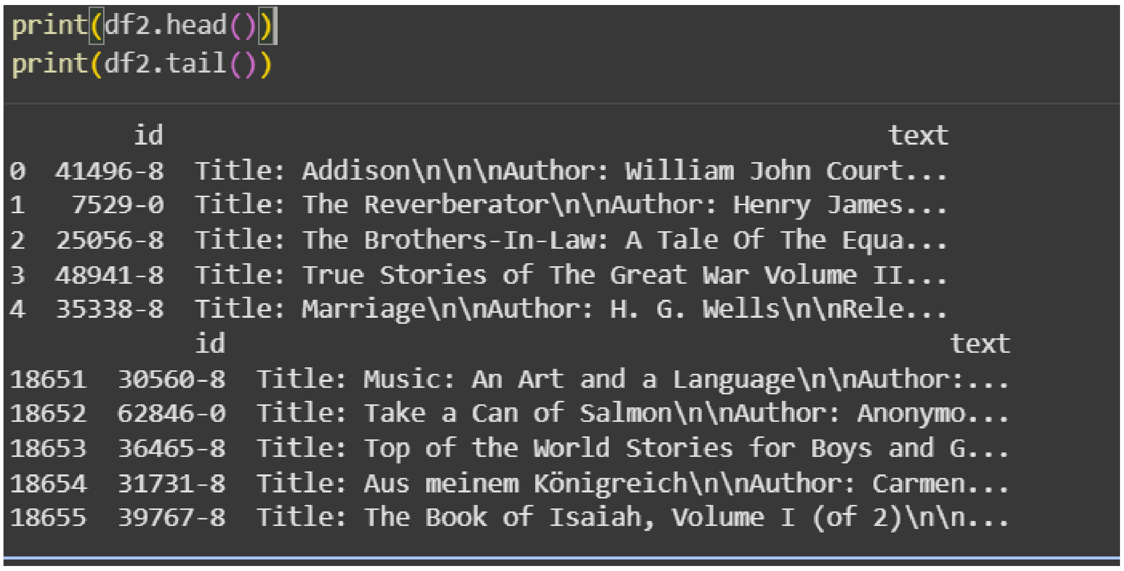
ST10150497 PDAN8412 PART 1

Recurrent Neural Networks

Dataset Justification

When searching for an appropriate dataset, I navigated to <huggingface.co> to search for a dataset that ideally had author names and their literary works included as columns. I came across the manu/project_gutenberg dataset <https://huggingface.co/datasets/manu/project_gutenberg> which included over 70,000 free eBooks. I decided to choose the English split of the database, which contained 61,300 rows. The dataset had 2 columns being "id" and "text", the text column would include each book's entire information from the cover to the end as well as author names, publications and copyright information. Therefore, each row was extremely large so I would have had a lot of cleaning to do.

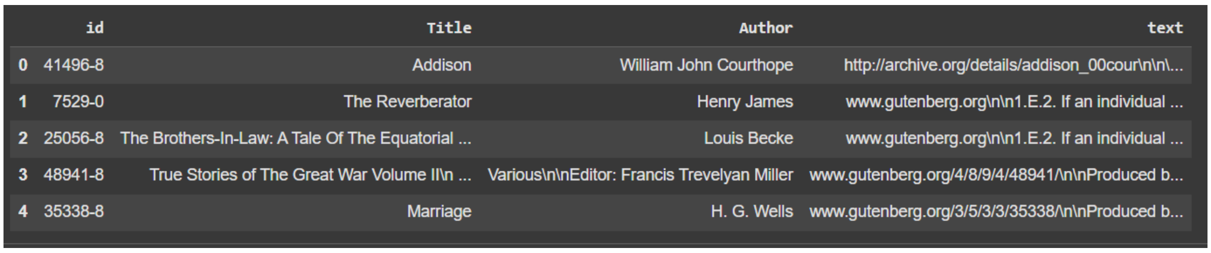


```
print(df2.head())
print(df2.tail())
```

```
         id                                               text
0   41496-8  Title: Addison\n\n\nAuthor: William John Court...
1    7529-0  Title: The Reverberator\n\nAuthor: Henry James...
2   25056-8  Title: The Brothers-In-Law: A Tale Of The Equa...
3   48941-8  Title: True Stories of The Great War Volume II...
4   35338-8  Title: Marriage\n\nAuthor: H. G. Wells\n\nRele...
             id                                               text
18651   30560-8  Title: Music: An Art and a Language\n\nAuthor:...
18652   62846-0  Title: Take a Can of Salmon\n\nAuthor: Anonymo...
18653   36465-8  Title: Top of the World Stories for Boys and G...
18654   31731-8  Title: Aus meinem Königreich\n\nAuthor: Carmen...
18655   39767-8  Title: The Book of Isaiah, Volume I (of 2)\n\n...
```

Figure 1. *Dataset head and tail before cleaning*

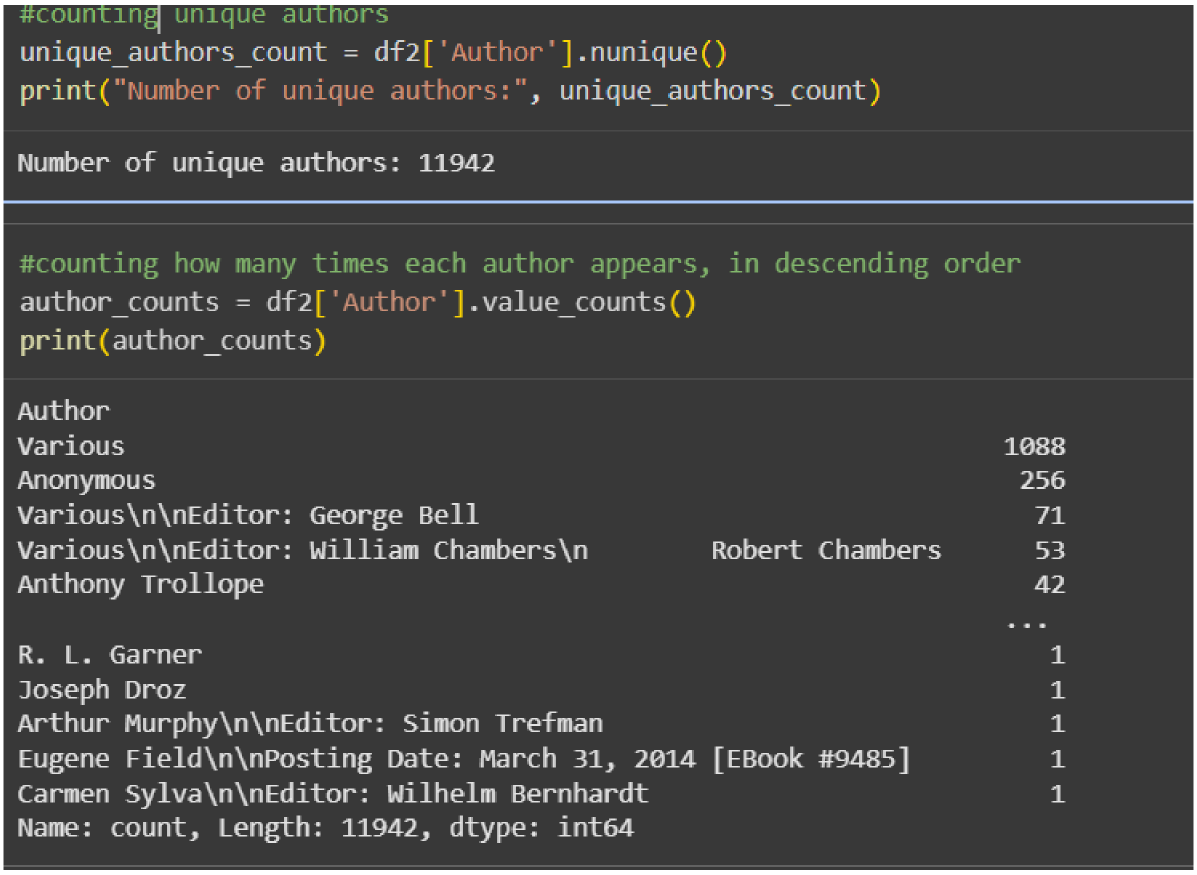
Figure 1. Shows the dataset before cleaning, there were only 2 rows being id and text and the text column had a lot of new lines and noise. I would have to begin feature engineering as well as removing unnecessary text that was written before the books actually started.



	id	Title	Author	text
0	41496-8	Addison	William John Courthope	http://archive.org/details/addison_00cour\n\n\...
1	7529-0	The Reverberator	Henry James	www.gutenberg.org\n\n1.E.2. If an individual ...
2	25056-8	The Brothers-In-Law: A Tale Of The Equatorial ...	Louis Becke	www.gutenberg.org\n\n1.E.2. If an individual ...
3	48941-8	True Stories of The Great War Volume II\n ...	Various\n\nEditor: Francis Trevelyan Miller	www.gutenberg.org/4/8/9/4/48941/\n\nProduced b...
4	35338-8	Marriage	H. G. Wells	www.gutenberg.org/3/5/3/3/35338/\n\nProduced b...

Figure 2. *Dataset head post feature engineering and text removal.*

I would engineer the "title" and "author" features from the text column, then go ahead and remove unnecessary text from the text column. I did this by remove all the text that came before a link, so all the text before "http" and or "www" would be removed as it would have nothing to do with the book. I would then move on to counting how many unique authors are in the dataset.

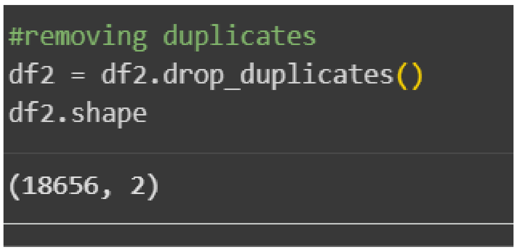


```
#counting unique authors
unique_authors_count = df2['Author'].nunique()
print("Number of unique authors:", unique_authors_count)

Number of unique authors: 11942
```

```
#counting how many times each author appears, in descending order
author_counts = df2['Author'].value_counts()
print(author_counts)

Author
Various                                                                 1088
Anonymous                                                                256
Various\n\nEditor: George Bell                                            71
Various\n\nEditor: William Chambers\n        Robert Chambers              53
Anthony Trollope                                                          42
                                                                        ...
R. L. Garner                                                               1
Joseph Droz                                                                1
Arthur Murphy\n\nEditor: Simon Trefman                                     1
Eugene Field\n\nPosting Date: March 31, 2014 [EBook #9485]                 1
Carmen Sylva\n\nEditor: Wilhelm Bernhardt                                  1
Name: count, Length: 11942, dtype: int64
```

Figure 3. *Counting the number of unique authors in the dataset.*

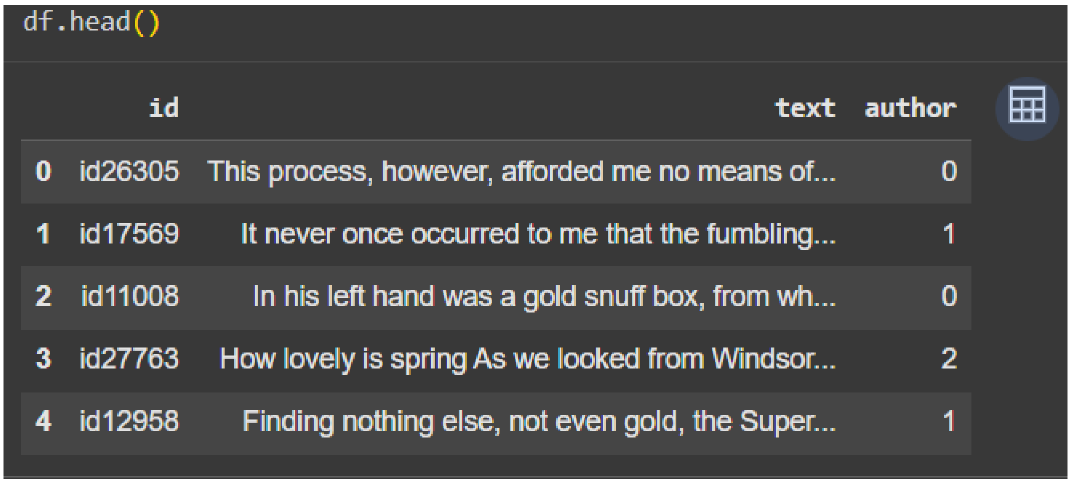
I used a sample of 30k rows from the 61.3k dataset as the entire dataset came out to 10GB in size as I was having memory issues attempting to analyse and clean the sample already. I would continuously run into out of memory issues with Visual Studio Code when analysing the sample, so the entire dataset was out of the question. During the cleaning process I removed duplicate rows which brought the dataset down to 18656 rows.



```
#removing duplicates
df2 = df2.drop_duplicates()
df2.shape

(18656, 2)
```

Figure 4. *Dropping duplicate rows*

Having 11942 in a dataset with 18656 was not ideal I would not have enough entries for that many classes, and the dataset was also extremely unbalanced. It did not help that the majority of the authors were labelled as anonymous or various. This is when I would decide not to move forward with this dataset. I use my reasons for not moving forward with this initial dataset as part of my justification for choosing the dataset I carried on with. I would continue to navigate huggingface and come across the spooky-author-identification dataset at < >. The dataset has 19,000 rows therefore it meets the https://huggingface.co/datasets/hkadxqq/spooky-author-identification 10,000-row minimum outlined in the task. There are three columns being id, text and author. Unlike the previous dataset this one already had an author column I could use as my target variable. The author column is already numerically encoded which removes the need for me to encode author names. The text column has the author's literary work, which is going to be used as the input variable. The dataset only has 3 author classes for its 19,000 rows meaning there will ideally be sufficient training data for each class.

