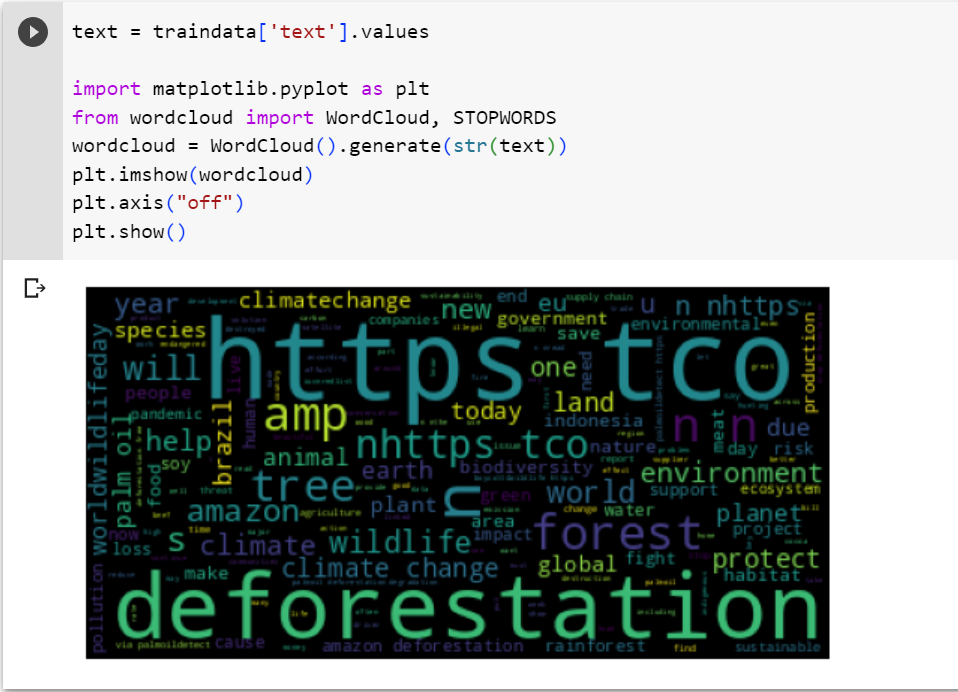
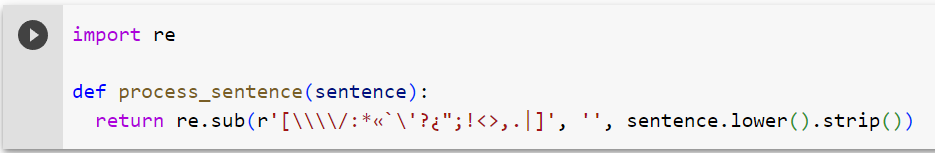
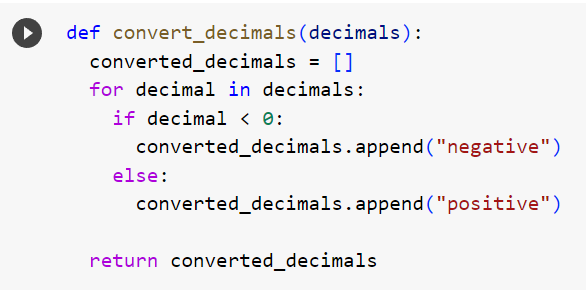
**Part 1: Train a neural network for sentiment analysis**

The two csv files were saved on my gdrive so I created code to get the data from my gdrive and read and save it to the variables traindata and testdata. Then I selected text as an input and distillbert\_valence as an output, as well as assigned data for null cells.

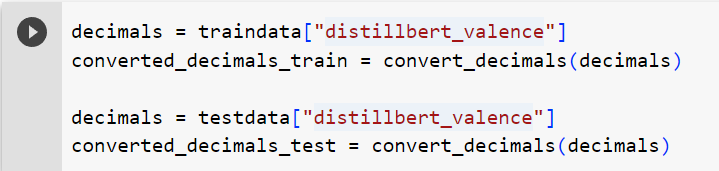


I created a variable called text and stored the data from the text column from the csv file. Then used matplotlib to create a word cloud of the positive and negative tweets posted by twitter users. This is an optional analysis step that gives a feel for the word frequencies in the data.

