Institute of Engineering and Technology

JK Lakshmipat University, Jaipur

CS1138: Machine Learning

Faculty Guide:

Mr. Arpan Gupta

Submitted By:

Aastha Gupta (2022btech002)
Jatin Choudhary (2022btech044)
Riya Singh (2022btech088)
Siddhi Nyati (2022btech101)

INDEX

ABSTRACT

This research utilizes logistic regression as well as LSTM, SVM, and Random Forest to derive useful information from social media information found on websites such as Instagram, Facebook, and Twitter. Utilizing machine learning algorithms, we explore sentiment analysis, trend detection, user profiling, and content classification.

Logistic regression is used for analysing sentiment, an important element in grasping public opinion and brand perception. By applying natural language processing (NLP) methods, we classify social media posts based on whether they convey positive, negative, or neutral feelings. This enables businesses to track shifts in customer happiness and opinions as time progresses, supporting smart decision-making and proactive interaction with users.

By adding logistic regression to our approach, we improve the depth of our investigation, giving companies more resources to uncover valuable information from social media data. This comprehensive strategy allows businesses to adjust to customer preferences, strategize effectively, and stay competitive in the digital realm of Facebook, Instagram, and Twitter.

INTRODUCTION

This significantly differentiates the current generation because the digital age, and social media, have turned these platforms into vibrant centres where people interact with all kinds of content, post experiences, and share their opinions. As a result, Facebook, Instagram, and Twitter have progressively developed as critical demystifies in the real-time domain into how the public feels, what it believes in and its habits. Sentiment analysis thus becomes a substantive approach for extracting knowledge from extensive internet-based conversation while detecting feelings underpinning several people's expressions.

Sentiment analysis is a technology under the umbrella of natural language processing that attempts to computationally analyse people's opinions, attitudes, emotions, and more as expressed in written text. Through understanding the emotional "tone," or polarity, that underlies any single sentiment expressed, sentiment analysis provides a critical understanding of public opinion, trends, and sentiment towards a brand.

While sentiment analysis does have a wide variety of applications, this technology is most critical due to its key role in business analytics. According to Ober Weis, the business sector utilises sentiment analysis to help identify customer sentiments, measure brand sentiment, and monitor confidence and exaggeration. Moreover, they also use sentiment analysis to track customer feedback, reviews, and social media mentions, which can meet their satisfaction and weak points in products or services.

Secondly, sentiment analysis is essential for market research and analysing the competitiveness of other businesses. It allows businesses to understand what consumers prefer and how they feel about products and services while showcasing market trends.

Furthermore, analysing sentiment on social media permits the assessment of public opinion about political figures and candidates, social issues and movements, and trends on the rise. As a result, quick strategic decision-making and PR are possible. Additionally, sentiment analysis is necessary for social media monitoring, where organizations can track brand mentions and sentiment trends. Therefore, they allow for real-time interaction with the audience when addressing negative feelings and amplifying positive ones.

Overall, sentiment analysis is a versatile tool for guiding the intricacies of a digital conversation, providing substantial information on public perception, customer choices, and market trends.

Implementable across various sectors, from marketing and consumer relationships to socio-political research and clinical medicine, sentiment analysis is a vital component of the contemporary data-oriented reality. Throughout this study, we are set to investigate sentiment analysis further, distinguish between its intricate patterns, and detail its prospective utility for administrative/course of business processes within a digital frame.

PROBLEM STATEMENT

In this study, we have strived to investigate and compare the effectiveness of different methodologies for sentiment analysis in social media platforms, namely Instagram, Facebook, and Twitter. Since our main goal was to compare the performance, accuracy, and efficiency of traditional natural language processing (NLP) techniques and a “novel” approach to Long Short-Term Memory (LSTM), our analysis is not limited by the considered methods, as we also took a comparative approach of Support Vector Machine (SVM) and Random Forest.

Therefore, we believe that the insights obtained by the comparative methodology of these approaches will allow stakeholders to make better decisions and develop a more sophisticated understanding of the public’s sentiment on social media.

LITERATURE REVIEW

Review Paper 1:

Title: Sentiment Analysis in Social Media Texts using SVM.

Author Information:

- Alexandra Balahur
- European Commission Joint Research Centre Vie E. Fermi 2749 21027 Ispra (VA), Italy
- alexandra.balahur@jrc.ec.europa.eu

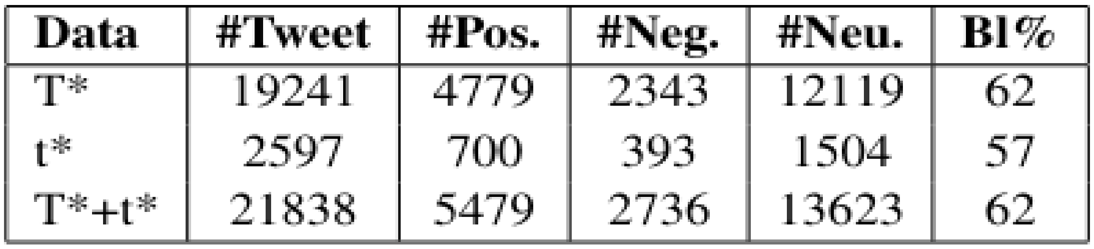
Introduction:

In the current digital era, social media sites such as Twitter have emerged as influential platforms for expressing public opinions. Each tweet reflects someone's personal reflections, emotions, and life events, generating a large collection of information for those who can decipher its significance. Sentiment analysis, a component of Natural Language Processing (NLP), is becoming increasingly important in this effort. It enables us to categorize the sentiment of this text data automatically - either as positive, negative, or neutral - offering valuable insights into public opinion.

Yet, conventional approaches to sentiment analysis frequently face challenges when dealing with the distinctive features of tweets. Contrary to formal writing, tweets are brief, casual, and filled with slang, hashtags, and typos. This paper suggests a new method for sentiment analysis that is tailored to deal with these challenges and efficiently analyse sentiment in tweets. Our approach seeks to give a more precise insight into public opinion on Twitter by taking into consideration the casual and concise nature of communication in this platform.

Dataset Used:

1. SemEval 2013: This dataset is likely from the SemEval-2013 Task 5, which focused on sentiment analysis in social media text. It likely contains tweets with sentiment labels (positive, negative, neutral).
2. Tweets labelled with basic emotions: This dataset is more specific and focuses on tweets labelled with basic emotions, which could include labels like happy, sad, angry, etc., besides positive and negative.
3. Blog sentences labelled with emotions: This dataset is included as a benchmark to compare the performance on tweets with a more formal style of text (blog sentences) also labeled with emotions.



Data	#Tweet	#Pos.	#Neg.	#Neu.	Bl%
T*	19241	4779	2343	12119	62
t*	2597	700	393	1504	57
T*+t*	21838	5479	2736	13623	62

Methodologies:

1. Pre-processing Tweets:

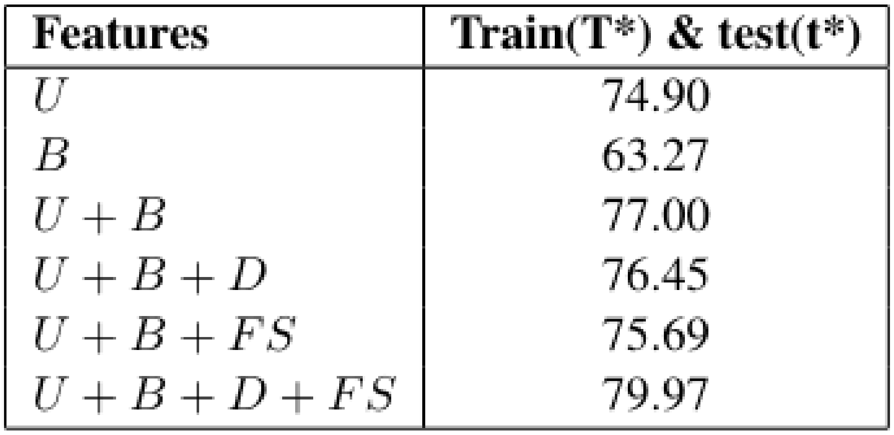
- Normalization: This stage addresses the informality of tweets by normalizing the text. This might involve:
  Replacing repeated punctuation signs (e.g., "!!!!" becomes "!")
  Expanding common abbreviations (e.g., "LOL" to "laugh out loud")
  Correcting or standardizing misspelled words
- Lexicon Replacement:
  Sentiment lexicons are dictionaries that map words to their sentiment polarity (positive, negative, or neutral). This stage replaces sentiment-bearing words in the tweet with a general label for their polarity (e.g., "happy" and "love" replaced with "positive").
- Handling Modifiers: Words like "not" or intensifiers can alter sentiment. This stage might involve.
  Replacing negation words ("not," "never") with a negation label
  Identifying and handling emphasis markers (e.g., all caps, exclamation points)

2. Sentiment Classification

- Feature Engineering: After pre-processing, relevant features are extracted from the normalized text to train a sentiment classifier. This could involve:
- Unigrams: Individual words in the tweet after pre-processing.
- Bigrams: Two-word sequences that can capture changes in sentiment due to negation or emphasis (e.g., "not good," "very happy").
- Machine Learning Model: The paper proposes using Support Vector Machines (SVM) as the sentiment classifier. SVMs are powerful tools for classification tasks and can effectively learn the relationship between the extracted features and sentiment labels.

Result and Discussion:

The system is evaluated on three datasets: SemEval 2013, tweets labelled with basic emotions, and blog sentences labelled with emotions. The results show that using unigrams and bigrams along with replacing sentiment-bearing words and modifiers with general labels achieves the best performance. This approach outperforms methods that don't use these techniques.



Features	Train(T*) & test(t*)
U	74.90
B	63.27
$U + B$	77.00
$U + B + D$	76.45
$U + B + FS$	75.69
$U + B + D + FS$	79.97

Table 3: Results in terms of accuracy for the different combination of features for the sentiment classification of tweets, using T* as training and t* as test set.

- The accuracy using SVM is found to be approx. 80% on SemEval data.

Comparison with my work: The research paper focused solely on sentiment analysis using Support Vector Machines (SVM) and achieved an accuracy of around 80%. In contrast, our study employed four models: Long Short-Term Memory (LSTM), SVM, Random Forest, and Natural Language Processing (NLP). However, our accuracies varied: LSTM at approximately 42%, SVM around 40%, Random Forest at about 55%, and NLP with the highest accuracy at 83%. While the research paper outperformed our study in overall accuracy, our approach offered a broader perspective by exploring multiple models. This allowed for a comparative analysis of different methodologies, revealing insights into their strengths and weaknesses in sentiment analysis. Although the SVM model in the research paper excelled, our study's inclusion of diverse techniques, notably NLP, showcased promising results, suggesting the potential for advanced language processing techniques to enhance sentiment analysis tasks.

Review Paper 2:

Title: Sentiment Analysis with SVM.

Author Information:

Munir Ahmad

Department of Computer Science Virtual University of Pakistan

Shabib Aftab

Department of Computer Science Virtual University of Pakistan

Iftikhar Ali

Department of Computer Science Virtual University of Pakistan

Introduction:

The volume of textual data from social media sites like Facebook and Twitter has increased the demand for effective text mining techniques, especially sentiment analysis. By knowing how customers feel about their goods or services, businesses can use sentiment analysis to improve their position in the market. The three primary methods for sentiment analysis are hybrid, machine learning, and lexicon-based methodologies. Machine learning techniques need training data for text sentiment categorization, whereas lexicon-based approaches use predefined dictionaries. Hybrid techniques incorporate both strategies. SentiStrength, SentiWordNet, Maximum Entropy, SVM, and hybrids like pSenti and SAIL are examples of common tools and methodologies. Precision, Recall, and F-Measure are used in this study's sentiment analysis utilizing Support Vector Machine (SVM) on pre-classified twitter datasets.

MATERIALS AND METHODS:

The purpose of this paper is to evaluate the effectiveness of Support Vector Machines (SVM) for textual data polarity detection (positive, negative, and neutral). Two Twitter datasets with pre-labels are taken into account for this analysis. The prelabeled tweets were selected as test data so that the effectiveness and accuracy of SVM could be examined. Weka will compute the difference between each tweet generated by this algorithm and the pre-labeled class based on the comparison of the output polarity. Precision, recall, and f measure will be used to gauge performance [1, 2, 3, 8,].

DATA SET:

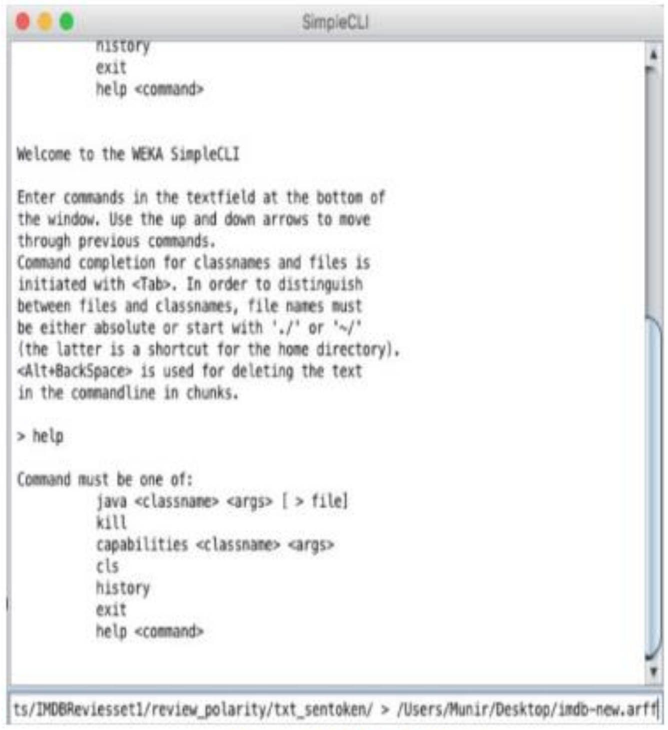
Two pre-labeled datasets of tweets are used in this research. First dataset contains the tweets about self-driving cars [5]. It contains 110 very negative, 685 slightly negative, 4245 neutral, 1444 slightly positive, 459 very positive and 213 irrelevant tweets

Table 1. Twitter dataset for self-driving cars

Class	Tweets
Very Negative	110
Slightly Negative	685
Neutral	4245
Slightly Positive	1444
Very Positive	459
Irrelevant	213
Total	7156

Table 2. Twitter dataset for Apple products

Class	Tweets
Negative	1218
Neutral	2162
Positive	423
Irrelevant	81
Total	3884



```
SimpleCLI
         history
         exit
         help <command>


Welcome to the WEKA SimpleCLI

Enter commands in the textfield at the bottom of
the window. Use the up and down arrows to move
through previous commands.
Command completion for classnames and files is
initiated with <Tab>. In order to distinguish
between files and classnames, file names must
be either absolute or start with './' or '~/'
(the latter is a shortcut for the home directory).
<Alt+BackSpace> is used for deleting the text
in the commandline in chunks.

> help

Command must be one of:
         java <classname> <args> [ > file]
         kill
         capabilities <classname> <args>
         cls
         history
         exit
         help <command>

ts/IMDBReviesset1/review_polarity/txt_sentoken/ > /Users/Munir/Desktop/imdb-new.arff
```

Fig 1: Simple CLI in Weka

In this stage, the normalized data is subjected to SVM for classification, and the results are displayed. Any supervised machine learning algorithm's performance can be carried out by giving the test data that has already been preclassified and contrasting the output polarities with the preclassified polarities. As input data, we have two pre-label tweet datasets. The precision, recall, and f measure are used to gauge the outcomes.