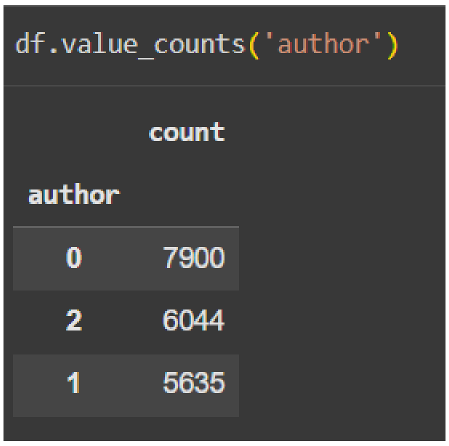


```
df.head()
```

	id	text	author
0	id26305	This process, however, afforded me no means of...	0
1	id17569	It never once occurred to me that the fumbling...	1
2	id11008	In his left hand was a gold snuff box, from wh...	0
3	id27763	How lovely is spring As we looked from Windsor...	2
4	id12958	Finding nothing else, not even gold, the Super...	1

Figure 5. *Initial head*

The initial head of the dataset suggests that there are not going to be as many preprocessing and cleaning steps for me to take in comparison to the previous dataset.



```
df.value_counts('author')
```

author	count
0	7900
2	6044
1	5635

Figure 6. *Class Distribution*

My final justification for this dataset is that there is no extreme class imbalance. The classes are not perfectly balanced with class 1 have 2000 less rows than class 0 however I plan to address with imbalance using class weights. If necessary, I will apply

SMOTE. SMOTE is a method for upsampling minority classes without overfitting. It accomplishes this by creating synthetic samples in feature space that are close to the other points that belong to the minority class(Maklin, 2022).

Analysis Plan

I will begin by finding an appropriate dataset for the analysis. The dataset should have a minimum of 10,000 rows and at the very least include author names as well as the works of the author regardless of whether they are already split into separate columns. Once I have identified an appropriate dataset, I will have to load it into my notebook. If the dataset is obtained from Kaggle I can just download it, place it in the same directory as my notebook and load it with pandas. Should the source be a website such as huggingface I will have to look into tutorials on how to load the dataset either from the website or YouTube. Once I have imported the required libraries I will load the dataset into a dataframe then save the dataframe to a csv so I can have offline access to the dataset.

I will then begin my analysis by inspecting the shape of the dataframe to ensure it was loaded properly by checking that the number of rows and column match what they should be where the dataset is hosted on. I will then inspect the head of the dataframe to see the first few rows and inspect if the columns are labelled. I will then inspect the data types, should there already be an author column I need to know its data type for the purpose of numeric encoding.

The next step will then be cleaning the data. I will begin by checking for missing values and duplicates and removing them. I will be dropping unnecessary columns such as “id” columns and any other column that does not provide information on author's writing style .Should the dataset not have an author column I will perform feature engineering on the appropriate column and create an author column.

Then I will inspect the class distribution so I can check for imbalances and depending on their severity how to address them. Should the imbalance be minor I will use class weights, should it be significant I will apply SMOTE. I will perform further feature engineering to extract word count, character count, word density and punctuation for my EDA. I will use the distribution of word count to assess how I will go about padding and adjusting maxlen. I will then plot the engineered features to visualize the differences in stylometric features across the classes.

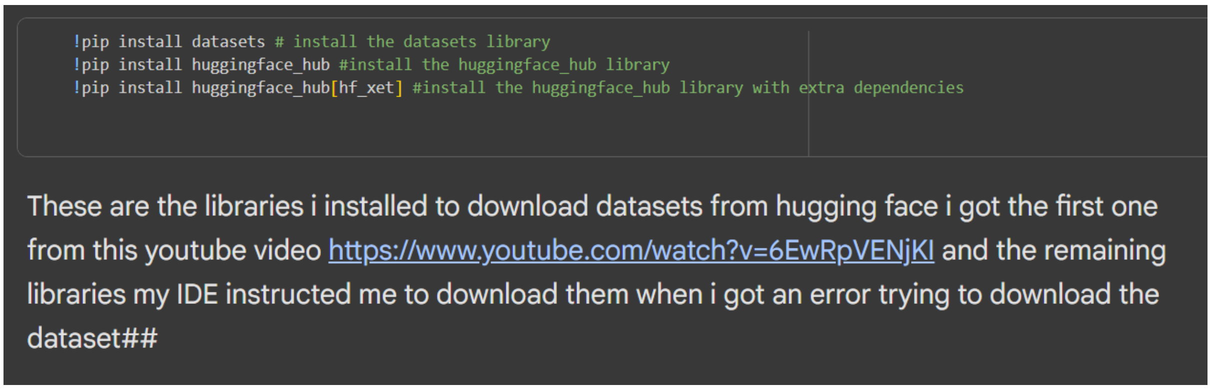
Going into preprocessing I will perform case normalization. Case normalization is the process of changing all the text's characters to the same case, usually lowercase. This ensures consistency in text representation(Mousavi, 2025). I will then begin with tokenizing the data. Tokenization is mostly used to transform textual data into a numerical representation that machine learning algorithms can understand (Ashraf, 2024). I will experiment with the number of words for my vocabulary size then fit the tokenizer to the column that has the author's text and then convert the text to sequences. Afterwards, I will print an index for the author text then print the same index from the sequence to check if the tokenization worked.

Since I will have visualised the word count distribution from my Exploratory Data Analysis, I will be able to select an appropriate maxlen for my padding sequence that I can experiment with. As I plan to tokenize the data I will apply a pad_sequences. Pad_sequences is used to ensure input sequences have the same length by padding them with a specific value (Srivatsavaya, 2023). After looking at the word count distribution in my EDA, I plan to pad my sequences to a number that allows for most of the input to be the same length.

I will then implement class_weights or SMOTE to address any potential class imbalances. Class_weights assigns majority class samples with lower weights and minority class samples with higher weights. This allows the model to pay more attention to the minority class so It gives better predictions for it (Abhinav, 2023).

Next I will carry out my 80/20 train_test split and move on to developing my model. I plan to make a keras.Sequential model for my RNN so I can stack layers linearly. Beginning with my embedding layer, then for my LSTM layers I will be using bidirectional LSTMs. BiLSTMs are an extension of LSTMs that involve two LSTMs running in parallel, one processing input from start to end and the other from end to start (GeeksforGeeks, 2025). I am using BiLSTMs for their higher contextual understanding and improved accuracy. I will then add a hidden dense layer and a final output dense layer with 3 neurons for the 3 classes. I will use the adam optimizer, sparse categorical_entropy for loss and for metric I will use accuracy, I will go into further detail on these in my report. After compiling the model, I fit the model to the training data and class weights and begin with 5 epochs and experiment with it. After training I will evaluate the model's accuracy, print a classification report and a confusion matrix.

Report

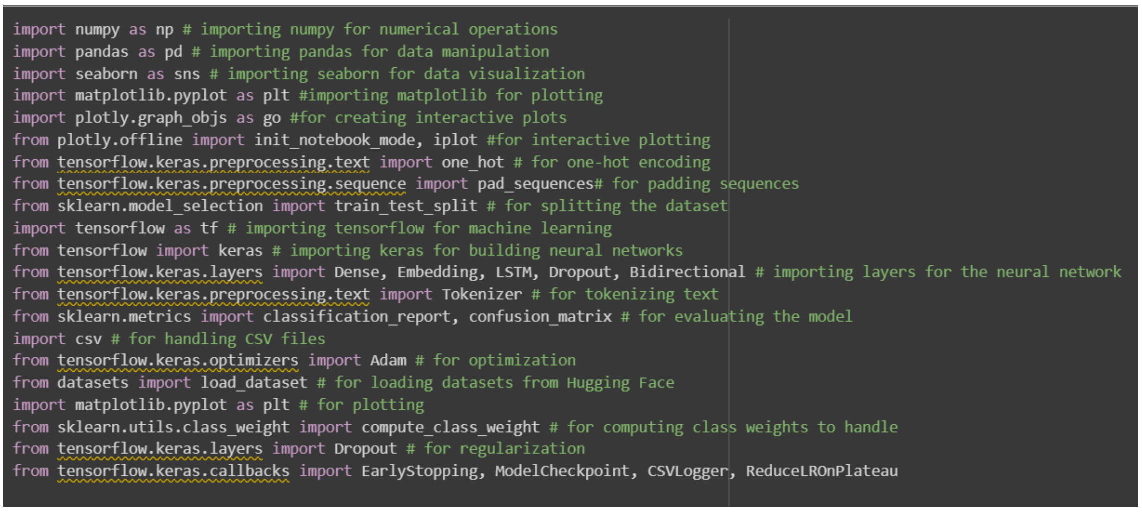


```
!pip install datasets # install the datasets library
!pip install huggingface_hub #install the huggingface_hub library
!pip install huggingface_hub[hf_xet] #install the huggingface_hub library with extra dependencies
```

These are the libraries i installed to download datasets from hugging face i got the first one from this youtube video https://www.youtube.com/watch?v=6EwRpVENjKI and the remaining libraries my IDE instructed me to download them when i got an error trying to download the dataset##

Figure 7. Library installation for hugging face

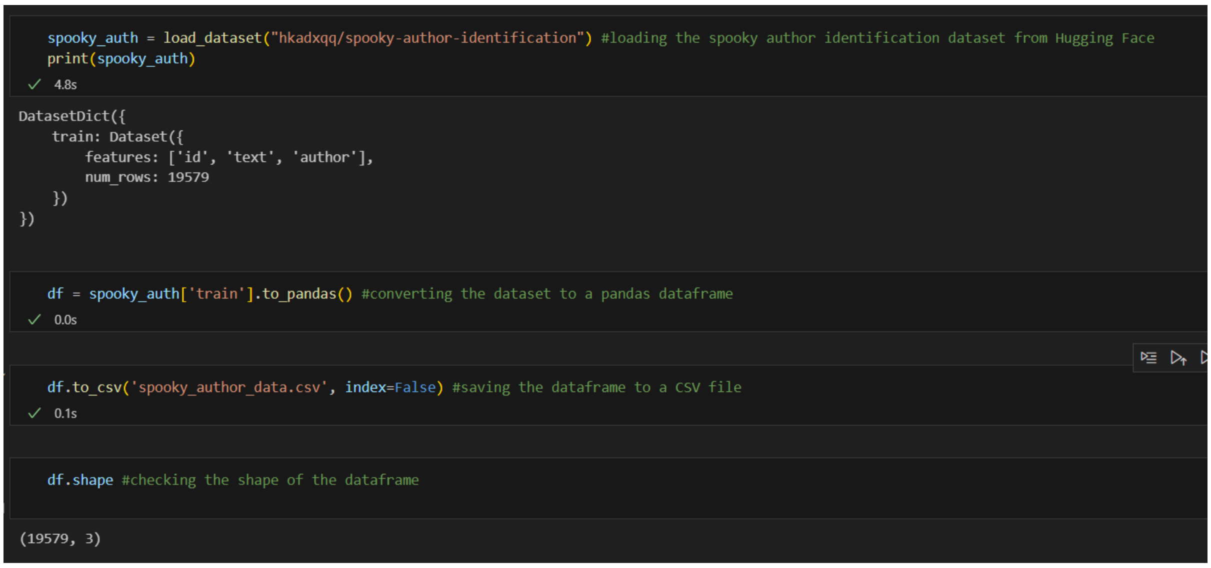
These are the prerequisite libraries I had to install to allow to be able to download datasets from hugging face. As hugging face does not allow you to directly download datasets I had to watch this video < https://www.youtube.com/watch?v=6EwRpVENjKI > to guide me on the download process. The video only mentioned downloading the "datasets" import, the following 2 imports I was instructed by my IDE to download them in an error message I received when trying to download the dataset.



```
import numpy as np # importing numpy for numerical operations
import pandas as pd # importing pandas for data manipulation
import seaborn as sns # importing seaborn for data visualization
import matplotlib.pyplot as plt #importing matplotlib for plotting
import plotly.graph_objs as go #for creating interactive plots
from plotly.offline import init_notebook_mode, iplot #for interactive plotting
from tensorflow.keras.preprocessing.text import one_hot # for one-hot encoding
from tensorflow.keras.preprocessing.sequence import pad_sequences# for padding sequences
from sklearn.model_selection import train_test_split # for splitting the dataset
import tensorflow as tf # importing tensorflow for machine learning
from tensorflow import keras # importing keras for building neural networks
from tensorflow.keras.layers import Dense, Embedding, LSTM, Dropout, Bidirectional # importing layers for the neural network
from tensorflow.keras.preprocessing.text import Tokenizer # for tokenizing text
from sklearn.metrics import classification_report, confusion_matrix # for evaluating the model
import csv # for handling CSV files
from tensorflow.keras.optimizers import Adam # for optimization
from datasets import load_dataset # for loading datasets from Hugging Face
import matplotlib.pyplot as plt # for plotting
from sklearn.utils.class_weight import compute_class_weight # for computing class weights to handle
from tensorflow.keras.layers import Dropout # for regularization
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, CSVLogger, ReduceLROnPlateau
```