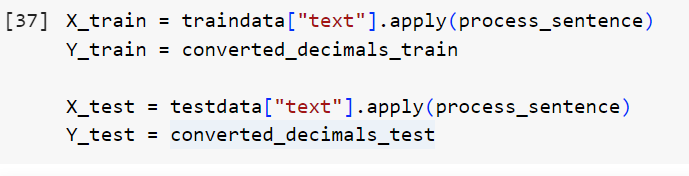
I imported the regular expressions package and created a method to preprocess the dataset to make tokens as uniform as possible. This is by removing special characters, like punctuation marks, white spaces and making all words lower case.



I created a method that changes floating point numbers to the strings negative and positive depending on whether they are positive or negative numbers.



I applied the method to both the traindata and testdata distillbert\_valence columns

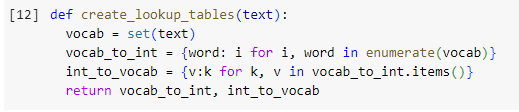


I applied my method to the text column in both pandas data frames to make all tweets lower case and assigned it to X for training. I then assigned the distillbert\_valence column to Y to use as the labels.



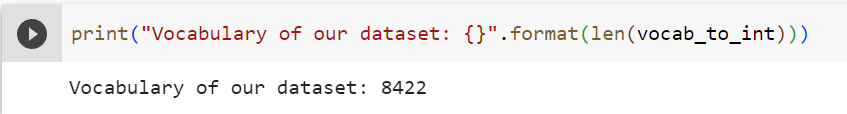
I specify that the labels that will be used correspond to the set of individual strings in y train data.

I split the training text data into strings that correspond to individual words to further analyze the tweets and added a backup character to use in case it comes across anything unseen.

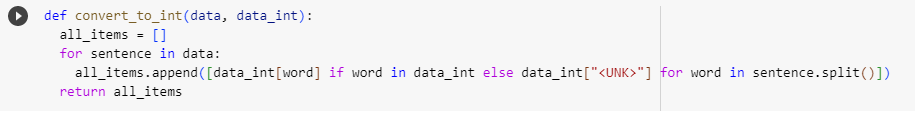


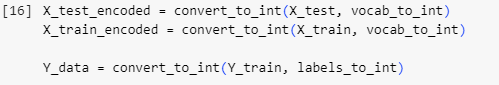
I created a method for a lookup table and specified that the variable vocab corresponds to the set of individual strings in the input text. The lookup tables allow for the words to be converted to integer then back to words, making it easy for the learning model to understand.

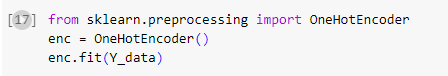
I applied the lookup table method to both elements and Y to convert between numbers and words from the train dataset.



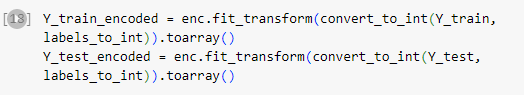
I printed the size of the vocabulary for the element's dataset

I created a function that uses the lookup tables to convert each tweet text into a numeric representation to one hot encode the data

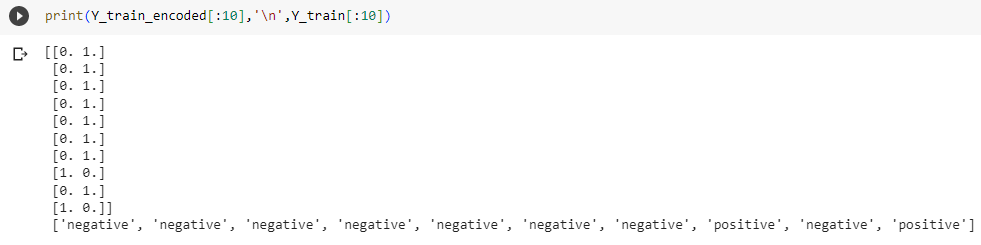
I applied the function to the x data for train and text and the y train data.