<u>RESULT:</u>

This section focuses on the results and comparative analysis of SVM in different measures for both datasets. For comparison, three evaluation parameters are used in this study: Precision, Recall and F Measure.

The precision can be calculated using TP and FP rate as shown below:

$$Precision = \frac{TP}{(TP + FP)}$$

TP is used for sentences, which are correctly classified, and FP is for those sentences, which are wrongly classified.

Recall can be calculated as shown below:

$$Recall = \frac{TP}{(TP + FN)}$$

FN is used for non-classified sentences and TP is for correctly classified sentences (as explained above).

F-measure can be computed as bellow:

$$\mathrm{F - measure} = \frac{Precision * Recall * 2}{(Precision + Recall)}$$

Table 3. Class wise Precision, Recall and F-Measure for First Dataset

Class	Precision	Recall	F-Measure
Very Negative	0.224	0.1	0.138
Slightly Negative	0.247	0.184	0.211
Neutral	0.708	0.841	0.769
Slightly Positive	0.428	0.305	0.356
Very Positive	0.278	0.237	0.256
Irrelevant	0.225	0.136	0.17
Average	0.558	0.599	0.572

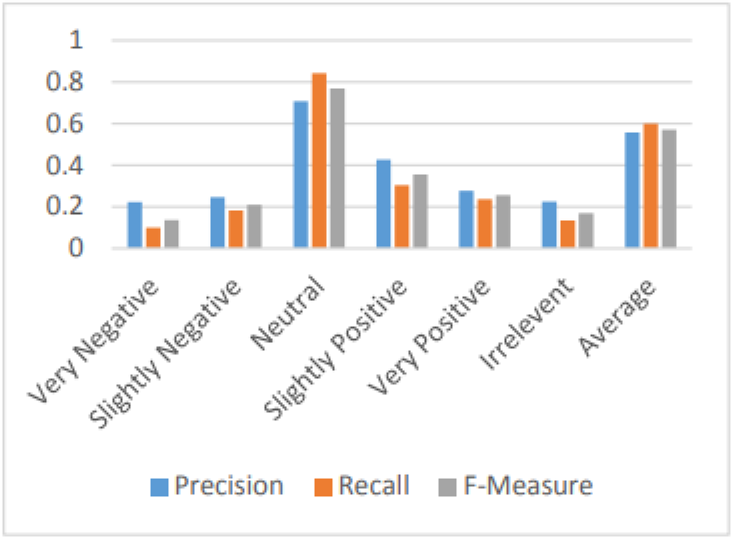


Fig 3: Twitter dataset for self-driving cars

In comparison to other classes, the neutral class had a higher score in Precision, Recall, and F-Measure—70.8%, 84.1%, and 76.9%, respectively.

The second dataset, derived from reference [6], comprises tweets pertaining to 'apple' products. The averages for Precision, Recall, and F-Measure are 70.2%, 71.2%, and 69.9%, respectively, based on the results.

Table 4. Class wise Precision, Recall and F-Measure for Second Dataset

Class	Precision	Recall	F-Measure
Negative	0.732	0.602	0.661
Neutral	0.729	0.859	0.789
Positive	0.548	0.376	0.446
Irrelative	0.318	0.173	0.224
Average	0.702	0.712	0.699

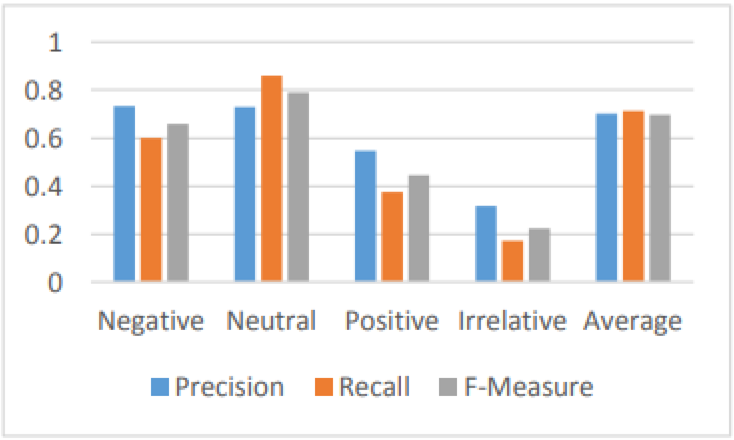


Fig 4: Twitter dataset for apple

Table 5. SVM Accuracy

Datasets	Accuracy %
Self-Driving Cars	59.91%
Apple	71.2%

CONCLUSION:

We have examined the Support Vector Machine's (SVM) sentiment analysis performance in this work. In order to analyze SVM performance, we have utilized two prior

tweets that were categorized; the first dataset included tweets about self-driving cars, while the second dataset focused on Apple products. Weka is a tool used for comparison and performance analysis. Three metrics are used to measure results: f-measure, precision, and recall. The average precision, recall, and f-measure for the first dataset are 55.8%, 59.9%, and 57.2%, respectively, based on the results. The average Precision, Recall, and F-Measure for the second dataset are, respectively, 70.2%, 71.2%, and 69.9%. Complete results are displayed both graphically and tabularly. The outcomes unequivocally demonstrate how SVM performance is dependent on the input dataset. The reliance of performance on

Review Paper 3:

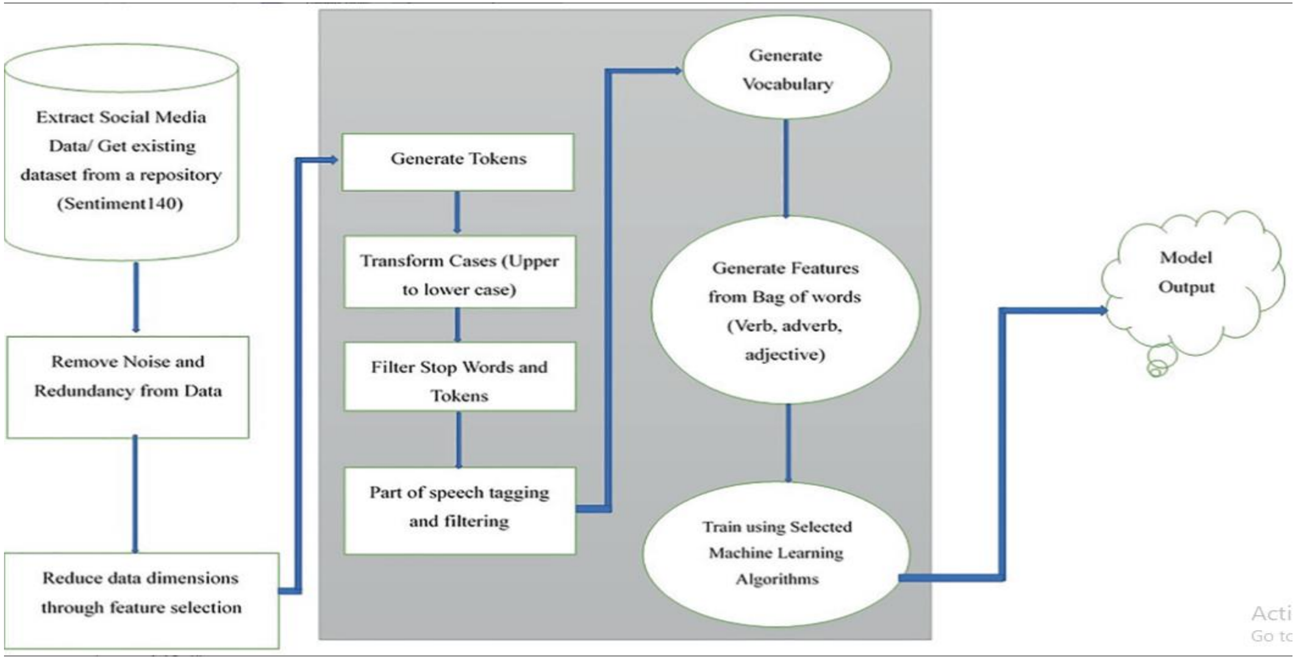
Title: Sentiment analysis on social media tweets using dimensionality reduction and natural language processing.

Author Information:

- Authors: Omuya EO, Okeyo G, Kimwele M
- Year: 22 September 2022
- Title: Sentiment analysis on social media tweets using dimensionality reduction and natural language processing
- Journal Homepage: https://onlinelibrary.wiley.com/doi/full/10.1002/eng2.12579

Introduction:

The article underscores the transformative impact of the internet, particularly through social media, on communication and information sharing. It emphasizes the significance of sentiment analysis in understanding public opinion and its wide-ranging applications across various fields, including business intelligence. Introducing a novel approach, the article proposes a model that amalgamates dimensionality reduction, NLP, and machine learning techniques to enhance sentiment analysis accuracy. By leveraging PCA and IG for feature selection, NLP for data preprocessing and sentiment analysis, and part-of-speech tagging for improved accuracy, the model demonstrates superior performance compared to existing methods. The model's innovation lies in its ability to streamline feature evaluations and select pertinent features for training, thereby boosting accuracy. The article follows a structured approach, exploring related work, detailing the proposed model and algorithm, presenting experimental findings, and concluding with insights for future research.



Extract Social Media Data/ Get existing dataset from a repository (Sentiment140)
Remove Noise and Redundancy from Data
Reduce data dimensions through feature selection
Generate Tokens
Transform Cases (Upper to lower case)
Filter Stop Words and Tokens
Part of speech tagging and filtering
Generate Vocabulary
Generate Features from Bag of words (Verb, adverb, adjective)
Train using Selected Machine Learning Algorithms
Model Output

Model containing:

1.**Data Collection:** Obtain data from sentiment140 repository via APIs. Which contains 1,600,000 tweets extracted using the twitter APIs. The tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment. In the research they sampled 1500 instances from the Sentiment140 dataset for their experiments.

2.**Data Preprocessing:** Clean data by removing noise and irrelevant features.

3.**Feature Selection:** Use PCA and IG to reduce dimensions and select relevant features.

a). **Principal Component Analysis (PCA):** Identify significant components by analyzing feature correlations.

b). **Information Gain (IG):** Determine relevant features based on calculated IG with a set threshold.

6. **Sentiment Analysis:**

- **Tokenization:** Break statements into tokens (symbols, words, phrases).

- **Case Transformation:** Convert tokens to lowercase.

- **Stop Words Filtering:** Remove insignificant words.

- **Part-of-Speech Tagging and Filtering**: Assign grammatical classes to tokens (verb, adjective, adverb).

- **Generating Vocabulary:** Create a bag of verbs, adjectives, and adverbs based on token occurrence rates.

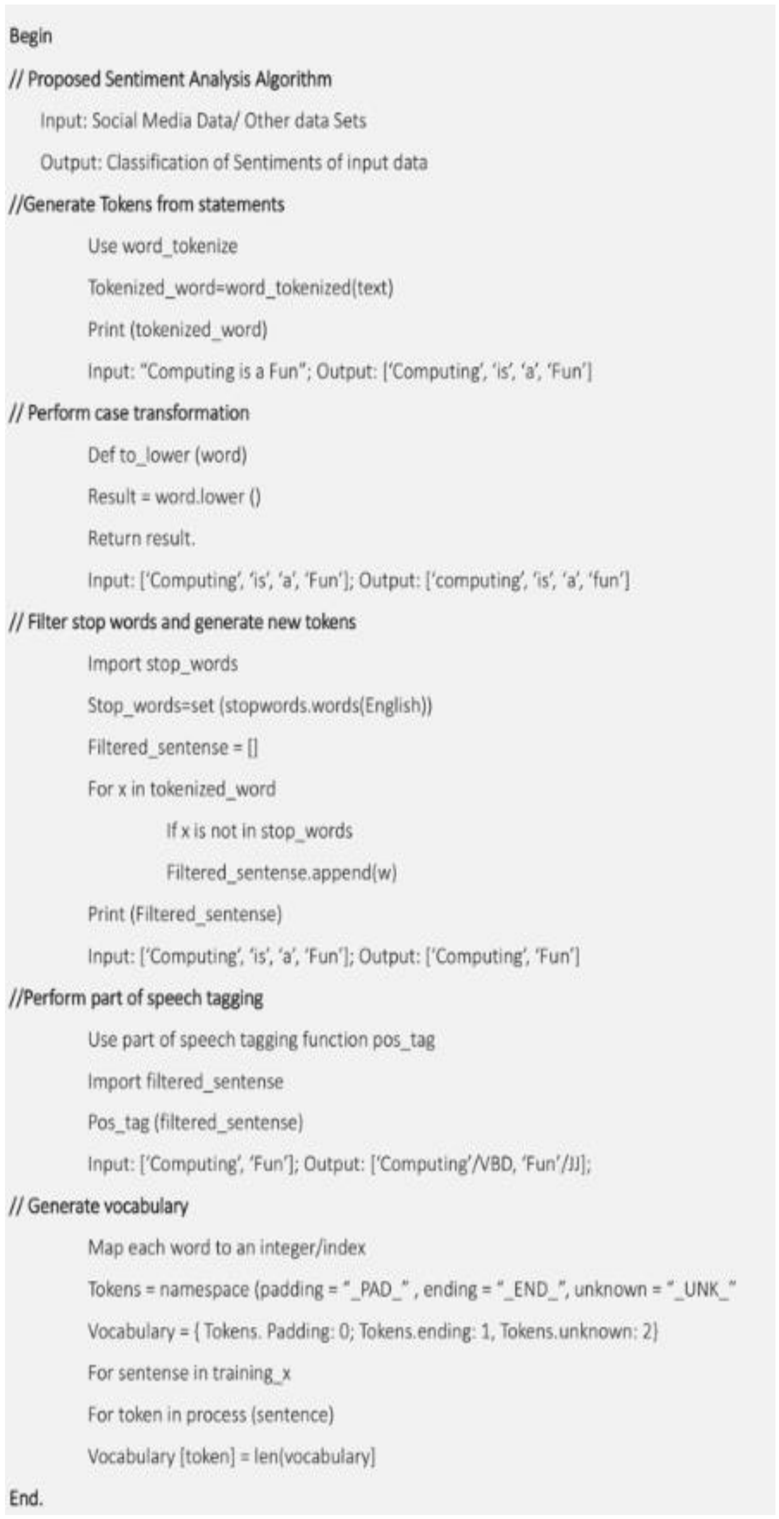
7. **Model Training**: Train sentiment analysis model using NB, SVM, and K-nearest neighbor algorithms on 70% of selected features.

8.**Sentiment Identification:** Utilize trained model to classify tweet polarity in test and new data.

9. **Analysis and Evaluation:** Assess model performance using metrics like accuracy, precision, recall, and F-measure.

