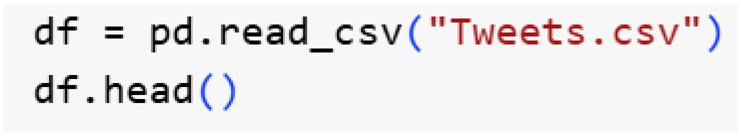
FINAL PROJECT:

1.

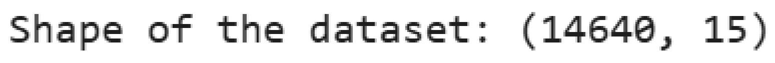
a. Exploratory Data Analysis

Load the dataset, check dimensions, and handle null values.

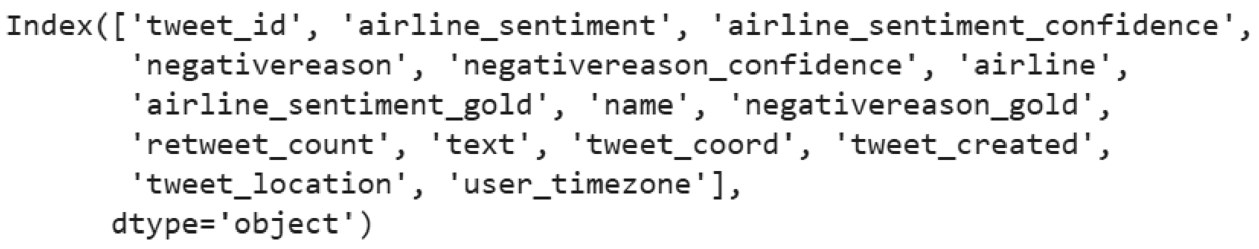


```
df = pd.read_csv("Tweets.csv")
df.head()
```

We can see a list of all the columns and samples.

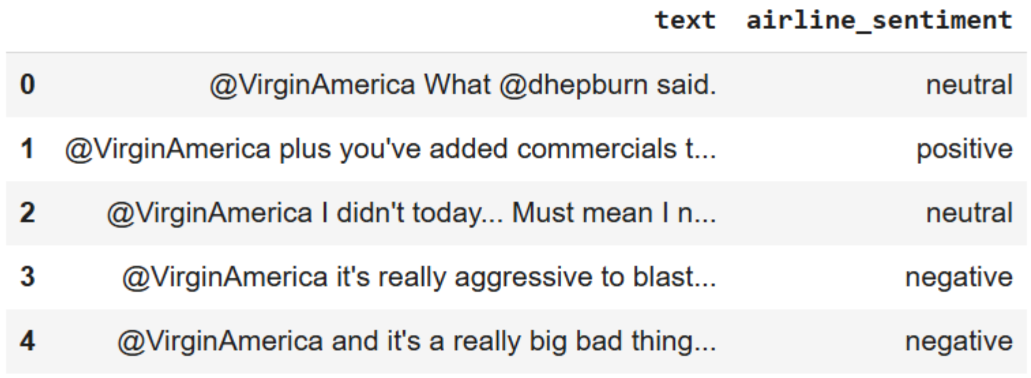


```
Shape of the dataset: (14640, 15)
```



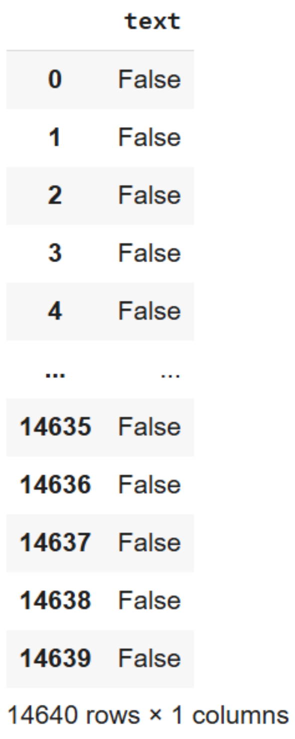
```
Index(['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence',
       'negativereason', 'negativereason_confidence', 'airline',
       'airline_sentiment_gold', 'name', 'negativereason_gold',
       'retweet_count', 'text', 'tweet_coord', 'tweet_created',
       'tweet_location', 'user_timezone'],
      dtype='object')
```

The original assignment had 15 columns, but here we are dealing with 2 major columns: ‘text’ and ‘airline_sentiment’.



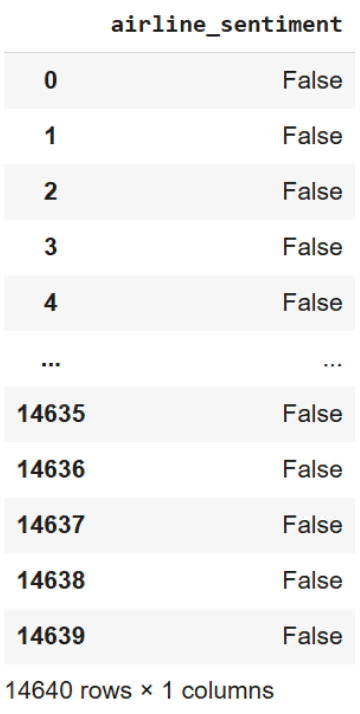
	text	airline_sentiment
0	@VirginAmerica What @dhepburn said.	neutral
1	@VirginAmerica plus you've added commercials t...	positive
2	@VirginAmerica I didn't today... Must mean I n...	neutral
3	@VirginAmerica it's really aggressive to blast...	negative
4	@VirginAmerica and it's a really big bad thing...	negative

We have checked individual nan values of required columns for the assignment/Project.



	text
0	False
1	False
2	False
3	False
4	False
...	...
14635	False
14636	False
14637	False
14638	False
14639	False

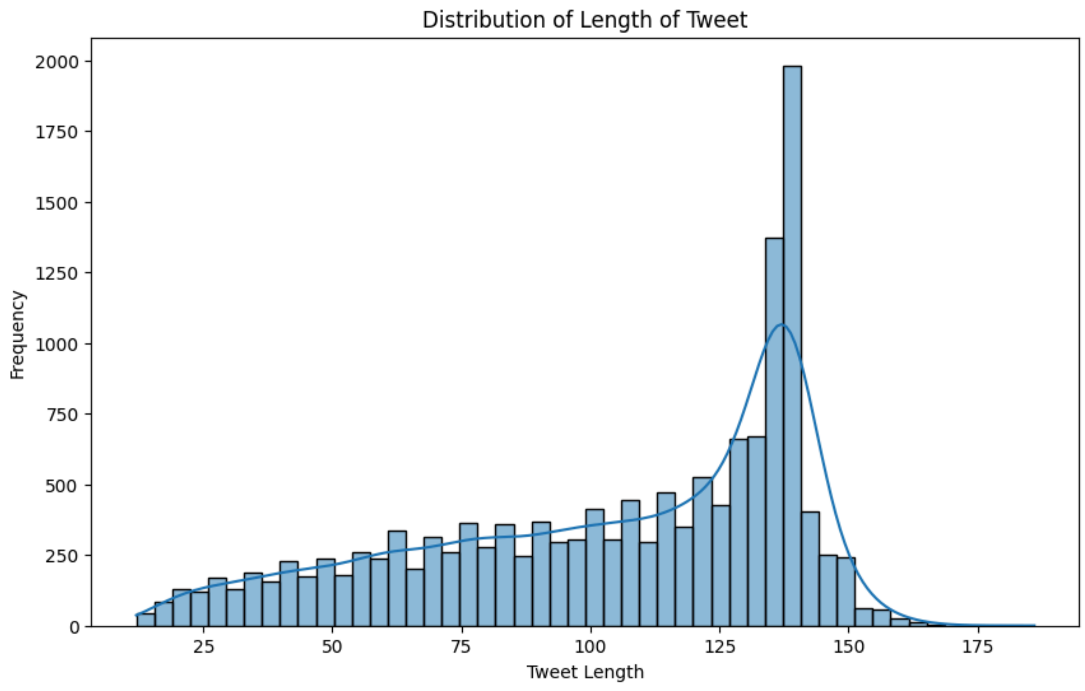
14640 rows × 1 columns



	airline_sentiment
0	False
1	False
2	False
3	False
4	False
...	...
14635	False
14636	False
14637	False
14638	False
14639	False

14640 rows × 1 columns

Explore post/tweet length distributions and label proportions:



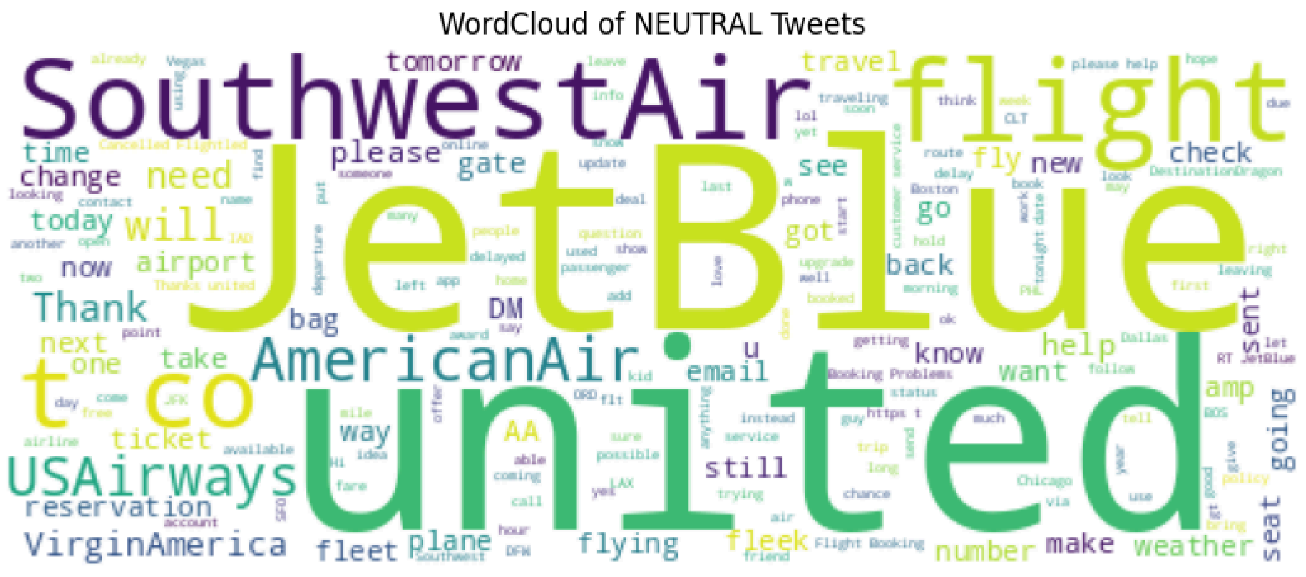
The distribution is right-skewed, not perfectly normal.