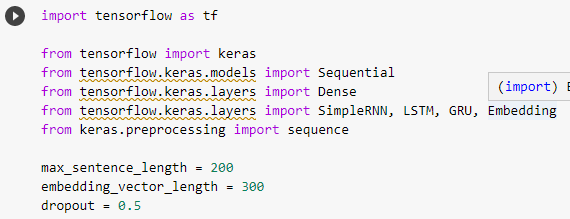


I imported sklearn’s onehotencoder and fitted it on y data so it can see what categories there are and what shape the resulting matrix needs to take.



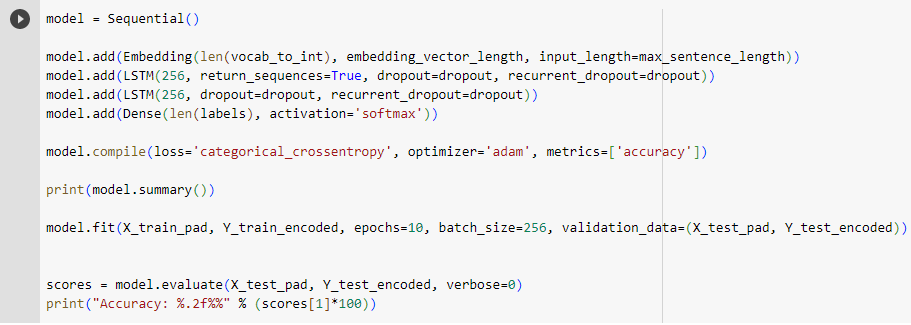
I applied the encoder to y train and test data and converted the resulting representations into arrays, as this is the format keras will need