THE PROPOSED SENTIMENT ANALYSIS ALGORITHM

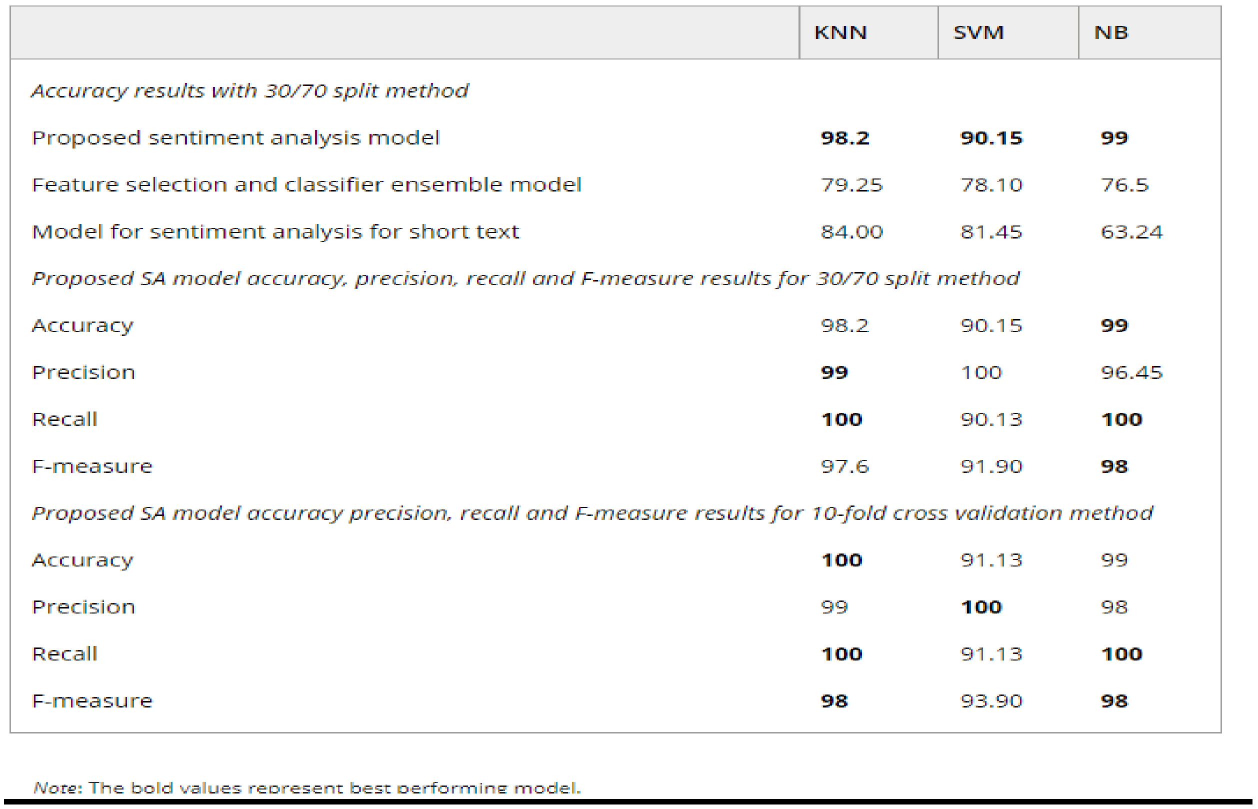
- The proposed algorithm for sentiment analysis starts with data preprocessing, preparing social media or other data for model training and testing. It then performs feature selection using PCA and IG to filter relevant features. These features undergo sentiment analysis, including tokenization and generating a bag of words. Finally, the algorithm trains a sentiment analysis model using NB, SVM, and K-nearest neighbor algorithms to generate a classifier. This classifier is tested and used to analyze new data, with its performance evaluated and compared to other sentiment analysis models.



```
Begin
// Proposed Sentiment Analysis Algorithm
    Input: Social Media Data/ Other data Sets
    Output: Classification of Sentiments of input data
//Generate Tokens from statements
          Use word_tokenize
          Tokenized_word=word_tokenized(text)
          Print (tokenized_word)
          Input: "Computing is a Fun"; Output: ['Computing', 'is', 'a', 'Fun']
// Perform case transformation
          Def to_lower (word)
          Result = word.lower ()
          Return result.
          Input: ['Computing', 'is', 'a', 'Fun']; Output: ['computing', 'is', 'a', 'fun']
// Filter stop words and generate new tokens
          Import stop_words
          Stop_words=set (stopwords.words(English))
          Filtered_sentense = []
          For x in tokenized_word
                    If x is not in stop_words
                    Filtered_sentense.append(w)
          Print (Filtered_sentense)
          Input: ['Computing', 'is', 'a', 'Fun']; Output: ['Computing', 'Fun']
//Perform part of speech tagging
          Use part of speech tagging function pos_tag
          Import filtered_sentense
          Pos_tag (filtered_sentense)
          Input: ['Computing', 'Fun']; Output: ['Computing'/VBD, 'Fun'/JJ];
// Generate vocabulary
          Map each word to an integer/index
          Tokens = namespace (padding = "_PAD_" , ending = "_END_", unknown = "_UNK_"
          Vocabulary = { Tokens. Padding: 0; Tokens.ending: 1, Tokens.unknown: 2}
          For sentense in training_x
          For token in process (sentence)
          Vocabulary [token] = len(vocabulary]
End.
```

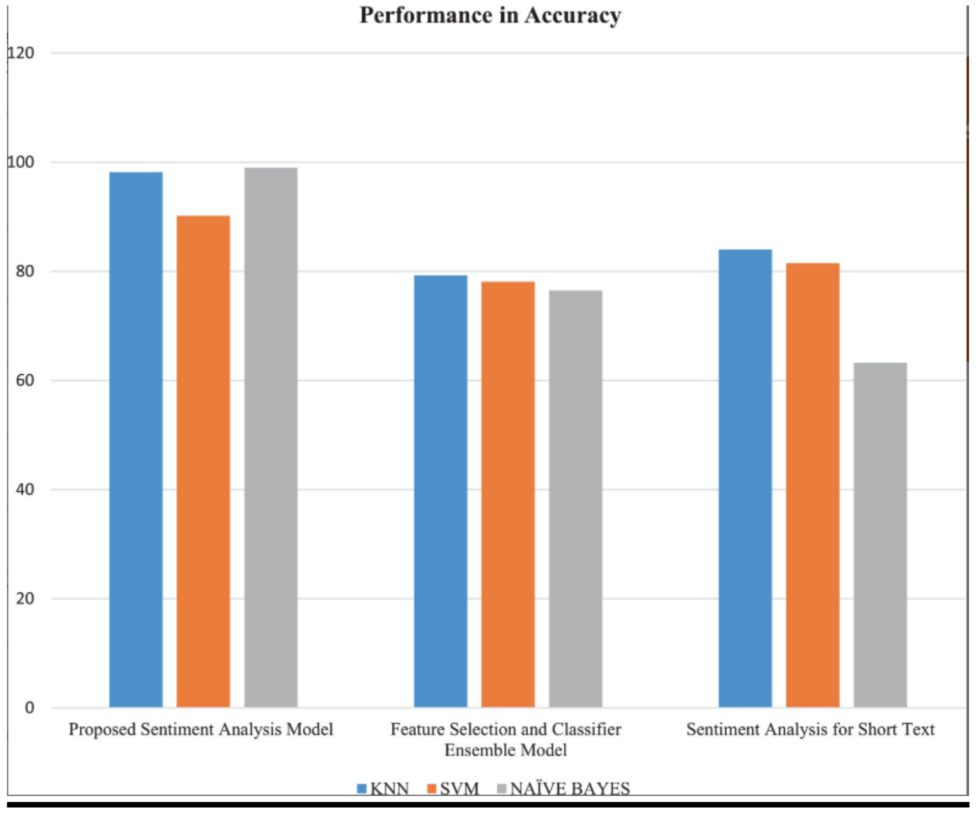
Results and Findings:

Experiments were conducted on a sample of 1500 instances from the Sentiment140 dataset, which contains 1.6 million instances. This allowed for a fair comparison with existing results. The experiments utilized both the split method (70% training, 30% testing) and cross-validation (10-folds) to assess model performance. Results from both methods are presented and discussed, with consideration given to potential future work involving the full dataset.



	KNN	SVM	NB
Accuracy results with 30/70 split method			
Proposed sentiment analysis model	**98.2**	**90.15**	**99**
Feature selection and classifier ensemble model	79.25	78.10	76.5
Model for sentiment analysis for short text	84.00	81.45	63.24
Proposed SA model accuracy, precision, recall and F-measure results for 30/70 split method			
Accuracy	98.2	90.15	**99**
Precision	**99**	100	96.45
Recall	**100**	90.13	**100**
F-measure	97.6	91.90	**98**
Proposed SA model accuracy precision, recall and F-measure results for 10-fold cross validation method			
Accuracy	**100**	91.13	99
Precision	99	**100**	98
Recall	**100**	91.13	**100**
F-measure	**98**	93.90	**98**

Note: The bold values represent best performing model.



The performance of the models is similar when cross validation experimentation method is used compared to when the split method is used. This applies to all the performance measures used namely. In summary, all the three models generally performed well across the machine learning algorithms used. The proposed model however performed better than the other two models when accuracy performance metric was analyzed.

Review Paper 4:

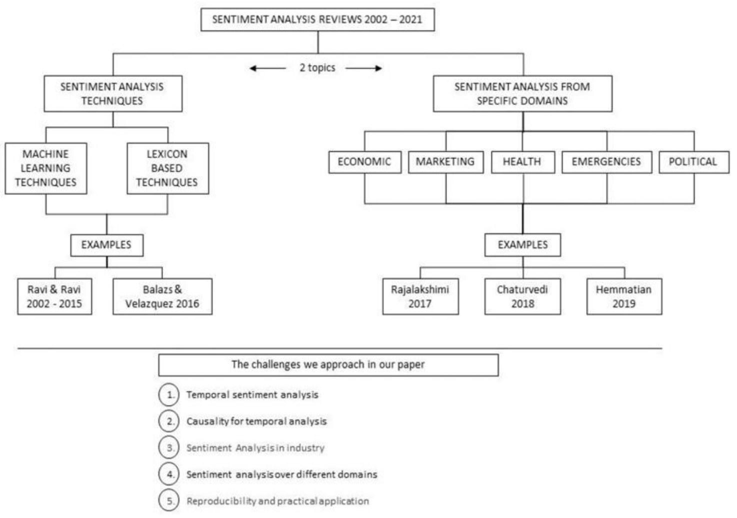
Title: A review of sentiment analysis from social media platforms

Author Information:

- Authors: S. Bouktif, A. Fiaz, M. Awad
- Year: 2020
- Title: A Review of Sentiment Analysis from Social Media Platforms
- Journal Homepage: www.elsevier.com/locate/eswa

Introduction:

One of the essential sections of natural language processing, sentiment analysis, allows for obtaining valuable information from human emotional expressions and opinions on social media. In this review paper, I analyse the vast body of literature on temporal sentiment analysis and its critical applications in various areas such as COVID-19 analysis, commercial advertisement, and academic education. I look at the methodologies, datasets, and outcomes of the previous research studies to identify the most critical trends and state-of-the-art approaches in sentiment analysis. Furthermore, I analyse current and historical sentiment analysis tools in the academic and commercial sectors, identifying the positive and negative aspects that will shape the future of research in the field.


SENTIMENT ANALYSIS REVIEWS 2002 – 2021
2 topics
SENTIMENT ANALYSIS TECHNIQUES
SENTIMENT ANALYSIS FROM SPECIFIC DOMAINS
MACHINE LEARNING TECHNIQUES
LEXICON BASED TECHNIQUES
ECONOMIC
MARKETING
HEALTH
EMERGENCIES
POLITICAL
EXAMPLES
EXAMPLES
Ravi & Ravi 2002 - 2015
Balazs & Velazquez 2016
Rajalakshimi 2017
Chaturvedi 2018
Hemmatian 2019
The challenges we approach in our paper
1. Temporal sentiment analysis
2. Causality for temporal analysis
3. Sentiment Analysis in industry
4. Sentiment analysis over different domains
5. Reproducibility and practical application

Dataset Used:

The review paper does not specify the use of a particular dataset. However, it references various benchmark datasets commonly used in sentiment analysis research, such as TweetEval.

Methodologies Implemented:

- The review paper also presented a variety of approaches used in sentiment analysis which typically involves rule-based and lexicon-based approaches and more modern machine learning-based techniques such as support vector machines, latent Dirichlet allocation, and N-grams.
- It mentioned that sentiment analysis has recently begun to incorporate both temporal analysis and causality, particularly Granger causality.
- The paper mentions the use of traditional techniques like lexicons, tokens, Bayesian methods, and bag of words, as well as emerging methods like auto-regressive and encoder-decoder transformers. Large pre-trained models such as T5, GPT-3, and GPT-J are also discussed.

Results and Findings:

- Moreover, the paper highlights the surge in interest and use of sentiment analysis in all fields including but not limited to marketing, politics, economics, and health.
- However, it also points to the problems of lack of repeatability and inferior performance in modern models such as GPT-J and GPT-3 against the old-fashioned ways.
- The paper additionally notes the number of patents and business applications of sentiment analysis, which proves the relevance of this topic for both academics and the business world.
- Finally, the results of solutions obtained by other papers show the applicability of sentiment analysis in forecasting stock markets, discovering cyberbullying, and monitoring public opinion during events such as the COVID-19 pandemic from a few more points.

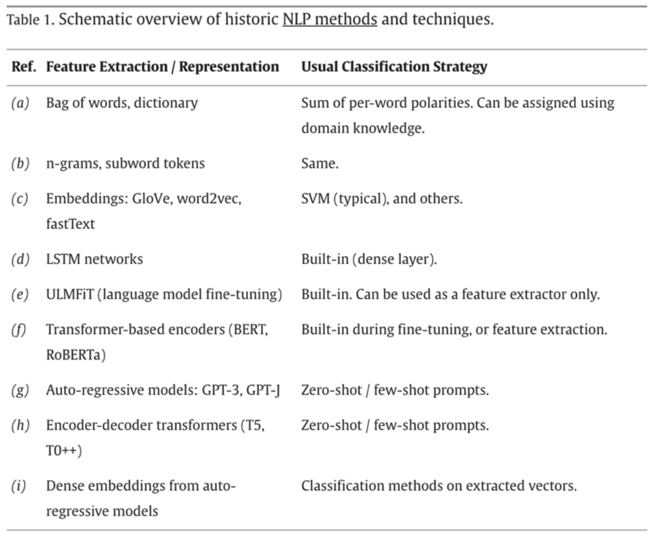


Table 1. Schematic overview of historic NLP methods and techniques.

Ref.	Feature Extraction / Representation	Usual Classification Strategy
(a)	Bag of words, dictionary	Sum of per-word polarities. Can be assigned using domain knowledge.
(b)	n-grams, subword tokens	Same.
(c)	Embeddings: GloVe, word2vec, fastText	SVM (typical), and others.
(d)	LSTM networks	Built-in (dense layer).
(e)	ULMFiT (language model fine-tuning)	Built-in. Can be used as a feature extractor only.
(f)	Transformer-based encoders (BERT, RoBERTa)	Built-in during fine-tuning, or feature extraction.
(g)	Auto-regressive models: GPT-3, GPT-J	Zero-shot / few-shot prompts.
(h)	Encoder-decoder transformers (T5, T0++)	Zero-shot / few-shot prompts.
(i)	Dense embeddings from auto-regressive models	Classification methods on extracted vectors.