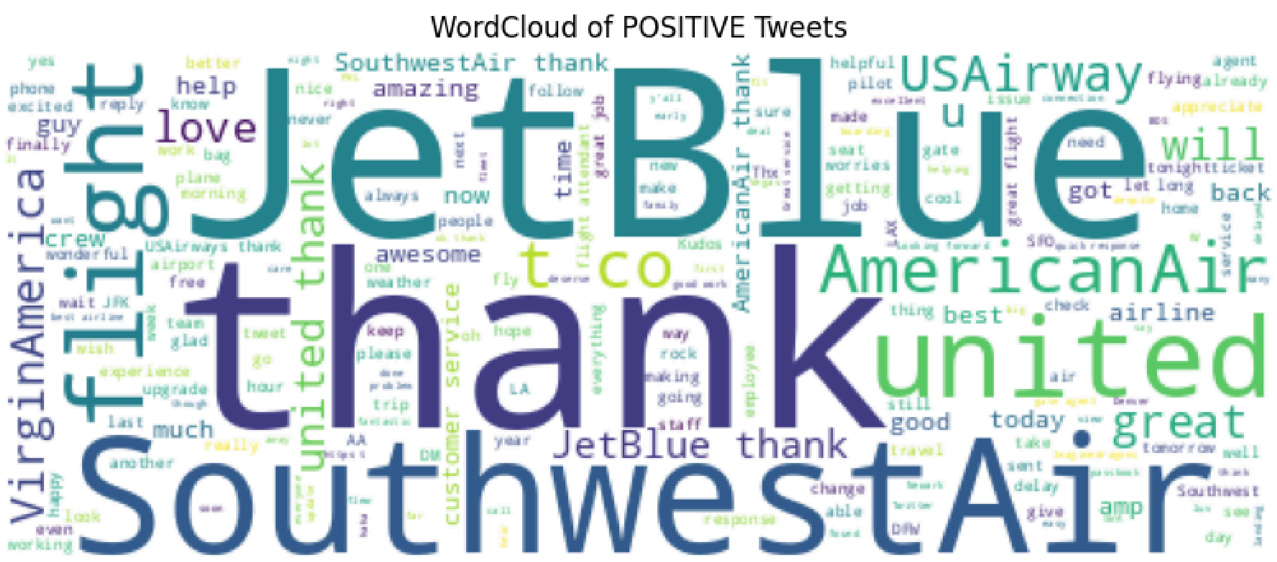
airline_sentiment	count
negative	9178
neutral	3099
positive	2363

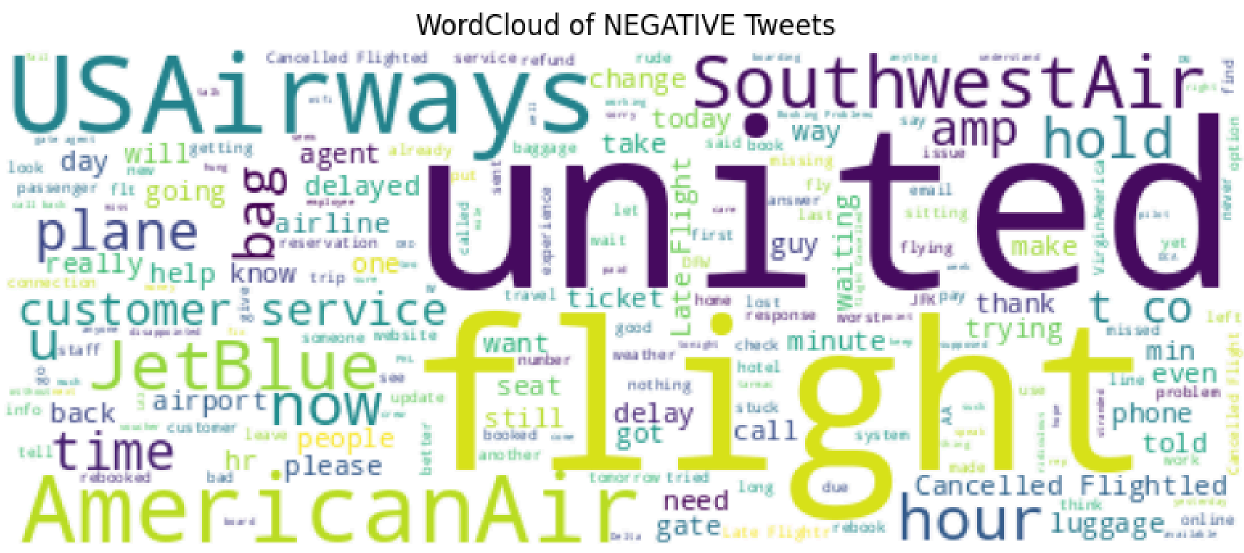
The Labels also seem heavily one-sided, with more than 64% of samples being tweets with negative sentiment.

b. Text Visualization

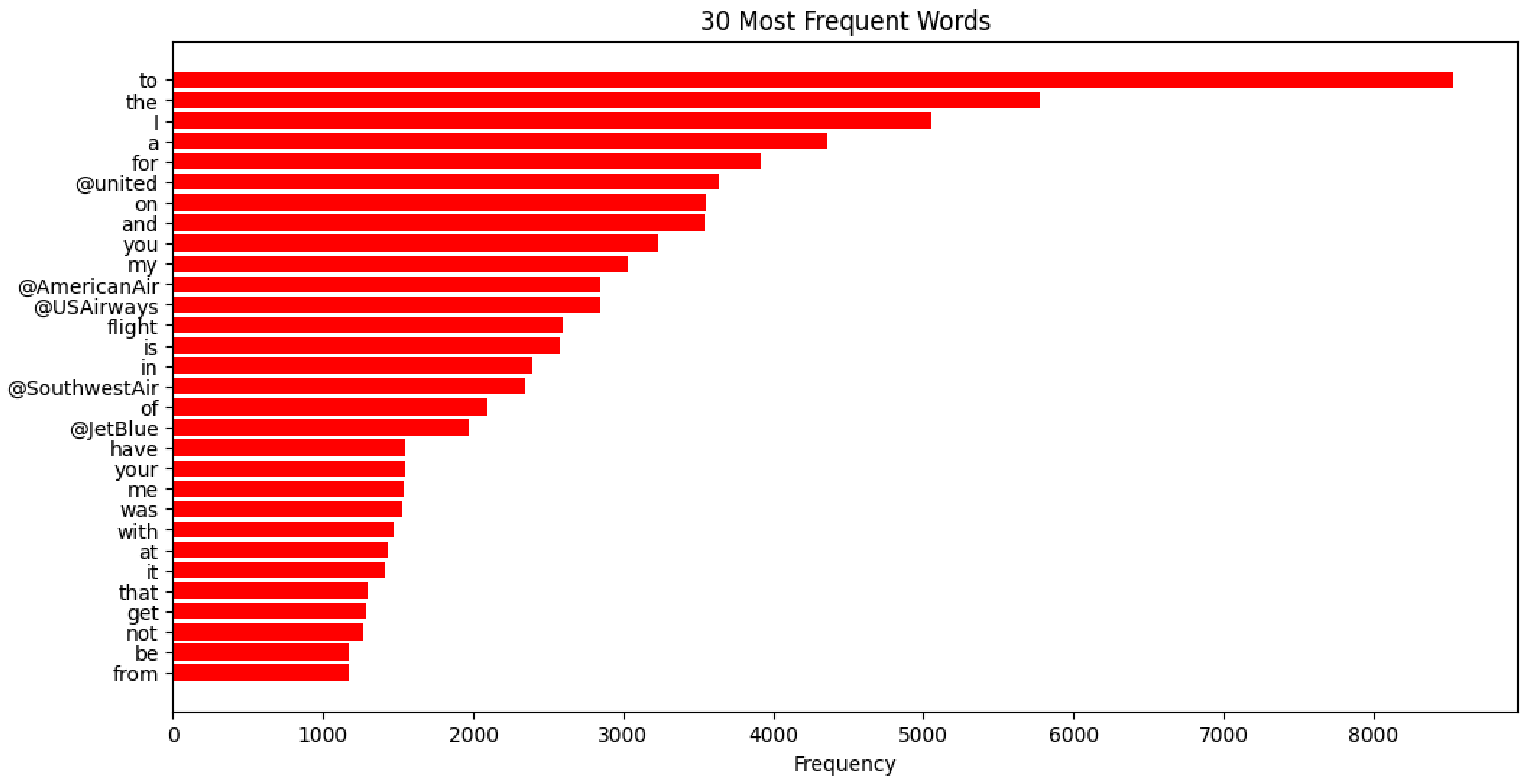
Word clouds per class:



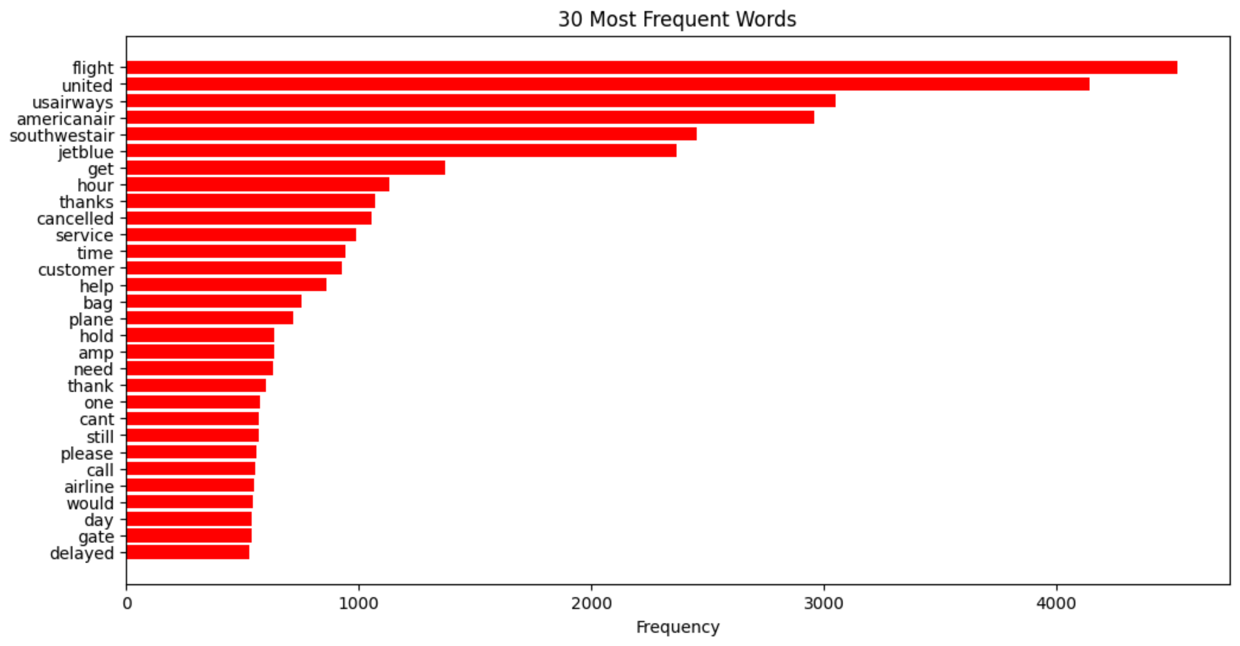




Top n frequent word occurrence:

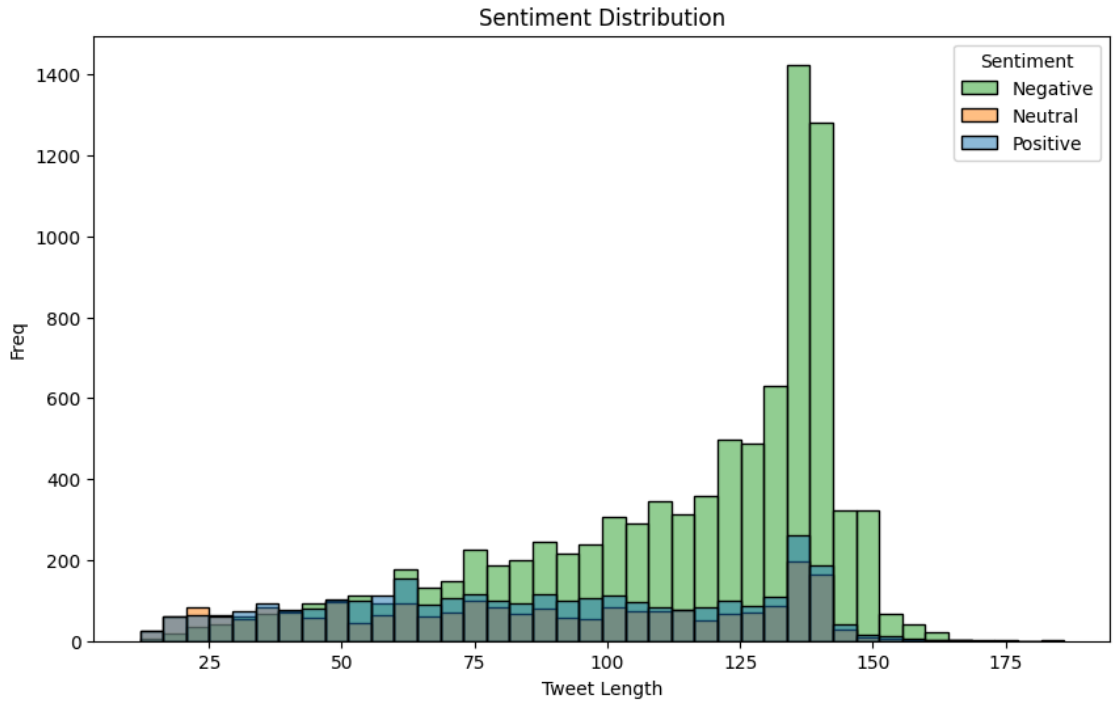


Here we observe a lot of stop words or words with not much significance or importance are in most frequent words use in the dataset.



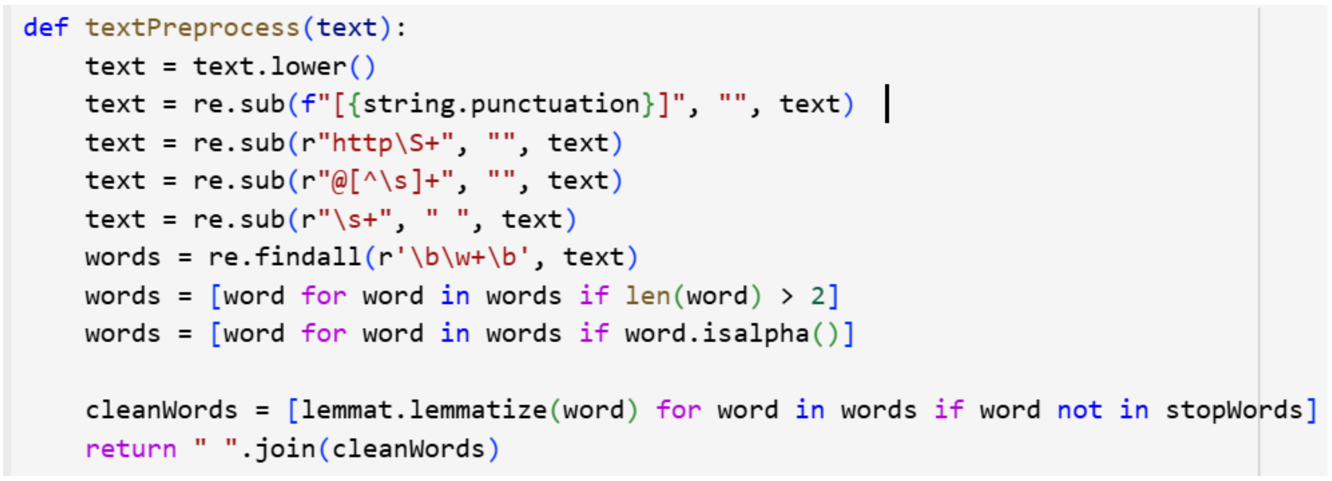
The above image with proper data cleaning and preprocessing.

Tweet/post length histograms:



c. Preprocessing & Feature Engineering

Clean, tokenize: lowercasing, punctuation removal, stopwords, stemming/lemmatization:



```
def textPreprocess(text):
    text = text.lower()
    text = re.sub(f"[{string.punctuation}]", "", text)
    text = re.sub(r"http\S+", "", text)
    text = re.sub(r"@[^\s]+", "", text)
    text = re.sub(r"\s+", " ", text)
    words = re.findall(r'\b\w+\b', text)
    words = [word for word in words if len(word) > 2]
    words = [word for word in words if word.isalpha()]

    cleanWords = [lemmat.lemmatize(word) for word in words if word not in stopWords]
    return " ".join(cleanWords)
```

All the preprocessing steps mentioned are removed using the attached method on the text column. The text is converted to lowercase to ensure uniformity. All punctuation marks, URLs, shortwords less than 3 characters, and alphanumeric tokens are removed. Remaining are lemmatised and stopwords are eliminated from the dataset.

Create at least two representations: - TF-IDF
Bag of Words:
Used scikit-learb library for using Bag of words model. Doesnt give importance to order or context in the sentence, all words are treated equally. It is a frequency based model. No fixed number of dimensions is set as max_features was initialised, so it can results in high dimension sparse matrix. Very sensitive to common words. The resultant matrix is just word cound.
The dimension we got is 10,140.

```
BOWvectorizer = CountVectorizer(stop_words='english')
```

```
DtstBOW.shape[1]
```

```
10140
```

TF-IDF:
Used scikit-learb library for using Bag of words model. Doesnt give importance to order or context in the sentence, all words are treated equally, but there is a triking difference between Bag of words and TF-IDF. TF-IDF weight seach word based on its importance in a sample also considers its relevance across the dataset, while Bag of Words does not. The output is a sparse matrix with normalised weights. Common words have reduced weights. The dimension we got is 10,140. No fixed number of dimensions is set as max_features was initialised, so it can results in high dimension sparse matrix.

```
tfidf = TfidfVectorizer(stop_words='english')
```

```
DtstTFIDF.shape[1]
```

```
10140
```

Part 2: Model Building
For interprettable models, we use Logistic Regression with L2 regularisation and Naive Bayes classifier.

For Black Box models, we used a simple Deep Neural Network and an LSTM built on PyTorch.

For interpretable/simple models, and DNN, we used both TF-IDF as embedding layers, where as these vector representations could be very sparse for blackbox models like LSTM, RNN, BERT, so these models have been avoided for LSTM.

