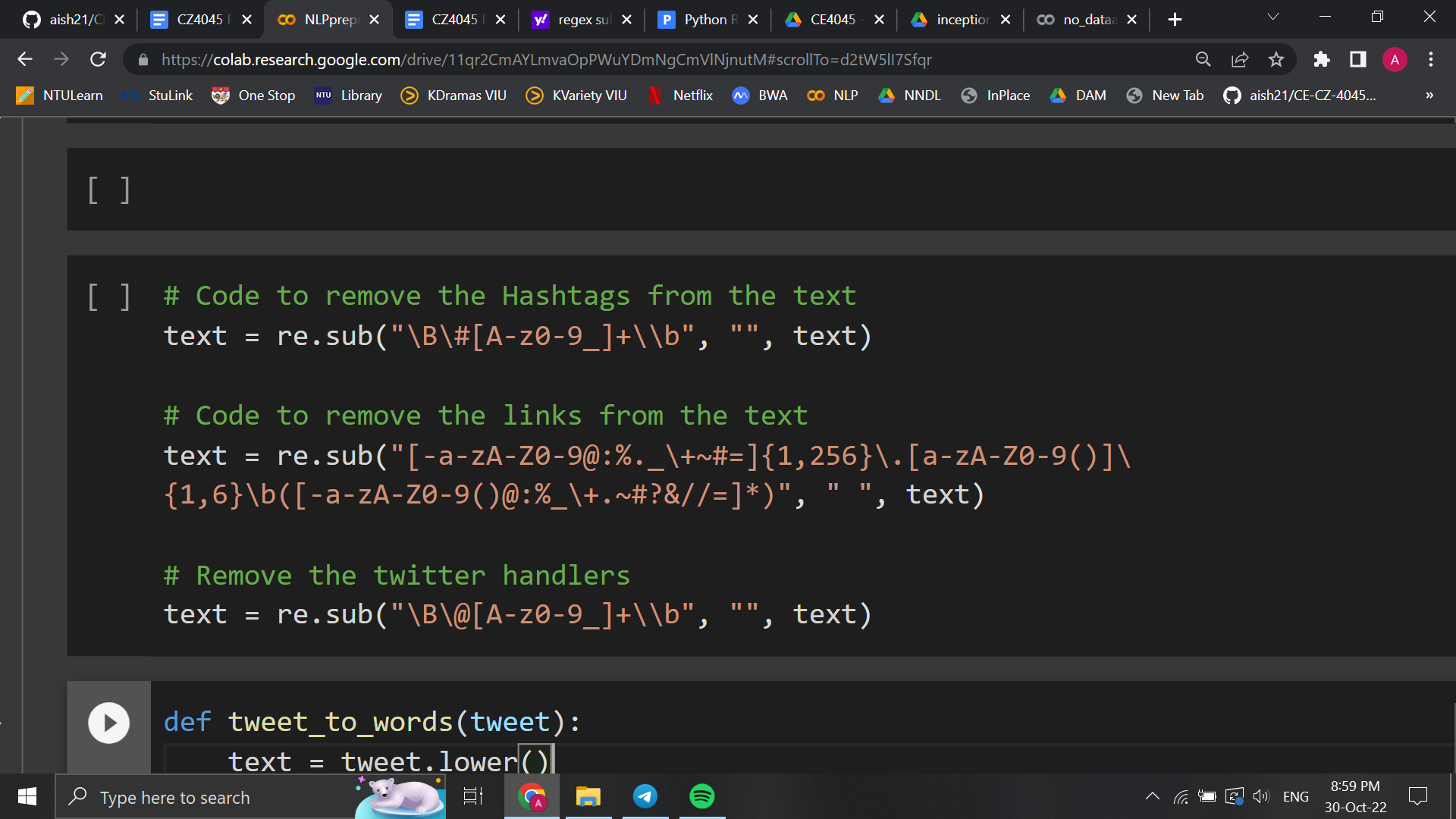
The corpus was converted to lowercase to standardize the text



*Fig 2.1.2: Code snippet for lowercase conversion*

1. **Removal of hashtags, links and twitter handles**

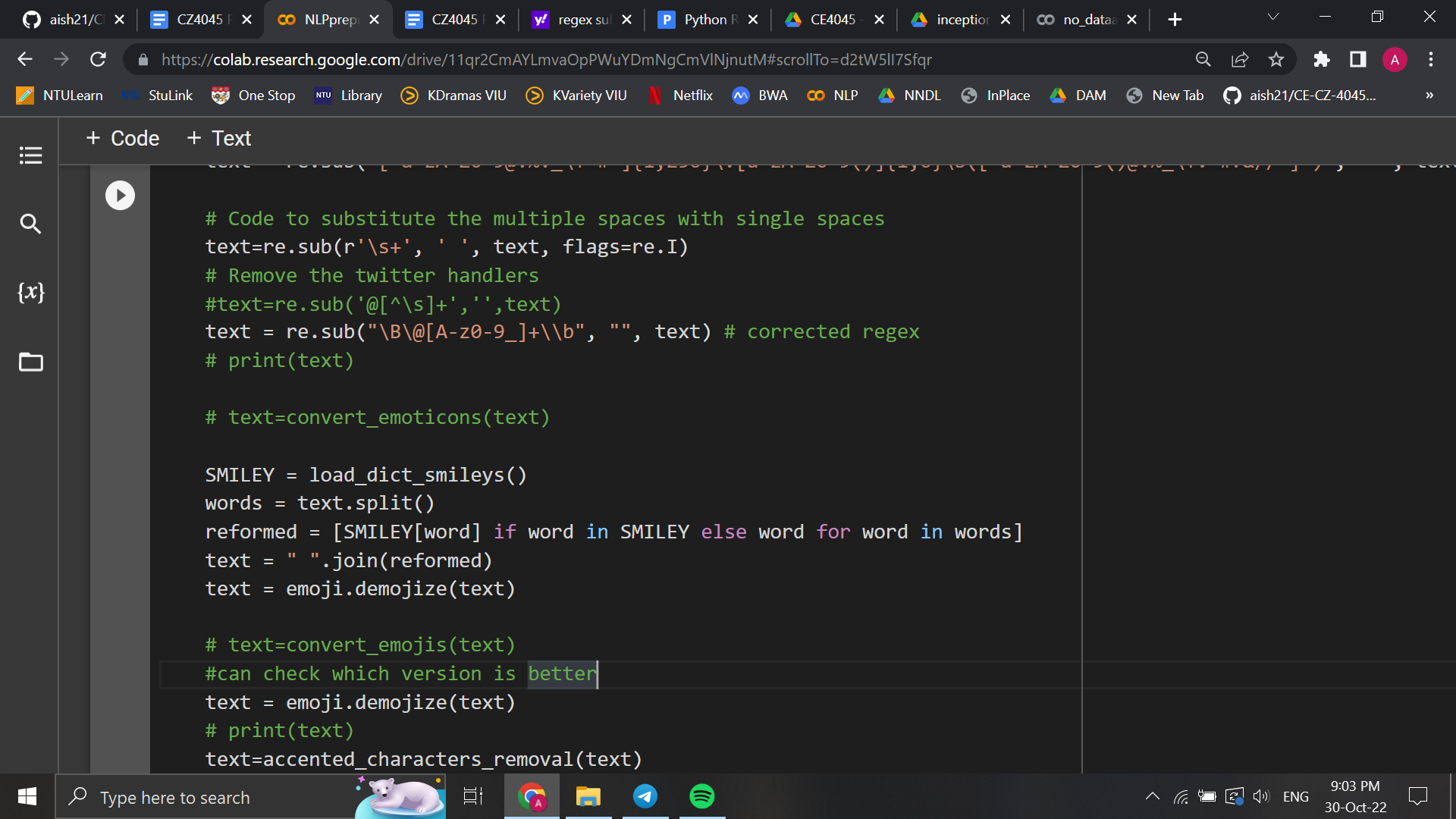
As these characters do not contribute to the sentiment of the tweet, they were removed using regex sub function.



*Fig 2.1.3: Code snippets for removal of hashtags, links and twitter handles*

1. **Conversion of emojis and emoticons**

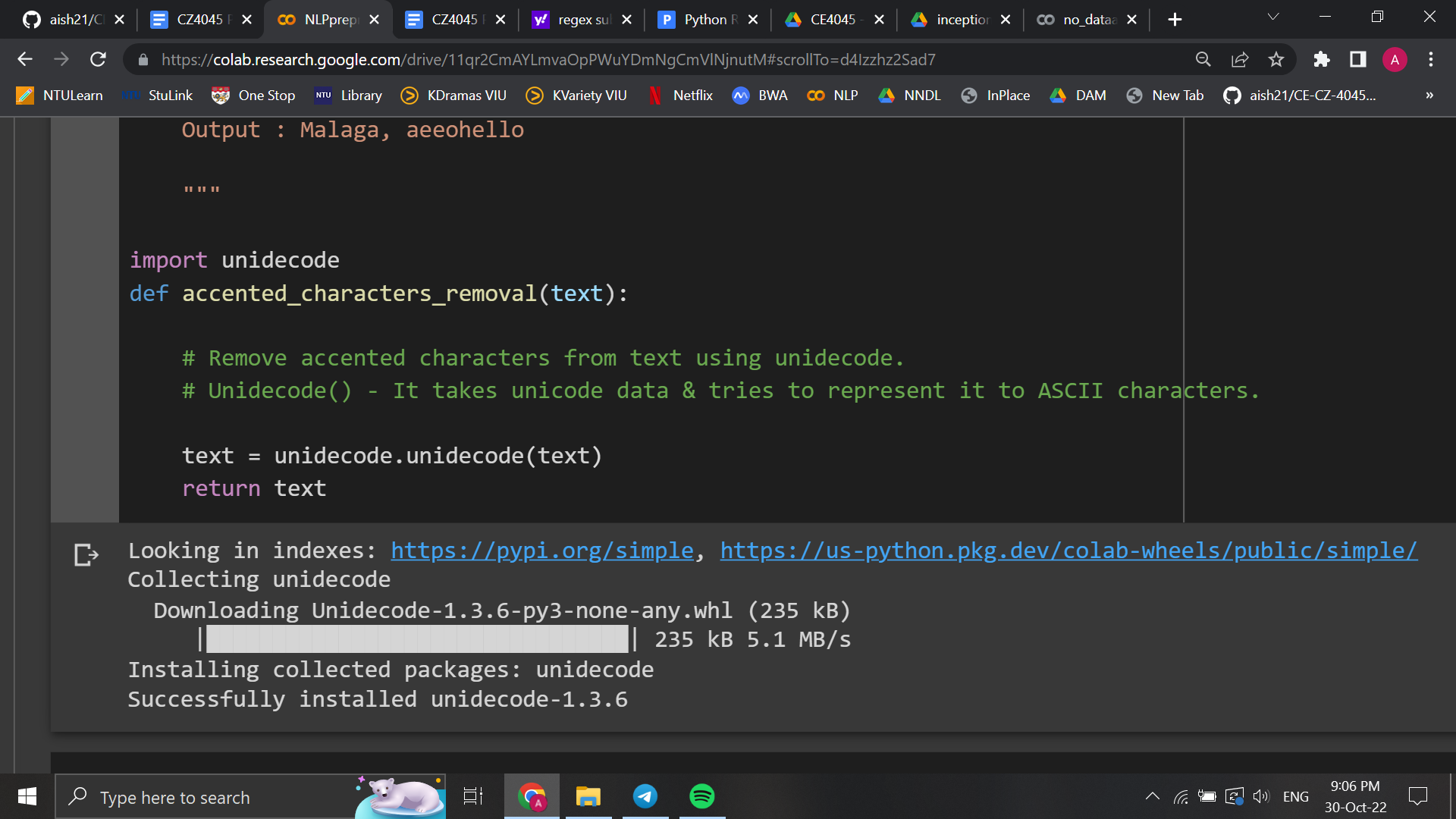
As emojis and emoticons can be an indicator of the sentiment of a tweet, we converted them to text. This would make the processed content more informative with respect to the sentiment. The demojize function of python’s emoji library was used to convert the emojis. We also created a dictionary containing the mapping of emoticons to text, and it was used to convert the emoticons.