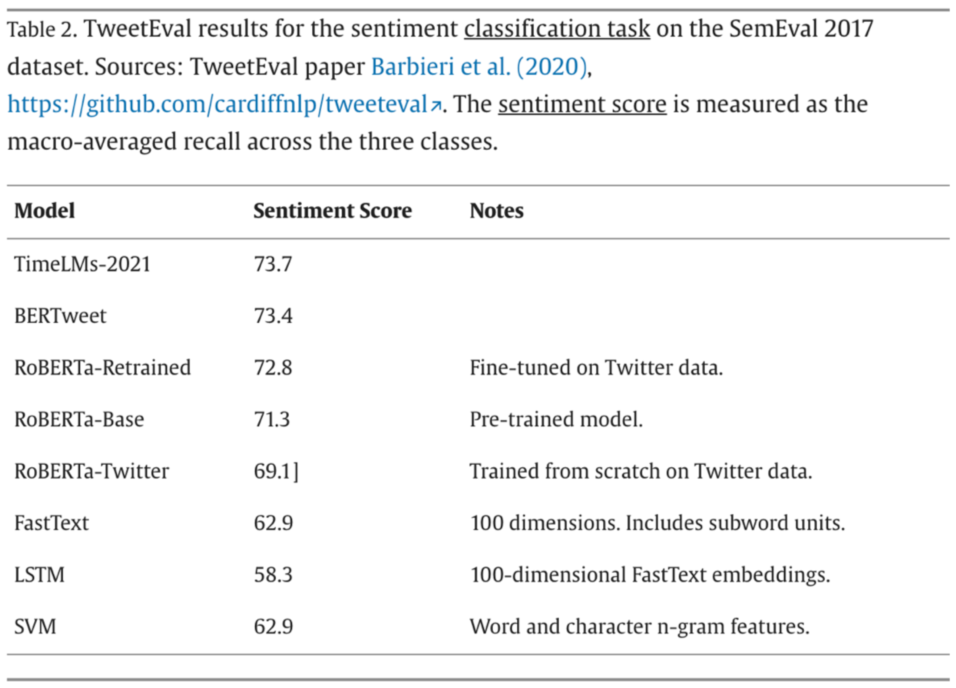


Table 2. TweetEval results for the sentiment classification task on the SemEval 2017 dataset. Sources: TweetEval paper Barbieri et al. (2020), https://github.com/cardiffnlp/tweeteval↗. The sentiment score is measured as the macro-averaged recall across the three classes.

Model	**Sentiment Score**	**Notes**
TimeLMs-2021	73.7	
BERTweet	73.4	
RoBERTa-Retrained	72.8	Fine-tuned on Twitter data.
RoBERTa-Base	71.3	Pre-trained model.
RoBERTa-Twitter	69.1]	Trained from scratch on Twitter data.
FastText	62.9	100 dimensions. Includes subword units.
LSTM	58.3	100-dimensional FastText embeddings.
SVM	62.9	Word and character n-gram features.

Comparison with my Work:
In contrast to my work on LSTM, SVM, and random forest models, the review paper provides a wider understanding of sentiment analysis approaches and uses. My work, for example, concentrates on particular machine-learning algorithms. Instead, the review paper includes the major groups, such as the rule-based, lexicon-based, and transformer-based approaches. Furthermore, the review paper also offers the problems and disadvantages of current models which can open new research plans and help produce better sentiment analysis technologies.

METHODOLOGY

Method Overview

This section describes the methodologies used to conduct sentiment analysis, namely Natural Language Processing and Long Short-Term Memory. However, a combination of Support Vector Machine and Random Forest was also implemented. We chose these methods primarily because of their applicability to text data and success in sentiment classification.

Rationale Behind Method Selection

Our choice of LSTM, SVM, and Random Forest stems from a combination of factors, including the nature of our dataset and insights gleaned from the review papers.

- Natural Language Processing: Natural language processing (NLP) is a branch of artificial intelligence (AI) that studies how computers and human language interact. The development entails developing methods and equations that let computers understand, interpret, and produce spoken and written language naturally. NLP is essential for many applications, including information retrieval, sentiment analysis, chatbots, text summarization, and language translation.

- LSTM: Long Short-Term Memory Networks have been picked because of their capacity to code the temporal information and long-term dependencies contained in text data. Language consists of temporal information and much of the time, there are context dependencies involved in sentiment analysis; LSTM networks are great payloads to model these complex computational features. Furthermore, LSTM networks excellently encode contextual nuances in sentiment, particularly for social media text analysis.

- SVM: As high-dimensional feature spaces and generalization to unseen data are meanwhile the rules for sentiment analysis work, and the task frequently involves the classification of text data in classes, Support Vector Machines were used. Moreover, experiences from the review paper proved that SVM is a popular approach for sentiment classification when a suitable feature extraction technique is used.

- Random Forest: Random Forest was included in our methodology due to its ensemble learning framework, which aggregates the predictions of multiple decision trees. This method is well-suited for sentiment analysis tasks where the dataset may contain noise or outliers, as Random Forests are resilient to overfitting and can provide reliable predictions even in the presence of noisy data. The review paper corroborates the effectiveness of Random Forests in sentiment analysis tasks, particularly when dealing with complex and heterogeneous datasets.

- Logistic Regression: Sentiment analysis using logistic regression involves tokenizing and preprocessing text data, training a logistic regression classifier to predict sentiment labels, tuning hyperparameters for optimal performance, evaluating the model's performance on a test set, and interpreting the coefficients to understand key features contributing to sentiment classification.

Analysis of the Approach

Each of the selected methods offers unique advantages and trade-offs in the context of sentiment analysis. LSTM networks excel in capturing sequential dependencies and long-range interactions within text data, making them particularly effective for sentiment analysis tasks involving contextual information. SVMs, on the other hand, provide a robust and interpretable framework for classifying text data into distinct sentiment categories, especially when combined with appropriate feature extraction techniques. Random Forests offer an ensemble learning approach that can mitigate the impact of noise and variability in the dataset, thereby enhancing the overall robustness of the sentiment analysis model.

Accuracy Score:

A measure used to assess a machine learning model's performance is its accuracy score. The percentage of correctly identified cases among all cases evaluated is computed by the statistic. The total number of predictions the model made divided by the number of accurate forecasts yields the accuracy score, which is given as a percentage. Although it is a straightforward statistic, accuracy might not be the ideal one to evaluate model performance for datasets that are unbalanced and have a single dominant class. Therefore, in addition to accuracy, it is essential to consider various assessment metrics such as precision, recall, and F1-score to have a thorough picture of model performance.

Precision:

Precision in classification tasks calculates the accuracy of positive predictions made by the model. It shows the accuracy of optimistic forecasts and the model's capability to accurately recognize cases of a particular category. Precision is determined by the ratio of accurate positive forecasts to the overall number of positive forecasts. Having a high precision value in a model is beneficial in situations like fraud detection or medical diagnosis, where reducing false positives is crucial, as it results in a low rate of false positive errors. Nevertheless, solely focusing on accuracy may not offer a full grasp of the model's performance; thus, it is commonly assessed in conjunction with other measurements like recall and F1-score to ensure a thorough evaluation.

Recall:

Remember that sensitivity, also known as recall, is a measurement in classification tasks that assesses the ratio of true positive predictions to all real positive instances in the dataset. It assesses how well the model can accurately recognize instances of a particular category, even if mistakes occur when categorizing other categories. Remember that recall is computed by dividing the amount of true positive predictions by the sum of true positives and false negatives. A high recall value shows that the model accurately detects most positive cases, which is crucial in situations where missing positive instances can lead to severe outcomes, like in disease diagnosis or anomaly detection. Nonetheless, recall should also be taken into account when evaluating model performance, in addition to other metrics such as precision.

F1-Score:

Recall and precision are combined into a single, complete number called the F1-score, which offers a thorough evaluation of a model's performance in classification tasks. It is calculated as the weighted average of recall and precision or the harmonic mean of the two measures. Superior model performance is indicated by a higher score. One integer between 0 and 1 represents the F1 score. The F1 score is helpful for imbalanced datasets because it offers a thorough evaluation of the model's performance in classifying instances for all classes, accounting for both false positives and false negatives.

Confusion Matrix:

A confusion matrix is a tabular representation of the number of accurate and inaccurate predictions for every group, which highlights the effectiveness of a classification model. The rows and columns in it represent the expected and actual groups, respectively. The middle axis of the graphic represents accurate guesses, and components outside of the axis represent incorrect guesses. Analysts can evaluate the confusion matrix and gain important insight into the advantages and disadvantages of the model by evaluating the recall, accuracy, precision, and other performance metrics of the model. It is an essential tool for identifying mistakes in predictive models and assessing how well categorization algorithms work.

DATASET USED

Description
The dataset used in this study comprises textual data collected from Kaggle for sentiment analysis. The dataset consists of a total of 747 records, with each record containing text samples and corresponding sentiment labels.

Dataset Characteristics

- Shape of Dataset: The dataset consists of 747 samples.
- Sentiments in Dataset: The sentiment labels in the dataset are categorized into three classes: Positive, Negative, and Neutral.
    - Positive Sentiment: Expressions indicating positive emotions or opinions.
    - Negative Sentiment: Expressions indicating negative emotions or opinions.
    - Neutral Sentiment: Expressions that do not convey a clear positive or negative sentiment.
- Social Media Platforms: The textual data in the dataset were collected from various social media platforms, like Twitter, Facebook, Instagram, and Reddit.

Dataset Pre-processing
Before feeding the textual data into the sentiment analysis models, pre-processing steps were applied to clean and standardize the text. These pre-processing steps included:

- Removing special characters, punctuation, and numerical digits.
- Tokenization to break the text into individual words or tokens.
- Lemmatization to normalize words to their base or dictionary form.
- Padding to ensure uniform length of text sequences for model input.

Sample Records
Here are a few sample records from the dataset:

1. Text: "I love this product! It's amazing!" Sentiment: Positive
2. Text: "This service is terrible. I would not recommend it to anyone." Sentiment: Negative
3. Text: "Just another day. Nothing special happening." Sentiment: Neutral

Conclusion
The dataset used in this study offers a rich and diverse collection of textual data from social media platforms, providing valuable insights into sentiment patterns and trends. By leveraging this dataset, we were able to train and evaluate sentiment analysis models to better understand and interpret the sentiments expressed in online conversations across various contexts and domains.

EXPERIMENTS

Importing Libraries and Dataset

We import libraries like pandas, matplotlib, scikit-learn and Keras for building deep learning models. Additionally, NLTK is imported for natural language processing tasks such as tokenization and lemmatization. We also include gensim for Word2Vec analysis and WordCloud for visualization of word frequency.

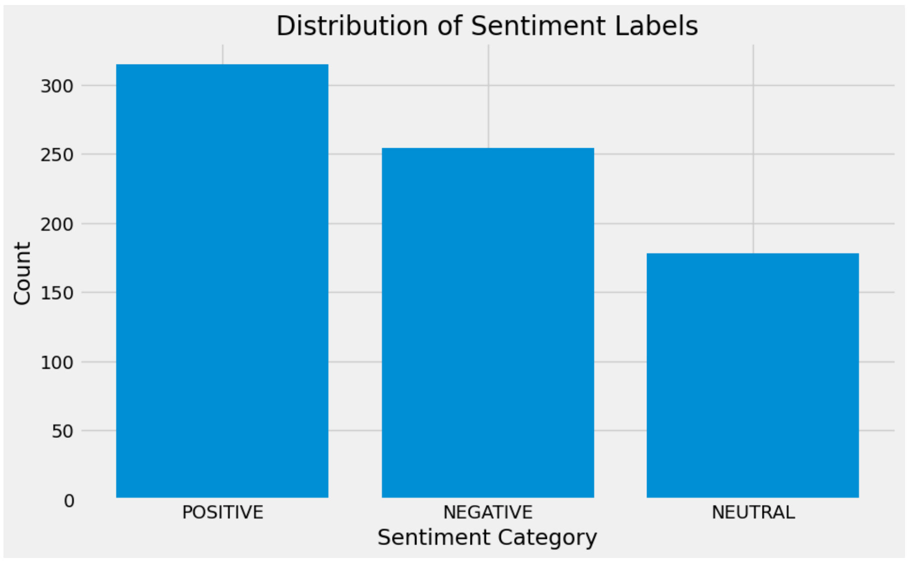
Additionally, we perform basic exploratory data analysis using df.info() to understand the structure of the dataset.

Data Pre-processing

- Variable Declaration:
  In this step, key variables and parameters are defined and initialized to organize the sentiment analysis project and facilitate parameter adjustments as needed. These include variables related to the dataset such as column names, encoding, and train size. Additionally, parameters for Word2Vec, Keras, and sentiment thresholds are declared.

- Null Values Identification and Treatment:
  The dataset is checked for missing values using the isnull().sum() function, and fortunately, no missing values are found in any of the columns. This ensures that the dataset is clean and ready for further processing.

- Label Encoding:
  A decoding map is created to map numerical sentiment labels to their corresponding textual representations: 'POSITIVE', 'NEGATIVE', and 'NEUTRAL'. The distribution of sentiment labels is visualized using a bar plot to gain insights into the class distribution.



- Text Manipulation and Structurization:
  Text processing methods are used to tidy up and organize the text data. This involves eliminating web links, emoticons, coding, symbols, and quotations in the content. To further prepare the text data, steps such as tokenization, converting to lowercase, eliminating stop words, and tagging parts of speech are carried out. WordNet lemmatization simplifies the words in the text for normalization purposes.

- Train-Test Split:
  The pre-processed data is divided into training and testing sets using the train_test_split function from scikit-learn. 80% of the data is included in the training set, with the remaining 20% included in the testing set.