Figure 8. *Library imports*

These are all the libraries I made use of. There are python libraries, Scikit-learn libraries and Keras libraries.



```
spooky_auth = load_dataset("hkadxqq/spooky-author-identification") #loading the spooky author identification dataset from Hugging Face
print(spooky_auth)
```
✓ 4.8s

```
DatasetDict({
    train: Dataset({
        features: ['id', 'text', 'author'],
        num_rows: 19579
    })
})
```

```
df = spooky_auth['train'].to_pandas() #converting the dataset to a pandas dataframe
```
✓ 0.0s

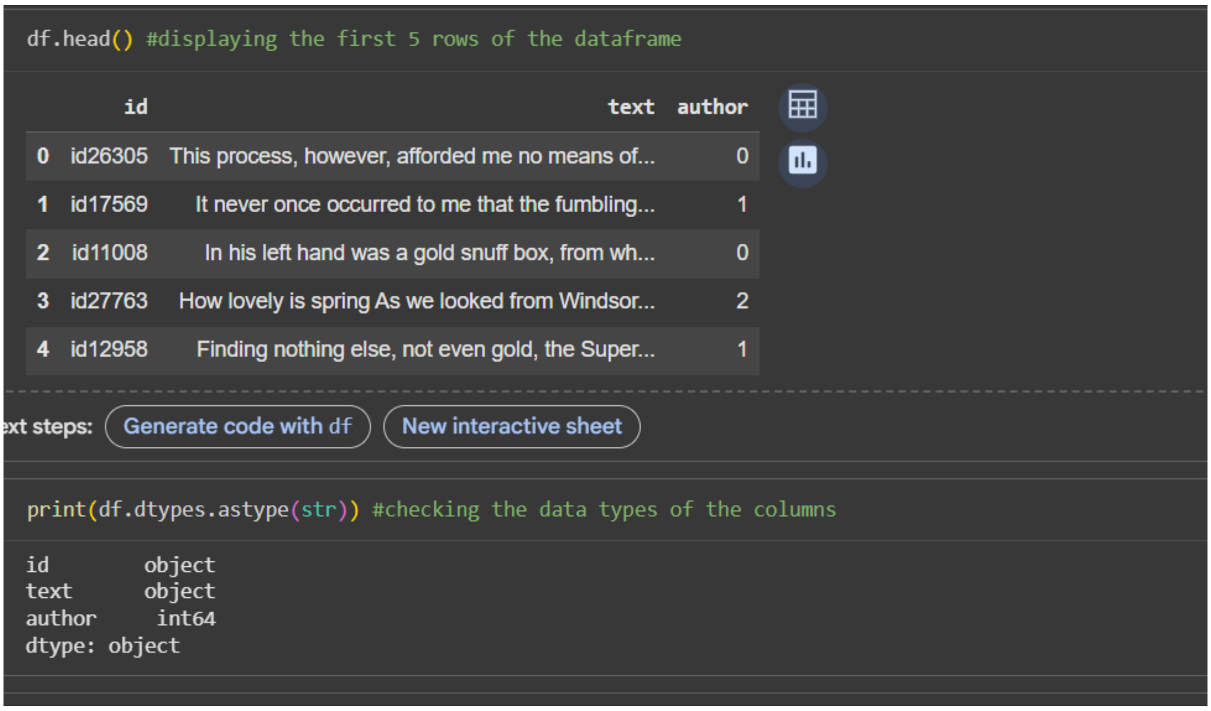
```
df.to_csv('spooky_author_data.csv', index=False) #saving the dataframe to a CSV file
```
✓ 0.1s

```
df.shape #checking the shape of the dataframe
```

```
(19579, 3)
```

Figure 9. *Downloading the huggingface dataset*

Now we get into the process of downloading the dataset from huggingface. After installing the necessary libraries, I just had to load the dataset by providing the name of the person who uploaded it then the name of the dataset. I downloaded the full 19579 row database there was no reason to sample it. After downloading it I converted it to a dataframe, then exported the dataframe into a csv so I could have the dataset in the same directory as my notebook. Then I printed the shape of the dataframe to ensure it matched with the original dataset.



```
df.head() #displaying the first 5 rows of the dataframe
```

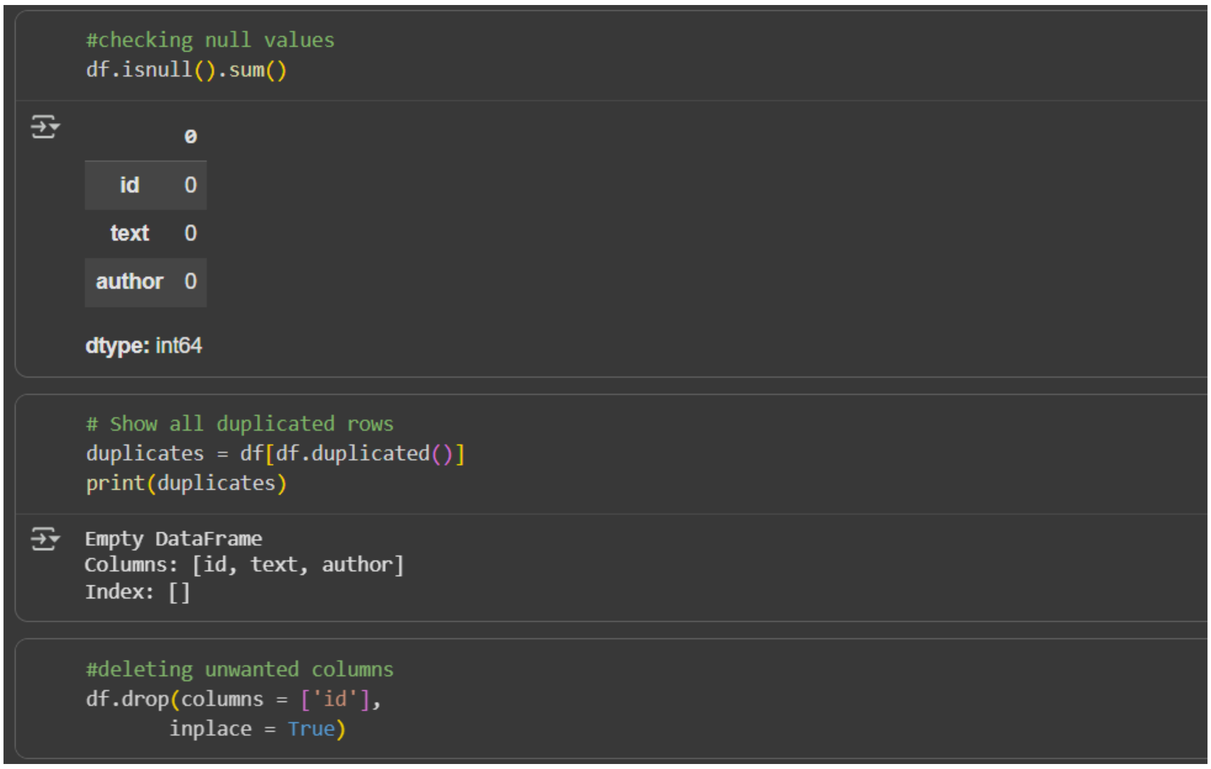
	id	text	author
0	id26305	This process, however, afforded me no means of...	0
1	id17569	It never once occurred to me that the fumbling...	1
2	id11008	In his left hand was a gold snuff box, from wh...	0
3	id27763	How lovely is spring As we looked from Windsor...	2
4	id12958	Finding nothing else, not even gold, the Super...	1

```
print(df.dtypes.astype(str)) #checking the data types of the columns
```

```
id         object
text       object
author      int64
dtype: object
```

Figure 10. *Data head and datatypes*

Here I am displaying the first 5 rows of the dataset to get a feel for it. We can see there is an id column that uniquely identifies each row, a text column that has the author's work and an author column for the authors. The id column is irrelevant and will be removed. The text column will be used for feature engineering and model training. We can see from the data types that the author column is already numerically encoded. If it were not I would have performed label encoding on it.



```
#checking null values
df.isnull().sum()
```

```
        0
id      0
text    0
author  0

dtype: int64
```

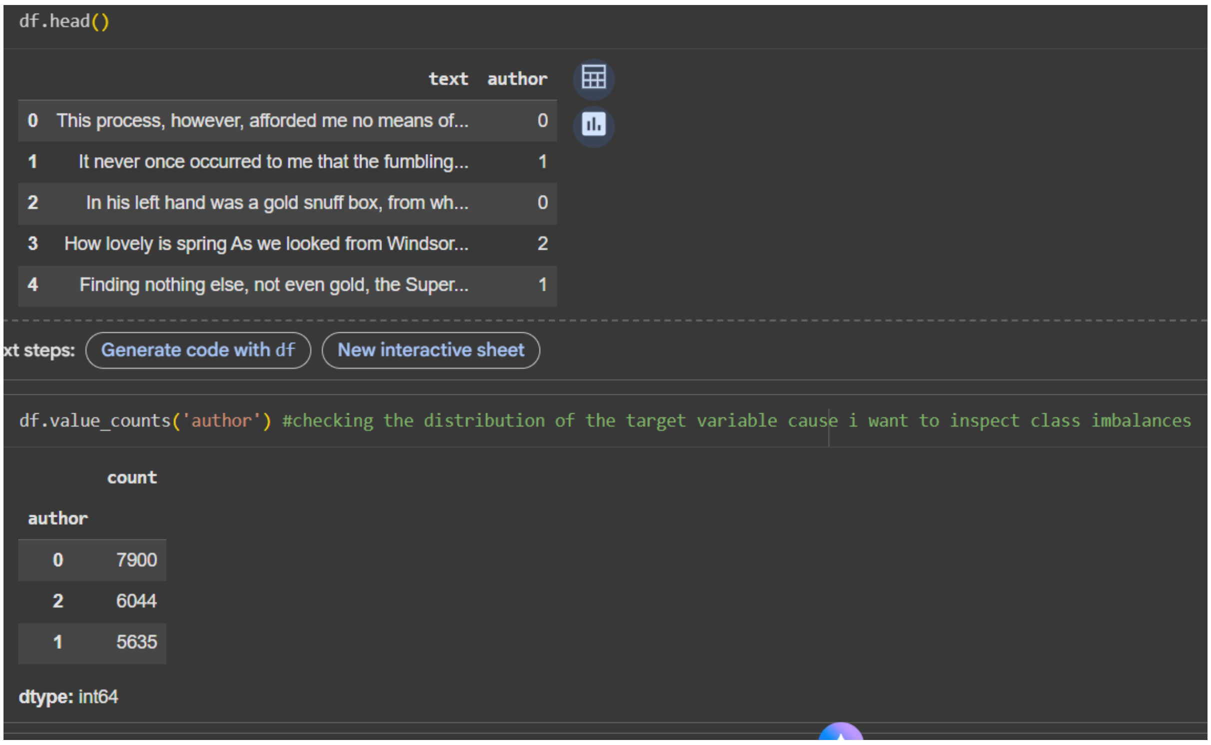
```
# Show all duplicated rows
duplicates = df[df.duplicated()]
print(duplicates)
```

```
Empty DataFrame
Columns: [id, text, author]
Index: []
```

```
#deleting unwanted columns
df.drop(columns = ['id'],
        inplace = True)
```