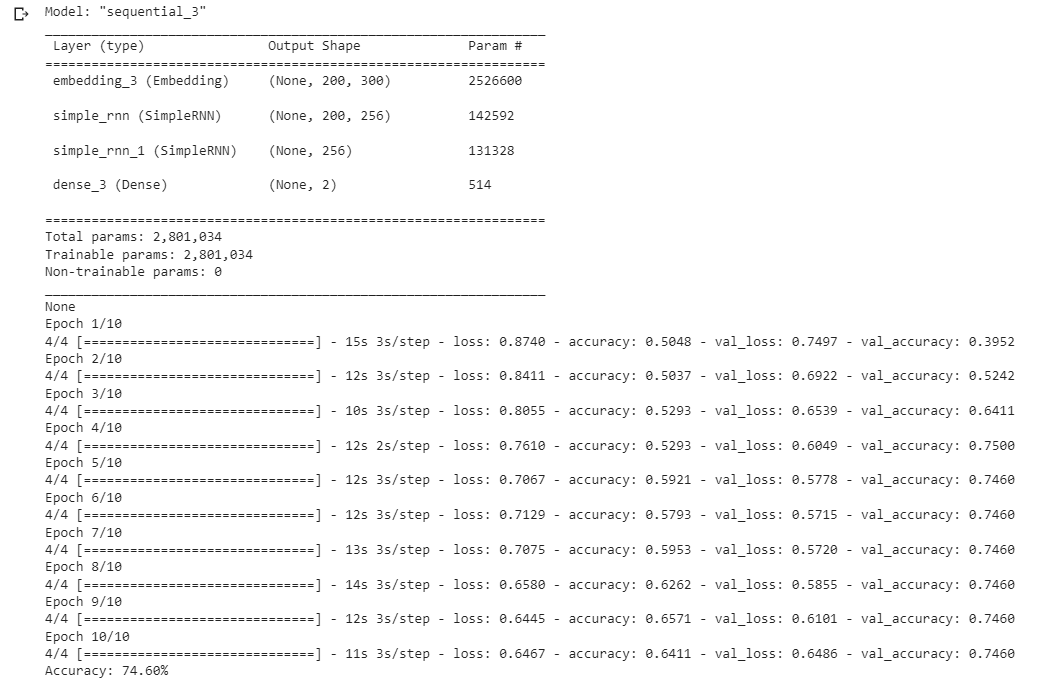
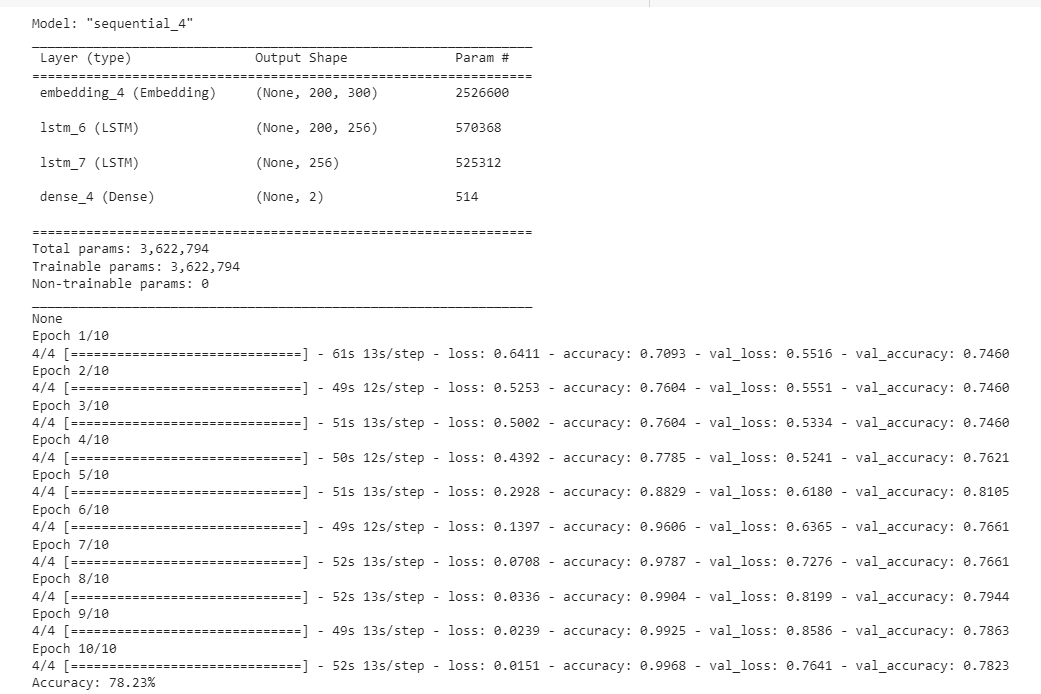
I displayed a sample of the encoded data and normal data to check if it has been correctly encoded



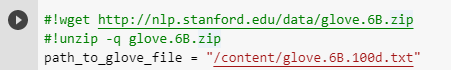
I imported tensor flow and other packages and specified hyperparameters. The sequence length has been set to 200 and captures an average amount of linguistic content and does not take too long to train. The embedding vector also captures an average context of co-occurring words, and the 0.5 drop out helps prevent the model from overfitting.

I padded the x train and test data to the length of the max sentence length as keras cannot deal with variably sized sequences. This pads up sequences with random symbols that are shorter than the set max sentence length parameter.

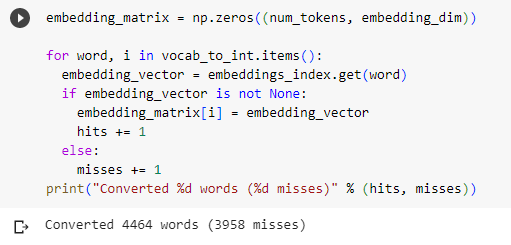
I changed the hyperparameters multiple times but the best result I was able to get was with a max sentence length of 200, an embedding vector length of 300 and a dropout of 0.5. I initialized the model and added an embedding layer that maps the length of the vocabulary to the embedding vector length. I also added two long short-term memory layers and a dense layer and compiled the model. I printed the model summary and trained the model with the train data and used the x test pad and y test pad for the validation. The metric I used was accuracy as the data only has two classes and it helps by determining the percentage of correct predictions made by the model. Lastly, I evaluated the model and displayed its overall accuracy.