Model Training:

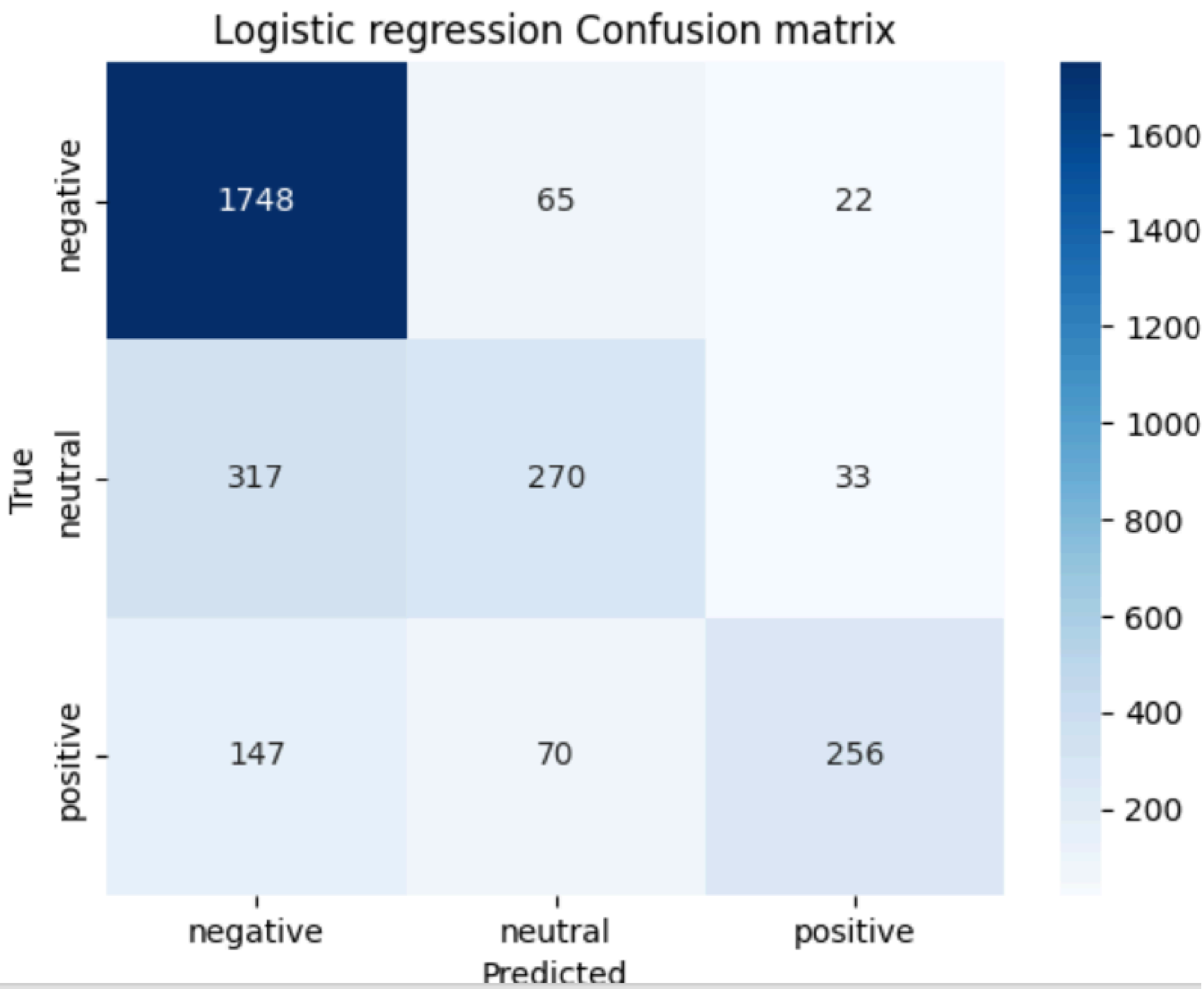
Logistic Regression & Naive Bayes:

For Logistic Regression, we used 'liblinear' solver with L2 regularizations and split the dataset into 80-20 split.

The below snippet is Logistic Regression with TFIDF.

```
LR(DtrTFIDF, DtstTFIDF, Ltr, Ltst)
```

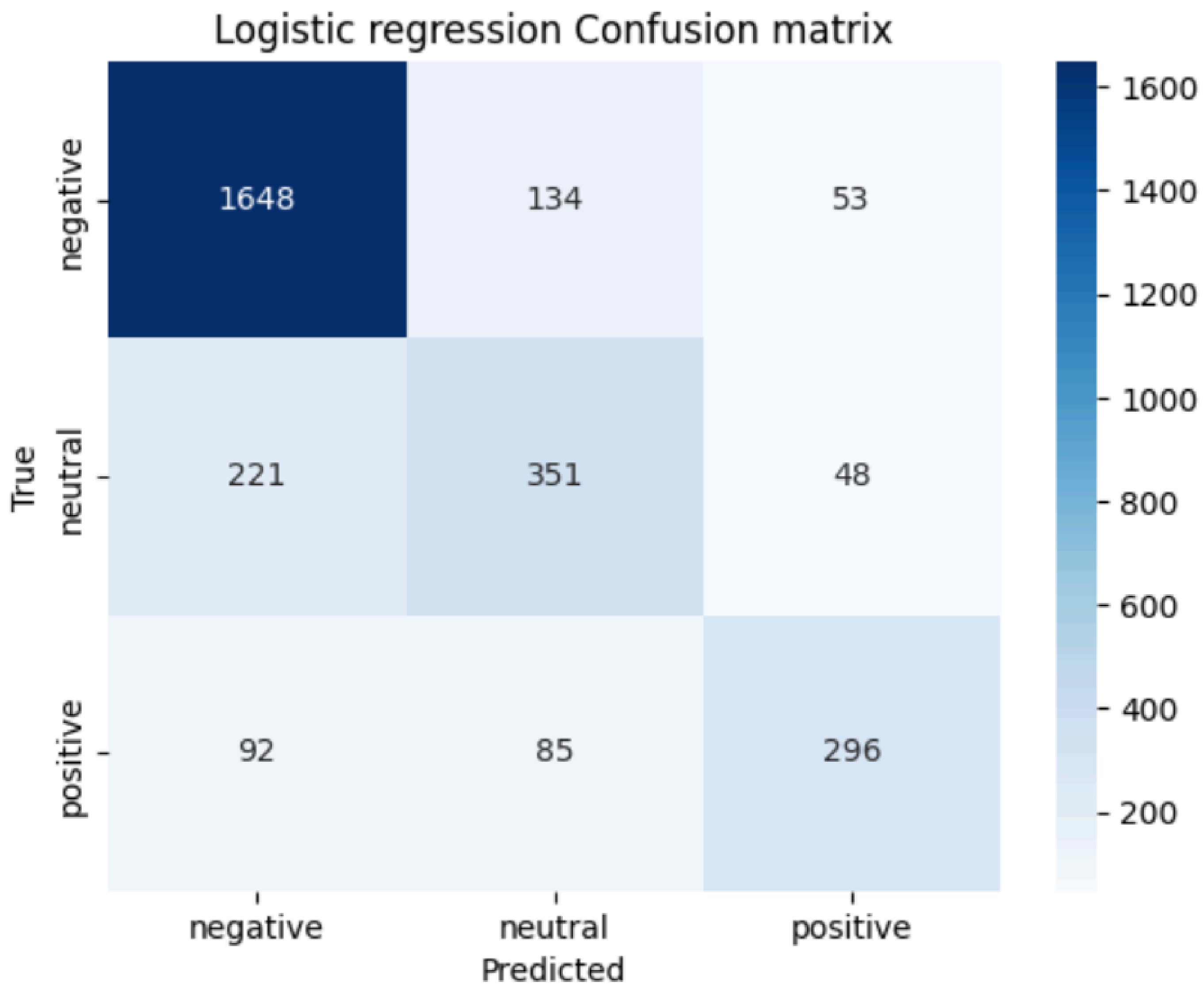
```
              precision    recall  f1-score   support

    negative       0.79      0.95      0.86      1835
     neutral       0.67      0.44      0.53       620
    positive       0.82      0.54      0.65       473

    accuracy                           0.78      2928
   macro avg       0.76      0.64      0.68      2928
weighted avg       0.77      0.78      0.76      2928
```



The below snippet is Logistic Regression with Bag of Words.

```
[298] LR(DtrBOW, DtstBOW, Ltr, Ltst)
```

```
              precision    recall  f1-score   support

    negative       0.84      0.90      0.87      1835
     neutral       0.62      0.57      0.59       620
    positive       0.75      0.63      0.68       473

    accuracy                           0.78      2928
   macro avg       0.73      0.70      0.71      2928
weighted avg       0.78      0.78      0.78      2928
```



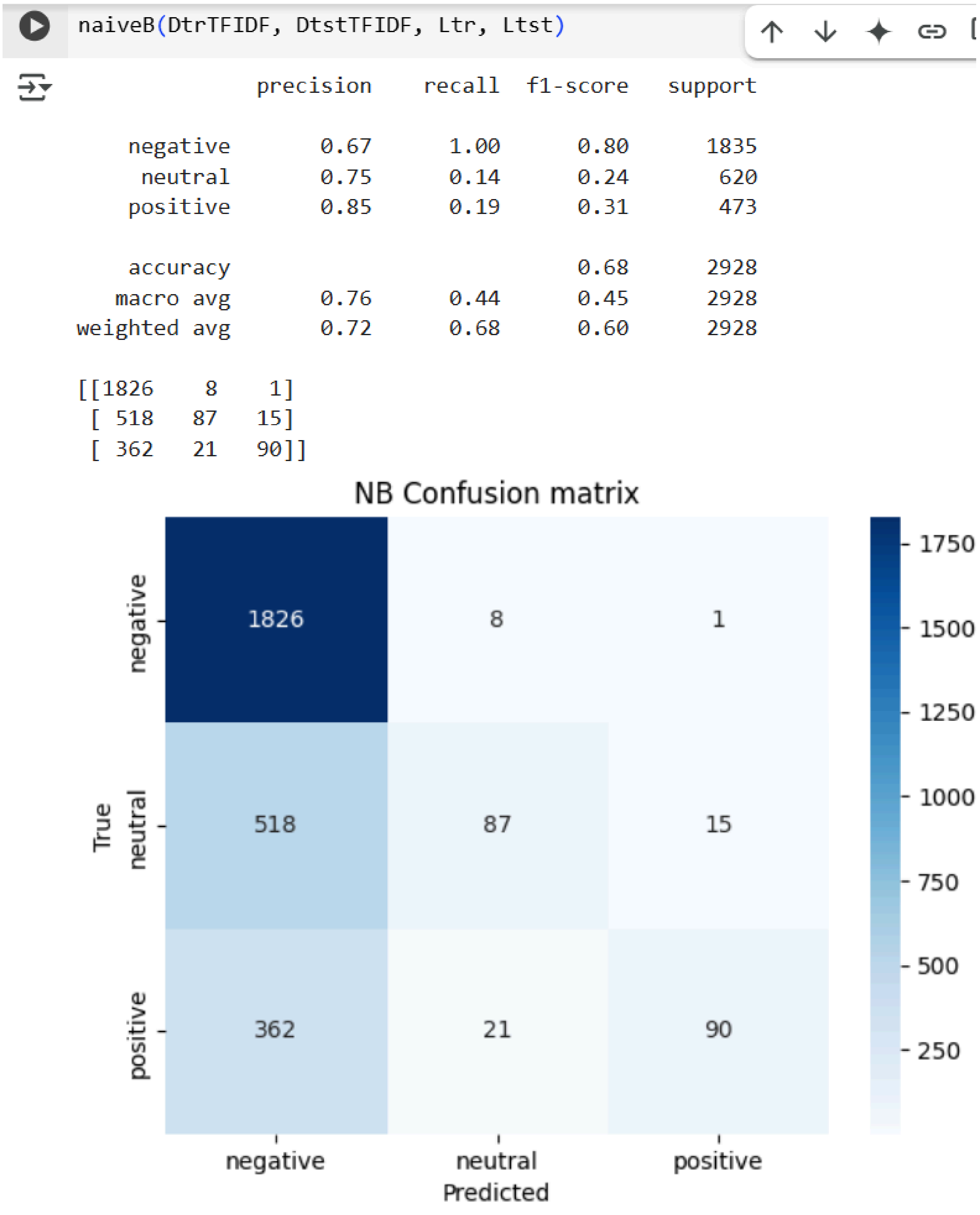
Both these models have identical accuracies which is 78% but classwise performances is very different. Bag of words model has more balance precision and recall and TFIDF model over predicts the negative class.