*Fig 2.1.4:Code snippet for conversion of emojis and emoticons*

1. **Removal of accented characters**

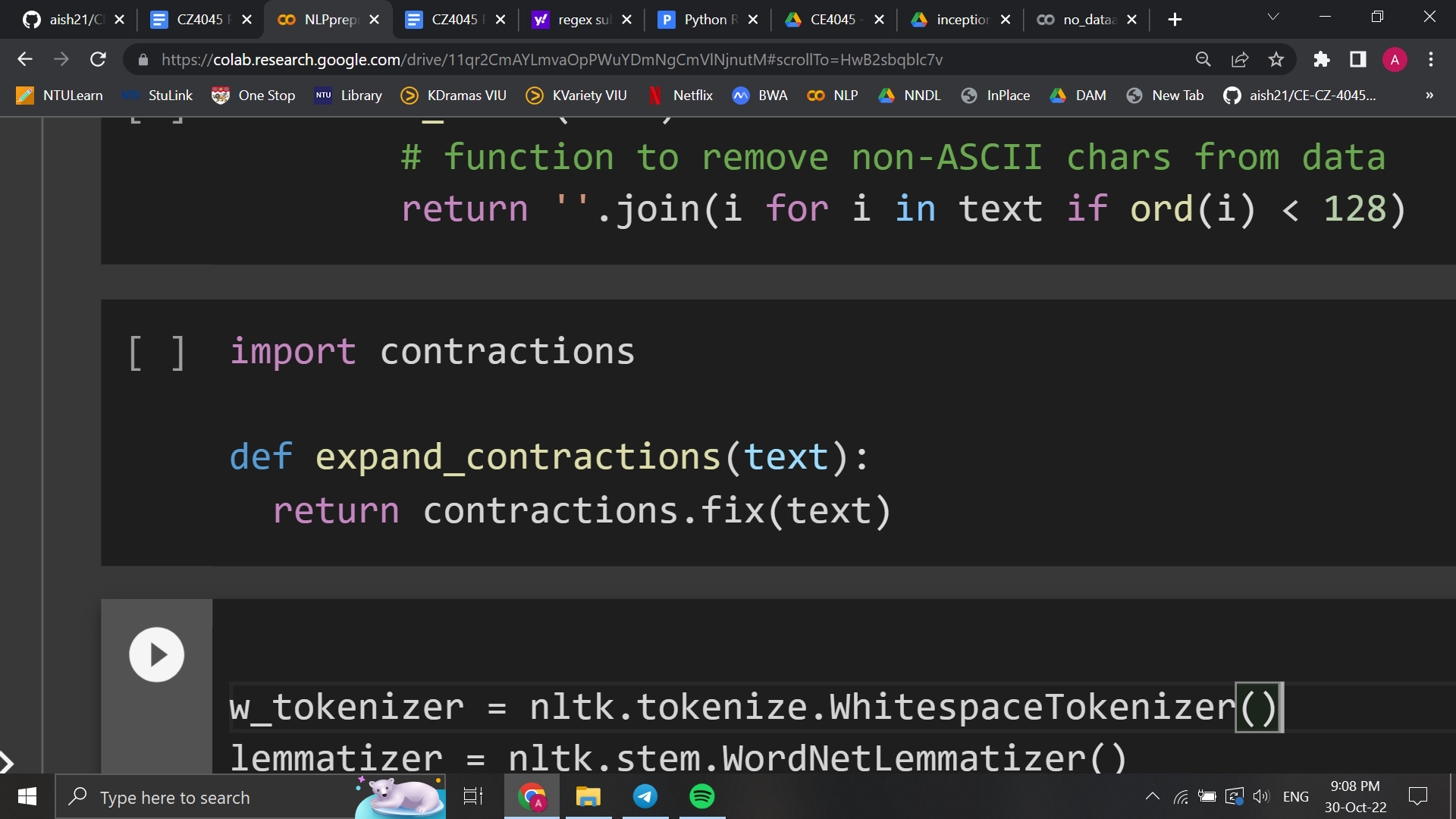
While the use of accents are limited, it is possible for them to occur in a free-text corpus. They can get induced because of keyboard settings or typing style. In order to analyze the English language, it is important that they are removed



*Fig 2.1.5:Code snippet for removal of accented characters*

1. **Expansion of contractions**

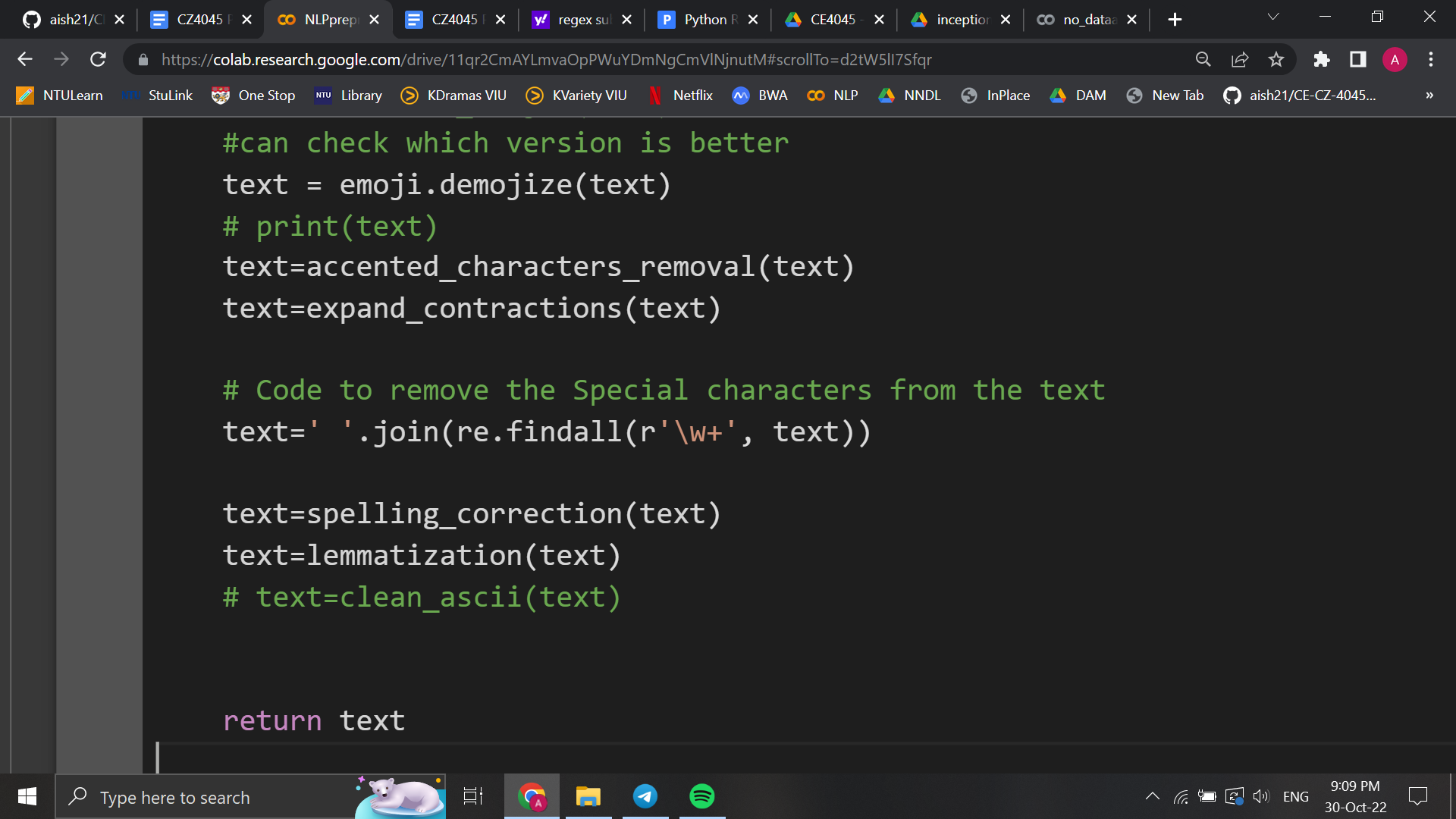
Often multiple words are combined to form a contraction. An apostrophe is used to indicate the missing letters. Contractions in English often exist in written and spoken forms- especially in unstructured data such as tweets. Expanding these contractions to their original form helps with text standardization. This was performed using the python contractions library



*Fig 2.1.6:Code snippet for expansion of contractions*

1. **Removal of special characters**

Non-alphanumeric characters add no value to text understanding and add noise to the algorithm. Thus, these characters have been removed using regex



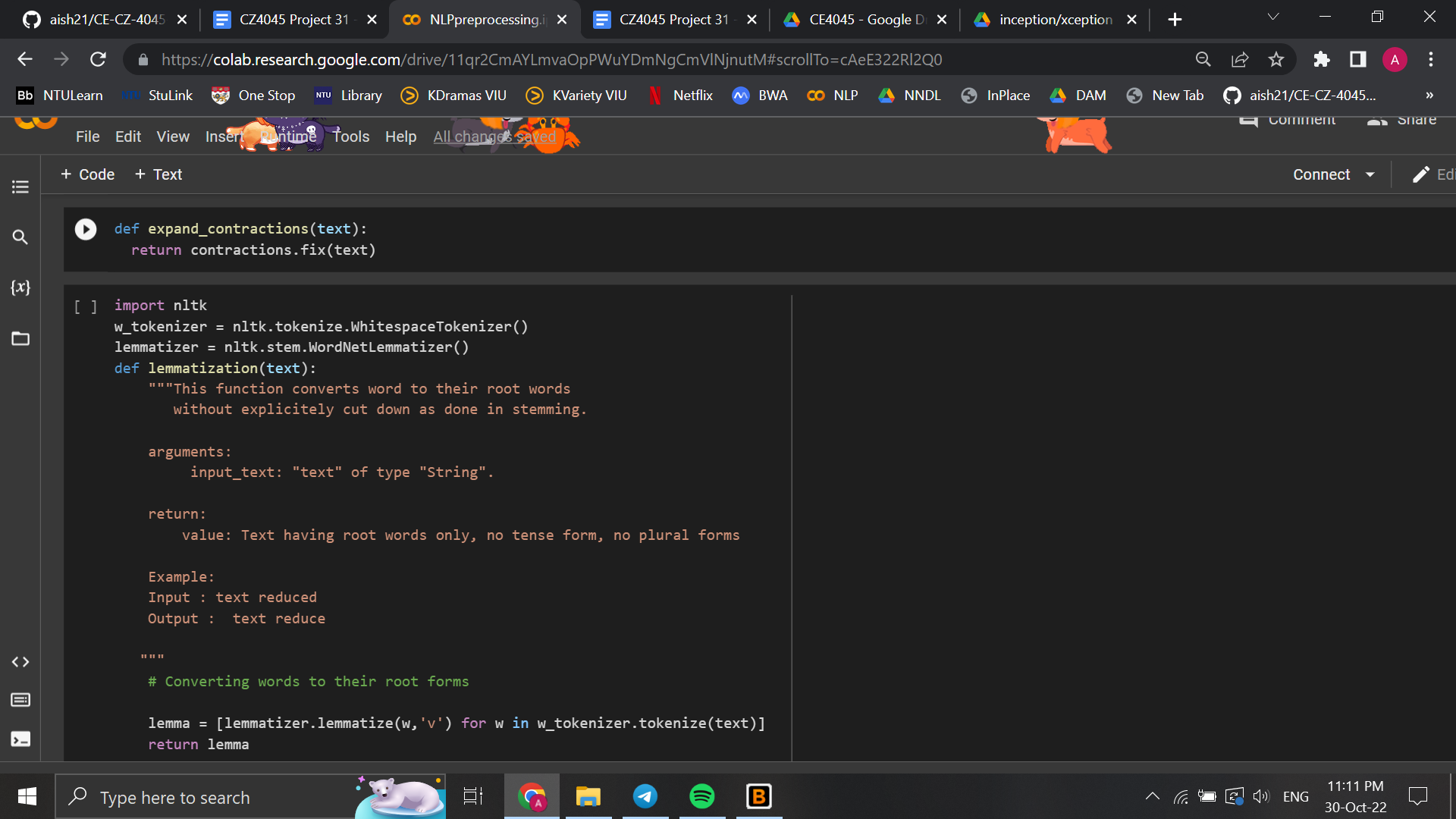
*Fig 2.1.7 :Code snippet for removal of special characters*

1. **Tokenization**

Tokenization is the process of breaking down raw text into smaller chunks. This is done as the meaning of the sentence can be interpreted by analyzing the words in the text. Moreover, this is the first step for lemmatization. Word-level tokenization was performed using NLTK’s Whitespace tokenizer