Figure 11. *Data cleaning*

Now we move on to the data cleaning. I begin by inspecting for null and duplicate values and as we can see there are not any, so no removing needs to be done there. I then removed the id column and used the inplace function so I didn't have to reassign the dataframe.



```
df.head()
```

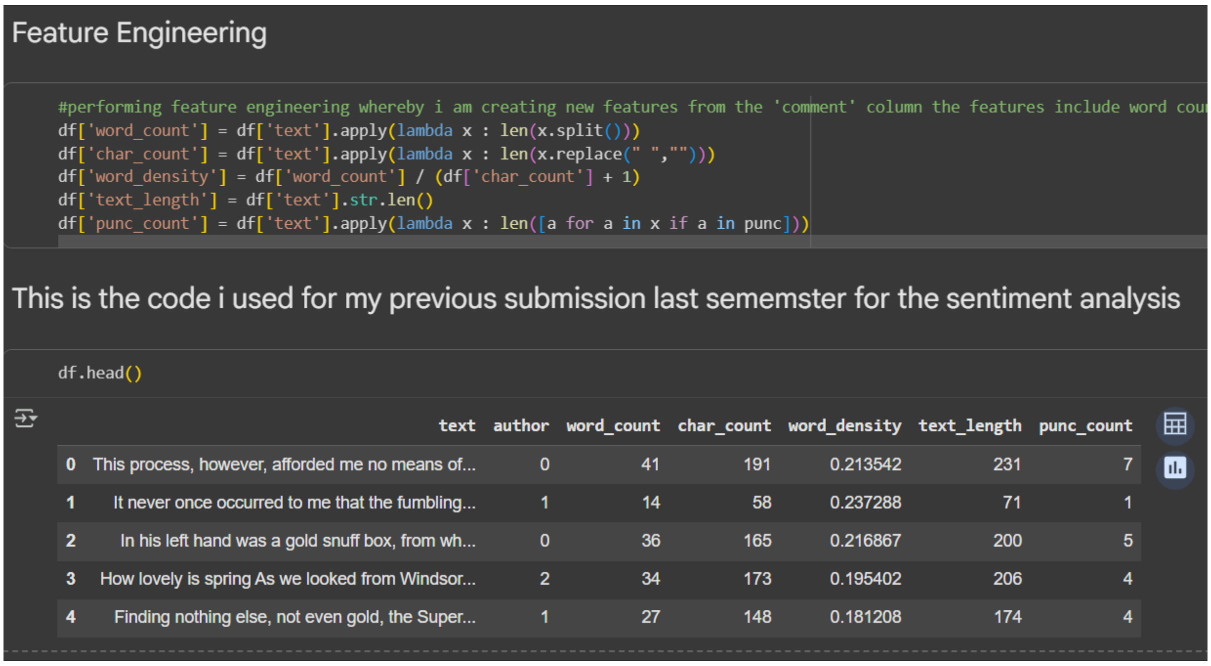
```
                                                text  author
0  This process, however, afforded me no means of...       0
1     It never once occurred to me that the fumbling...     1
2      In his left hand was a gold snuff box, from wh...    0
3   How lovely is spring As we looked from Windsor...       2
4     Finding nothing else, not even gold, the Super...     1
```

```
df.value_counts('author') #checking the distribution of the target variable cause i want to inspect class imbalances
```

```
        count
author
0        7900
2        6044
1        5635

dtype: int64
```

Figure 12. *Head inspection and class distribution*

Inspecting the data head to see that the id column has indeed been deleted. Then I'm viewing the class distribution using value counts to check for imbalances. As we can see there is a minor imbalance within the classes therefore I will be fitting class_weights to the model. As I explained in my plan class_weights assigns higher weights to minority classes and lower weights to majority classes. This allows it to make better predictions for minority classes.



Feature Engineering

```
#performing feature engineering whereby i am creating new features from the 'comment' column the features include word cou
df['word_count'] = df['text'].apply(lambda x : len(x.split()))
df['char_count'] = df['text'].apply(lambda x : len(x.replace(" ","")))
df['word_density'] = df['word_count'] / (df['char_count'] + 1)
df['text_length'] = df['text'].str.len()
df['punc_count'] = df['text'].apply(lambda x : len([a for a in x if a in punc]))
```

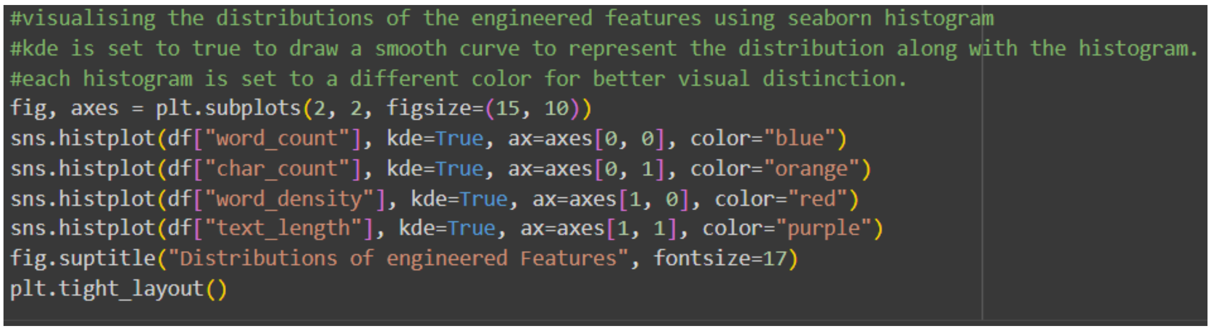
This is the code i used for my previous submission last sememster for the sentiment analysis

```
df.head()
```

```
                                                text  author  word_count  char_count  word_density  text_length  punc_count
0  This process, however, afforded me no means of...       0          41         191      0.213542          231           7
1     It never once occurred to me that the fumbling...     1          14          58      0.237288           71           1
2      In his left hand was a gold snuff box, from wh...    0          36         165      0.216867          200           5
3   How lovely is spring As we looked from Windsor...       2          34         173      0.195402          206           4
4     Finding nothing else, not even gold, the Super...     1          27         148      0.181208          174           4
```

Figure 13. *Feature engineering.*

Here I am engineering stylometric features that I am going to use for my exploratory data analysis. I reused the code I submitted for the previous semester's sentiment analysis. The features are word count, character count, word density, text length and punctuation count. After engineering the features I displayed the data head to see if the engineering worked and to see the distribution across the first 5 rows.



```
#visualising the distributions of the engineered features using seaborn histogram
#kde is set to true to draw a smooth curve to represent the distribution along with the histogram.
#each histogram is set to a different color for better visual distinction.
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
sns.histplot(df["word_count"], kde=True, ax=axes[0, 0], color="blue")
sns.histplot(df["char_count"], kde=True, ax=axes[0, 1], color="orange")
sns.histplot(df["word_density"], kde=True, ax=axes[1, 0], color="red")
sns.histplot(df["text_length"], kde=True, ax=axes[1, 1], color="purple")
fig.suptitle("Distributions of engineered Features", fontsize=17)
plt.tight_layout()
```

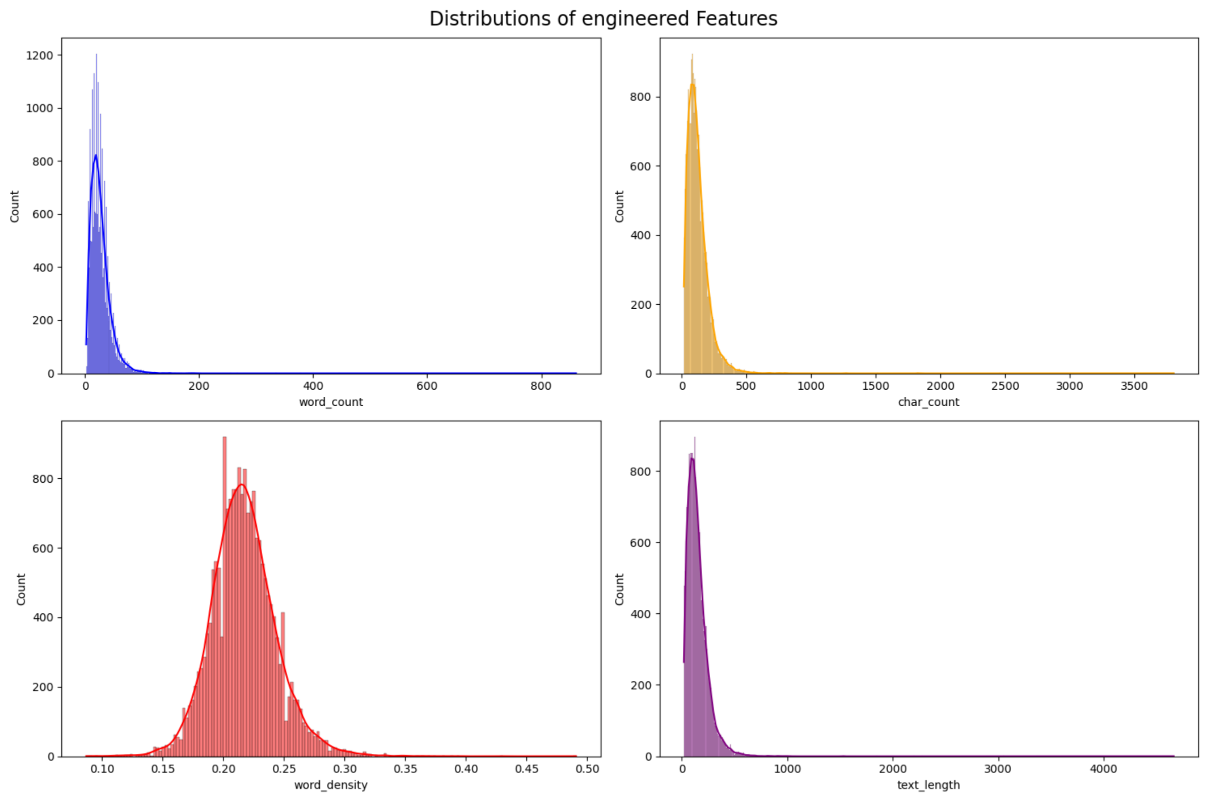
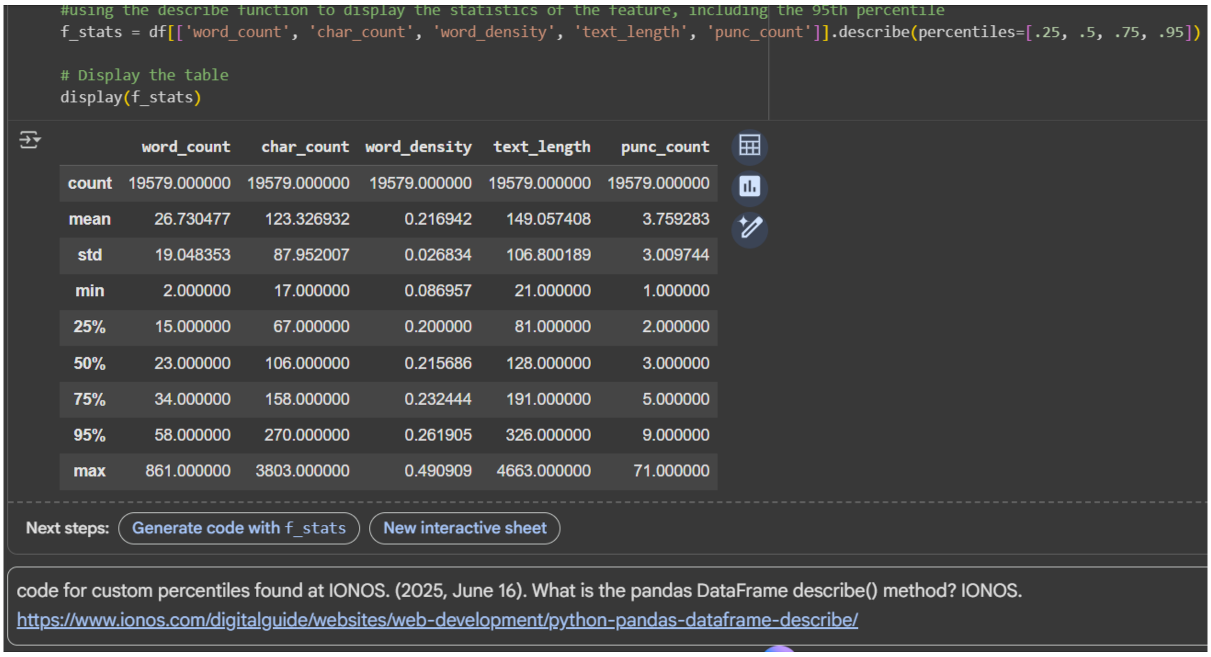