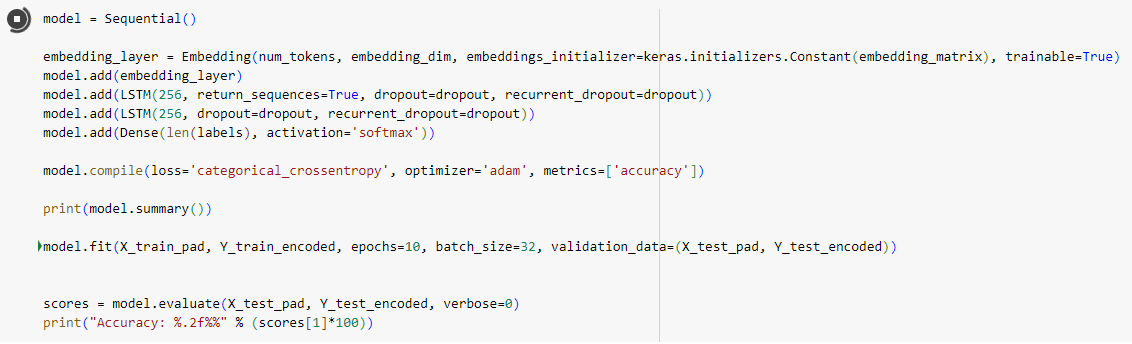
I decided to use a long short-term memory network because it is typically used for text processing. It is also better when compared to using a simple recurrent neural network as it uses special units in addition to standard units and is efficient at modelling complex sequential data (Great Learning Team, 2022). They have a memory cell that remembers data for extended periods of time and therefore are much better at handling long-term dependencies (Great Learning Team, 2022). They may be slow compared to using a simple recurrent neural network but from testing both out, long short-term memory network had a higher accuracy than simple neural network. LSTM had an accuracy of 78.23% and SimpleRNN had an accuracy of 74.60%. The results are above.

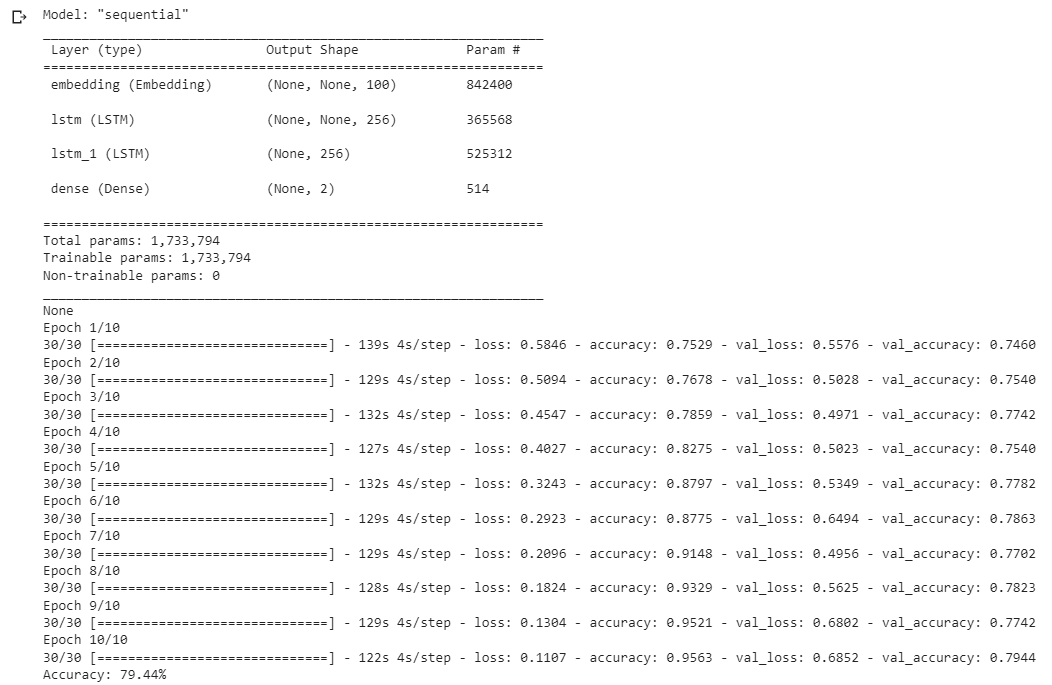
**Part 2: Vary the knowledge representation**