1. **Lemmatization**

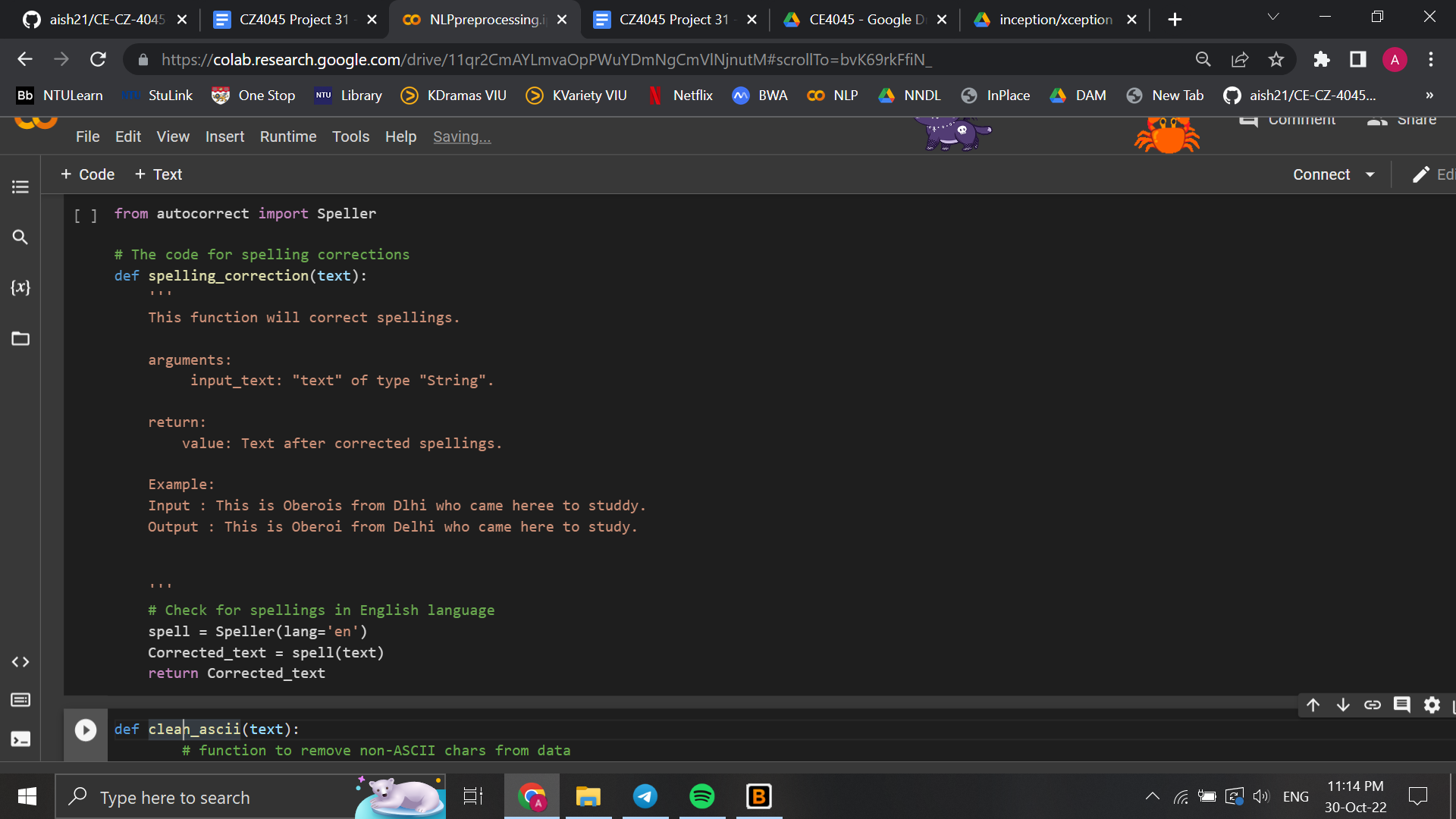
Lemmatization is the process of root form of the inflected words. While lemmatization is slower than stemming, conversion is done properly using vocabulary. Thus lemmatization was chosen over stemming. The Wordnet lemmatizer of python’s NLTK package was used for this task



*Fig 2.1.8 :Code snippet for lemmatization and tokenization*

1. **Spelling Correction**

In tweets, it is highly possible for spelling mistakes to be made. Python’s autocorrect package was used in an attempt to correct these spelling mistakes



*Fig 2.1.9:Code snippet for spelling correction*

While lemmatization and spelling correction have its advantages, they can produce unexpected results in certain cases. Thus, we experimented with different types of pre-processing (primarily with and without spelling correction) to verify its effectiveness

Stopwords were not removed as words such as ‘not’ are important for classifying the sentiment. Removing such words would change the sentiment of the processed sentence

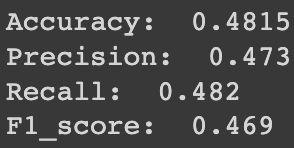
### 2.2 Motivation for Classification Approach

#### 2.2.1 Machine Learning models

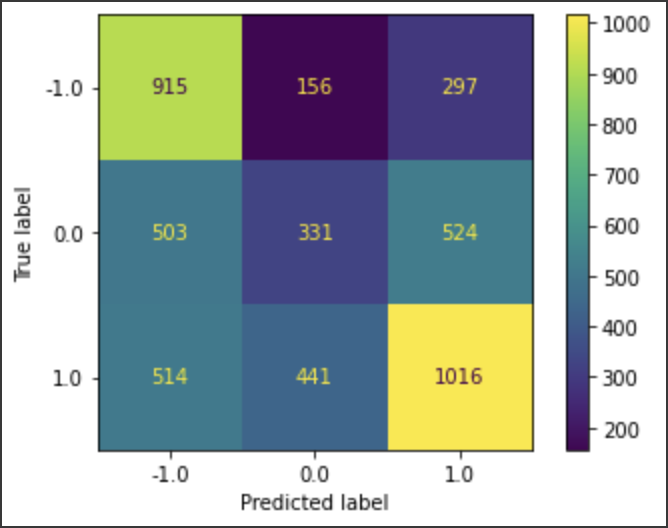
##### Gaussian Naive Bayes

The Naive Bayes classifier separates data into different classes according to Bayes’ Theorem with the assumption that the predictors are independent of each other. The algorithm is quite simple and can thus build very quickly. Moreover, it requires less training data and performs better as compared to other well known classification approaches. As tf-idf vectors have been constructed, the features aren’t discrete and have a continuous value. Thus Gaussian Naive Bayes model have been specifically used

However, the assumption of independence is quite simplistic. It is not completely realistic to assume the features found being independent. Thus, in this case, Naive Bayes performs as a poor estimator. It is also observed that when spelling correction was applied to Naive Bayes, the accuracy decreased slightly.



*Fig 2.2.1.1 : Evaluation metrics for Naive Bayes*

**

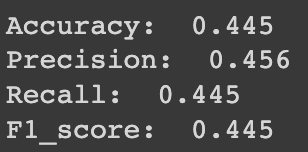
*Fig 2.2.1.2: Confusion matrix for Naive Bayes (without spelling correction)*

##### Random Forest

Random forest is an ensemble of decision tree classifiers. Each tree independently makes a decision and the final class is determined by max voting. Feature randomness and bagging are used to create an uncorrelated forest of trees. Thus, we have trees that are trained on not only different sets of data but also use different features to make decisions. They are thus suitable for dealing with high-dimensional noisy data

However, predictions made by random forest are not as easy to interpret as compared to decision tree classifiers. If the data contains groups of correlated features of similar relevance to output, smaller groups are favored over larger ones. Moreover, they require a significant amount of memory for storage.

After performing a grid search to identify the optimal depth of the tree and the number of estimators, the hyperparameters were set as ‘None’ and ‘50’ respectively



*Fig 2.2.1.3: Evaluation metrics for Random Forest*

**

*Fig 2.2.1.4: Confusion matrix for Random Forest*

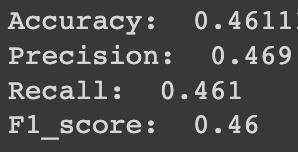
#### 2.2.2 Lexicon-based Sentiment Analysis

##### VADER (Valence Aware Dictionary for Sentiment Reasoning)

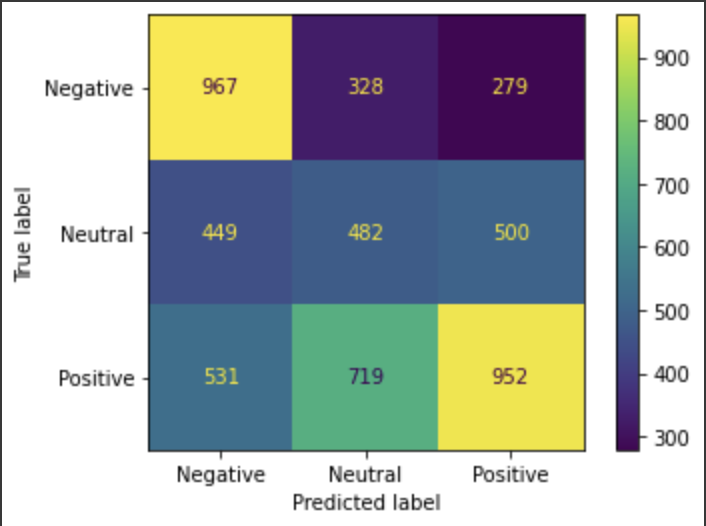
VADER is a lexicon and rule-based feeling analysis instrument that is explicitly sensitive to suppositions communicated in web-based media. VADER utilizes a mix of lexical highlights (e.g., words) that are, for the most part, marked by their semantic direction as one or the other positive or negative. Thus, VADER not only tells us about the Polarity score, it also tells us how positive or negative a conclusion is.

VADER is optimized for social media data and can yield good results when used with data from Twitter, Facebook, etc. Unfortunately, the main drawback with this rule-based approach is that the method only cares about individual words and completely ignores the context in which it is used.

After performing sentiment analysis using the VADER model, we obtained the following results, which again were not too encouraging:



*Fig 2.2.2.1: Evaluation metrics for VADER*

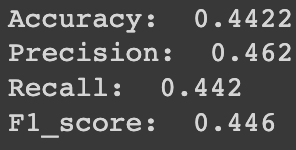
**

*Fig 2.2.2.2: Confusion matrix for VADER*

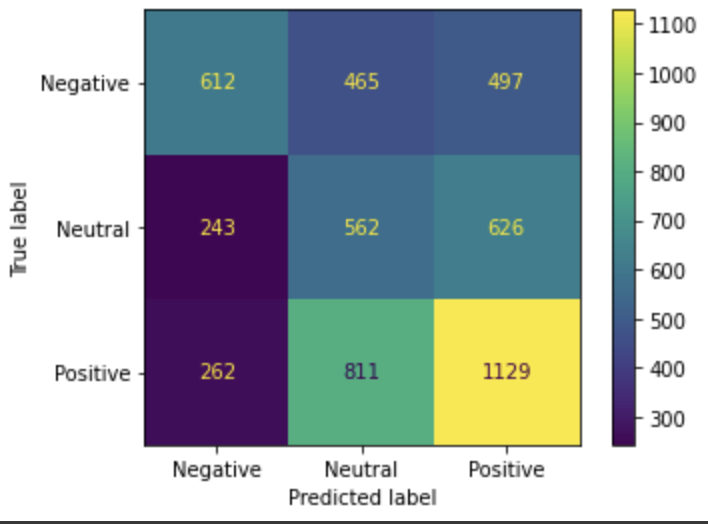
##### TextBlob

Textblob is a Python NLP library that uses the database of Natural Language Toolkit’s (NLTK) WordNet. It uses NLTK because it is simple, easy to deploy, will use up fewer resources, gives dependency parsing, and can even be used for small applications.

Textblob is applicable for complex analysis and working with textual data. When a sentence is passed into Textblob, it gives two outputs which are polarity and subjectivity. There are several different use cases for Textblob including: noun phrase extraction, part of speech tagging, tokenization, word inflection, lemmatization, WordNet integration, and more. Textblob is mostly used to carry out the task of sentiment analysis using its pre-trained inbuilt classifier. Unfortunately, the results for TextBlob in Figure 2.2.2.3 are still unsatisfactory.



*Fig 2.2.2.3: Evaluation metrics for TextBlob*

**

*Fig 2.2.2.4: Confusion matrix for TextBlob*

1. **Subjectivity Detection**

Subjectivity detection is a popular subtask in the Sentiment Analysis domain. Subjectivity/subjective sentences generally refer to personal opinion, emotion or judgment, whereas objective refers to factual information. TextBlob calculates subjectivity by looking at the 'intensity'. Intensity determines if a word modifies the next word. For English, adverbs are used as modifiers.

TextBlob model was used to calculate subjectivity of each tweet where the output lies within [0,1] with 0 indicating almost no opinion and 1 indicating highly opinionated tweet

1. **Polarity Detection**

Polarity Detection is another popular subtask along with subjectivity where Polarity refers to the overall sentiment conveyed by a particular text, phrase or word and polarity detection aims to differentiate the opinion into ‘positive’ and ‘negative’.

TextBlob model was used to calculate the polarity of each tweet where the output lies between [-1,1], with -1 referring to negative sentiment and +1 referring to positive sentiment.

#### 2.2.3 Deep Learning models

##### Bi-Long Short-Term Memory (LSTM)+Attention

*Concept*

LSTM (Long Short Term Memory) is a type of recurrent neural network (RNN) that has been widely used for sequential prediction problems. It is made up of neurons, otherwise known as memory cells, that contain weights and three types of gates: input, forget and output gate. It was developed in order to solve the disadvantages of RNN, hence LSTM is capable of handling long-term dependencies and improves the problem of vanishing gradients (loss with respect to a particular set of weights). LSTM also has an excellent time complexity of O(1) to update each weight. As a result of the property of selectively remembering patterns for long periods of time, we include more regulating parameters and gates that govern the flow and mixing of inputs based on training weights using LSTM, allowing for greater flexibility in managing the outputs.

*Implementation*

After several runs and tests on various neural network architectures, the group decided to go with bidirectional LSTM models. Bidirectional LSTM (Bi-LSTM) enables neural networks to have both backward and forward information about the sequence at each time step. Using bidirectional will run inputs in two directions, one from the past to the future and one from the future to the past, maintaining information from both the past and the future at every point in time. Using knowledge from the future will help the network comprehend what the next word is and hence better forecast the outputs. We required such a network since predicting the word in a phrase coupled with its context made predicting its class easier and helped improve the accuracy.

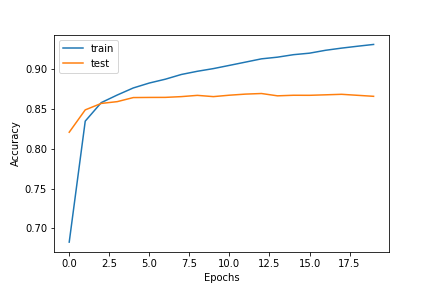
Furthermore, Keras's attention layer is based on the notion of cognitive attention. This effect allows the neural network to concentrate on relevant bits of the input data while fading out the remainder. The idea behind this method is to draw the focus and attention to a tiny yet significant portion of the data. Because they store information in a memory, LSTMs allow RNN to remember inputs over time. Because of this trait, it is critical to emphasize significant sections of the phrase when deciding so that we may shift our focus to vital elements and efficiently employ computer resources across each sentence. The attention class accepts a layer and then performs the necessary functions, building the layer by adding the necessary weights and biases based on the input shape, and then obtaining the required outputs.