TF-IDF model is more biased towards predicting negative especially for neutral and positive sentiments.

In Bag of words, neutral tweets are confused with negative.

Naive Byaes:

NB with TF-IDF.



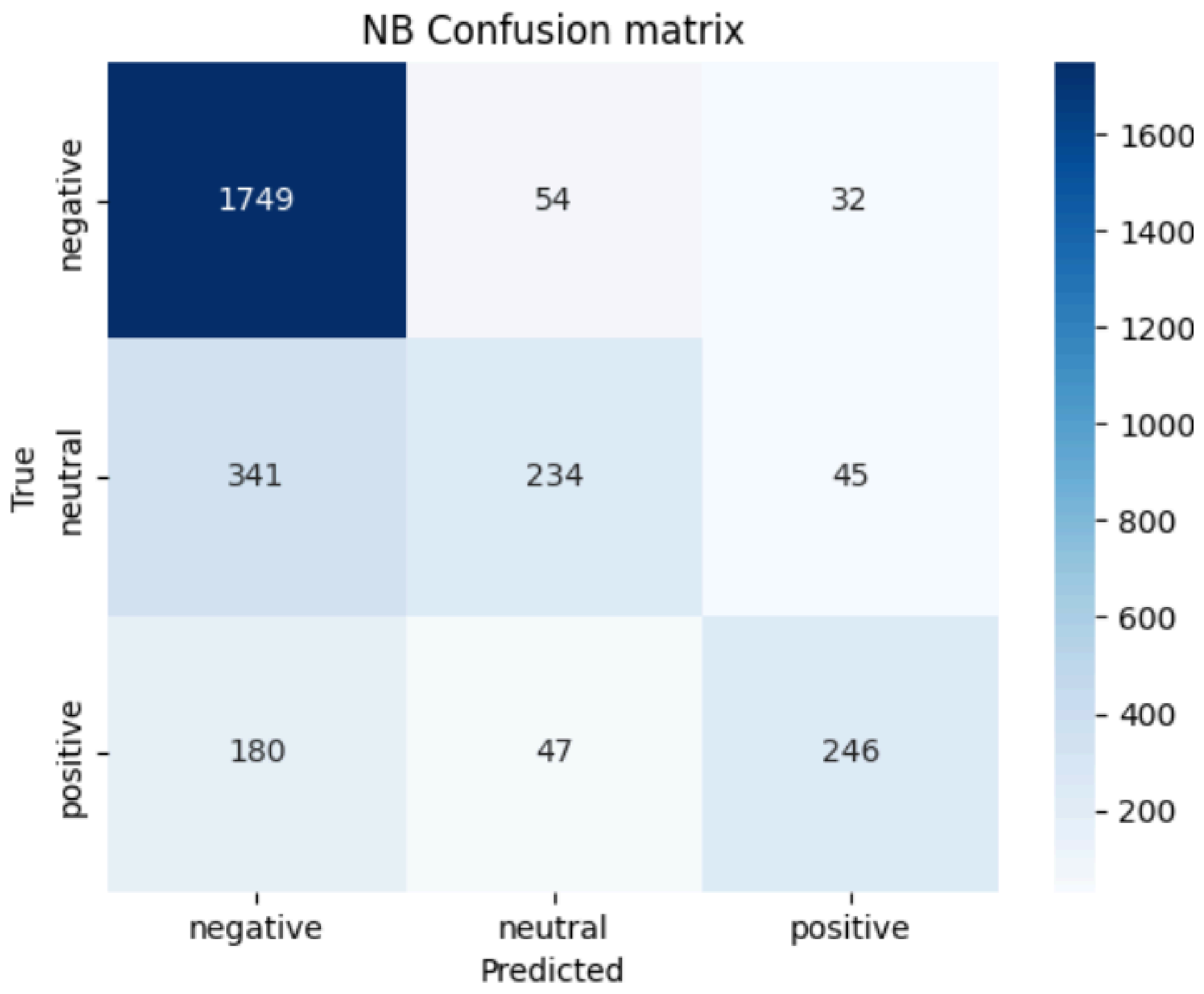
```
naiveB(DtrTFIDF, DtstTFIDF, Ltr, Ltst)
```

```
              precision    recall  f1-score   support

    negative       0.67      1.00      0.80      1835
     neutral       0.75      0.14      0.24       620
    positive       0.85      0.19      0.31       473

    accuracy                           0.68      2928
   macro avg       0.76      0.44      0.45      2928
weighted avg       0.72      0.68      0.60      2928

[[1826    8    1]
 [ 518   87   15]
 [ 362   21   90]]
```

NB with Bag of words.

```
[301] naiveB(DtrBOW, DtstBOW, Ltr, Ltst)
```

```
              precision    recall  f1-score   support

    negative       0.77      0.95      0.85      1835
     neutral       0.70      0.38      0.49       620
    positive       0.76      0.52      0.62       473

    accuracy                           0.76      2928
   macro avg       0.74      0.62      0.65      2928
weighted avg       0.75      0.76      0.74      2928

[[1749   54   32]
 [ 341  234   45]
 [ 180   47  246]]
```

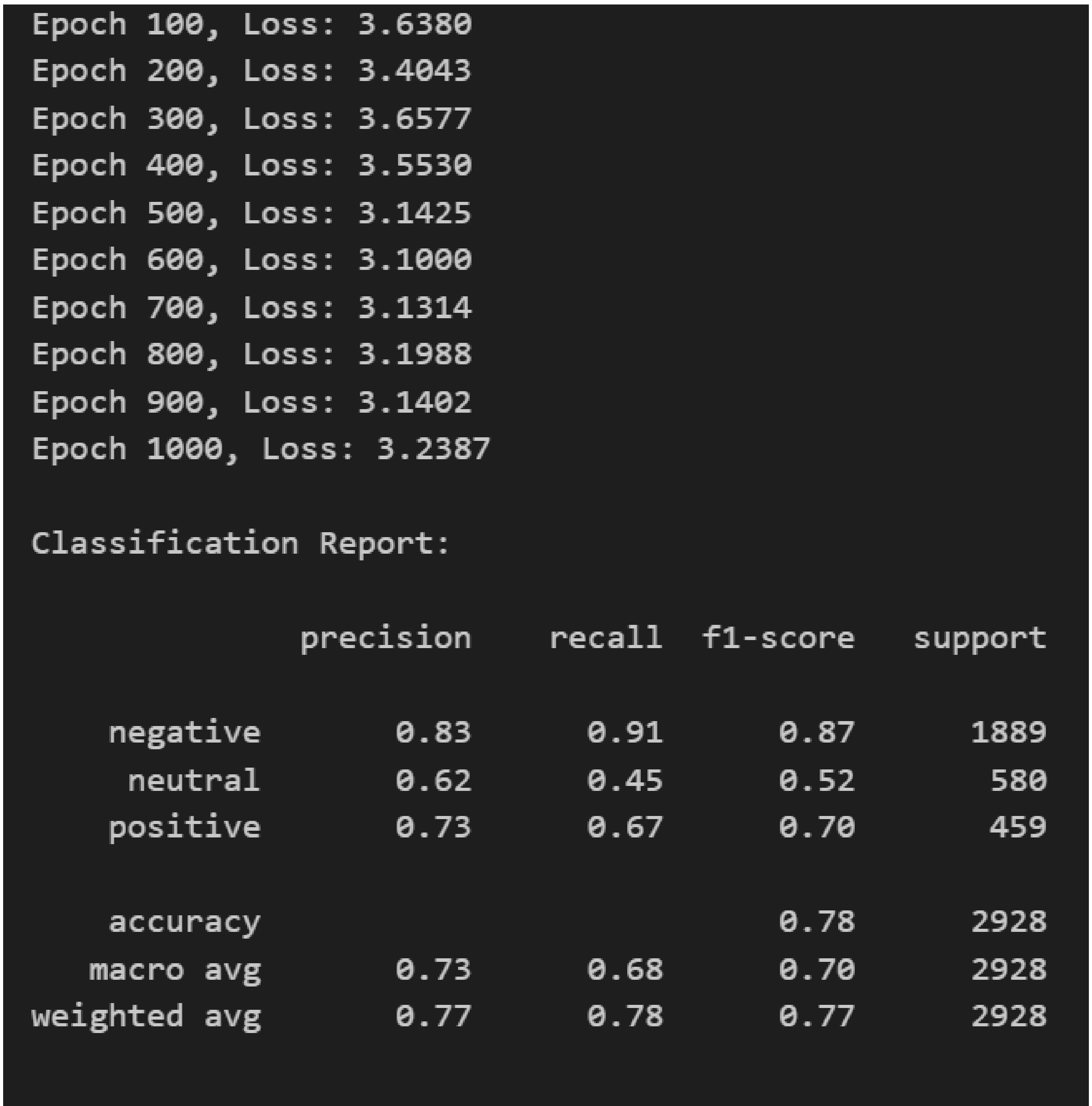


Bag of words has better accuracy, precision, recall for all classes in comparison with TF-IDF. TF-IDF often downweights common but informative words, which might be causing the results to go down.