I downloaded pre-trained GloVe embeddings and unzipped the downloaded file

I used the embeddings to create an embedding index which matches words in the vocabulary to their respective embedding representation. I then created an embedding matrix from the embedding index, which converts the words it can and reports back matching words and non-matching words.

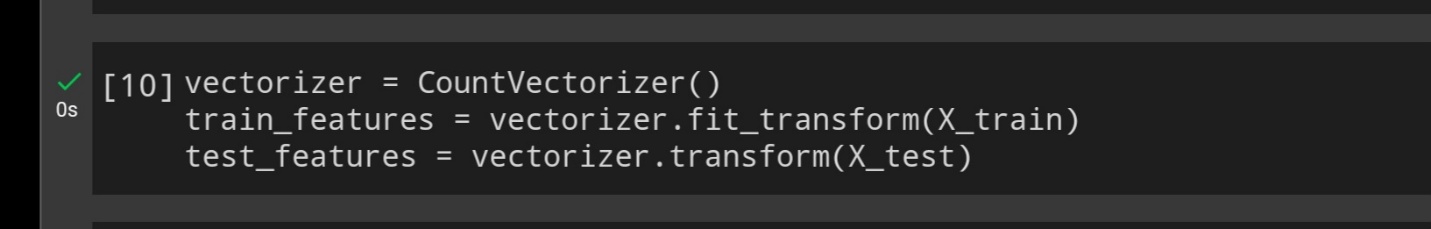
I defined a new embedding layer based on the embedding matrix and used the same model from my neural network for training.

After 10 epochs, I got a validation accuracy of 79.44%. Based on this, the embeddings have helped my model from its baseline of 78.23% and is better to use than a simple embedding layer.

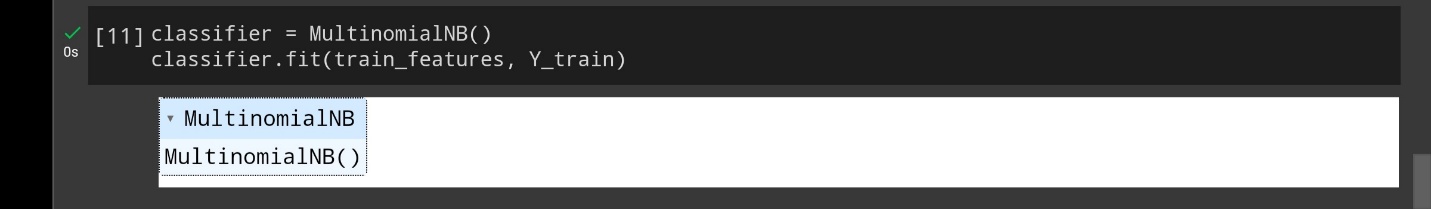
The model I chose for my embedding layer was GloVe. GloVe is used for word representation and stands for global vectors. It is a type of word embedding that encodes the co-occurrence probability of two words as vector differences and uses the Euclidean separation between two words to evaluate the semantic similarity of the corresponding words. I decided to use GloVe because even though it is slower than Word2vec, it includes global statistics to obtain word vectors, but Word2vec only relies on local statistics. GloVe also creates a lower-dimensional matrix hence a preferable word embedding can be acquired by reducing the modification loss and is easier to train over more data. Lastly, GloVe seems to perform better in word relations and named entity identification problems.