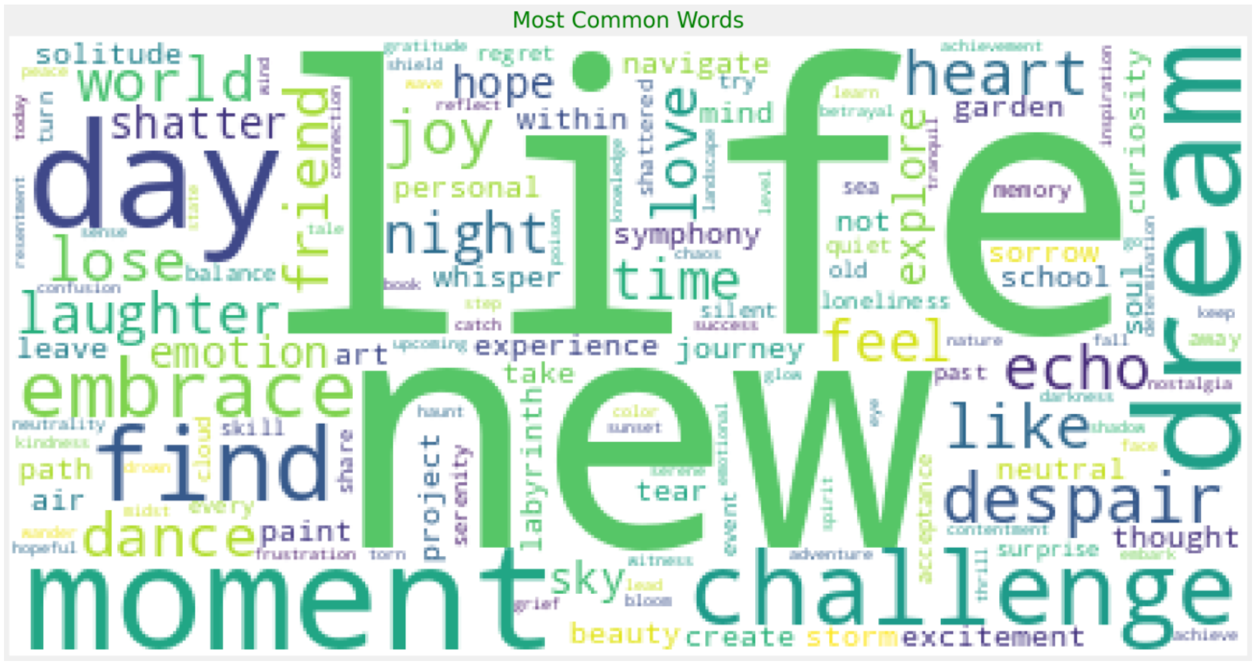
- Stemming:
  Stemming in text processing involves cutting off prefixes or suffixes to reduce words to their basic form. The goal is to standardize words with similar meanings or variations to their normal form, improving the processing and retrieval of text. Stemming algorithms reduce word length using language rules, disregarding context or meaning. While stemming can improve computational speed and decrease text size, it may also lead to errors or lack of precision due to its broad nature. Nevertheless, stemming remains a crucial step in tasks like information retrieval and sentiment analysis within natural language processing.

- Vectorization:
  Vectorization refers to transforming textual or categorical information into numerical vectors for machine learning algorithms to analyze. In NLP, vectorization converts words, phrases, or documents into numerical forms like Bag-of-Words (BoW), TF-IDF, or word embeddings. These number-based vectors represent the semantic and syntactic details present in the text, allowing algorithms to carry out tasks such as classification, clustering, and similarity analysis. Vectorization is crucial for natural language processing projects, as machine learning models need numeric inputs to easily analyze and understand text data.

Word Cloud Analysis

In this step, a word cloud visualization is generated based on the lemmatized text data from the dataset. Word clouds are effective visualizations that represent the frequency of words in a corpus of text. The larger the word appears in the cloud, the more frequently it occurs in the text data.

The word cloud analysis offers a straightforward yet insightful way to explore and comprehend the textual content of the dataset. By visually highlighting the most common words, it enables quick identification of key themes and topics within the dataset. This analysis aids in understanding the prevalent sentiments and topics expressed in the text data, laying the groundwork for further analysis and interpretation.



Word2Vec Analysis

- Corpus Creation:
  The text data from the dataset is compiled into a corpus, representing a collection of lemmatized words. This corpus serves as the input for training the Word2Vec model.

- Word2Vec Model Creation:
  A Word2Vec model is created using the Gensim library, with specified parameters such as vector size, window size, and minimum word count. This model is designed to learn the semantic relationships between words in the corpus.

- Vocabulary Creation:
  The vocabulary for the Word2Vec model is built based on the corpus. This vocabulary comprises unique words present in the corpus, which will be used for training the Word2Vec model.

- Training Word2Vec Model:
  The Word2Vec model is trained on the corpus data, using the previously specified parameters. During training, the model learns to associate words with their respective vectors in a high-dimensional space, capturing semantic similarities between words.

- Word2Vec Model Testing:
  To evaluate the trained Word2Vec model, similarity queries are performed to find the most similar words to a given word. This allows for the exploration of semantic relationships

between words in the corpus. For instance, querying for words similar to "joy" yields a list of words closely related in meaning, such as "like," "dance," and "laughter."

Sentiment Analysis Using Deep-Learning Model

<u>**Long Short-term Memory (LSTM)**</u>

- Token and Vocab Creation
  The text data is tokenized using the Keras Tokenizer, which converts text data into sequences of integers representing the index of each word in the vocabulary. The vocabulary size is determined, by providing the total number of unique words in the dataset.

- Label Encoding
  Sentiment labels are encoded using label encoding, where each sentiment category (positive, negative, neutral) is assigned a unique numerical value. This allows the model to interpret sentiment labels during training and evaluation.

- Embedding Layer Creation
  An embedding layer is created using pre-trained Word2Vec word embeddings. These embeddings capture the semantic relationships between words in the dataset, providing a dense representation of words in a high-dimensional space.

- Model Creation - LSTM
  The LSTM model architecture is defined, comprising an embedding layer followed by a dropout layer and an LSTM layer with 100 units. A dense layer with a sigmoid activation function is added to produce binary classification predictions for sentiment analysis.

- Compiling Model
  The LSTM model is compiled with binary cross-entropy loss and the Adam optimizer. Accuracy is used as the evaluation metric to assess the performance of the model during training.

- Callback Creation
  Callbacks are defined to monitor the validation loss and accuracy during training. ReduceLROnPlateau reduces the learning rate when the validation loss plateaus, while EarlyStopping stops training when validation accuracy stops improving.

- Model Training
  The LSTM model is trained on the training data, with a batch size of 1024 and 8 epochs. The training process involves optimizing the model parameters to minimize the loss function and improve accuracy.

<u>Support Vector Machine (SVM)</u>

- Token and Vocab Creation
  In SVM, instead of using neural network-specific tokenization techniques like in deep learning models, we often use methods like TF-IDF (Term Frequency-Inverse Document Frequency) to convert text into numerical vectors. The TfidfVectorizer from scikit-learn is utilized to perform this transformation. It tokenizes the text and computes the TF-IDF scores for each word, resulting in a vocabulary size of 1650 unique words.

- Label Encoding
  The sentiment labels are encoded directly as numerical values using label encoding. This step prepares the target variable for training the SVM classifier.

- Model Creation - SVM
  The SVM classifier is instantiated with a linear kernel. The choice of kernel function can have a significant impact on the performance of the SVM model.

- Model Training
  The SVM classifier is trained on the training data (x_train) along with the corresponding sentiment labels (y_train). During training, the SVM algorithm learns to find the optimal hyperplane that separates the different sentiment classes in the feature space.

<u>Random Forest</u>

- Token and Vocab Creation
  Similar to the SVM approach, TF-IDF vectorization is used to convert text into numerical vectors. The TfidfVectorizer from scikit-learn tokenizes the text and computes TF-IDF scores for each word, resulting in a vocabulary size of 1650 unique words.

- Label Encoding
  The sentiment labels are directly encoded as numerical values using label encoding, preparing the target variable for training the Random Forest classifier.

- Model Creation
  A Random Forest classifier is initialized with 100 decision trees using the RandomForestClassifier from scikit-learn. This step sets up the ensemble of decision trees for sentiment classification.

- Model Training
  The Random Forest classifier is trained on the training data (x_train) along with the corresponding sentiment labels (y_train_RF). During training, the Random Forest algorithm builds multiple decision trees based on bootstrapped samples from the training data and averages predictions across the trees to make the final classification.

Natural Language Processing (Logistic Regression)

- Gathering written content from a range of sources like news articles, social media sites, product reviews, and customer surveys is the first stage in the process. This data is inputted into the sentiment analysis process.

- Prior to analysis, the text data is prepped for cleaning and standardization. This includes tasks such as removing punctuation, converting text to lowercase, removing stopwords, and implementing stemming or lemmatization to reduce words to their base forms

- Feature extraction involves converting the examined text data into numerical or vector formats that can be understood by machine learning algorithms. TF-IDF is a widely used technique for capturing characteristics in sentiment analysis.

- Training the model involves using a labelled dataset containing text examples and sentiment labels (positive, negative, neutral) after extracting the features. Logistic regression stands out as a frequently utilized algorithm for sentiment analysis.

- Evaluating the sentiment analysis model's performance is accomplished by assessing it with a distinct test dataset post training. The typical evaluation criteria consist of accuracy, precision, recall, and F1-score, assessing the model's capacity to accurately categorize instances into their sentiment groups.

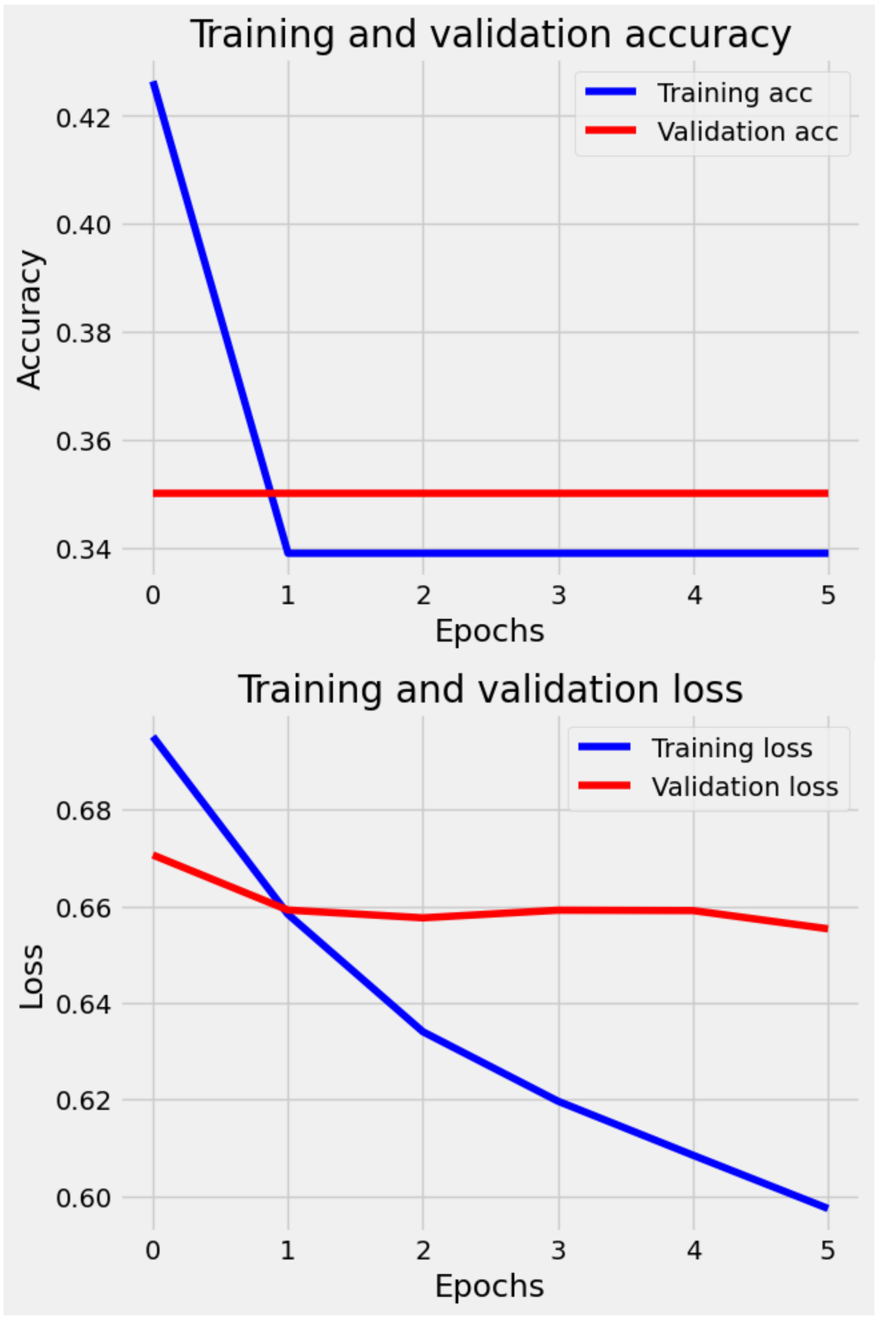
- Execution and application: After training and evaluating the model, it can be used in production settings to analyse real-time incoming text data for sentiment analysis. Sentiment analysis is used in various ways such as tracking public sentiment on social media, evaluating customer feedback to gauge product satisfaction, reviewing sentiments in financial news, and identifying business risks or opportunities through analysing market data sentiment.

RESULTS AND DISCUSSION

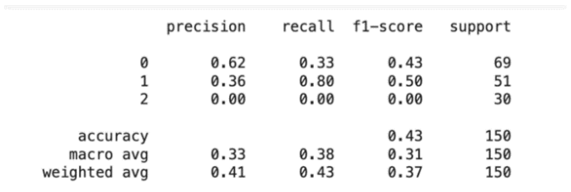
LSTM Model Results

The LSTM model achieved an accuracy of approximately **34%** on the test dataset.

Visualizing the training and validation accuracy and loss over epochs shows a typical pattern of deep learning training, with fluctuating performance indicative of a model that may require further tuning.