Figure 14. *Distribution of engineered features*

Once again reusing the code from the previously mentioned sentiment analysis I plotted a histogram to visualise the distribution of the features. The y-axis is the number of rows and the x-axes are the features. To give a clear interpretation of the histograms, I provided a statistics table in ***Figure 15.***



```
#using the describe function to display the statistics of the feature, including the 95th percentile
f_stats = df[['word_count', 'char_count', 'word_density', 'text_length', 'punc_count']].describe(percentiles=[.25, .5, .75, .95])

# Display the table
display(f_stats)
```

	word_count	char_count	word_density	text_length	punc_count
count	19579.000000	19579.000000	19579.000000	19579.000000	19579.000000
mean	26.730477	123.326932	0.216942	149.057408	3.759283
std	19.048353	87.952007	0.026834	106.800189	3.009744
min	2.000000	17.000000	0.086957	21.000000	1.000000
25%	15.000000	67.000000	0.200000	81.000000	2.000000
50%	23.000000	106.000000	0.215686	128.000000	3.000000
75%	34.000000	158.000000	0.232444	191.000000	5.000000
95%	58.000000	270.000000	0.261905	326.000000	9.000000
max	861.000000	3803.000000	0.490909	4663.000000	71.000000

Next steps: Generate code with f_stats New interactive sheet

code for custom percentiles found at IONOS. (2025, June 16). What is the pandas DataFrame describe() method? IONOS. https://www.ionos.com/digitalguide/websites/web-development/python-pandas-dataframe-describe/

Figure 15. Statistics table

With this table we can interpret the histograms better. We can also use the word_count statistic to help determine the maxlen for when applying padding. As we can see 95% percent of the word_count for all the rows is 58 words with the largest number of words for an entry being 861.

```
fig = go.Figure(data=traces, layout=layout) #creating a figure with the traces and layout
iplot(fig) #displaying the plotly histogram
```

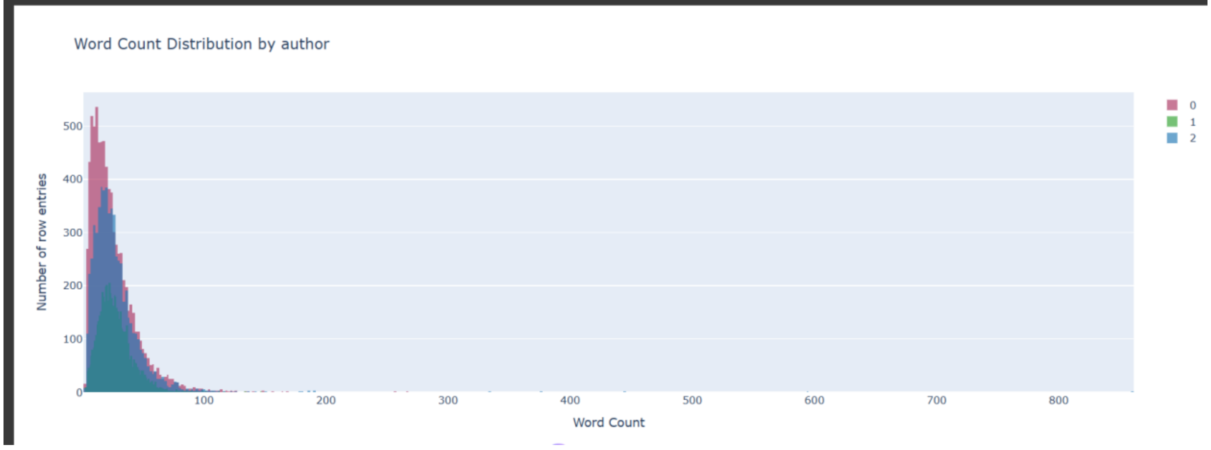
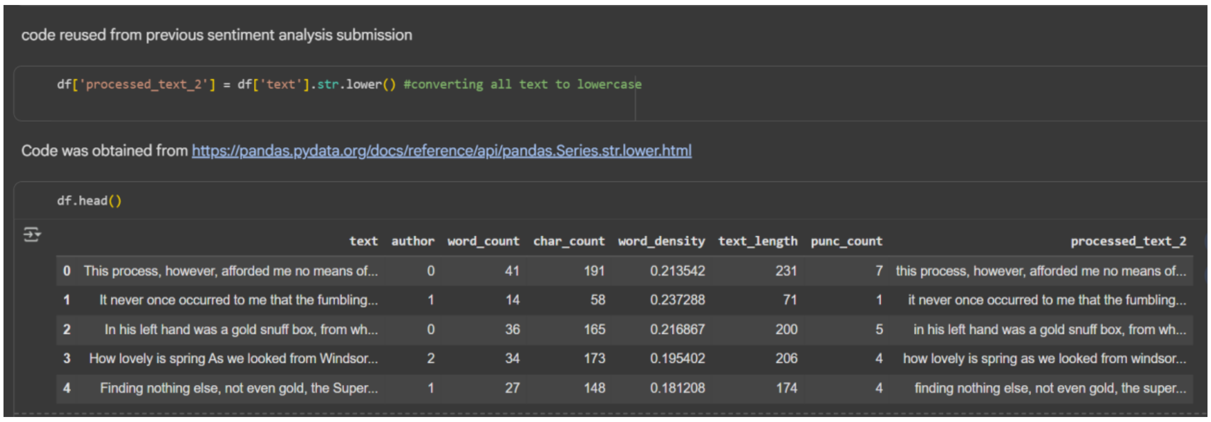


Figure 16. Word count distribution by author

Here we can see the word count distribution across the three author classes. From this we can see author 0 has a really high number of entries (563) with a low word count of 12 words. It makes sense for author 0 to have this high number of entries as in the dataset it has the highest number of entries followed by author 2 then 1. This chart gives us an enhanced visual of the class distribution so we can better understand the imbalance.


code reused from previous sentiment analysis submission

```
df['processed_text_2'] = df['text'].str.lower() #converting all text to lowercase
```

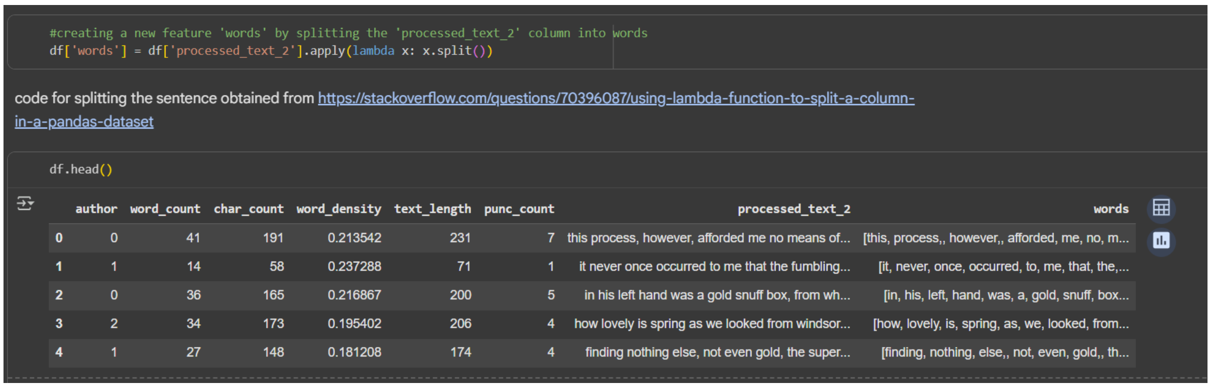
Code was obtained from https://pandas.pydata.org/docs/reference/api/pandas.Series.str.lower.html

```
df.head()
```

```
                                               text  author  word_count  char_count  word_density  text_length  punc_count                                  processed_text_2
0   This process, however, afforded me no means of...       0          41         191      0.213542          231           7  this process, however, afforded me no means of...
1      It never once occurred to me that the fumbling...       1          14          58      0.237288           71           1     it never once occurred to me that the fumbling...
2       In his left hand was a gold snuff box, from wh...       0          36         165      0.216867          200           5      in his left hand was a gold snuff box, from wh...
3    How lovely is spring As we looked from Windsor...       2          34         173      0.195402          206           4   how lovely is spring as we looked from windsor...
4       Finding nothing else, not even gold, the Super...       1          27         148      0.181208          174           4      finding nothing else, not even gold, the super...
```

Figure 18. Case normalization

Moving on to the data preprocessing I begin by performing case normalization. I converted all the text to lowercase for consistency across all the text. Then I printed the data head to see if the case normalization worked.

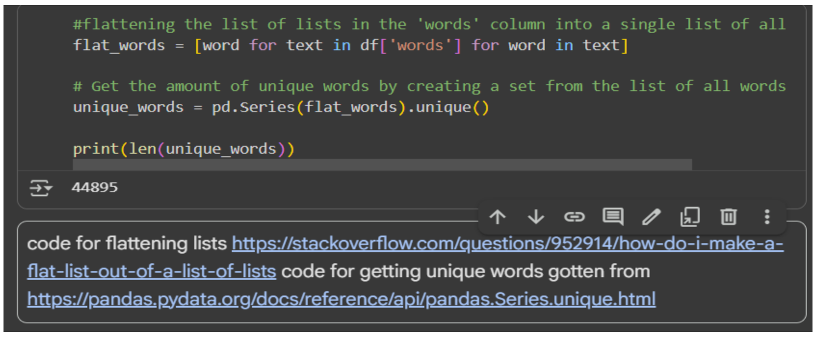


```
#creating a new feature 'words' by splitting the 'processed_text_2' column into words
df['words'] = df['processed_text_2'].apply(lambda x: x.split())
```

code for splitting the sentence obtained from https://stackoverflow.com/questions/70396087/using-lambda-function-to-split-a-column-in-a-pandas-dataset

```
df.head()
```

```
   author  word_count  char_count  word_density  text_length  punc_count                                  processed_text_2                                              words
0       0          41         191      0.213542          231           7  this process, however, afforded me no means of...  [this, process,, however,, afforded, me, no, m...
1       1          14          58      0.237288           71           1     it never once occurred to me that the fumbling...     [it, never, once, occurred, to, me, that, the,...
2       0          36         165      0.216867          200           5      in his left hand was a gold snuff box, from wh...      [in, his, left, hand, was, a, gold, snuff, box...
3       2          34         173      0.195402          206           4   how lovely is spring as we looked from windsor...    [how, lovely, is, spring, as, we, looked, from...
4       1          27         148      0.181208          174           4      finding nothing else, not even gold, the super...     [finding, nothing, else,, not, even, gold,, th...
```



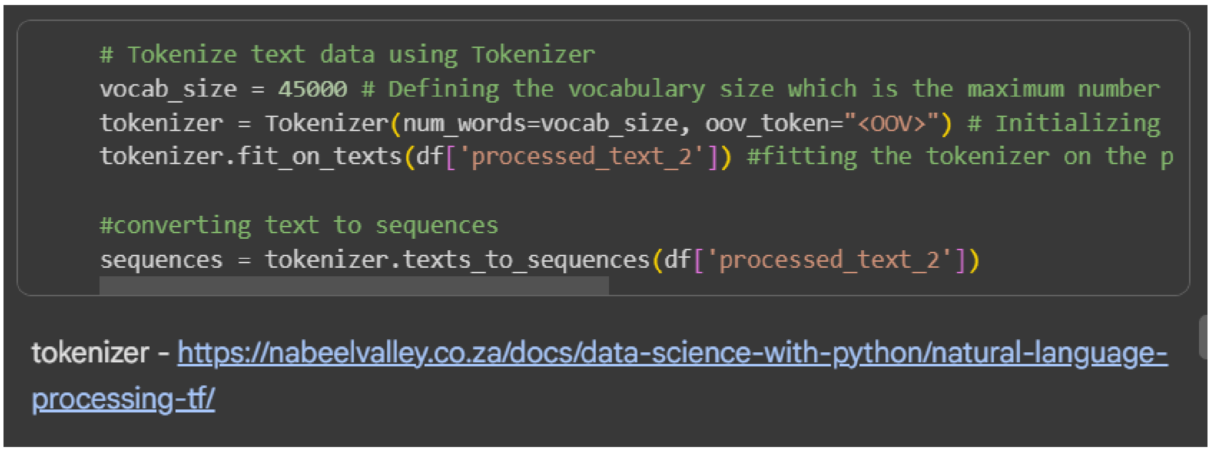
```
#flattening the list of lists in the 'words' column into a single list of all
flat_words = [word for text in df['words'] for word in text]

# Get the amount of unique words by creating a set from the list of all words
unique_words = pd.Series(flat_words).unique()

print(len(unique_words))
```

```
44895
```

code for flattening lists https://stackoverflow.com/questions/952914/how-do-i-make-a-flat-list-out-of-a-list-of-lists code for getting unique words gotten from https://pandas.pydata.org/docs/reference/api/pandas.Series.unique.html

Figure 19. *Feature engineering and unique word count*

In this above figure I carried out further feature engineering. I made a new words column where the words from processed_text_2 are split and placed into lists. I did this so I could then flatten the list and count the number of unique words in the column so when I began tokenization I would have an idea of what to set my vocab size. From the output we can see the number of unique words is 44895 so my vocab size should be just above that.