To create the final architecture, we introduced an embedding layer and then passed the Bidirectional LSTM layer to the attention class. To provide the best parameter values to the neural network, extensive trials were conducted with hyperparameter tuning techniques to obtain the best model. The hyperparameter technique used was GridSearchCV. This function may be used to loop through supplied hyperparameters and fit the estimator (model) to training data. As a consequence, the best hyperparameters from the list may be picked at the end. The GridSearchCV function is given predefined hyperparameter values. This is accomplished by creating a dictionary in which we mention a certain hyperparameter as well as the values it can take. GridSearchCV uses the Cross-Validation approach to analyze the model for each combination of the dictionary values. The following hyperparameters were tuned -

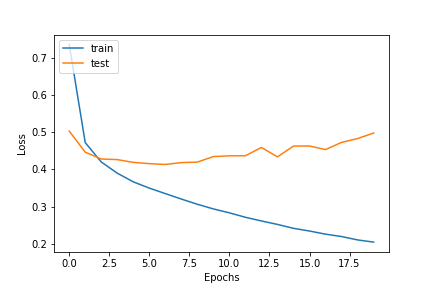
1. **Batch Size:** It is widely acknowledged that employing a large batch size leads to poor generalization. Smaller batch sizes have been shown to have faster convergence and to allow the model to begin learning before viewing all of the data. The drawback of using a smaller batch size is that the model will not always converge to the global optima. The batch sizes available in this project are 512, 256, 128 and 64. These parameters were chosen based on the team's observations of data and analysis of experiments conducted with various model designs. The optimal batch size value that resulted was 512.
2. **Number of Epochs:** One of the significant concerns encountered in the testing described and explained above was overfitting. When the number of epochs used to train a neural network model exceeds the required amount, the training model learns patterns that are very specific to the sample data. As a result, the model is unable to perform well on new data. The number of epochs available in this project is 20, 50, 100, and 200. These parameters were chosen based on the team's observations of data and analysis of experiments conducted with various model designs. The ideal figure for the number of epochs that resulted was 20.
3. **Optimization Algorithms**: The technique of iteratively training the model to achieve a maximum and minimal function evaluation is known as optimization. Given the classification challenge for sentiment prediction, it is vital to select the appropriate optimization technique to meet the demands of the problem statement. During the experimentation, the team thought that the Adamax optimizer would be the best fit for this problem; nevertheless, in order to enhance efficiency, the team opted to make this a variable for GridSearchCV to guarantee that the decision was right. The algorithms used were 'SGD,' 'RMSprop,' 'Adagrad,' 'Adadelta,' 'Adam,' 'Adamax,' and 'Nadam.' The resulting optimization method was named 'Adamax.'
4. **Learning Rate:** The learning rate is a hyperparameter that governs how quickly an algorithm updates parameter estimates or learns parameter values. At the end of each batch, the learning rate decides how much weight is updated. The learning rate values available in this project are 0.001, 0.01, 0.03, 0.05, 0.07, 0.09, 0.1, 0.2. These parameters were chosen based on the team's observations of data and analysis of experiments conducted with various model designs. The optimal value for the learning rate that resulted was 0.01.
5. **Network Weight:** Nodes in neural networks are made up of weighted parameters that are used to build a weighted sum of the inputs. Weight initialization is the process of converting a neural network's weights to small random numbers that serve as the starting point for the neural network's optimization. The network weight initialization for the LSTM Bidirectional layers was completed in this project. The default value is 'glorot uniform,' although the GridSearchCV algorithm also accepted the following values: 'uniform,' 'lecun uniform,' 'normal,' 'zero,' 'glorot normal,' 'glorot uniform,' 'he normal,' 'he uniform.' These parameters were chosen based on the team's observations of data and analysis of experiments conducted with various model designs. The ideal value for network weight initialization that resulted was 'he\_uniform,' which differed from the default hyperparameter value.
6. **Dropout Regularization**: Dropout is a regularization technique that approximates concurrent training of several neural networks. Certain layer outputs are ignored or "dropped out" at random during training. This makes each layer seem and be recognized as having a distinct number of nodes and connection to the prior layer. The dropout regularization values available in this project are 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9. These parameters were chosen based on the team's observations of data and analysis of experiments conducted with various model designs. The resulting optimum value for the dropout regularization came out to be 0.5.
7. **Embedding Size and Input and Output Dimensions**: These hyperparameter settings were essential in addressing the overfitting problem seen in the studies. They were tweaked using a trial and error process, in which the team began with certain beginning values based on the data and results collected from the studies. Based on the criteria and observable improvements in model performance, these variables were increased and decremented. Finally, the values chosen were (256, 128) for the embedding layer and 256 and 128 units for the two LSTM layers.

*Evaluation metrics:*

Accuracy and loss plots were plotted for the models. The results shown below were the best obtained plots after tuning the hyperparameters. It can be seen that there is still a significant amount of overfitting taking place in the model.

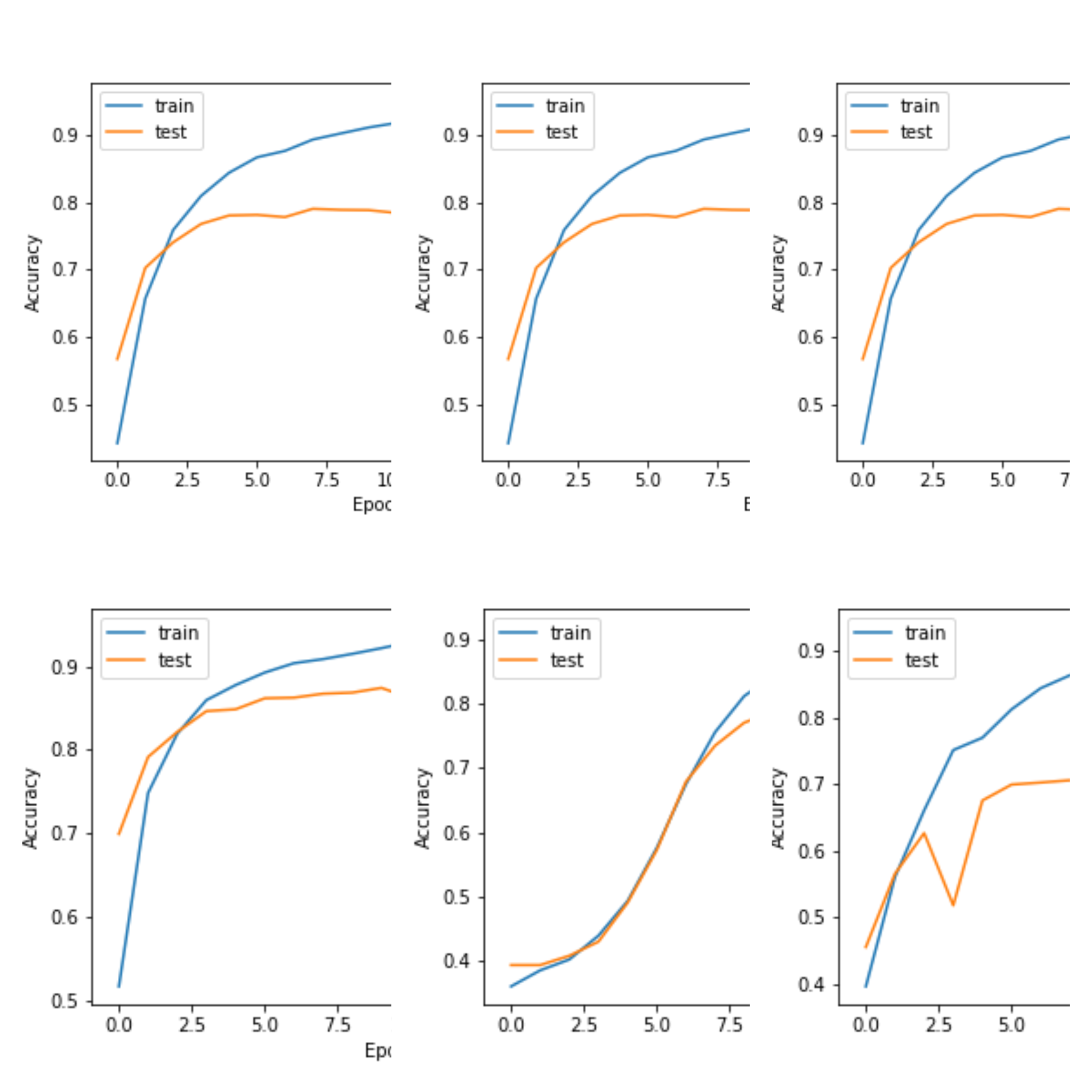


*Fig 2.2.3.A.1:Accuracy Plot*

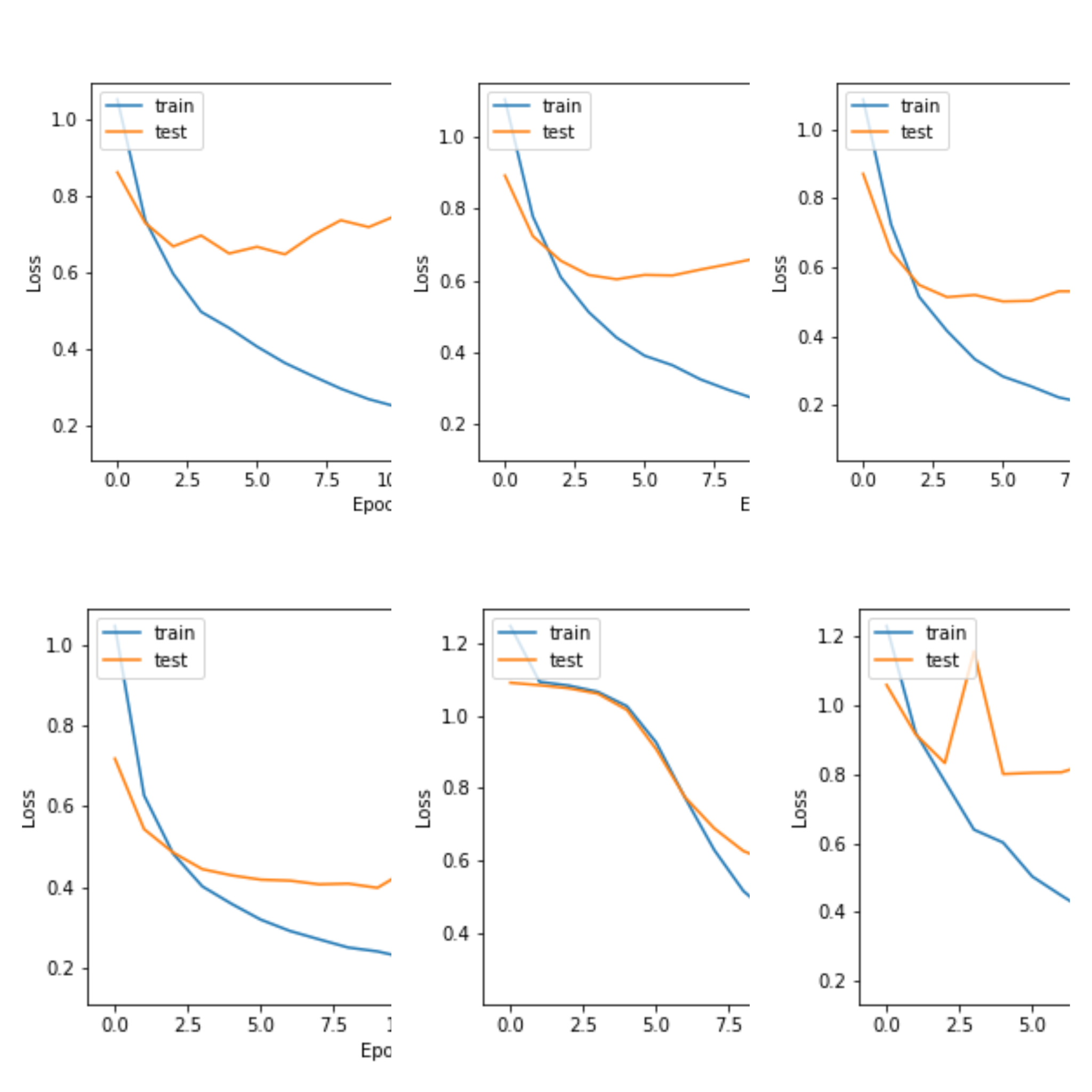


*Fig 2.2.3.A.2:Loss Plot*

In order to test our model further, we decided to apply the model on individual team dataframes to see if that affects the results -



*Fig 2.2.3.A.3:Accuracy plots of individual team models*

**

*Fig 2.2.3.A.4:Loss plots of individual team models*

Despite testing the models on different teams, only Manchester City data seemed to perform the best out of all the teams (when compared to the overall model) based on the accuracy and loss plots. It is evident from the plots that despite exhaustive methods to tune the model, the best model too overfits to a great extent.

##### RoBERTa Embeddings

*Concept:*

RoBERTa is a retraining of the BERT model with improved training methodology. The key innovation of the BERT model was the application of bidirectional training of Transformer architectures to language models. This bidirectional training aims to learn a deeper sense of language context and flow in contrast to previous efforts which looked at a text sequence in a single direction.