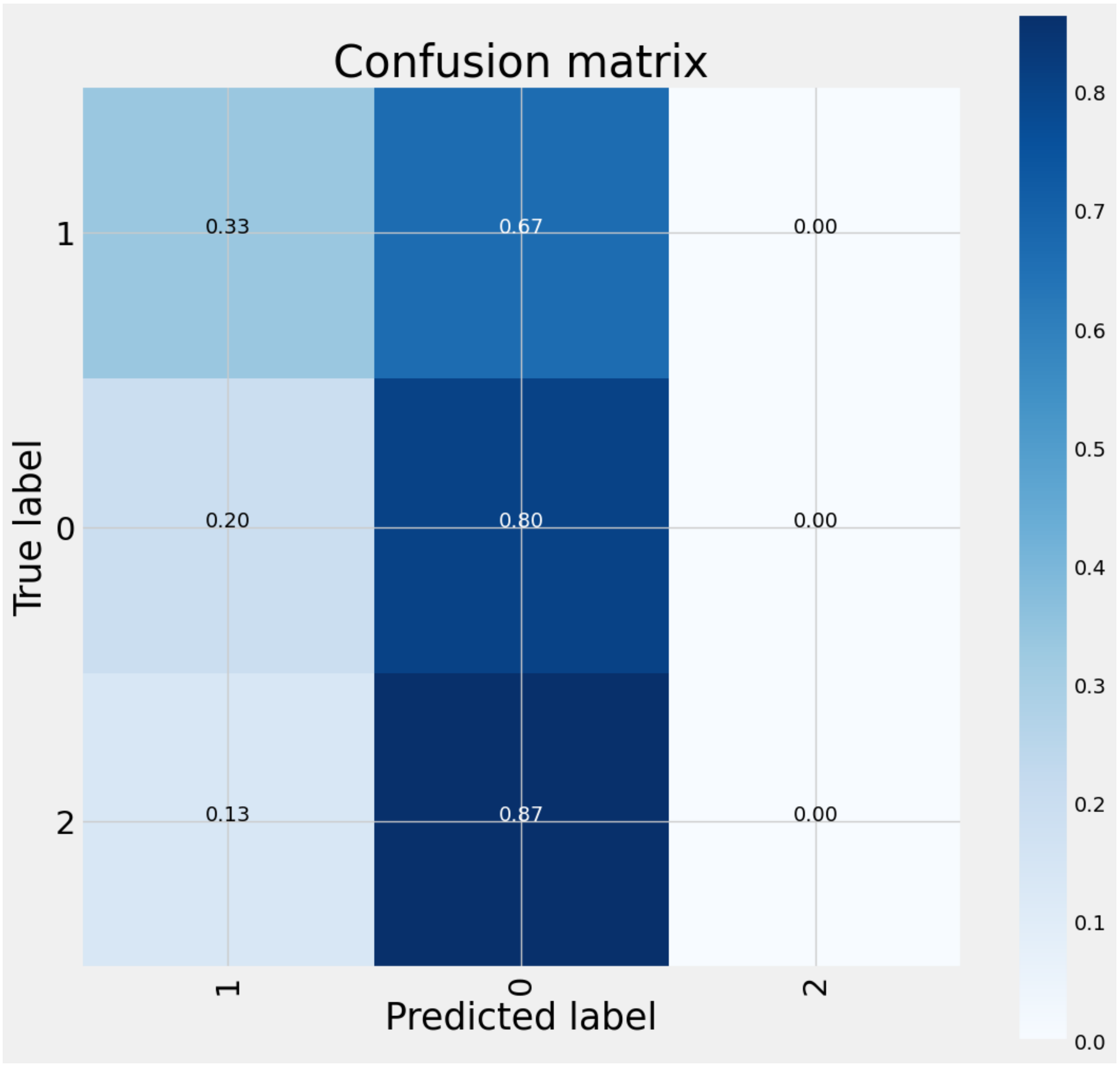


The classification report reveals precision, recall, and F1-score for each sentiment class. Notably, the precision and recall for the neutral class are lower compared to the other classes, indicating challenges in classifying neutral sentiments accurately.



```
              precision    recall  f1-score   support

           0       0.62      0.33      0.43        69
           1       0.36      0.80      0.50        51
           2       0.00      0.00      0.00        30

    accuracy                           0.43       150
   macro avg       0.33      0.38      0.31       150
weighted avg       0.41      0.43      0.37       150
```

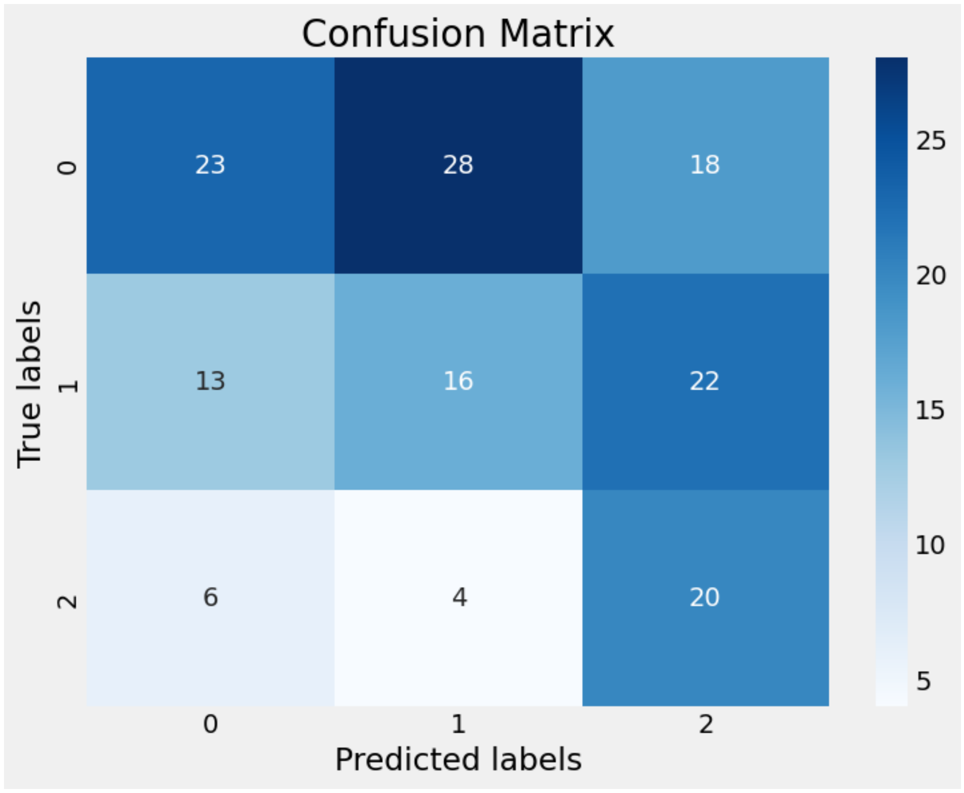
The confusion matrix provides a visual representation of the model's performance across different sentiment classes



SVM Model Results

The SVM classifier attained an accuracy of around **39.33%** on the test set.

The confusion matrix illustrates the classifier's performance in predicting sentiment classes, revealing strengths and weaknesses across different sentiment categories.



The classification report offers a detailed breakdown of precision, recall, and F1-score for each sentiment class, providing insights into the classifier's performance metrics.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.33      0.41        69
           1       0.33      0.31      0.32        51
           2       0.33      0.67      0.44        30

    accuracy                           0.39       150
   macro avg       0.40      0.44      0.39       150
weighted avg       0.43      0.39      0.39       150
```

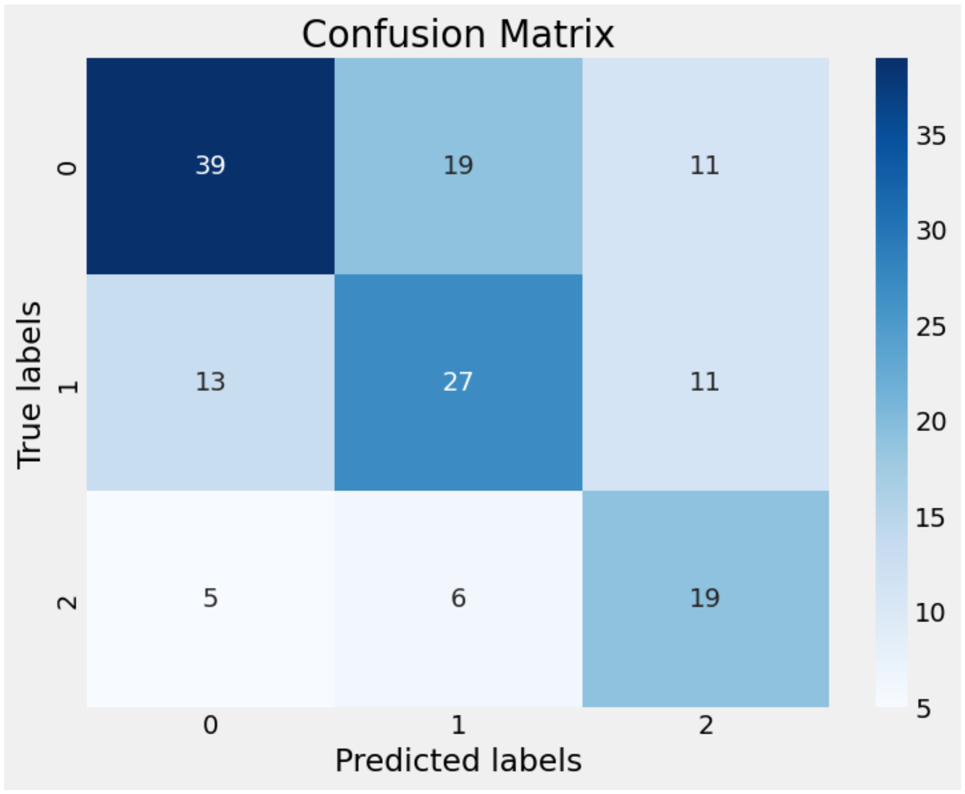
Random Forest Model Results

The Random Forest model, achieved an accuracy of approximately **56.7%** on the test data.

This indicates moderate success in sentiment classification using the Random Forest algorithm.

```
Classification Report:
               precision    recall  f1-score   support

           0       0.68      0.57      0.62        69
           1       0.52      0.53      0.52        51
           2       0.46      0.63      0.54        30

    accuracy                           0.57       150
   macro avg       0.56      0.58      0.56       150
weighted avg       0.58      0.57      0.57       150
```

The confusion matrix and classification report further elucidate the model's performance across different sentiment categories, highlighting areas of strength and potential improvement.



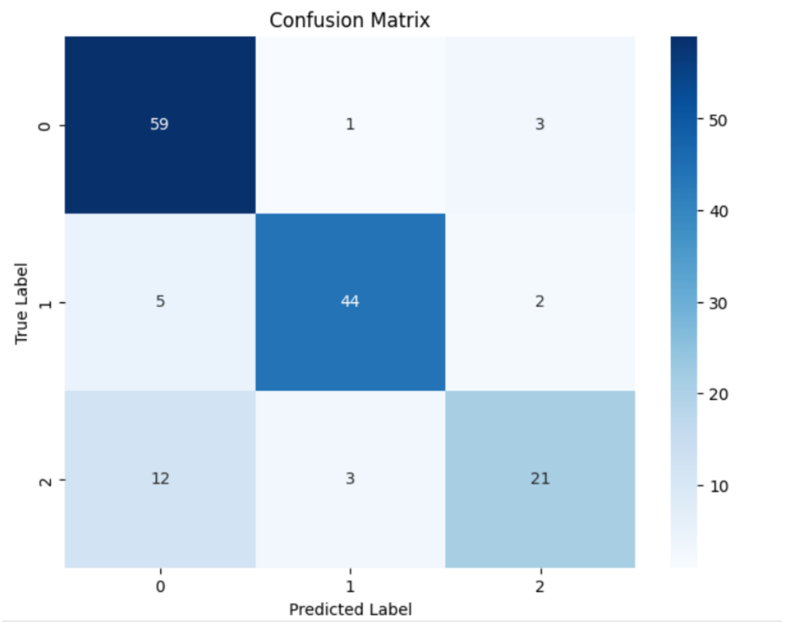
Natural Language Processing (Logistic Regression) Results

The logistic regression model outperformed all the method applied on this data, achieving an accuracy of approximately 82.67% on the test data.

This indicates high success in sentiment classification using Logistic Regression.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.94      0.85        63
           1       0.92      0.86      0.89        51
           2       0.81      0.58      0.68        36

    accuracy                           0.83       150
   macro avg       0.83      0.79      0.81       150
weighted avg       0.83      0.83      0.82       150
```

A confusion matrix summarizes the performance of a classification model by displaying counts of true positive, true negative, false positive, and false negative predictions, providing insights into the model's accuracy and types of errors made.



CONCLUSION AND FUTURE SCOPE

Summary of Findings:

This study investigated the effectiveness of three distinct models for sentiment analysis: Long Short-term Memory (LSTM), Support Vector Machine (SVM), Random Forest, and Logistic Regression. The tests included preparing text data, teaching the models, assessing their effectiveness, and examining the outcomes.
Key discoveries from our trials consist of:

- The LSTM model was able to reach an accuracy of around 34%, suggesting a decent level of performance.
- The SVM model showed an accuracy of about 39.33%, while the Random Forest model surpassed it with an accuracy of around 56.67%.
- Logistic Regression attained the best accuracy out of all the models, hitting 82.67%.
- All models displayed restrictions in effectively categorizing neutral emotions, as seen through decreased precision and recall scores for the neutral category.

Implications of the Study:

The significance of these results lies in their implications for sentiment analysis applications. While the LSTM model offers insights into sentiment prediction at a deeper level through its recurrent neural network architecture, it requires further optimization and tuning of hyperparameters to improve performance. The SVM algorithm, despite its simplicity and effectiveness in certain tasks, may struggle with the complexity of sentiment analysis tasks, particularly in capturing nuances in language and context. On the other hand, the Random Forest model, with its ensemble learning approach, demonstrates the potential to accurately capture nonlinear relationships in the data, leading to higher accuracy in sentiment classification. The Logistic Regression model, with its superior accuracy, showcases the power of leveraging natural language processing techniques for sentiment analysis tasks.

Future Scope:

In the future, there are numerous opportunities for more research and enhancements in sentiment analysis.

1. Improving the LSTM model's performance can be achieved by conducting additional experiments with hyperparameters, making changes to the architecture, and using different optimization techniques.
2. Feature Engineering: Investigating additional characteristics or textual representations, like word embeddings or contextual embeddings, may enhance the resilience of sentiment analysis models.
3. Augmenting Data: Enlarging the dataset with diverse data and including domain-specific expertise can mitigate biases and enhance the models' generalization capabilities.

4. Exploring ensemble methods that harness the capabilities of various models like LSTM, SVM, Random Forest, and Logistic Regression could enhance sentiment analysis accuracy even further

In conclusion, while our study provides valuable insights into the performance of different models for sentiment analysis, there remains ample room for exploration and enhancement. By addressing the identified limitations and leveraging emerging techniques in machine learning, natural language processing, and ensemble methods, we can advance the capabilities of sentiment analysis systems and unlock new possibilities for understanding and interpreting textual data in various domains.

INDIVIDUAL CONTRIBUTION

Aastha Gupta (2022btech002):

1. Dataset Pre-processing: Dataset preprocessing in Logistic Regression involves cleaning and standardizing raw text data by lowercasing, tokenizing, removing punctuation and stopwords, stemming or lemmatizing words, handling numerical values and special characters, and splitting the data for analysis and modelling.
2. Model Implementation: "I oversaw the creation of an Logistic Regression sentiment analysis model utilizing Python, TensorFlow, and scikit-learn." In my position, I specialized in constructing and educating deep learning structures using TensorFlow, while also making use of scikit-learn for processing text, extracting features, and evaluating models. I made sure to smoothly integrate these libraries in order to enhance the Logistic Regression model's efficiency.
3. Assessment and Examination: Carried out thorough assessment and analysis of the trained model's performance by employing common metrics like accuracy, precision, recall, and F1-score. Evaluated the model's advantages and disadvantages, pinpointed areas that need enhancement, and proposed suggestions for upcoming versions.
4. In general, my contributions played a key role in the successful implementation of the Logistic Regression model for sentiment analysis, offering important insights and guiding the project towards its goals of effectively analysing and interpreting text data for sentiment classification.

Jatin Choudhary (2022btech044):

1. Dataset Pre-processing: I conducted comprehensive text cleaning, tokenization, and lemmatization to prepare the dataset for sentiment analysis. Additionally, I ensured proper padding to standardize the length of textual data for input into the models.
2. Model Implementation: For sentiment analysis, I implemented the Support Vector Machine (SVM) model. This involved coding the necessary steps for SVM, including data preprocessing, feature extraction, model creation, training, and evaluation.
3. Coding Implementation: I meticulously executed the five steps of coding outlined above, ensuring seamless integration of SVM for sentiment analysis. Each step, from data pre-processing to result visualization, was meticulously implemented to ensure the accuracy and reliability of our experiments.