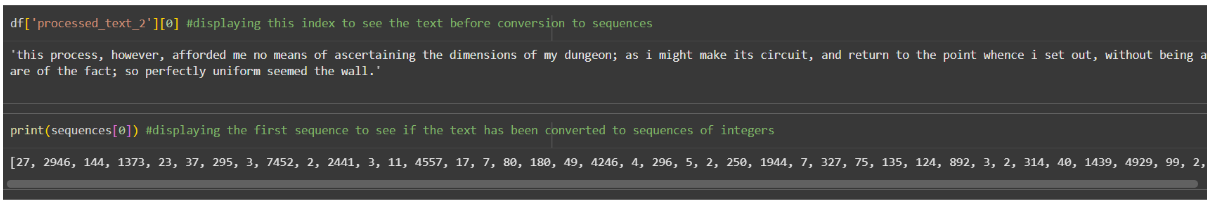


```
# Tokenize text data using Tokenizer
vocab_size = 45000 # Defining the vocabulary size which is the maximum number
tokenizer = Tokenizer(num_words=vocab_size, oov_token="<OOV>") # Initializing
tokenizer.fit_on_texts(df['processed_text_2']) #fitting the tokenizer on the p

#converting text to sequences
sequences = tokenizer.texts_to_sequences(df['processed_text_2'])
```

tokenizer - https://nabeelvalley.co.za/docs/data-science-with-python/natural-language-processing-tf/

Figure 20. *Vocab size declaration and tokenization*

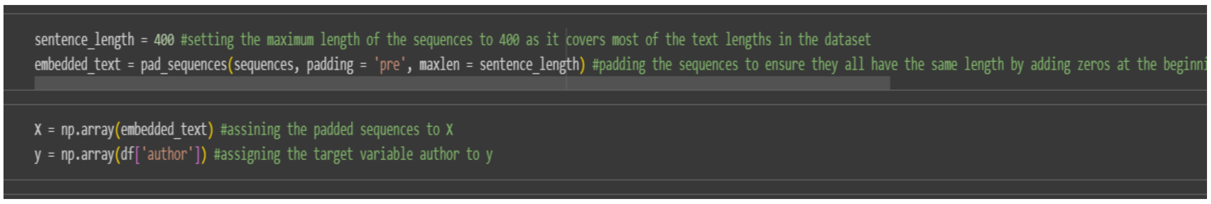
As the number of unique words in the processed_text_2 column is 44895, I have set the vocab_size to 45,000, Vocabulary size in this case refers to the total number of unique tokens (Balakrishnan, 2024). As I have 44895 unique words I will be assigning them to 45,000 unique tokens to make sure every unique word has a unique token. I am doing this as I want most of each author's stylistic cues to be captured.



```
df['processed_text_2'][0] #displaying this index to see the text before conversion to sequences

'this process, however, afforded me no means of ascertaining the dimensions of my dungeon; as i might make its circuit, and return to the point whence i set out, without being a
are of the fact; so perfectly uniform seemed the wall.'

print(sequences[0]) #displaying the first sequence to see if the text has been converted to sequences of integers

[27, 2946, 144, 1373, 23, 37, 295, 3, 7452, 2, 2441, 3, 11, 4557, 17, 7, 80, 180, 49, 4246, 4, 296, 5, 2, 250, 1944, 7, 327, 75, 135, 124, 892, 3, 2, 314, 40, 1439, 4929, 99, 2,
```

Figure 21. *Checking to see if tokenization worked*

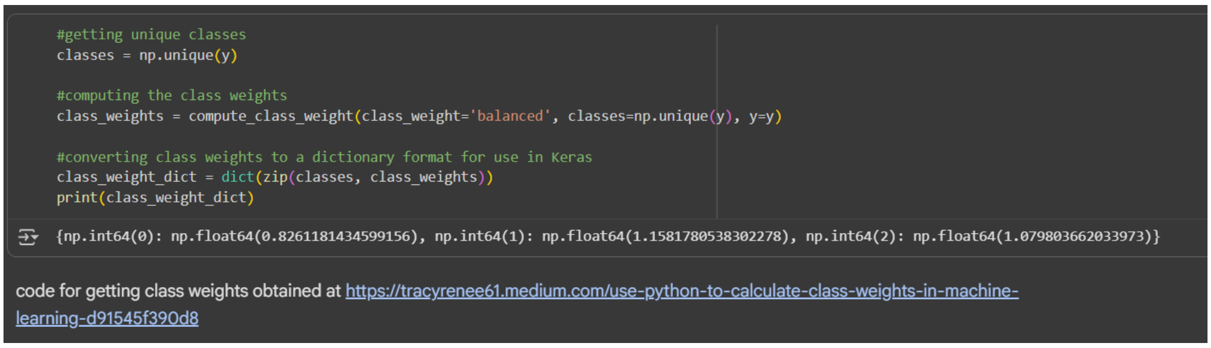
This is where I was just checking to see if the tokenization worked by printing the first index of the processed_text_2 column and its corresponding index in sequences. We can see that tokenization. The tokenization has worked we can even see even see that the word "of" which appears several times has been assigned the token "3". At the beginning of the notebook, I referenced the primary YouTube tutorial I used to develop the RNN < https://www.youtube.com/watch?v=Hrt_gl6r2SI&t=1607s >. In the video one-hot encoding was used instead of tokenization, after training my model and evaluating it I researched on how I could improve model accuracy and that is when I decided to switch out one-hot encoding for tokenization. With code obtained from(Nabeel Valley, 2023). I replaced the one-hot encoding with tokenization which made my model more accurate.



```
sentence_length = 400 #setting the maximum length of the sequences to 400 as it covers most of the text lengths in the dataset
embedded_text = pad_sequences(sequences, padding = 'pre', maxlen = sentence_length) #padding the sequences to ensure they all have the same length by adding zeros at the beginni

X = np.array(embedded_text) #assining the padded sequences to X
y = np.array(df['author']) #assigning the target variable author to y
```

Figure 22. *Text embedding and variable assigning*

From the statistics table in Figure 15, we saw that 95% of the text in the text column had a length of 58 words with the largest sentence being 851 words. Therefore, I decided to set the sentence length to 400 as this would be able to encompass the full length of almost every sentence in the text column. After that would be the tokenized text, which is sequences to 400 tokens, so most of the text would have the same length. In order to guarantee consistent input dimensions, maintain positional information, maximize computational resources, and avoid information loss during training, padding sequences are required to be used(eitca, 2023). Then I converted the embedded text into a numpy array and assigned it to X the input and did this same for author and assigned it to y. Then I converted the embedded text into a numpy array and assigned it to X the input and did the same for author and assigned it to y the output. This was done as numpy arrays provide a consistent and standardized data format that scikit-learn classifiers can easily handle they are also very efficient for numerical computation(etica, 2023b).



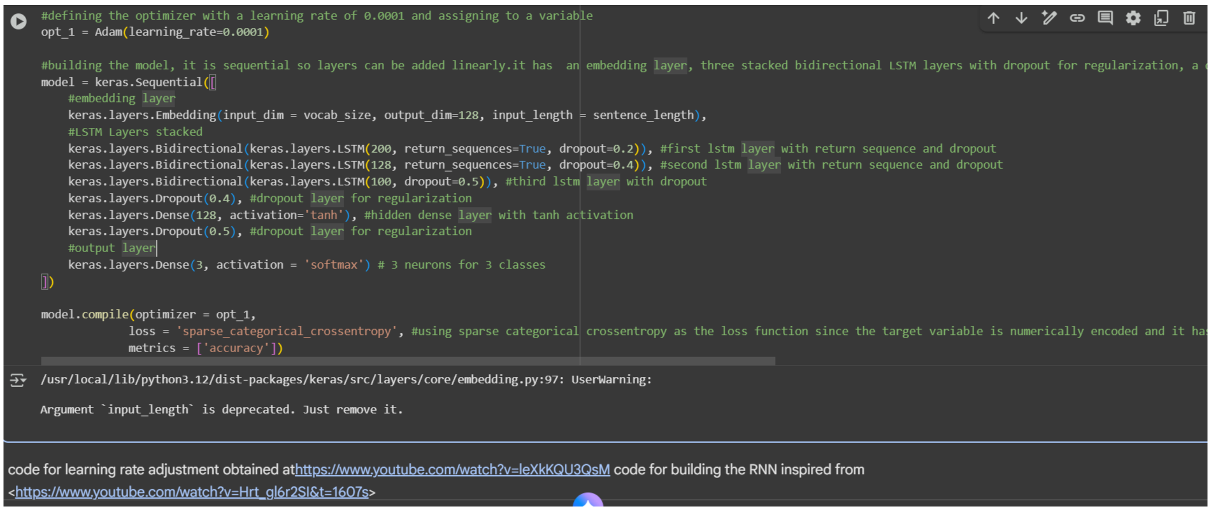
```
#getting unique classes
classes = np.unique(y)

#computing the class weights
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y), y=y)

#converting class weights to a dictionary format for use in Keras
class_weight_dict = dict(zip(classes, class_weights))
print(class_weight_dict)
```

```
{np.int64(0): np.float64(0.8261181434599156), np.int64(1): np.float64(1.1581780538302278), np.int64(2): np.float64(1.079803662033973)}
```

code for getting class weights obtained at https://tracyrenee61.medium.com/use-python-to-calculate-class-weights-in-machine-learning-d91545f390d8

Figure 23. *Computing class weights.*

Once again, I displayed the class distribution, then I moved on to computing class weights. As detailed in my analysis plan, I am going to address this imbalance using class weights. Class weights assign higher weights to minority classes and lower weights to majority classes (Abhinav, 2023). From the output we can see that class 1 which has the lowest number of entries has the highest weight at 1.1582 and class 0 with the highest number of entries has a weight of 0.8261.



```
#defining the optimizer with a learning rate of 0.0001 and assigning to a variable
opt_1 = Adam(learning_rate=0.0001)

#building the model, it is sequential so layers can be added linearly.it has  an embedding layer, three stacked bidirectional LSTM layers with dropout for regularization, a
model = keras.Sequential([
    #embedding layer
    keras.layers.Embedding(input_dim = vocab_size, output_dim=128, input_length = sentence_length),
    #LSTM Layers stacked
    keras.layers.Bidirectional(keras.layers.LSTM(200, return_sequences=True, dropout=0.2)), #first lstm layer with return sequence and dropout
    keras.layers.Bidirectional(keras.layers.LSTM(128, return_sequences=True, dropout=0.4)), #second lstm layer with return sequence and dropout
    keras.layers.Bidirectional(keras.layers.LSTM(100, dropout=0.5)), #third lstm layer with dropout
    keras.layers.Dropout(0.4), #dropout layer for regularization
    keras.layers.Dense(128, activation='tanh'), #hidden dense layer with tanh activation
    keras.layers.Dropout(0.5), #dropout layer for regularization
    #output layer
    keras.layers.Dense(3, activation = 'softmax') # 3 neurons for 3 classes
])

model.compile(optimizer = opt_1,
              loss = 'sparse_categorical_crossentropy', #using sparse categorical crossentropy as the loss function since the target variable is numerically encoded and it has
              metrics = ['accuracy'])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning:

Argument `input_length` is deprecated. Just remove it.
```

code for learning rate adjustment obtained athttps://www.youtube.com/watch?v=leXkKQU3QsM code for building the RNN inspired from <https://www.youtube.com/watch?v=Hrt_gl6r2SI&t=1607s>