BERT also incorporates other NLP methodologies from the architecture of Transformers such as the Attention mechanism which allows the model to place more weightage on critical words of a sentence to predict the next one. BERT also makes use of Masked Language Modeling (MLM), where 15% of the words in each text sequence is “masked” or hidden. During training, the model tries to predict the original value of the masked words. Lastly, BERT makes use of Next Sentence Prediction (NSP) during the training phase, where the model receives pairs of sentences as input and learns to predict if the second sentence is the subsequent sentence in the original document.

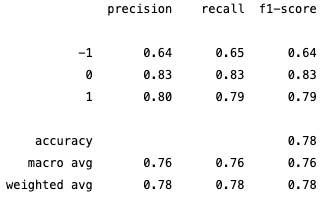
To improve this training procedure, RoBERTa removes the NSP mechanism and introduces dynamic masking so that the masked words change during model training. More importantly, RoBERTa uses 160 GB of text for pretraining from various sources. These innovations result in a pre-trained model that improves on BERT.

*Implementation approach:*

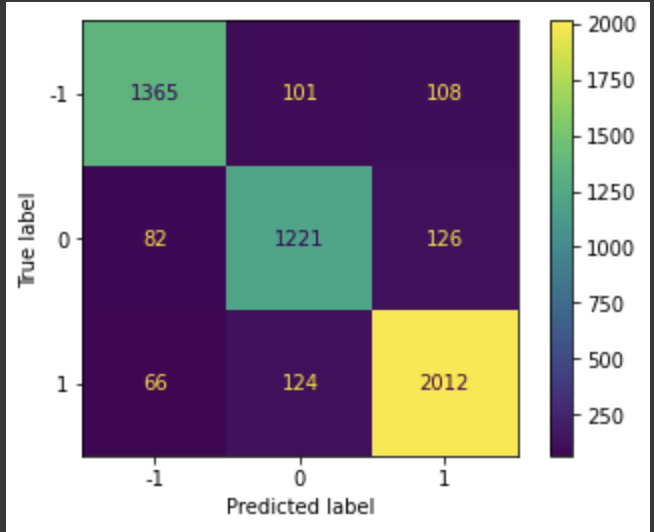
In order to use the RoBERTa model for sentiment classification, we add a FeedForward layer followed by a fully connected layer after the RoBERTa transformer output for classification.

In our project, we use a RoBERTa-Base model which is further pretrained on twitter text data. After pretraining, we found it easier to fine-tune it on our smaller dataset of premier league tweets.

*Evaluation metrics:*



*Fig 2.2.3.B.1: Evaluation metrics for RoBERTa*

**

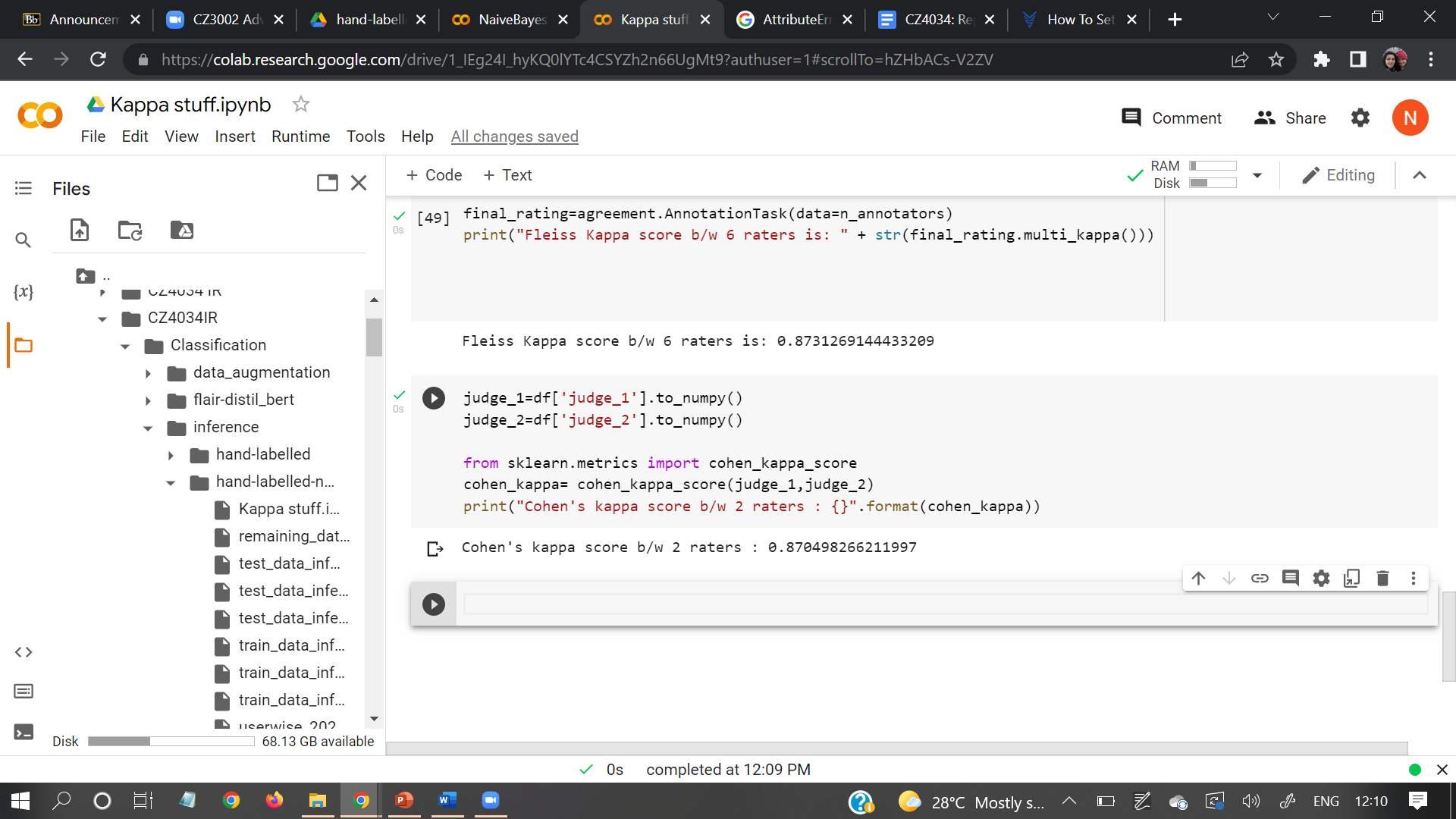
*Fig 2.2.3.B.2: Confusion matrix for RoBERTa*

*Conclusion:*

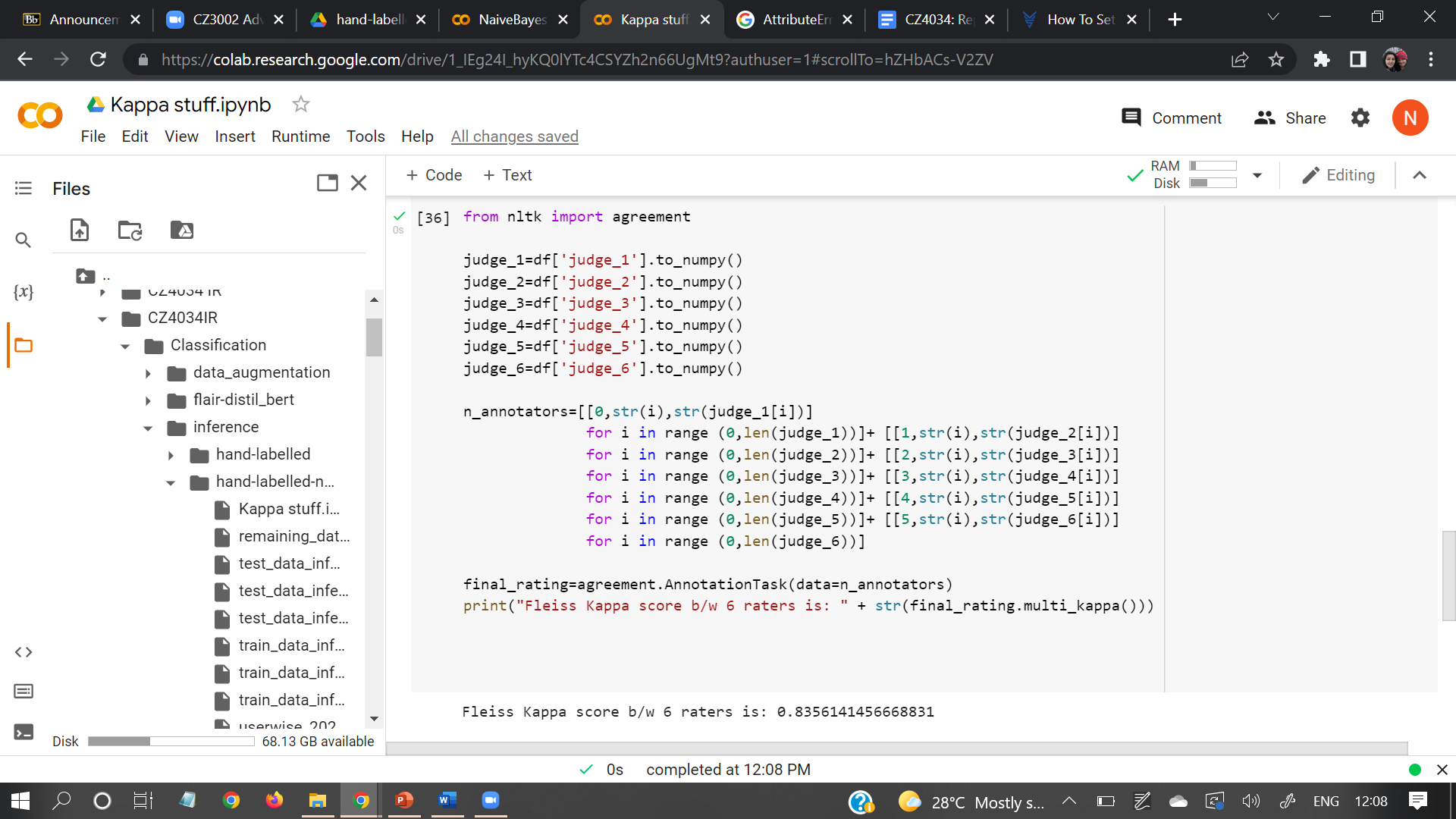
1. The RoBERTa model performance was significantly better than all other models used which goes on to show the accuracy and effectiveness of Bi-directional LSTM and attention layers in classifying text.
2. Further innovations can be made to this RoBERTa model by fine tuning its hyperparameters and making use of transfer learning which is discussed in the Innovation section

### 2.3 Building Evaluation Dataset by Manually Labeling

We manually labeled 10% of the entire tweets dataset for the evaluation dataset. This was performed by all 6 team members and we have ensured that the inter-annotator agreement was above 80%. Inter-annotator agreement is a measure of how well two (or more) annotators can make the same annotation decision for a certain category. We have used Cohen kappa and Fleiss kappa to calculate the kappa score between 2 judges as well as among 6 judges respectively.

****

*Fig 2.3.1:Cohen’s kappa score calculation*

****

*Fig 2.3.2: Fleiss’s kappa score calculation*

As the kappa measure is approximately 0.83, which is above 0.8, our evaluation dataset is considered as a good inter-annotator agreement

### 2.4 Evaluation metrics and discussion of results

| **Models** | **F1** | **Precision** | **Recall** | **Accuracy** |
| --- | --- | --- | --- | --- |
| Gaussian Naive Bayes | 0.469 | 0.473 | 0.482 | 0.4815 |
| Random Forest | 0.445 | 0.456 | 0.445 | 0.445 |
| VADER | 0.460 | 0.469 | 0.461 | 0.4611 |
| TextBlob | 0.446 | 0.462 | 0.442 | 0.4422 |
| Bi-LSTM+Attention Neural Network | 0.506 | 0.335 | 0.506 | 0.386 |
| Roberta Model & Embeddings  *(with Enhancements)* | 0.78 | 0.78 | 0.78 | 0.78 |

*Table 3: Summary Table of Models and Respective Evaluation Metrics*

### 2.5 Discussion of performance metrics

#### 2.5.1 Average Tweet Length

The average length of a tweet in our dataset is 66 words.