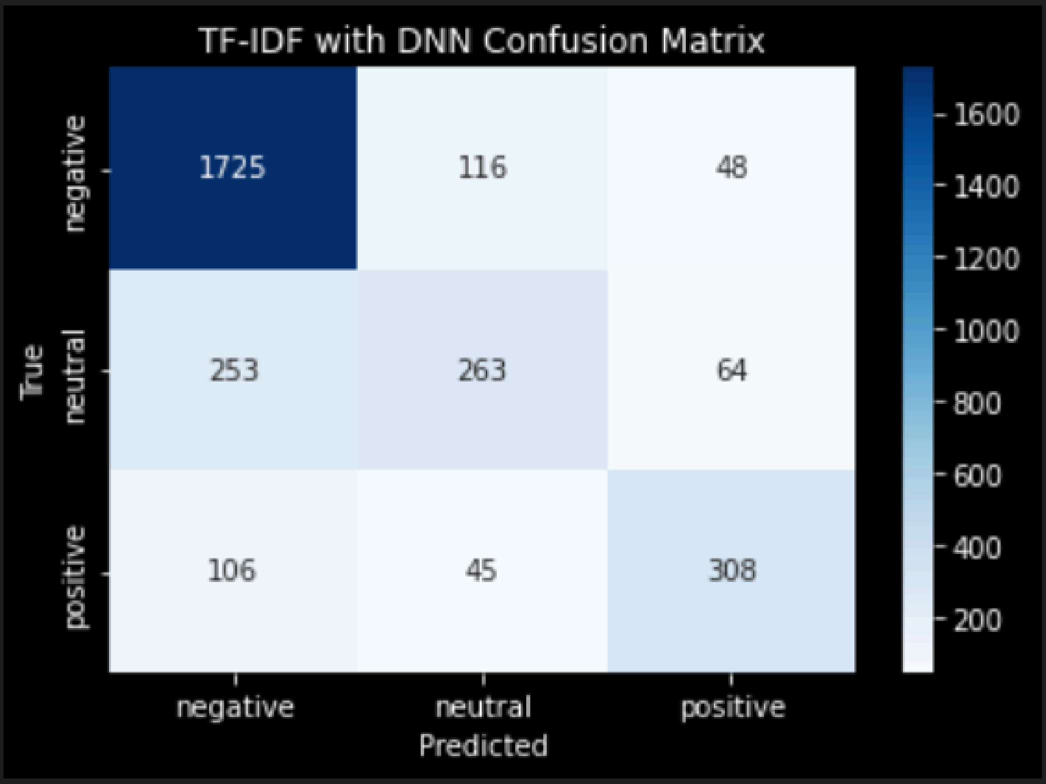
Bag of words + Naive Bayes is performing better than TF-IDF, it has more balanced and better class generalisation across all the classes.

Black box models:

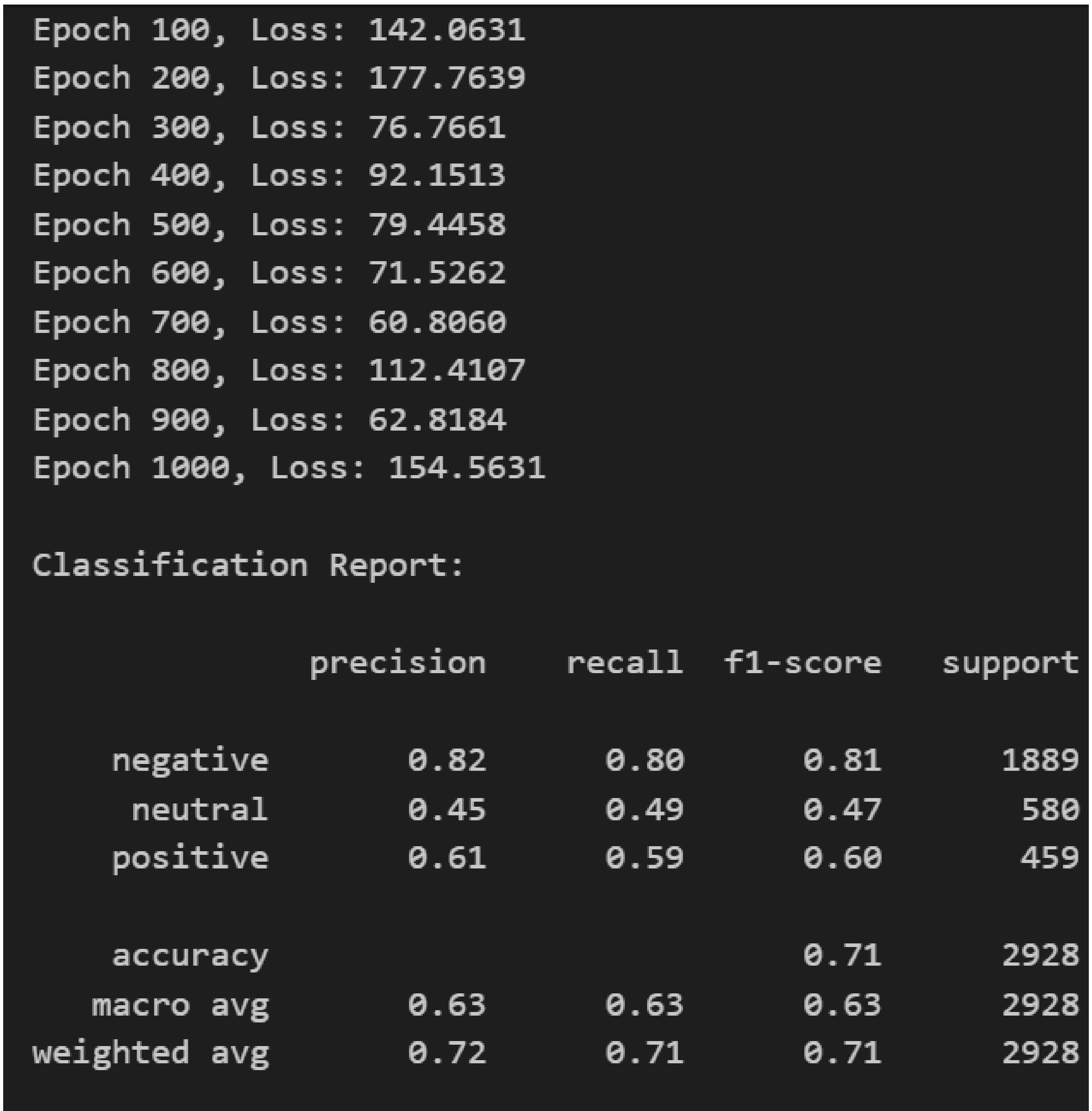
DNN with TF-IDF:



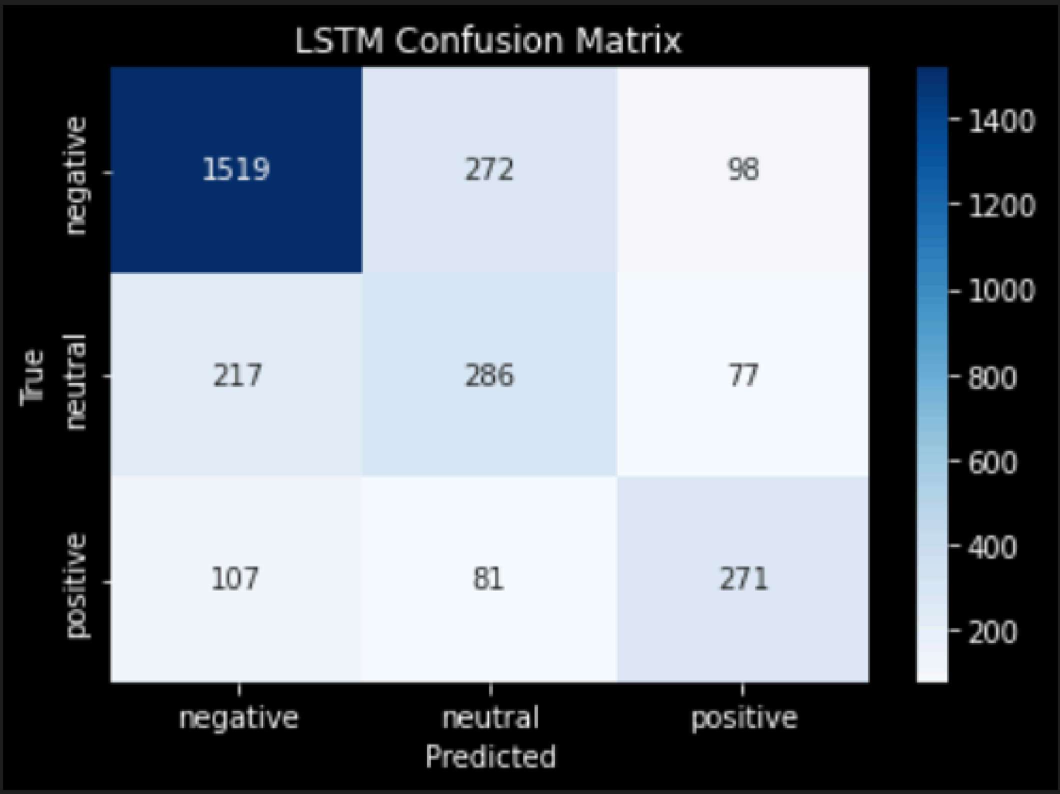
```
Epoch 100, Loss: 3.6380
Epoch 200, Loss: 3.4043
Epoch 300, Loss: 3.6577
Epoch 400, Loss: 3.5530
Epoch 500, Loss: 3.1425
Epoch 600, Loss: 3.1000
Epoch 700, Loss: 3.1314
Epoch 800, Loss: 3.1988
Epoch 900, Loss: 3.1402
Epoch 1000, Loss: 3.2387

Classification Report:

              precision    recall  f1-score   support

    negative       0.83      0.91      0.87      1889
     neutral       0.62      0.45      0.52       580
    positive       0.73      0.67      0.70       459

    accuracy                           0.78      2928
   macro avg       0.73      0.68      0.70      2928
weighted avg       0.77      0.78      0.77      2928
```