Figure 24. *Building the RNN*

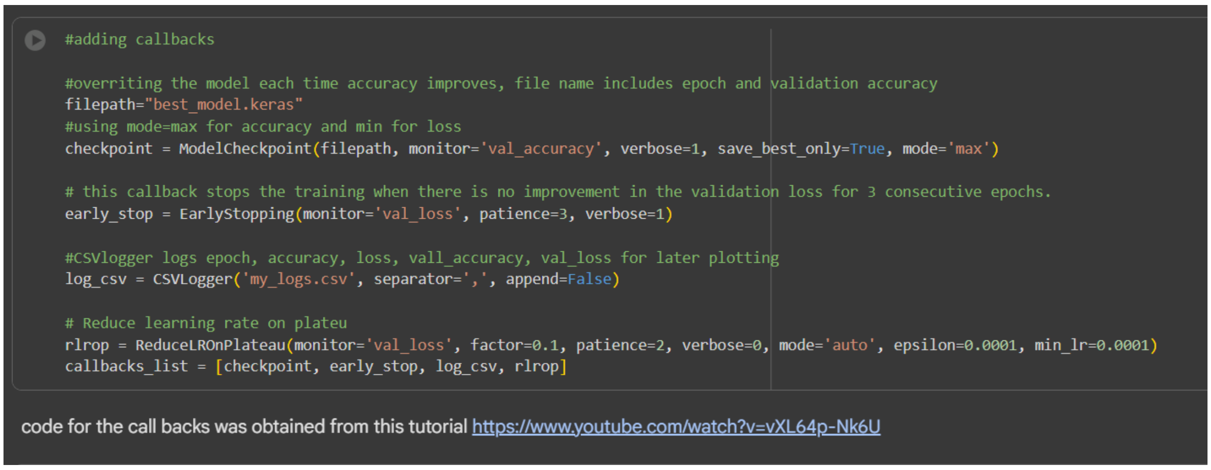
This is where I began the development of my RNN. I assigned the learning rate of my Adam optimizer so I can adjust it. I did this as I saw in this video < https://www.youtube.com/watch?v=leXkKQU3QsM > that adjusting the learning rate of the optimizer is a form of hyperparameter tuning that can improve model performance. I experimented with various values and found that the learning rate of 0.0001 improved my model's accuracy. The amount that a machine learning model modifies its parameters at each step of its optimization algorithm is determined by the hyperparameter learning rate(Belcic & Stryker, 2015). In the previously mentioned YouTube video standard LSTM layers were used but as I stated in my analysis plan I planned to use bidirectional LSTM for their higher accuracy and better contextual understanding. In the first two LSTM layers I set the return_sequences to true. When return_sequences=True is set, an LSTM returns the entire output sequence rather than just the last one, enabling subsequent layers to utilize the entire information history to learn complex patterns(etica, 2023a). This is useful in my case as the authors writing styles does fall under complex patterns, which my machine is attempting to learn. Setting the return_sequence to true is quite beneficial when stacking multiple LSTM layers in Natural Language processing with tensorflow(etica, 2023a).

Each layer in the LSTM has dropout. During each training iteration, dropout randomly disables (or "drops") a portion of neurons. The network performs better on unseen data as a result of learning more generalized features and avoiding being overly reliant on any one node (Kashyap, 2024). I implemented dropout after fitting and evaluating the model then noticing it was overfitting. My last training epoch would have an accuracy of 96% while the model accuracy would be around 75%, I would then implement dropout in each layer to combat the overfitting. Initially, I began with setting a dropout of 0.5 for each layer then after further experimentation, I would find the dropout figures that provided me with the best model accuracy.

The activation function for my hidden dense layer was initially relu as that was the one used in the tutorial that I followed, then I tried using tanh instead and it improved my model's accuracy. When deciding whether or not to activate a neuron, an activation function is used. Using more straightforward mathematical processes, it will determine whether or not the neuron's input to the network is significant throughout the prediction process(Baheti, 2021). Essentially an activation function decides whether a neuron will be activated based on the output of a previous neuron.

The output layer is a dense neural network with 3 neurons as there are 3 classes being predicted and it uses the activation function softmax. I opted to use softmax instead of sigmoid as softmax is used for multi-class classification and sigmoid is used for binary classification (MYSCALE, 2024).

I used sparse categorical cross-entropy as the loss as I have more than 2 classes and the target classes are integers. For multi class classification tasks, sparse categorical cross-entropy is typically used. As well as when target labels are given as numbers rather than vectors that have been one-hot encoded (Chand, 2023). I used accuracy as the metric. The model's performance across all classes is described by the metric accuracy. It works well when every class is equally important. The ratio of accurate predictions to total predictions is how it is computed (Das & Skelton, 2025).



```
#adding callbacks

#overriting the model each time accuracy improves, file name includes epoch and validation accuracy
filepath="best_model.keras"
#using mode=max for accuracy and min for loss
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

# this callback stops the training when there is no improvement in the validation loss for 3 consecutive epochs.
early_stop = EarlyStopping(monitor='val_loss', patience=3, verbose=1)

#CSVlogger logs epoch, accuracy, loss, vall_accuracy, val_loss for later plotting
log_csv = CSVLogger('my_logs.csv', separator=',', append=False)

# Reduce learning rate on plateu
rlrop = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2, verbose=0, mode='auto', epsilon=0.0001, min_lr=0.0001)
callbacks_list = [checkpoint, early_stop, log_csv, rlrop]
```

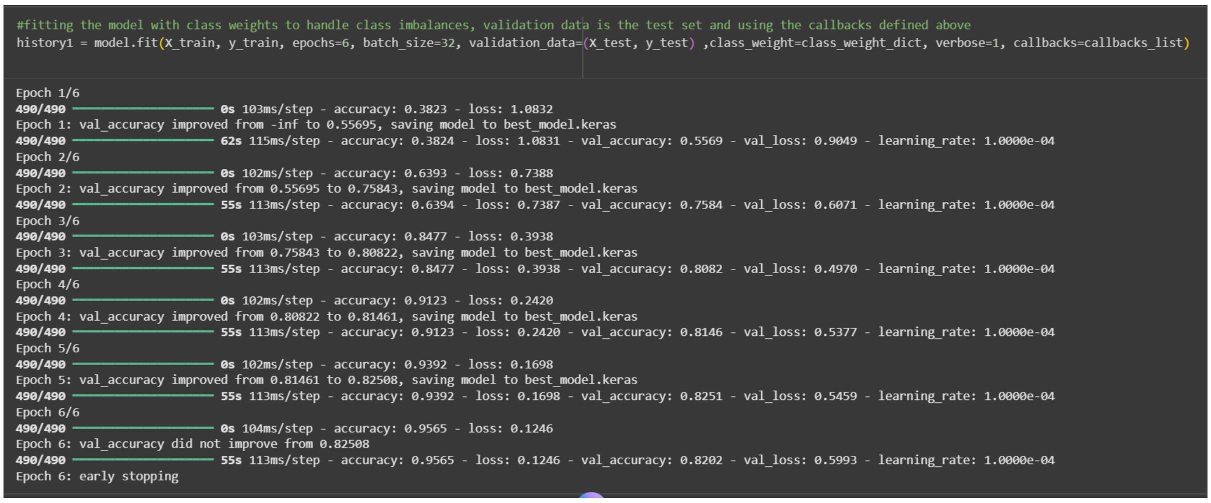
code for the call backs was obtained from this tutorial https://www.youtube.com/watch?v=vXL64p-Nk6U

Figure 25. *Callbacks*

To improve my model's accuracy I implemented callbacks. By using the ModelCheckpoint callback, a model or weights are saved (in a checkpoint file) at a predetermined interval. This allows the model or weights to be loaded later to continue training from the saved state(Keras, 2025b). In my case I am only saving the best model with the save_best_only=True function. And I set it to monitor my val_accuracy which means it saved the model based on which one has the highest accuracy on the validation set.

Then I implemented EarlyStopping. During training, earlystopping monitors the model's performance on a validation set and stops training when performance starts to deteriorate, which is a sign that the model is starting to overfit the training set (Cyborg, 2024).

The last callback I implemented would be ReduceLROnPlateau. This callback reduces the learning rate by the specified factor if no improvement is seen in the 'patience' number of epochs(Keras, 2025a).

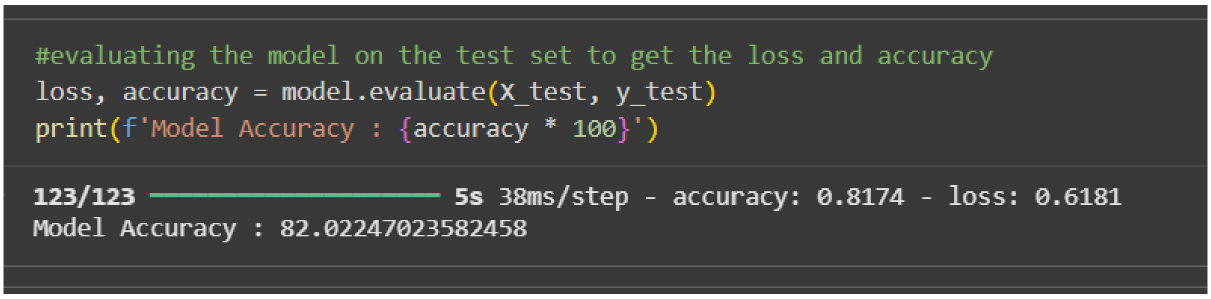


```
#fitting the model with class weights to handle class imbalances, validation data is the test set and using the callbacks defined above
history1 = model.fit(X_train, y_train, epochs=6, batch_size=32, validation_data=(X_test, y_test) ,class_weight=class_weight_dict, verbose=1, callbacks=callbacks_list)
```

```
Epoch 1/6
490/490 ━━━━━━━━━━━━━━━━━━━━ 0s 103ms/step - accuracy: 0.3823 - loss: 1.0832
Epoch 1: val_accuracy improved from -inf to 0.55695, saving model to best_model.keras
490/490 ━━━━━━━━━━━━━━━━━━━━ 62s 115ms/step - accuracy: 0.3824 - loss: 1.0831 - val_accuracy: 0.5569 - val_loss: 0.9049 - learning_rate: 1.0000e-04
Epoch 2/6
490/490 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step - accuracy: 0.6393 - loss: 0.7388
Epoch 2: val_accuracy improved from 0.55695 to 0.75843, saving model to best_model.keras
490/490 ━━━━━━━━━━━━━━━━━━━━ 55s 113ms/step - accuracy: 0.6394 - loss: 0.7387 - val_accuracy: 0.7584 - val_loss: 0.6071 - learning_rate: 1.0000e-04
Epoch 3/6
490/490 ━━━━━━━━━━━━━━━━━━━━ 0s 103ms/step - accuracy: 0.8477 - loss: 0.3938
Epoch 3: val_accuracy improved from 0.75843 to 0.80822, saving model to best_model.keras
490/490 ━━━━━━━━━━━━━━━━━━━━ 55s 113ms/step - accuracy: 0.8477 - loss: 0.3938 - val_accuracy: 0.8082 - val_loss: 0.4970 - learning_rate: 1.0000e-04
Epoch 4/6
490/490 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step - accuracy: 0.9123 - loss: 0.2420
Epoch 4: val_accuracy improved from 0.80822 to 0.81461, saving model to best_model.keras
490/490 ━━━━━━━━━━━━━━━━━━━━ 55s 113ms/step - accuracy: 0.9123 - loss: 0.2420 - val_accuracy: 0.8146 - val_loss: 0.5377 - learning_rate: 1.0000e-04
Epoch 5/6
490/490 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step - accuracy: 0.9392 - loss: 0.1698
Epoch 5: val_accuracy improved from 0.81461 to 0.82508, saving model to best_model.keras
490/490 ━━━━━━━━━━━━━━━━━━━━ 55s 113ms/step - accuracy: 0.9392 - loss: 0.1698 - val_accuracy: 0.8251 - val_loss: 0.5459 - learning_rate: 1.0000e-04
Epoch 6/6
490/490 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step - accuracy: 0.9565 - loss: 0.1246
Epoch 6: val_accuracy did not improve from 0.82508
490/490 ━━━━━━━━━━━━━━━━━━━━ 55s 113ms/step - accuracy: 0.9565 - loss: 0.1246 - val_accuracy: 0.8202 - val_loss: 0.5993 - learning_rate: 1.0000e-04
Epoch 6: early stopping
```

Figure 26. *Fitting the model*

Fitting the model with 6 epochs as this is what provided me the best accuracy. Batch size is 32, a validation set is included, class_weight is fitted to the model for class imbalance and the callbacks are also fit to the model. From the training epochs we can see the accuracy metric and loss being measured for the training set and validation set as well as the learning rate. Whenever a model improves in validation accuracy, the ModelCheckpoint callback saves the best model. We can also see early stopping being applied when the model's validation accuracy does not improve from 0.82508.