#### 2.5.2 Average Record Classification Metrics

We look at the time taken to run our selected LSTM+Attention Neural Network, for inference on the tweets and report the time taken to preprocess the tweets and make a prediction. These operations were performed on 5207 records.

| **Operation** | **Time to Run (sec)** | **Records/sec** |
| --- | --- | --- |
| Preprocessing | 240 | 122 |
| Inference | 198 | 7 seconds per epoch |

*Table 4: Time taken for Inference*

### 2.6 A simple UI for visualizing classified data

Instructions to run:

1. Install streamlit

pip install streamlit

python -m pip install streamlit

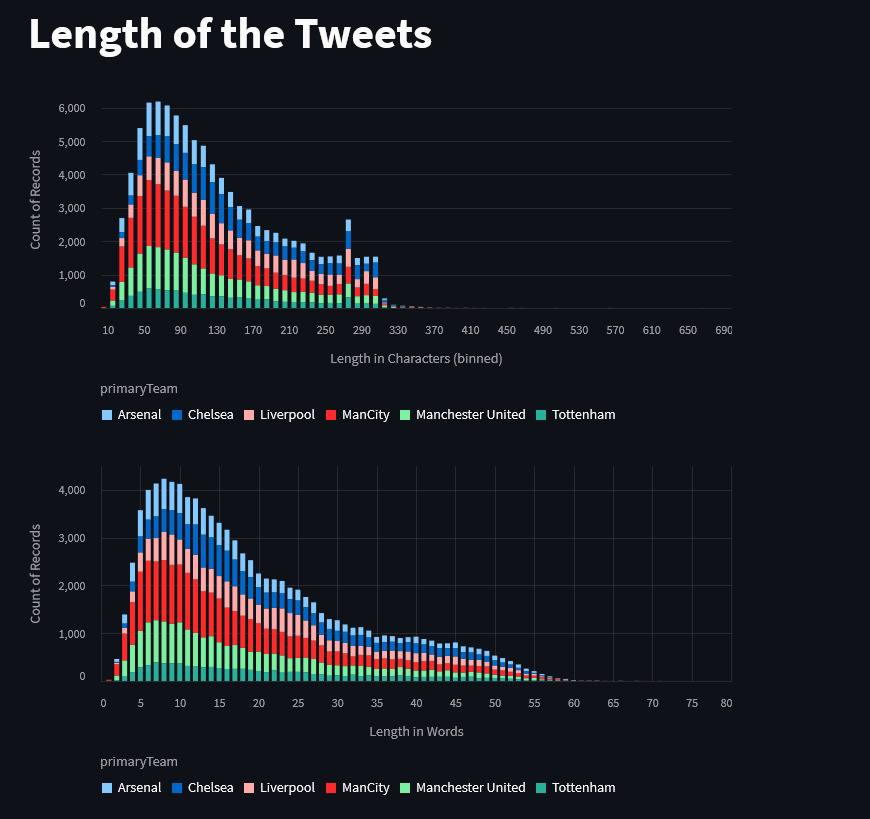
1. Navigate to the UI directory
2. Run streamlit

streamlit run .\main.py

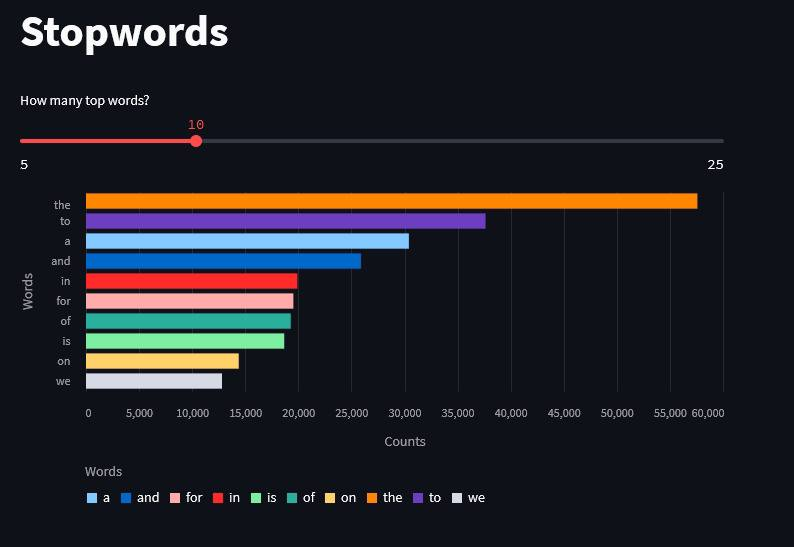
The visualization will open in your browser



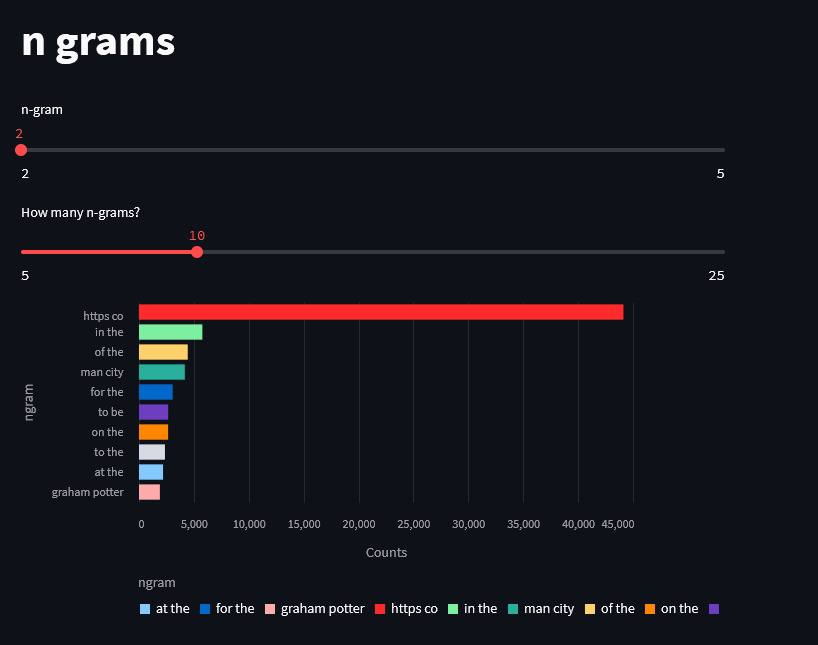
*Figure 2.6.1: UI of ratio of tweets by team*



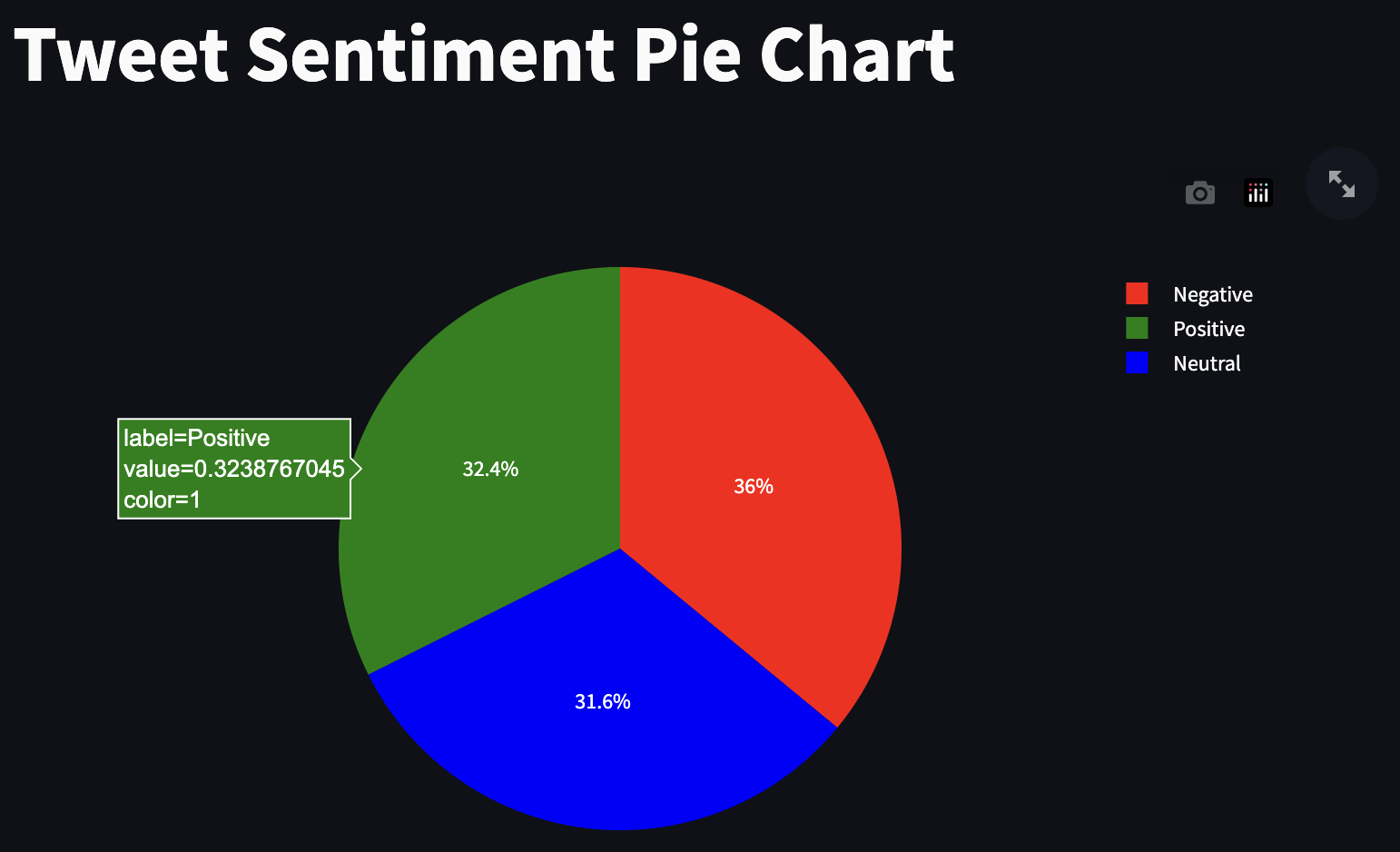
*Figure 2.6.2: UI plot of length of tweets*