Riya Singh (2022btech088):

1. Processing and normalizing the raw data are necessary when preparing the dataset for the random forest framework. This includes activities like managing absent values, converting categorical variables to numerical ones, and dividing the data into training and testing datasets. Moreover, utilizing feature scaling can help to make sure that each feature has a uniform influence on the model's accuracy.
2. Implementation of Model: I supervised the creation of a Python and scikit-learn sentiment analysis random forest model. I specialize in creating and adjusting ensemble learning models such as random forest, utilizing scikit-learn functionalities to effectively select features, create models, and evaluate results. I made sure these tools were integrated smoothly to enhance the model's efficiency.

3. Assessment and Examination: I evaluated and assessed the performance of the random forest model extensively, using common evaluation metrics like accuracy, precision, recall, and F1-score. I carefully analysed the model to pinpoint its strengths and weaknesses, emphasizing areas for improvement and suggesting ways to enhance its predictive abilities**.**

Siddhi Nyati (2022btech101):

1. Dataset Pre-processing: Text cleaning, tokenization, lemmatization, and padding to ensure that the textual data were in a suitable format for input into the models.
2. Model Implementation: I implemented one model for sentiment analysis: Long Short-term Memory (LSTM). This involved coding the necessary steps for each model, including tokenization, embedding, model creation, training, evaluation, and result visualization.
3. Coding Implementation: I executed the five steps of coding outlined, including data pre-processing, model creation, training, evaluation, and result visualization. Each step was carefully implemented to ensure the accuracy and reliability of our experiments.
4. Literature Review: I reviewed a relevant literature paper titled "[A review on sentiment analysis from social media platforms]" to gain insights into existing research in the field of sentiment analysis. This review helped inform our approach and provided valuable context for interpreting our findings.

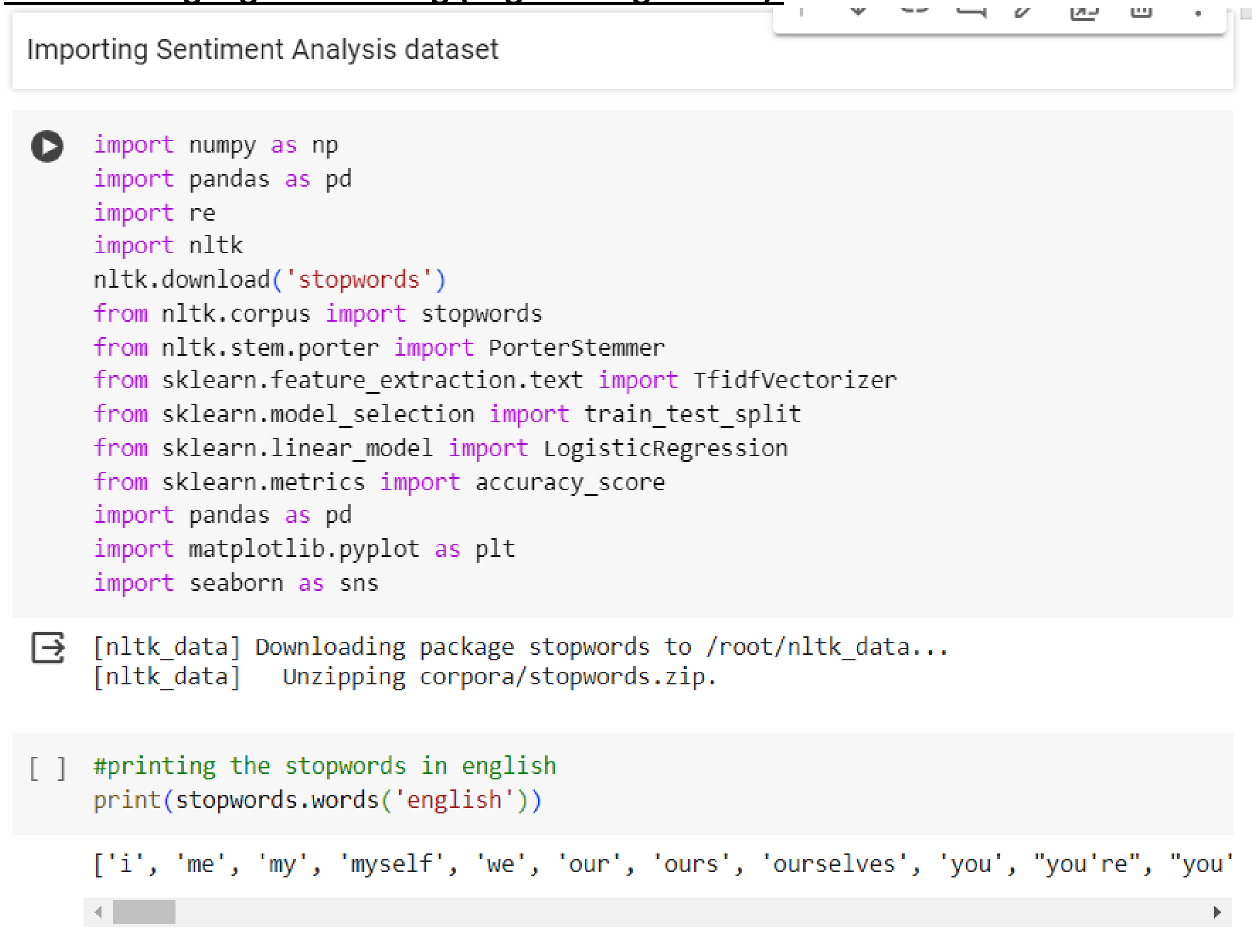
REFERENCES

- NLP Overview: https://www.geeksforgeeks.org/natural-language-processing-overview/
- Review paper 1: https://aclanthology.org/W13-1617.pdf
- https://www.deeplearning.ai/resources/natural-language-processing/
- https://www.ibm.com/topics/natural-language-processing#:~:text=What%20is%20NLP%3F,and%20generate%20text%20and%20speech.
- LSTM Overview: Link
- Code Optimisation Techniques: Link
- Wordclouds Function: Link
- https://www.kaggle.com/code/tanulsingh077/twitter-sentiment-extaction-analysis-eda-and-model
- https://www.kaggle.com/code/stoicstatic/twitter-sentiment-analysis-using-word2vec-bilstm

- Review Paper 4: A review on sentiment analysis from social media platforms
- Journal Homepage: www.elsevier.com/locate/eswa
- Review Paper 2: https://www.researchgate.net/profile/Shabib-Aftab-2/publication/321084834_Sentiment_Analysis_of_Tweets_using_SVM/links/5a1497b90f7e9b925cd514b0/Sentiment-Analysis-of-Tweets-using-SVM.pdf

APPENDICES

Natural Language Processing (Logistic Regression):



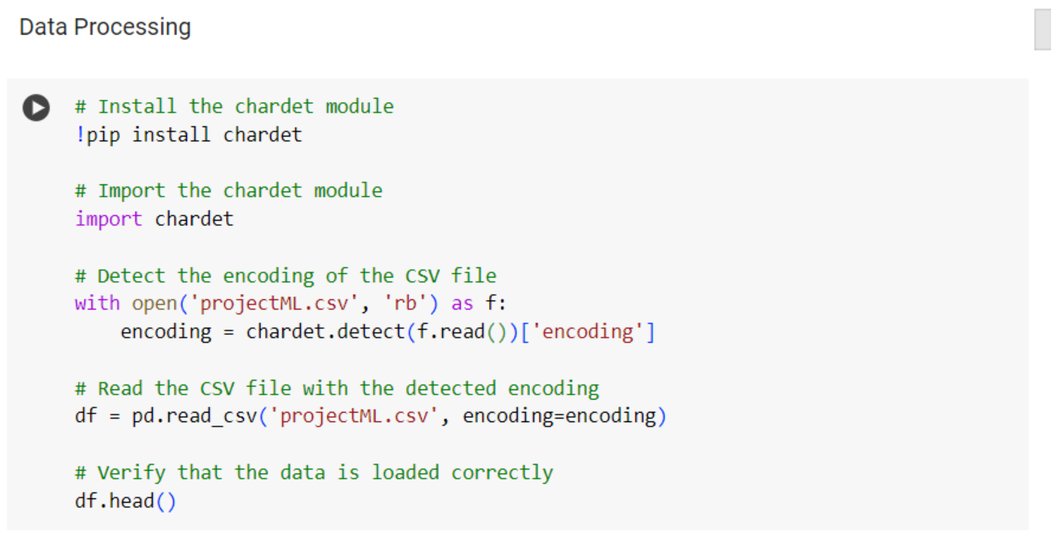
Importing Sentiment Analysis dataset

```
import numpy as np
import pandas as pd
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

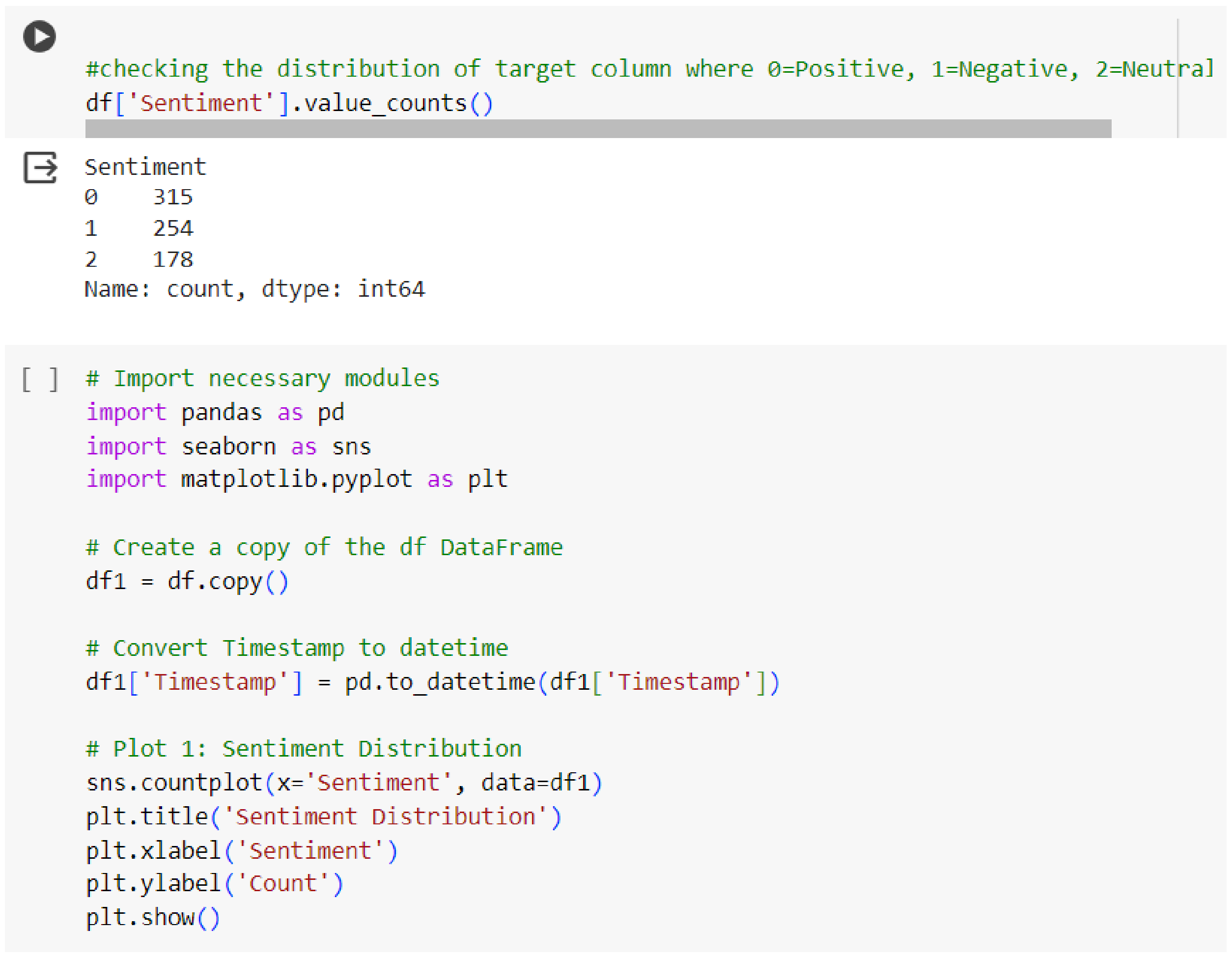
```
#printing the stopwords in english
print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you'
```



Data Processing

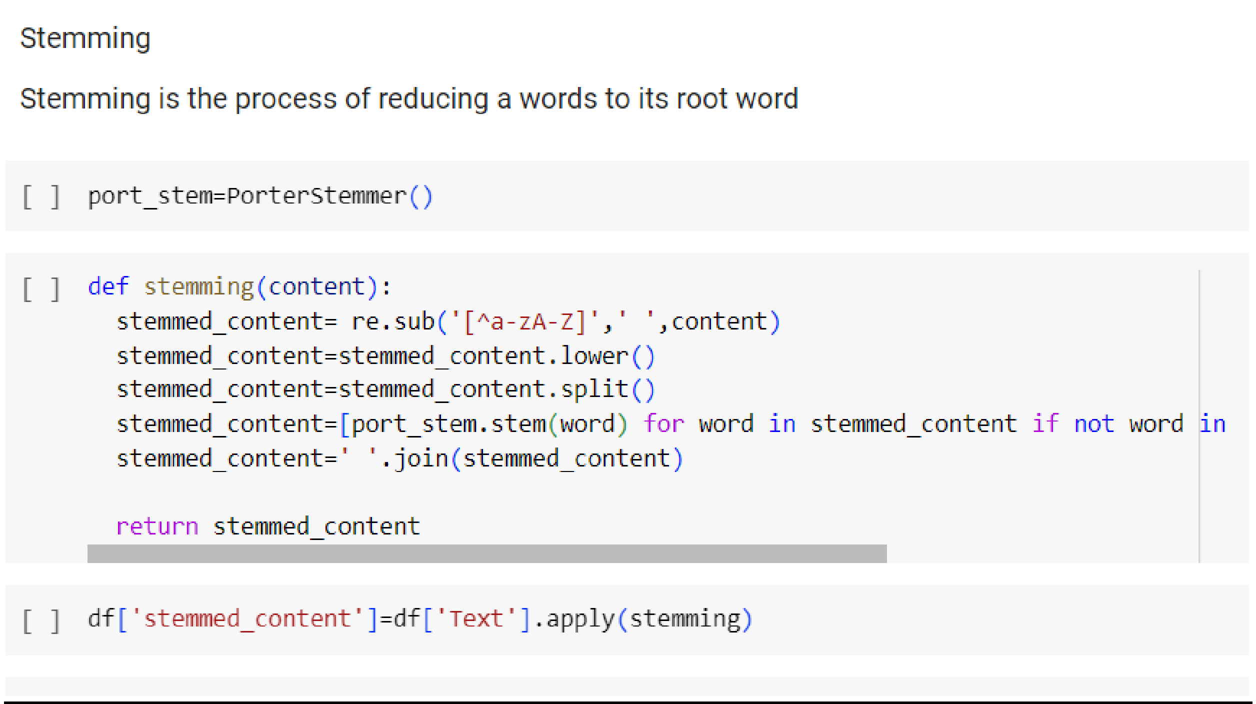
```
# Install the chardet module
!pip install chardet