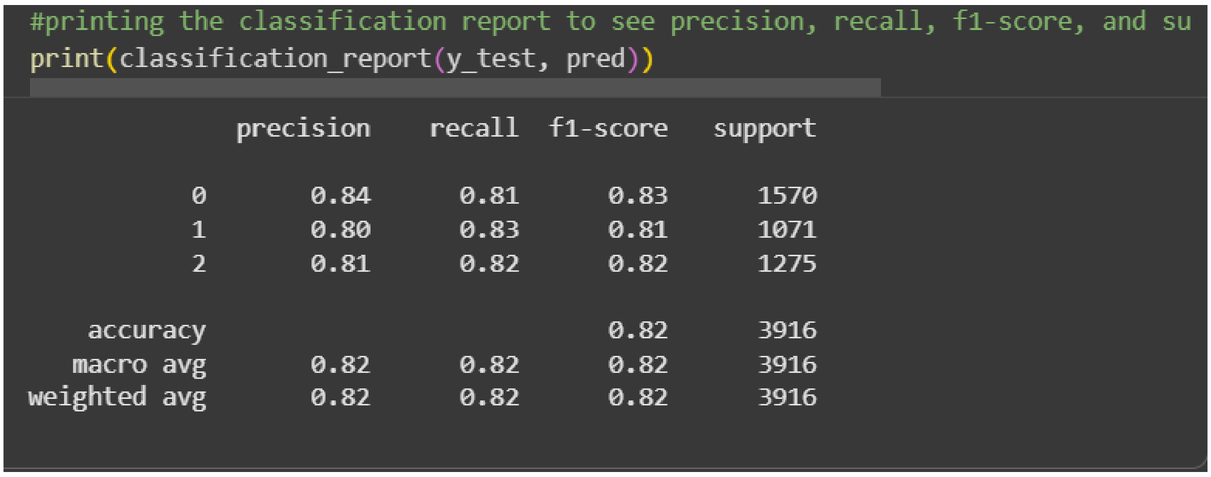


```
#evaluating the model on the test set to get the loss and accuracy
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Model Accuracy : {accuracy * 100}')
```

```
123/123 ━━━━━━━━━━━━━━━━━━━━ 5s 38ms/step - accuracy: 0.8174 - loss: 0.6181
Model Accuracy : 82.02247023582458
```

Figure 27. *Evaluating the model on the test set*

After evaluating the model, we can see an accuracy of 82.02 and a loss of 0.6181. This is a decent improvement of my initial accuracy of 73, where I did not have dropouts and callbacks. I also had not experimented with the number of neurons in my LSTM layers. I initially had a much higher number of neurons in each layer which was causing my model to overfit.

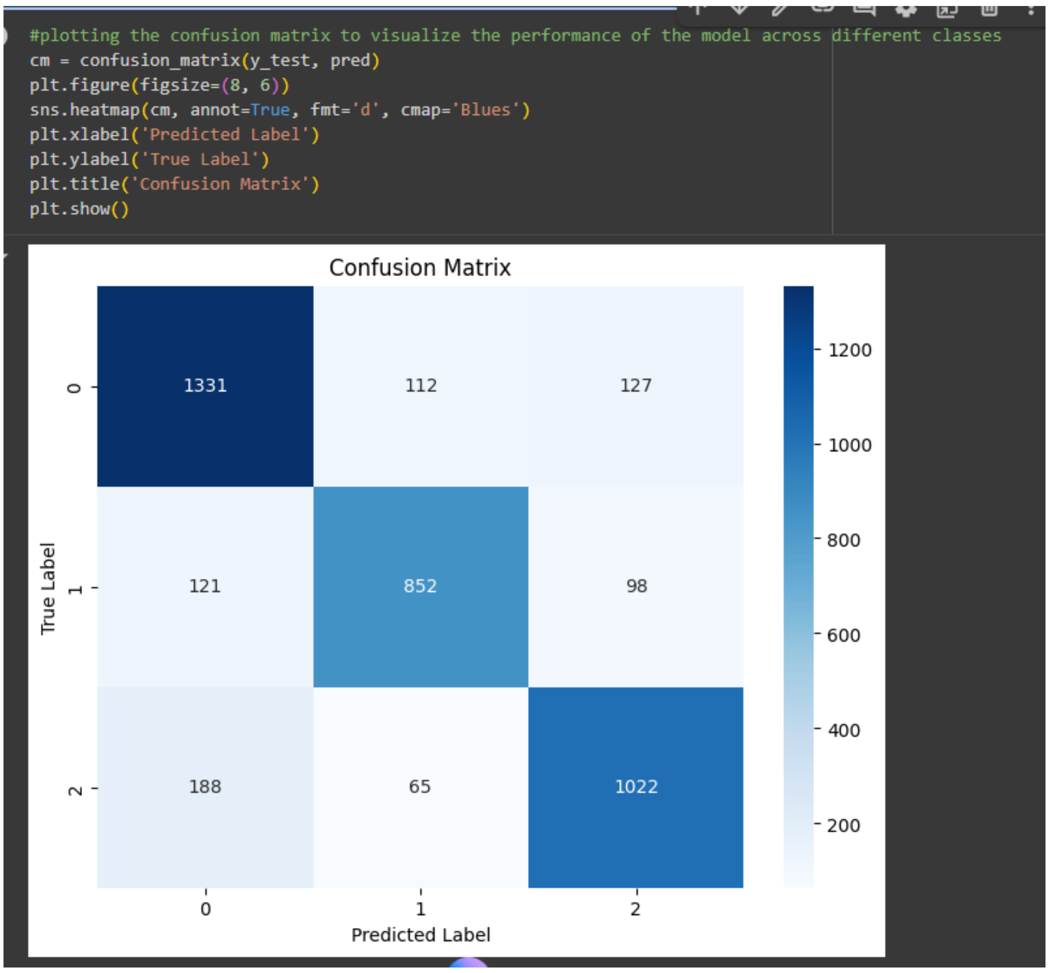


```
#printing the classification report to see precision, recall, f1-score, and su
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.81      0.83      1570
           1       0.80      0.83      0.81      1071
           2       0.81      0.82      0.82      1275

    accuracy                           0.82      3916
   macro avg       0.82      0.82      0.82      3916
weighted avg       0.82      0.82      0.82      3916
```

Figure 28. Classification Report

From this classification report we can see that the model's accuracy when predicting all 3 classes fell in the range of 0.81 – 0.84 with class 0 having the highest accuracy and class 1 having the lowest accuracy. This is as a result of class 0 having 2200 more row entries than class 1. However we can see that with class weights being applied the imbalance didn't have greatly significant effects.



```
#plotting the confusion matrix to visualize the performance of the model across different classes
cm = confusion_matrix(y_test, pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

True Label \ Predicted Label	0	1	2
0	1331	112	127
1	121	852	98
2	188	65	1022

Figure 29. *Confusion Matrix*

This confusion matrix shows the number of predictions the model made that were correct and incorrect. We can see that the model predicted the author was 2 when it was actually 0 154 times. This was where the model made the most prediction errors. The least amount of prediction errors was the model predicting 1 when the author was actually 2 being 82 times.

Bibliography

Abhinav, R. (2023, August 5). *Improving Class Imbalance with Class Weights in Machine Learning*. Medium. https://medium.com/@ravi.abhinav4/improving-class-imbalance-with-class-weights-in-machine-learning-af072fdd4aa4

Ashraf, A. (2024, January 30). *Tokenization in NLP : All you need to know*. Medium. https://medium.com/@abdallahashraf90x/tokenization-in-nlp-all-you-need-to-know-45c00cfa2df7

Baheti, P. (2021, May 21). *Activation Functions in Neural Networks [12 Types & Use Cases]*. V7labs. https://www.v7labs.com/blog/neural-networks-activation-functions

Balakrishnan, N. (2024, June 3). *The Crucial Role of Tokenization in Enhancing NLP Model Performance*. Medium. https://medium.com/@nivedha0702/a-comprehensive-analysis-of-tokenization-in-llm-be82a47d2cad

Belcic, I., & Stryker. (2015). *What is learning rate in machine learning?* . IBM. https://www.ibm.com/think/topics/learning-rate

Chand, S. (2023, July 20). *Choosing between Cross Entropy and Sparse Cross Entropy — The Only Guide you Need!* Medium. https://medium.com/@shireenchand/choosing-between-cross-entropy-and-sparse-cross-entropy-the-only-guide-you-need-abea92c84662

Cyborg. (2024, April 5). *What is Early Stopping in Deep Learning?* Medium. https://cyborgcodes.medium.com/what-is-early-stopping-in-deep-learning-eeb1e710a3cf

Das, A. F., & Skelton, J. (2025, May 7). *How to Evaluate Deep Learning Models: Key Metrics Explained*. Digitalocean. https://www.digitalocean.com/community/tutorials/deep-learning-metrics-precision-recall-accuracy

eitca. (2023, August 5). *Why is it necessary to pad sequences in natural language processing models?* Eitca. https://eitca.org/artificial-intelligence/eitc-ai-tff-tensorflow-fundamentals/natural-language-processing-with-tensorflow/training-a-model-to-recognize-sentiment-in-text/examination-review-training-a-model-to-recognize-sentiment-in-text/why-is-it-necessary-to-pad-sequences-in-natural-language-processing-models/

etica. (2023a, August 5). *What is the significance of setting the "return_sequences" parameter to true when stacking multiple LSTM layers?* Etica. https://eitca.org/artificial-intelligence/eitc-ai-tff-tensorflow-fundamentals/natural-language-processing-with-tensorflow/long-short-term-memory-for-nlp/examination-review-long-short-term-memory-for-nlp/what-is-the-significance-of-setting-the-return_sequences-parameter-to-true-when-stacking-multiple-lstm-layers/

etica. (2023b, August 7). *What is the advantage of converting data to a numpy array and using the reshape function when working with scikit-learn classifiers?* Etica. https://eitca.org/artificial-intelligence/eitc-ai-mlp-machine-learning-with-python/programming-machine-learning/k-nearest-neighbors-application/examination-review-k-nearest-neighbors-application/what-is-the-advantage-of-converting-data-to-a-numpy-array-and-using-the-reshape-function-when-working-with-scikit-learn-classifiers/

GeeksforGeeks. (2025, July 23). *Difference Between a Bidirectional LSTM and an LSTM*. GeeksforGeeks. https://www.geeksforgeeks.org/deep-learning/difference-between-a-bidirectional-lstm-and-an-lstm/

Kashyap, P. (2024, October 30). *Understanding Dropout in Deep Learning: A Guide to Reducing Overfitting*. Medium. https://medium.com/@piyushkashyap045/understanding-dropout-in-deep-learning-a-guide-to-reducing-overfitting-26cbb68d5575

Keras. (2025a, May). *ReduceLROnPlateau*. Keras. https://keras.io/api/callbacks/reduce_lr_on_plateau/

Keras. (2025b, June). *ModelCheckpoint*. Keras. https://keras.io/api/callbacks/model_checkpoint/

Maklin, C. (2022, May 14). *Synthetic Minority Over-sampling TEchnique (SMOTE)*. Medium. https://medium.com/@corymaklin/synthetic-minority-over-sampling-technique-smote-7d419696b88c

Mousavi, E. (2025, January 15). *NLP Series: Day 3 — Lowercasing and Removing Punctuations*. Medium. https://medium.com/@ebimsv/nlp-series-day-3-lowercasing-and-removing-punctuations-0703617cf7b6

MYSCALE. (2024, April 4). *Softmax vs. Sigmoid Functions: Understanding Neural Networks Variation*. MYSCALE. https://www.myscale.com/blog/neural-networks-softmax-sigmoid/

Nabeel Valley. (2023, September 3). *NLP WITH TENSORFLOW*. Nabeel Valley.

Srivatsavaya, P. (2023, August 31). *Tokenization in NLP : All you need to know*. Medium. https://medium.com/@prudhviraju.srivatsavaya/advantages-and-disadvantages-of-pad-sequences-keras-layer-4ea08a7eee5c