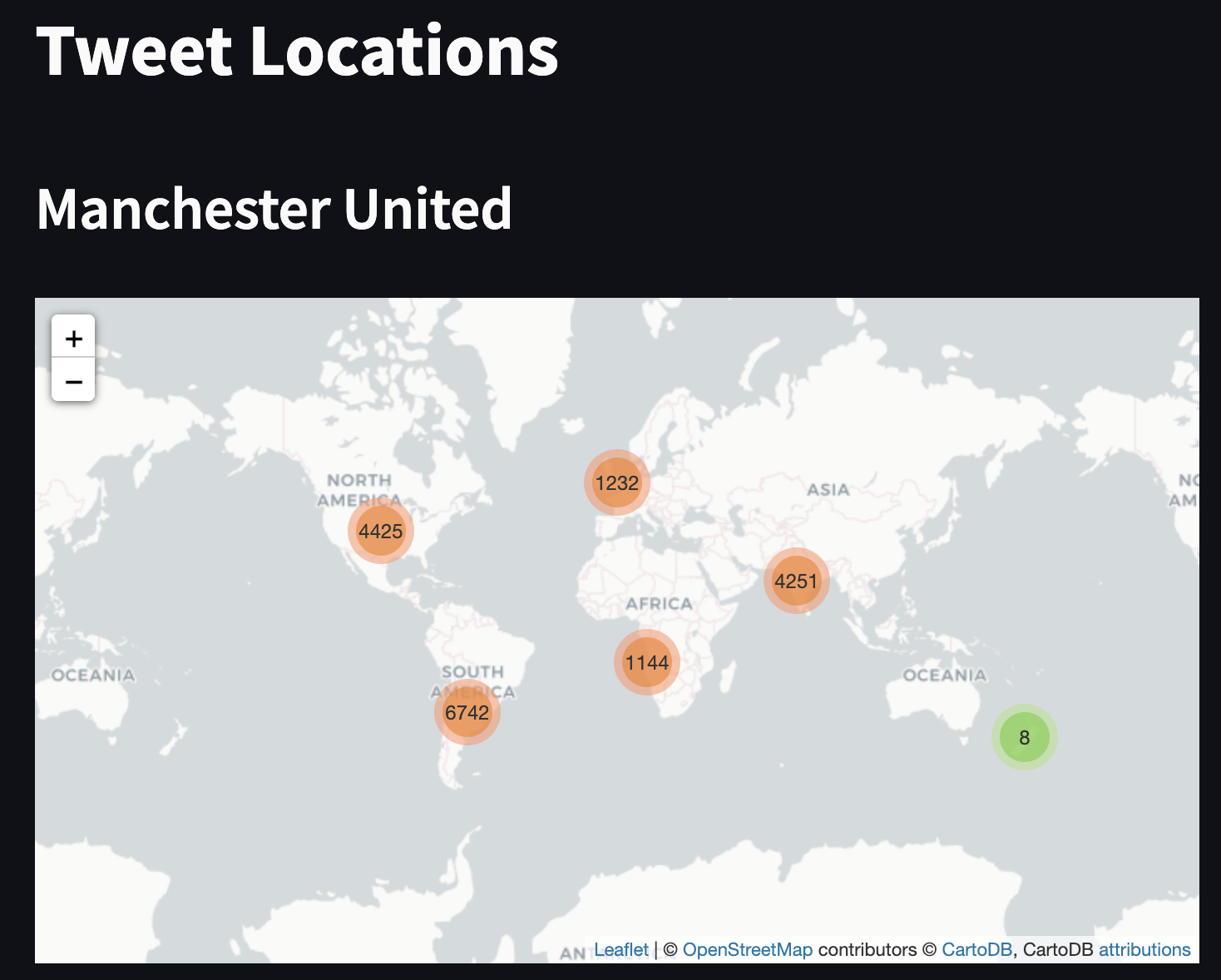
*Figure 2.6.3: UI plot of stopwords*



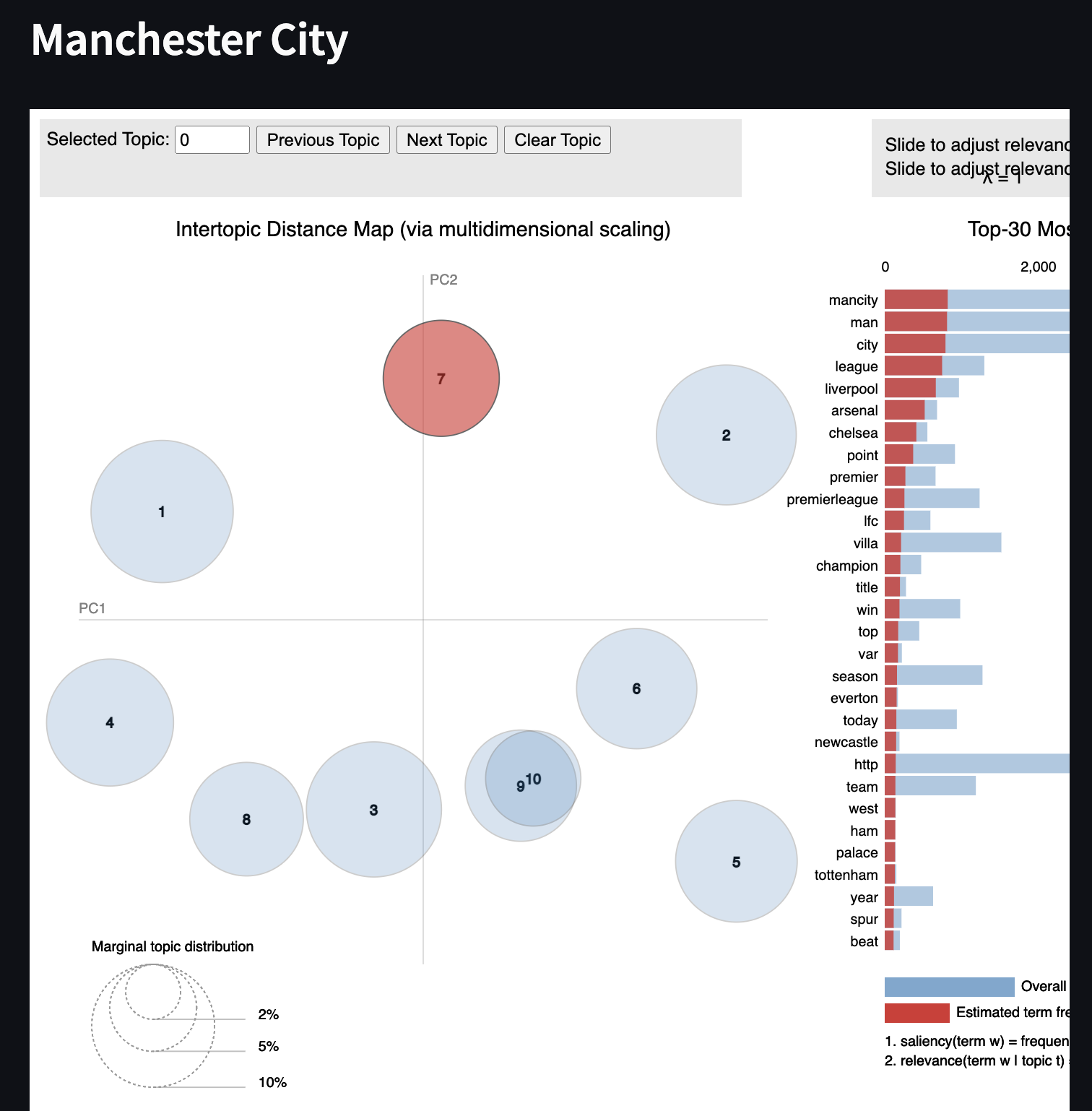
*Figure 2.6.4: UI plot of n-grams*

**

*Figure 2.6.5: UI Pie chart of tweet sentiments*

**

*Figure 2.6.6: UI map of tweet locations*

**

*Figure 2.6.7: UI LDA plot of Manchester City*

## Section 3: Explore some innovations for enhancing classification

### 3.1 Potential Reasons for Misclassification

By analyzing the classification results obtained by the models we explored earlier, we can say that the reasons for misclassification could be:

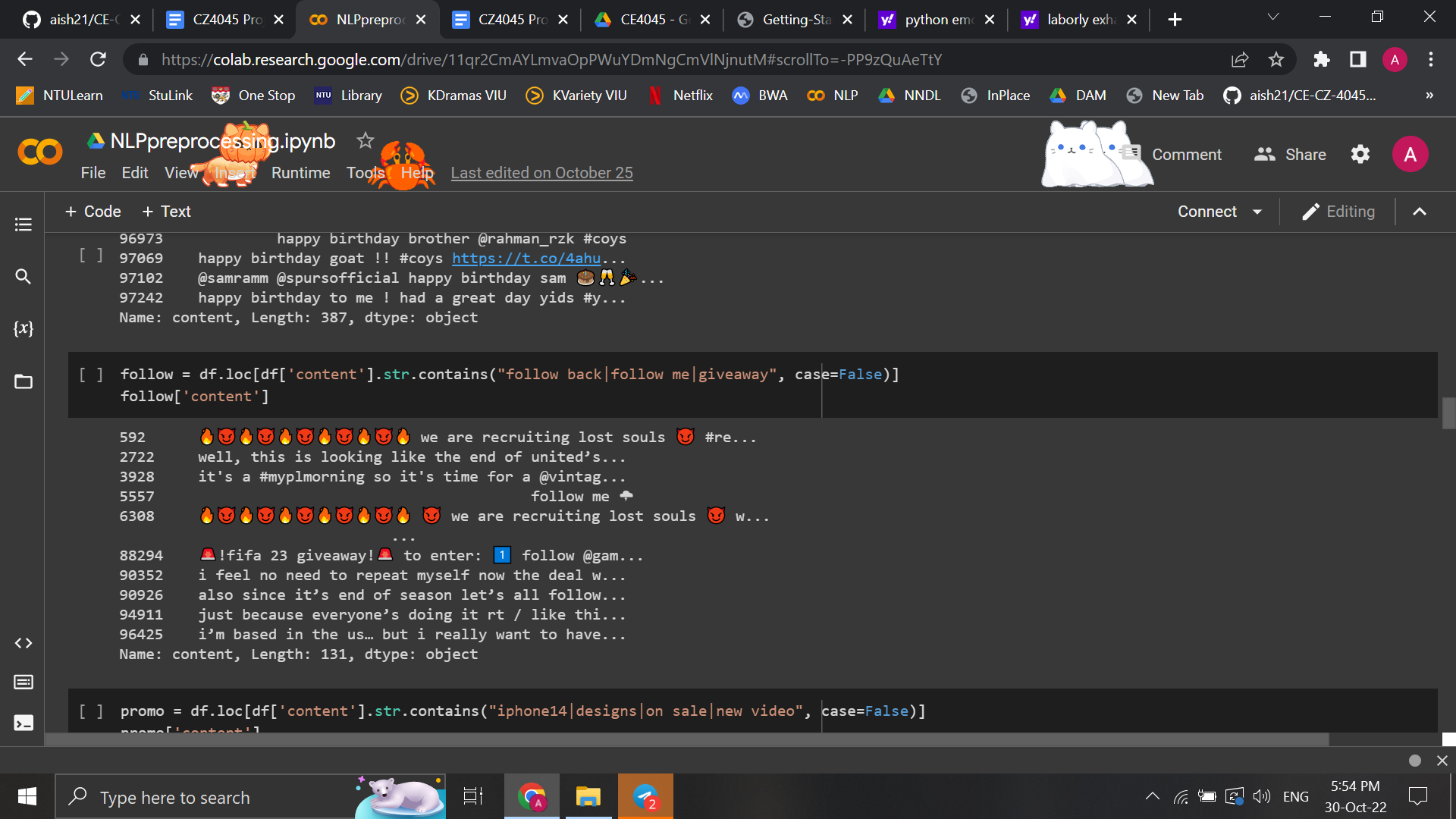
1. Similarity in the structure of tweets

This means that there could possibly be similar words found in the three classes of tweet. This causes difficulty for the model to classify unseen tweets, it is not able to grasp the key idea from the training data, thus it ends up misclassifying tweets.

To confirm this, we will perform Principal Component Analysis which is later discussed in Section 3.3.

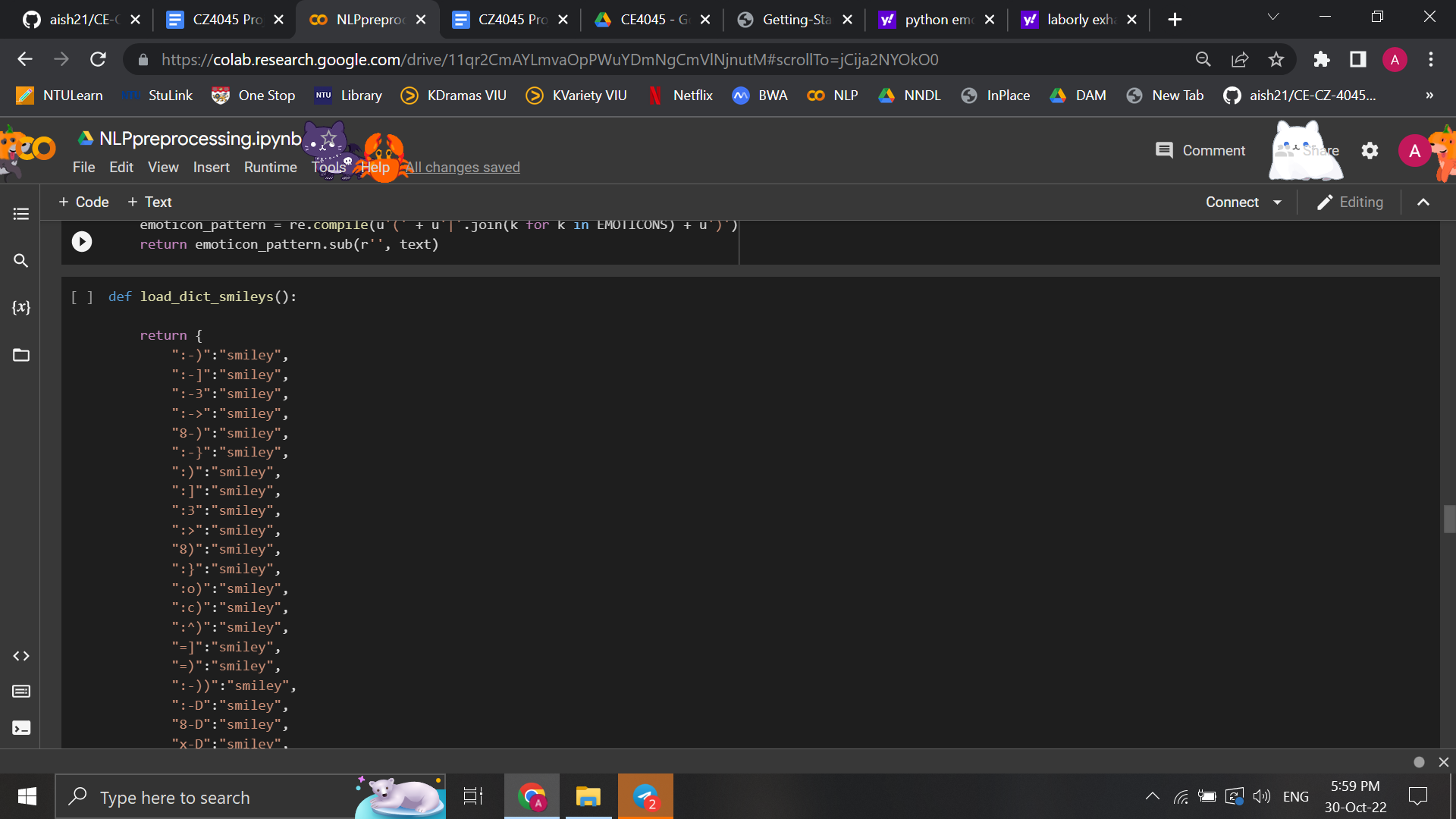
### 3.2 Innovations during Preprocessing

A simple innovation we added to the preprocessing step would be the implementation of our own algorithm to remove irrelevant keywords and emoticons. During the process of manually labeling tweets, we came across a lot of unrelated topics and even advertisements which are exhaustive to remove one by one. Hence, we selected the common keywords of irrelevant tweets and created a dataframe of tweets containing them as shown in Figure 3.2.1. We carefully scanned through the content of each data frame to ensure that we are not risking any loss of meaningful tweets.



*Figure 3.2.1: List of tweets containing keywords related to promotions*

In addition, we believe that emoji and emoticon handling is essential as removing them would lose valuable information on the sentiment of the tweet. As mentioned in Section 2.2 part 4, we created an emoticon dictionary to convert emoticons to a word that describes the emoticon. For instance, ":)" is converted to “smiley" and ":(" is converted to “sad", Figure 3.2.2 shows a part of the emoticon dictionary. Converting the emojis and emoticons into readable information helps improve the accuracy of the analysis.



*Figure 3.2.2: Small part of emoticon dictionary*

Furthermore, we assessed the effect of spelling correction as there may be possibilities in which it changes a correct word into an unintended one. We found that the tweets contained a significant proportion of Named-Entities which are not part of the English vocabulary and are assumed as being misspelled by NLTK’s spell-checker library. For instance, the word ‘Cristiano’ referring to the football player ‘Cristiano Ronaldo’ (aka GOAT) is used widely in tweets about Manchester United but this was being corrected by the spell checker to ‘Christian’ which completely changes the meaning/sentiment of the tweet. Similar issues were found with many other player/club names and acronyms like ‘Haaland’, ‘coyg’ (acronym used frequently for ‘Come on you gunners’), ‘nld’ (acronym used frequently for ‘North London Derby’), and etc. After observing these wrong corrections made by the spell-checker, we decided not to perform spelling correction during preprocessing.