Language models lack understanding, they manipulate tokens but have no semantic anchoring for what they say or do. False or misleading information could be given out when fed into the model as the model does not know what the difference between right and wrong data is and personal data could be leaked. They can create unfair discrimination as there are stereotypes and social biases that can deny or burden identities that differ. For example, a normal family consists of a child, female mother, and a male father. Deep learning models work as black boxes and supply no reasoning for their decisions although they assist in providing solutions. These models benefit users and help serve the public good but could do harm only if used in the wrong hands. They also leave a large carbon footprint when being used to train data and risk harming marginalized communities by reinforcing hegemonic viewpoints. It takes substantial amounts of resources and time to train language models from scratch compared to pre-trained language models. With pre-trained models, the weights and features can be used as a starting point. It is also a low-cost investment model as it needs less data and computational power to train the model and saves time and money.

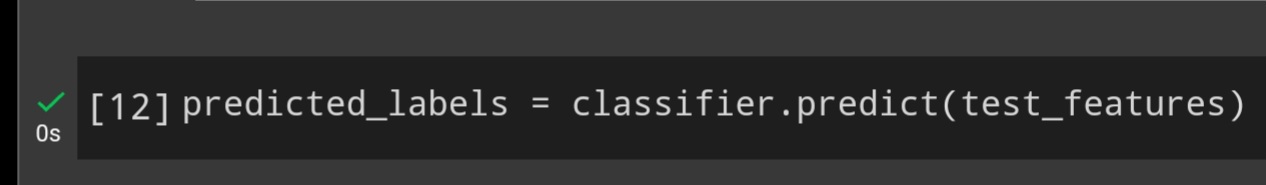
**Part 3: Create a probabilistic baseline**



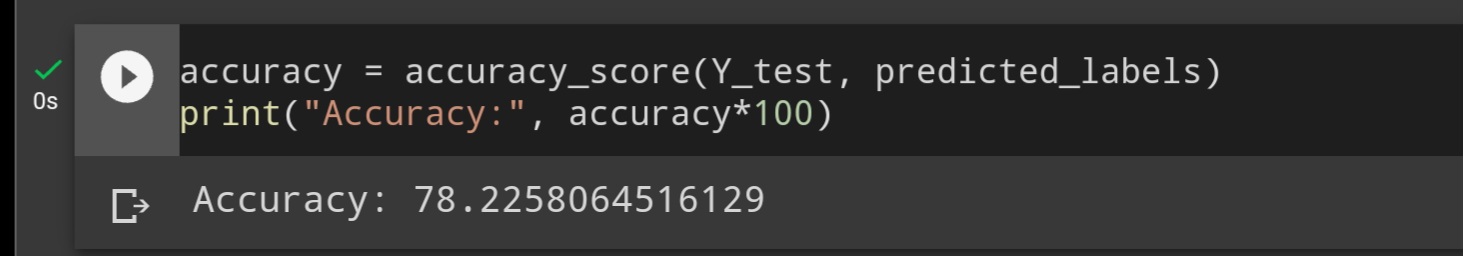
I used the CountVectorizer from scikit-learn to convert the tweets into a matrix of token counts. I then fitted the vectorizer on the X\_train and transformed both the X\_train and X\_test into feature vectors.



I trained a MultinomialNB classifier, which is a probabilistic classifier based on the Naive Bayes algorithm, using the train features and Y\_train.



I created a variable that predicts the labels for the test data



Finally, I used the trained classifier to predict the labels for the test features and calculated the accuracy of the classifier by comparing the predicted labels with the actual labels.

This model had an accuracy of 78.23%, which is the same percentage as using LSTM with a simple embedding layer. It has a lower accuracy compared to using GloVe with a 79.44% for accuracy. From this analysis, GloVe is the model that has the best accuracy and will give better results.

**References**

* Great Learning Team. (2022) Types of Neural Networks and Definition of Neural Network. Great Learning. 23 November. Available online: [Types of Neural Networks and Definition of Neural Network (mygreatlearning.com)](https://www.mygreatlearning.com/blog/types-of-neural-networks/) [Accessed 20/05/2023]