LSTM with learnable embedding layer:



```
Epoch 100, Loss: 142.0631
Epoch 200, Loss: 177.7639
Epoch 300, Loss: 76.7661
Epoch 400, Loss: 92.1513
Epoch 500, Loss: 79.4458
Epoch 600, Loss: 71.5262
Epoch 700, Loss: 60.8060
Epoch 800, Loss: 112.4107
Epoch 900, Loss: 62.8184
Epoch 1000, Loss: 154.5631

Classification Report:

              precision    recall  f1-score   support

    negative       0.82      0.80      0.81      1889
     neutral       0.45      0.49      0.47       580
    positive       0.61      0.59      0.60       459

    accuracy                           0.71      2928
   macro avg       0.63      0.63      0.63      2928
weighted avg       0.72      0.71      0.71      2928
```



b. Architecture & Implementation Details
Logistic Regression (TF-IDF / BoW):

Pipeline: Preprocessing and cleaning Text → sckit-learn vectorisation → FitTransform / Transform → pass the vector into LR model and Fit → Evaluations using Accuracy, Precision, Recall, F1, and Confusion matrix.

L2 Regularisation is used by default, and the liblinear solver is used. The dataset was split into an 80-20 split.

Naive Bayes (TF-IDF / BoW):

Pipeline: Preprocessing and cleaning Text → sckit-learn vectorisation → FitTransform / Transform → pass the vector in to Multivariate NB model and Fit → Evaluations using Accuracy, Precision, Recall, F1, and Confusion matrix.

The dataset was split into an 80-20 split.

DNN with TF-IDF:

Pipeline: Preprocessing and cleaning Text → sckit-learn vectorisation → FitTransform / Transform → pass the vector into the deep neural network with training → Evaluations using Accuracy, Precision, Recall, F1, and Confusion matrix.

The dataset was split into an 80-20 split.

Model architecture:
The model has one input layer, 2 hidden layers, and an output layer.

The starting layer is a TF-IDF vector → 1st Dense Hidden layer with 256 neurons and a ReLU activation function with 0.3 dropout → 2nd Dense Hidden layer with 64 neurons and a ReLU activation function with 0.2 dropout → Output layer with 3 classes signify 3 sentiments with activation functions.

The loss function used is CrossEntropyLoss() (multi-class classification) and Adam optimiser with a learning rate 0.001.
The batch size used is 32.
1000 epochs were used.
Build a custom Tensordata from torch.utils.data.

LSTM with learnable embedding layer:

This has an LSTM layer, an Embedding layer, and an output layer.
1st layer is a sequence of vocab IDs → Embedding is with learnable parameters with 32 vectors → Output of the embedding is an LSTM layer with 32 hidden units → The end hidden state is sent to a Full Dense layer with 3 neurons.

No activation functions.

The text is tokenised into words and mapped to IDs using a custom vocabulary. Padding is used with ID 0, and a custom tensor dataset is used.

The loss function used is CrossEntropyLoss() (multi-class classification) and Adam optimiser with a learning rate of 0.001.

The batch size used is 32.

The dataset was split into an 80-20 split.

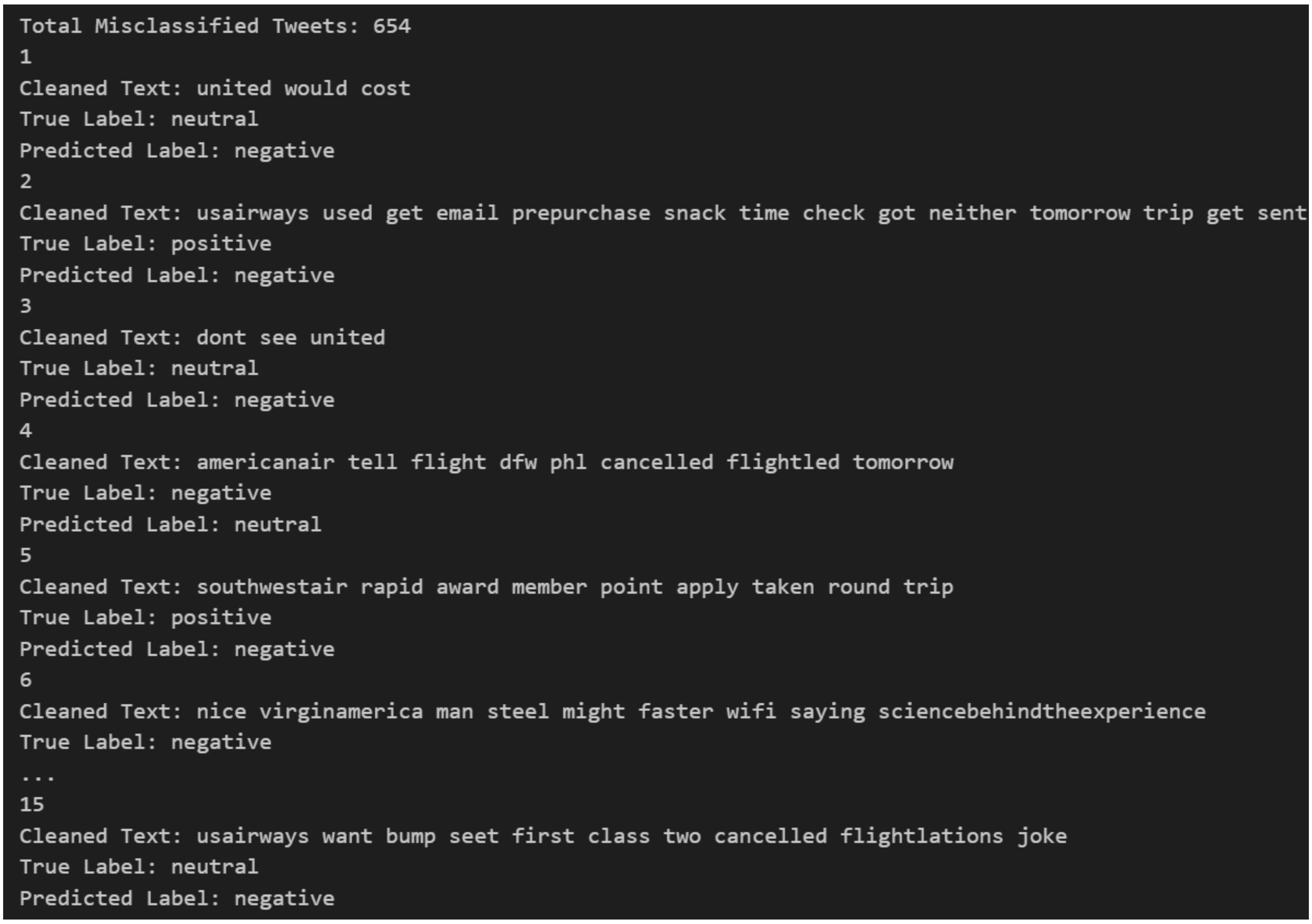
c. Initial Observations:

TF-IDF with DNN and LSTM performed better than better overall in comparison with other methods/models.

NB and Logistic regression are faster to train and easier to interpret, but showed noticeable imbalance, performing only on the dominant sentiment, which is negative. Also these models dont focus on contextual understanding, which isa huge downside.

The downside of Black box models is that they require many parameters and require tuning them, and are also very time-consuming.

Part-3:
a. Error Analysis:
We used Logistic Regression for doing the error analysis and studying misconstructions or misclassified sentiments.
We had a total of 654 misclassified sentiments.
One observation most of neutral tweets are classified as negative, which is due to the dataset being completely dominant in having negative tweets.



```
Total Misclassified Tweets: 654
1
Cleaned Text: united would cost
True Label: neutral
Predicted Label: negative
2
Cleaned Text: usairways used get email prepurchase snack time check got neither tomorrow trip get sent
True Label: positive
Predicted Label: negative
3
Cleaned Text: dont see united
True Label: neutral
Predicted Label: negative
4
Cleaned Text: americanair tell flight dfw phl cancelled flightled tomorrow
True Label: negative
Predicted Label: neutral
5
Cleaned Text: southwestair rapid award member point apply taken round trip
True Label: positive
Predicted Label: negative
6
Cleaned Text: nice virginamerica man steel might faster wifi saying sciencebehindtheexperience
True Label: negative
...
15
Cleaned Text: usairways want bump seet first class two cancelled flightlations joke
True Label: neutral
Predicted Label: negative
```

Upon examining the tweets, tweets that have negative word of any context 'dont see united', 'united would cost' have been straight away classified as negative though they are neutral. Even sarcasm or subtle complaints are marked as negative.