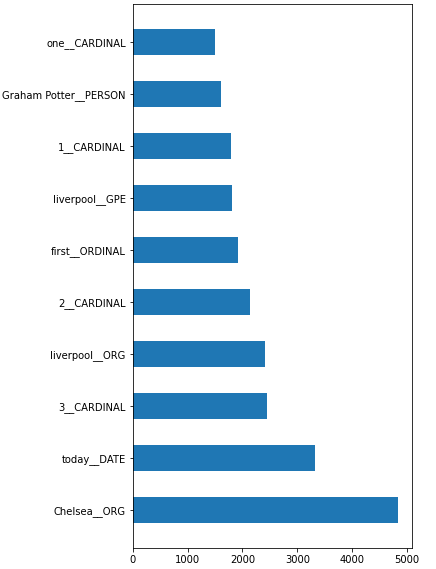
### 3.3 Named Entity Recognition

*Concept:*

Named Entity Recognition (NER) is the process of recognising and categorizing essential information (entities) in a text. An entity can be any word or series of words that consistently refers to the same idea or concept. Detected entities are classified into different categories based on the part of speech to which they belong as well as other context. Some categories are: cardinal (one, two, three) and ordinal (first, second, third) numbers; organization (ORG); geo-political entity (GPE); time; date; person; etc.

*Implementation approach:*

1. Use SpaCy to classify the pre-processed data. SpaCy is a pre-trained model that facilitates NER.
2. Count the number of occurrences of each named entity plus category and plot the ones that occur most commonly



*Figure 3.3.1*

*Conclusion:*

The SpaCy model is far from perfect and mislabels some entities. For example, Liverpool is correctly identified as an organization in 2,415 instances, but in 1,805 instances the model classifies Liverpool as a geo-political entity. Since the classification accuracy of NER is quite low, we ultimately decided to not use this model.

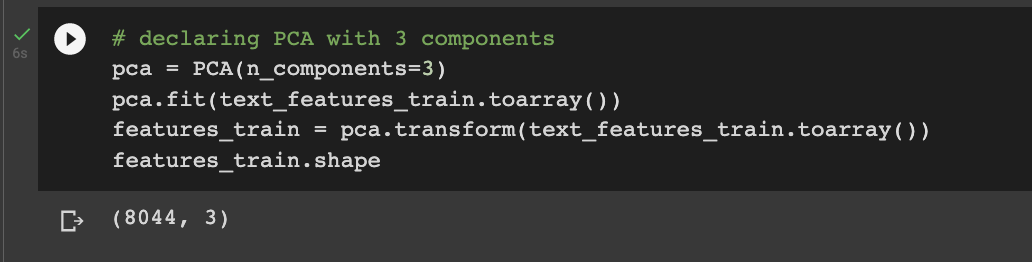
### 3.4 Principal Component Analysis

*Concept:*

Principal Component Analysis is a technique to reduce the number of dimensions in a dataset while trying to retain the most information. By using correlation between the dimensions it tries to provide the maximum number of variables that keeps the maximum amount of information out how the original data is distributed.

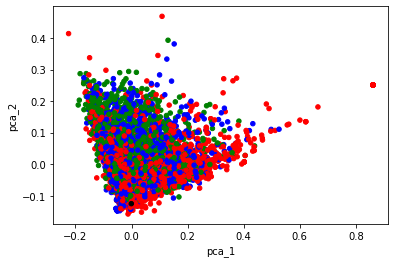
*Implementation approach:*

1. Convert the textual train data to TF IDF (Term Frequency and Inverse Document Frequency) vectors. These word frequency scores help in highlighting the words that are more ‘interesting’ (i.e., more frequent) in a particular tweet.
2. Fit and predict the vectors obtained on PCA, with n\_components set to 3. This means that we are trying to get the first three principal components of the data. The shape of the ‘features\_train’ variable tells us that for the 8044 tweets in the tr+aining dataset, we have generated the first 3 principle components.



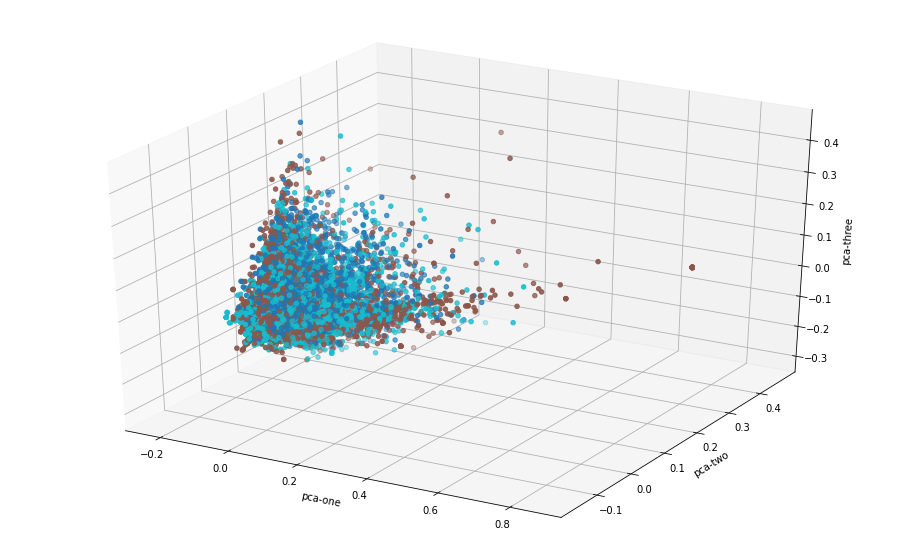
*Fig 3.4.1: Generating PCA components by fitting the model on our data*

1. Plotting a 2D visual distribution of the 3 classes of sentiment captured by the first 2 PCA components (as red, blue and green for 0, 1, -1):



*Fig 3.4.2: Visualization of the 3 sentiment classes by extracted PCA components*

1. Plotting a 3D visual distribution of the 3 classes of sentiment captured by the 3 PCA components as teal, blue and brown dots below:



*Fig 3.4.3: Visualization of the 3 sentiment classes by extracted PCA components (3D)*

*Conclusion:*

1. PCA plots highlight the similarity in the clusters of samples.
2. We notice that in the above plots, the 3 classes are overlapping, meaning that they have very similar words being used in the tweets.
3. The outliers in these figures are removed by plotting individual barplots for each PCA component and eliminating few of the extreme values.
4. This may also be happening since we have a data imbalance
   1. Since there is a higher number of tweets for the neutral class, the model is learning it too well.
   2. Thus, if it finds even a few similar words in the positive and negative class to the ones in the neutral class, it might place them in the same cluster as the neutral class.

### 3.5 Transfer Learning and Model Fine-Tuning

*Concept:*

Deep learning models generally achieve the best results for sentiment classification, but they do so only when trained on relatively larger datasets sampled from the target domain. This is probably due to the large number of trainable parameters in these models. Similarly for our case, our manually built neural network gave us the best results, however we wanted to attempt a few different innovations on the state of the art sentiment analysis techniques. This is in order to find the most optimal method of performing sentiment analysis for cases where there might not be enough data to train a deep learning model.

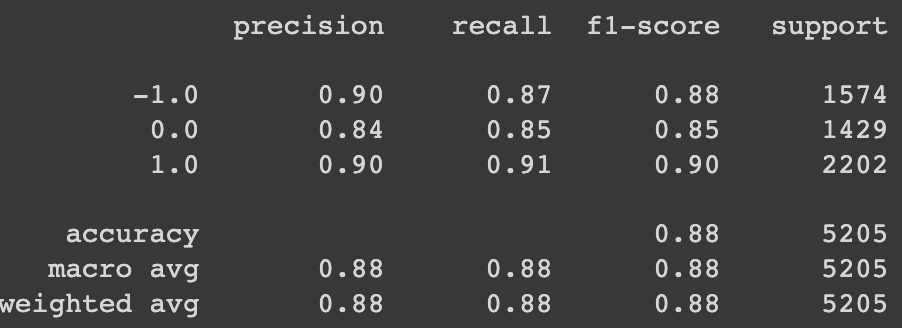
Transfer learning can be beneficial in such scenarios. It is applied by first pretraining the deep learning model on a large dataset from a different source domain, such that the pretrained model can then be fine-tuned on the dataset from the target domain. In doing so, the model is able to converge through training on fewer data instances than it would require when training from scratch.

*Implementation approach:*

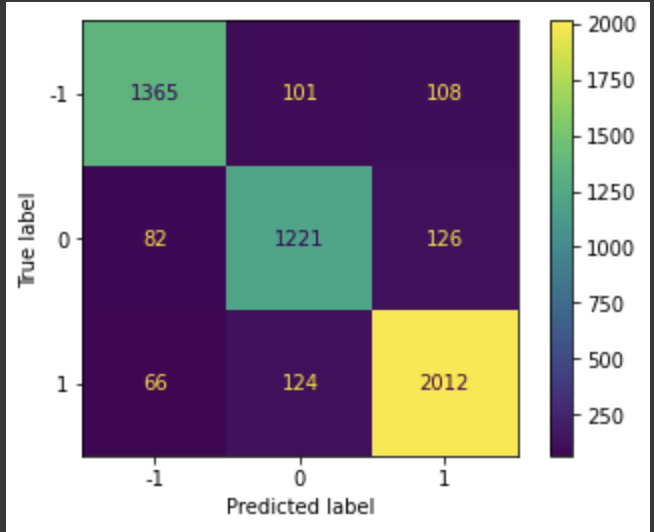
1. We begin with the RoBERTa-base model which is pretrained on a dataset composed of 160 GB of text data from five different sources.
2. The RoBERTa-base model is further fine-tuned for sentiment analysis on 58 million tweets acquired from the TweetEval dataset. We fine-tuned the model for 5 epochs using the SGD optimizer with learning rate decay with a small base learning rate of 1e-3.
3. The resulting model from part ‘b’ is then further fine-tuned on our hand-labeled dataset containing premier league tweets and sentiment. In this final step, we train the model for 15 epochs using the SGD optimizer with learning rate decay with a base learning rate of 1e-4 and momentum of 0.9.

*Conclusion:*

1. We found that the RoBERTa model with transfer learning resulted in a better performance on our evaluation dataset than all other models detailed in this report.
2. Using the transfer learning approach, we were able to achieve a 10% increase in our overall accuracy on the evaluation dataset as opposed to directly training the RoBERTa-base classifier on our training dataset as shown in the figure below.



*Figure 3.5.1: RoBERTa Model model with transfer learning evaluation metrics*

**

*Figure 3.5.2: RoBERTa model with transfer learning confusion matrix*

## Conclusion

We have successfully built a sentiment analysis model to analyze tweets regarding the top six Premier League teams. We built six different models and evaluated each of their performance on our preprocessed corpus. We discovered that RoBERTa Embeddings was our best model, reaching an accuracy of 78% as well as an F1 score of 0.78. Furthermore, we implemented innovative solutions to improve the models. The main innovations include principal component analysis, transfer learning along with hyperparameter fine tuning and NER. When we performed transfer learning on the RoBERTa model, this accuracy increased by 10%. We ensured Cohen’s and Fleiss’s kappa score was above 80% for our evaluation dataset to have a good inter-annotator agreement. Moreover, we built a UI using streamlit for better visualization of our classified data. We hope that this report will pique interests to extend and improve accuracy of our models for further research in the natural processing language field.

## Bibliography

The following table will consist of the URL to the various resources used in the report including the project youtube video link and the google drive links to the our analysis data, analysis results and source codes

| **Name** | **URL** |
| --- | --- |
| Corpus Webpage (Twitter) | <https://www.twitter.com/> |
| Documentation for Crawling (Scraping) Tool - SNScrape | <https://github.com/JustAnotherArchivist/snscrape> |
| Documentation for Classification Model - Roberta | <https://huggingface.co/docs/transformers/model_doc/roberta> |
| Link to Folder with all deliverables (Google Drive Link) | <https://drive.google.com/drive/folders/1pUDxH7emgjuILKIhbKrqfkWnqqewOsbm?usp=share_link> |
| Crawling Source Code & Data (Google Drive Link) | <https://drive.google.com/file/d/13yK5jfeROekROU7ZQfg1ZEl8wHUgTK7B/view?usp=share_link> |
| Classification Source Code & Data (Google Drive Link) | <https://drive.google.com/file/d/10YTQC1UWr1ZuBUngQe5BgtLb0Bc1-7gD/view?usp=share_link> |
| Youtube Link to Our Video Presentation | <https://youtu.be/Hkk8Csxahaw> |

*Table 5: Bibliography and resources*

## Work Distribution

| **Members** | **Work Responsibility** |
| --- | --- |
| Akshat Sharma | Data Labeling, Crawling, Preprocessing, Classification (Gaussian NB,Random Forest,Vader,TextBlob,Roberta), Report |
| Angelin Grace Wijaya | Data labeling, Preprocessing, Innovation, Report, Slides |
| Bhutra Divyansh | Data Labeling, User Interface, Innovation, Slides |
| Gupta Suhana | Data Labeling, Crawling, Innovation, Slides, Video |
| Singh Aishwarya | Data Labeling, EDA, Classification (Neural Networks), User Interface |

*Table 6: Work responsibilities*