Also, confusion between negative and positive is observed in tweets like cancellations or factual information.

b. Class Sensitivity & Confusion Patterns

Analyze per-class precision, recall, and F1-score:

Negative sentiment has has high precision across all the models, this can be attributed due to the imbalance nature of dataset

DNN evaluation analysis:

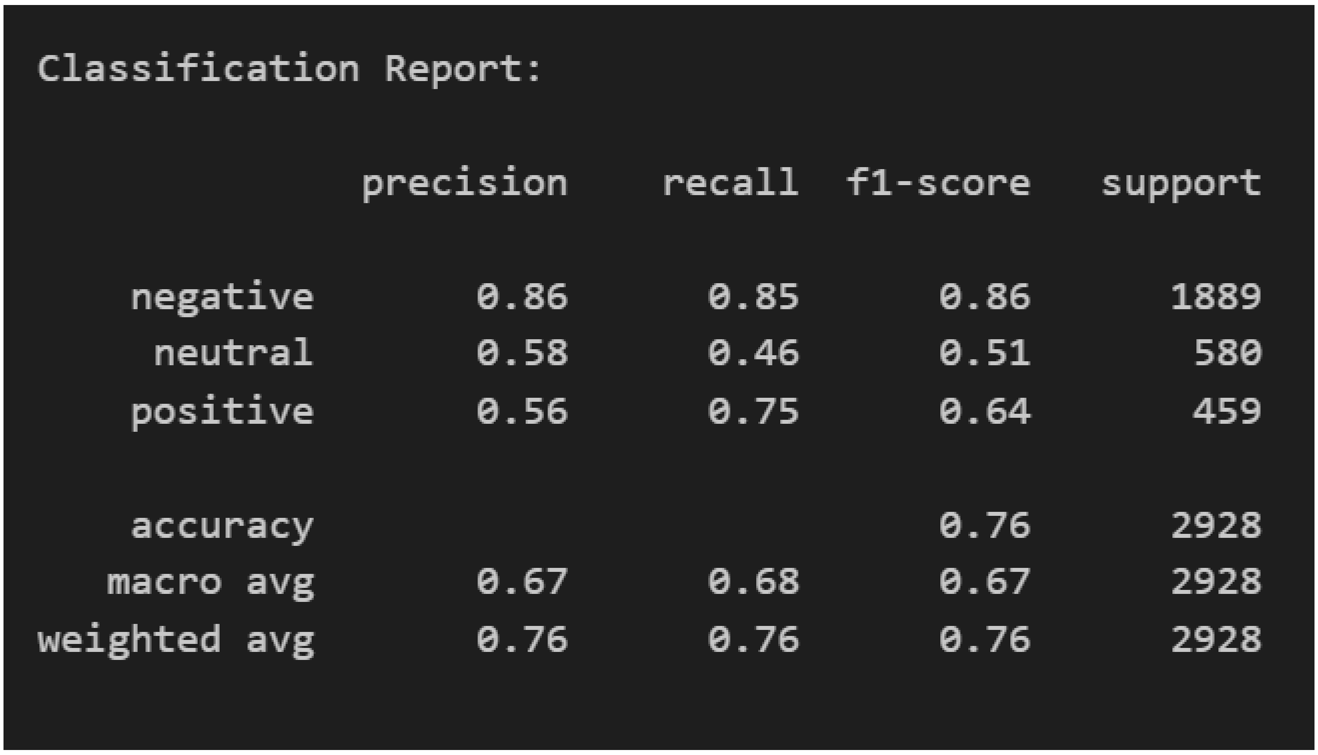
Has an accuracy around 70% with good balance between positive, negative, neutral sentiments.

Negative sentiment has a **precision of 84%**, which is again understandable due to the imbalance nature of dataset. Also the **F1 is 83%** and **recall is 83%**.

Positive sentiment has a **precision of 60%, recall is 47% and F1 is 53%.** This tells us that the model for the most part is good at classifying positive tweets from false positives, in comparison with other models like Logistic Regression and NB.

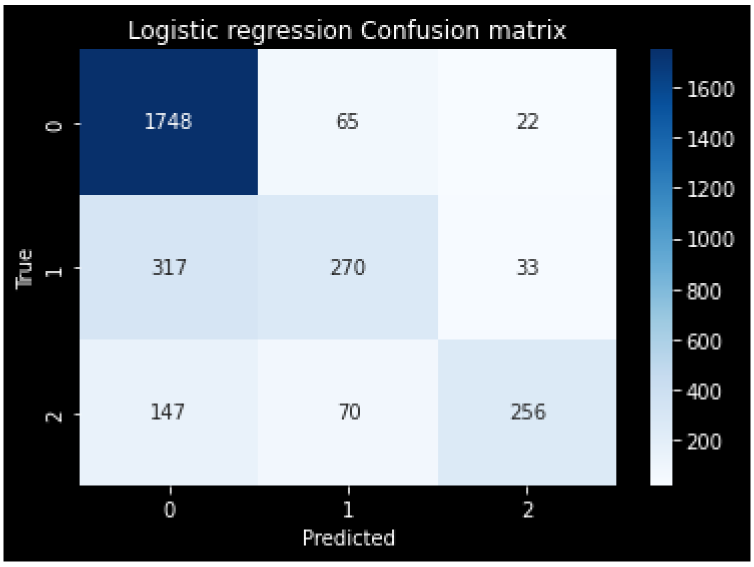
The neutral sentiment suffered in comparison with positive and negative. **41% precision, 49% recall, and F1 is 45%.** This shows that neutral tweets were marked as negative due to subtle tone and not completely shifting towards positive or negative, because tweets like these can sometimes get ambiguous and less expressive.

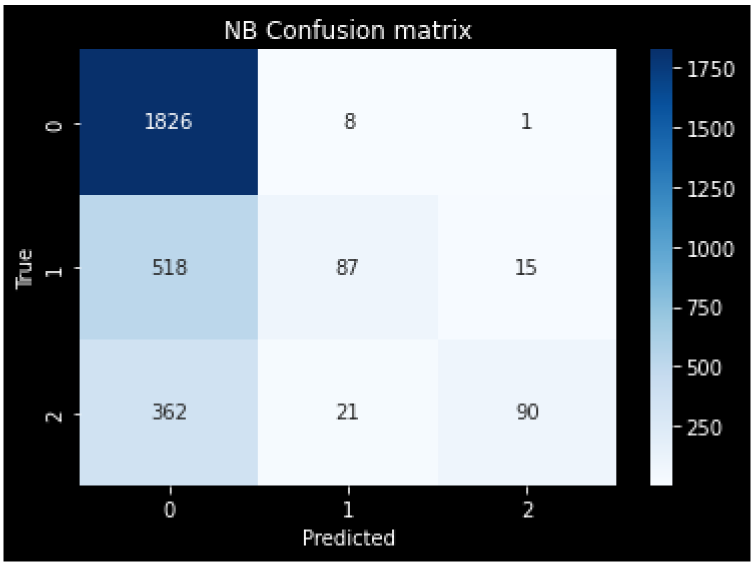
On the whole, DNN model has a good balance.

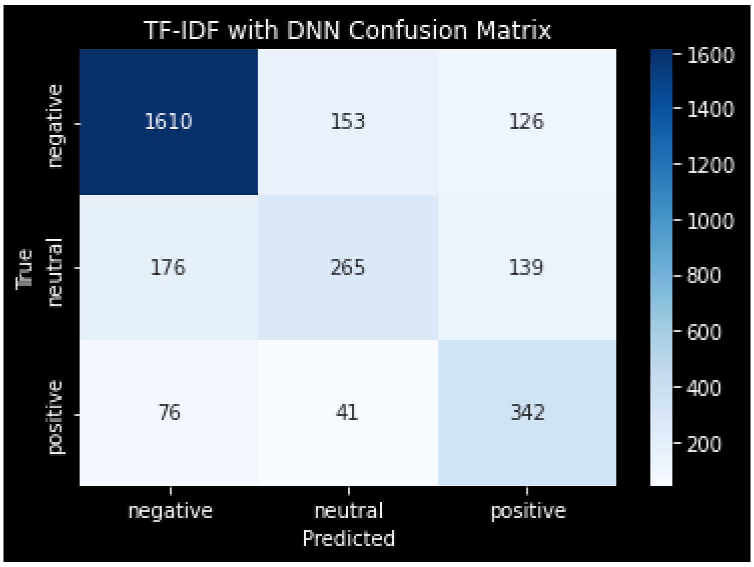


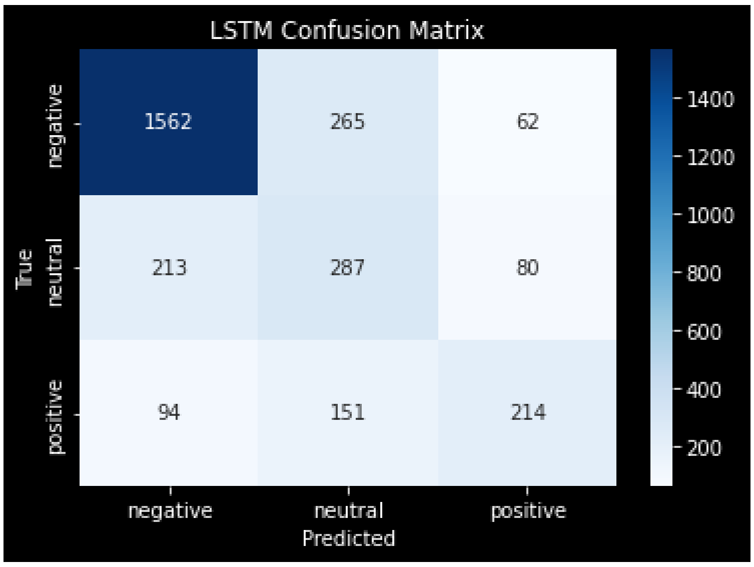
```
Classification Report:

              precision    recall  f1-score   support

    negative       0.86      0.85      0.86      1889
     neutral       0.58      0.46      0.51       580
    positive       0.56      0.75      0.64       459

    accuracy                           0.76      2928
   macro avg       0.67      0.68      0.67      2928
weighted avg       0.76      0.76      0.76      2928
```

Visualize confusion matrices for all models:









Discuss how class imbalance or text structure might impact model performance:
As the dataset is imbalance, there is inherit bias towards negative, which is observed in all models. This is clearly noticeable in negative sentence evaluation metric which has highest score than other sentiments across all the sentiments.

Structure too played an important role, many short tweets or subtle tweets are often misclassified. This can be overcome using models LSTM, RNN. Because tweets can sometimes be no so expressive or very subtle or sarcastic which makes the model missclassify.

. Explainability vs Performance: Comparison:
Compare explainable vs black-box models, is the performance gain from black-box models worth the loss in interpretability?

This project used 2 blacbox models and 2 interpreable models

Blacbox models are observed to be performing than interpretable models. This can be partly due ti vector representation because here used TF-IDF which does not contain contextual information and it captures only linear relationships.

Where as blackbox models are also scope to capture non linear relations and also they can learn from context.

But there is a trade off, which is BlackBox models are more computationally expensive nad requires a lot of training, many parameters to be taken care of and also they have to be carefully finetuned, which also takes a lot time. Interpretable model are better in this context.