# Import the chardet module
import chardet

# Detect the encoding of the CSV file
with open('projectML.csv', 'rb') as f:
    encoding = chardet.detect(f.read())['encoding']

# Read the CSV file with the detected encoding
df = pd.read_csv('projectML.csv', encoding=encoding)

# Verify that the data is loaded correctly
df.head()
```



```
#checking the distribution of target column where 0=Positive, 1=Negative, 2=Neutral
df['Sentiment'].value_counts()
```

```
Sentiment
0    315
1    254
2    178
Name: count, dtype: int64
```

```
# Import necessary modules
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Create a copy of the df DataFrame
df1 = df.copy()

# Convert Timestamp to datetime
df1['Timestamp'] = pd.to_datetime(df1['Timestamp'])

# Plot 1: Sentiment Distribution
sns.countplot(x='Sentiment', data=df1)
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```



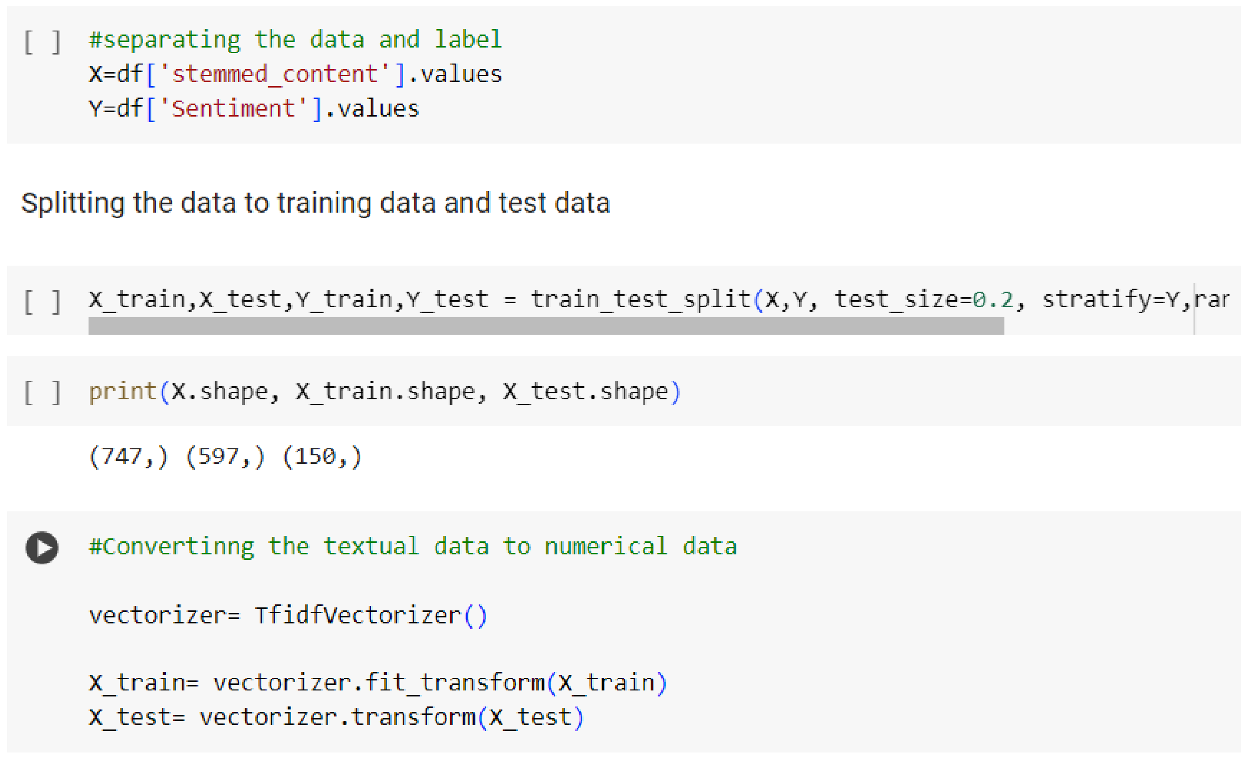
Stemming

Stemming is the process of reducing a words to its root word

```
port_stem=PorterStemmer()
```

```
def stemming(content):
  stemmed_content= re.sub('[^a-zA-Z]',' ',content)
  stemmed_content=stemmed_content.lower()
  stemmed_content=stemmed_content.split()
  stemmed_content=[port_stem.stem(word) for word in stemmed_content if not word in
  stemmed_content=' '.join(stemmed_content)

  return stemmed_content
```

```
df['stemmed_content']=df['Text'].apply(stemming)
```



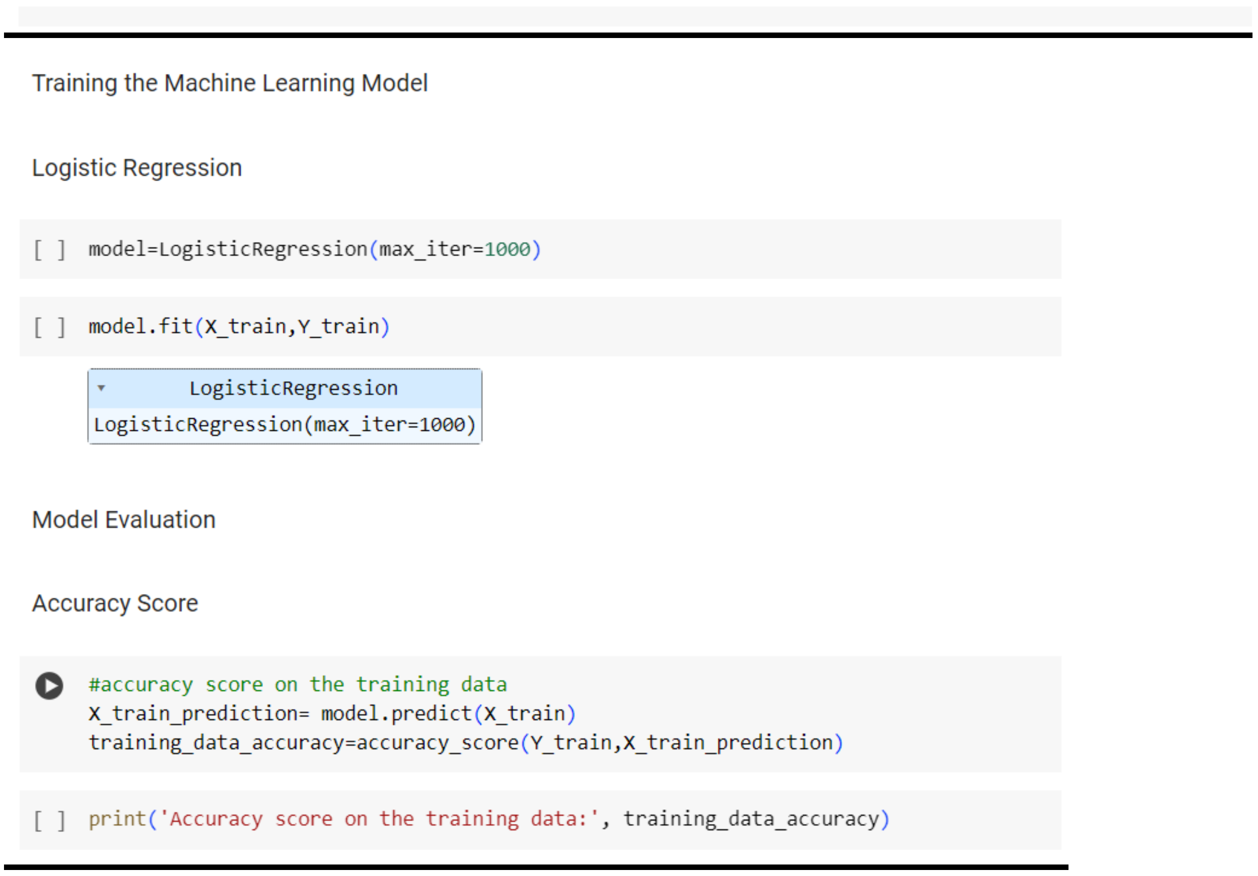
```
[ ] #separating the data and label
    X=df['stemmed_content'].values
    Y=df['Sentiment'].values
```

Splitting the data to training data and test data

```
[ ] X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size=0.2, stratify=Y,rar
```

```
[ ] print(X.shape, X_train.shape, X_test.shape)

    (747,) (597,) (150,)
```

```
    #Converting the textual data to numerical data

    vectorizer= TfidfVectorizer()

    X_train= vectorizer.fit_transform(X_train)
    X_test= vectorizer.transform(X_test)
```



Training the Machine Learning Model

Logistic Regression

```
[ ] model=LogisticRegression(max_iter=1000)
```

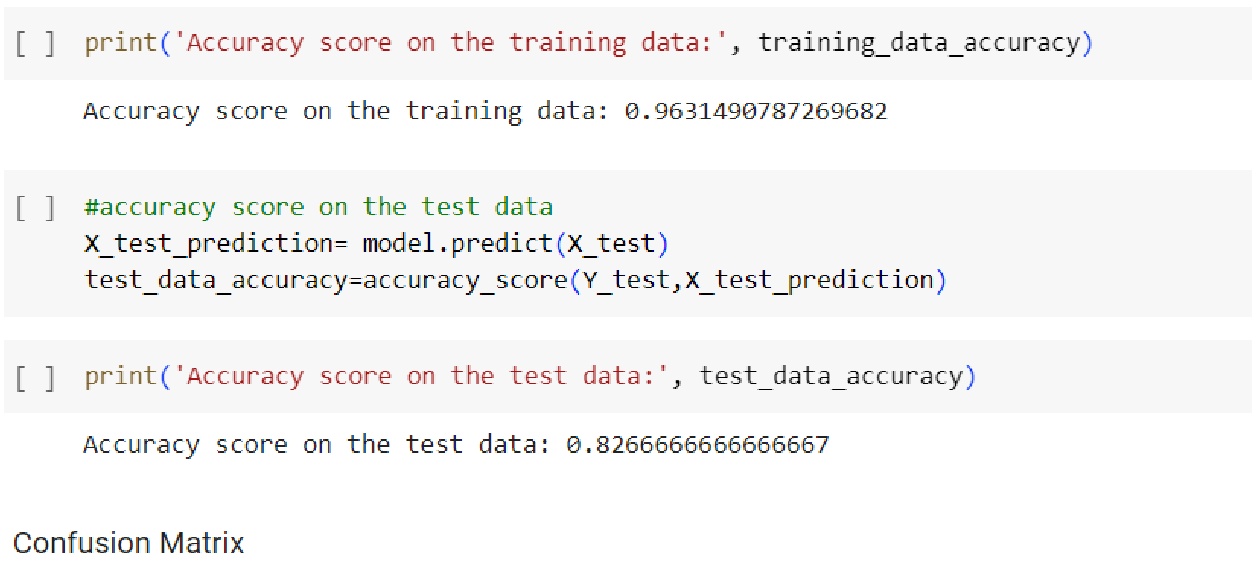
```
[ ] model.fit(X_train,Y_train)

         LogisticRegression
    LogisticRegression(max_iter=1000)
```

Model Evaluation

Accuracy Score

```
    #accuracy score on the training data
    X_train_prediction= model.predict(X_train)
    training_data_accuracy=accuracy_score(Y_train,X_train_prediction)
```

```
[ ] print('Accuracy score on the training data:', training_data_accuracy)
```

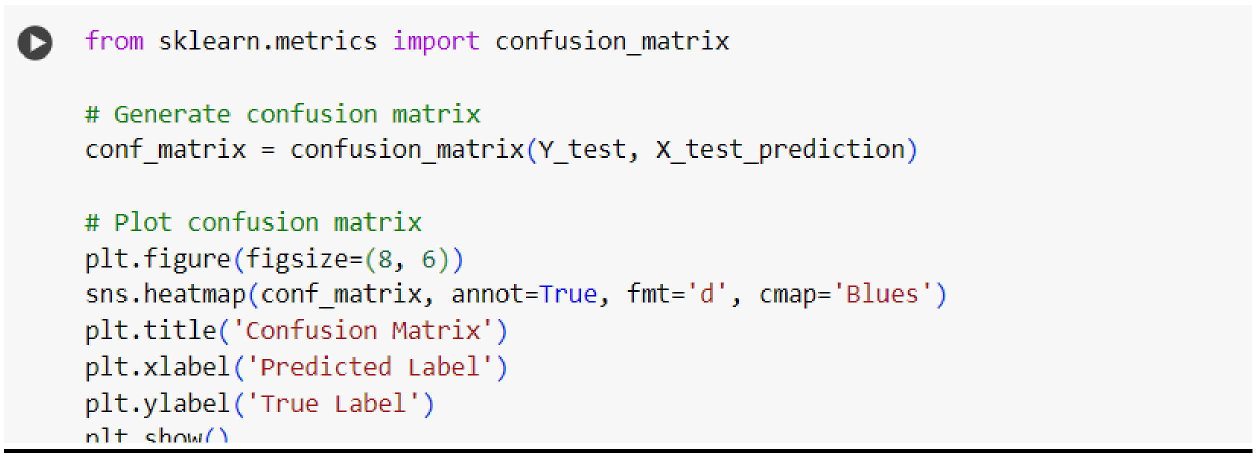


```
[ ] print('Accuracy score on the training data:', training_data_accuracy)

    Accuracy score on the training data: 0.9631490787269682
```

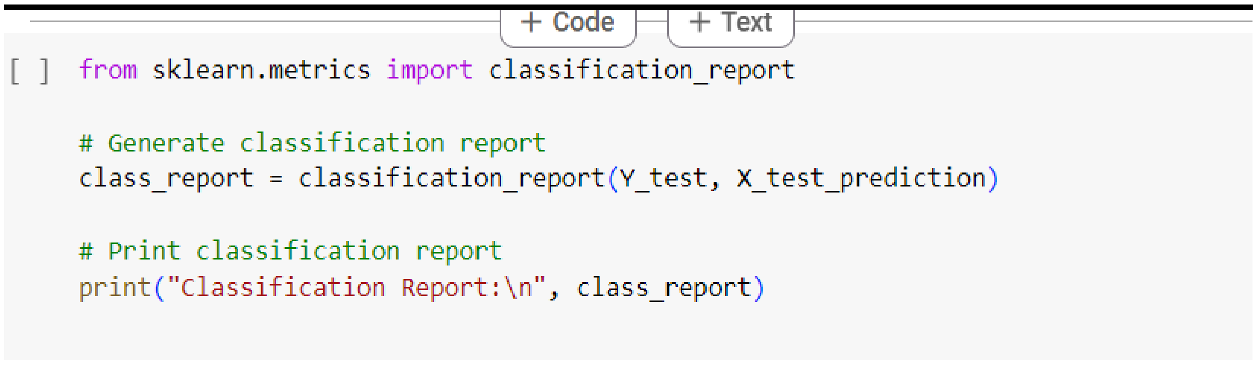
```
[ ] #accuracy score on the test data
    X_test_prediction= model.predict(X_test)
    test_data_accuracy=accuracy_score(Y_test,X_test_prediction)
```

```
[ ] print('Accuracy score on the test data:', test_data_accuracy)

    Accuracy score on the test data: 0.8266666666666667
```

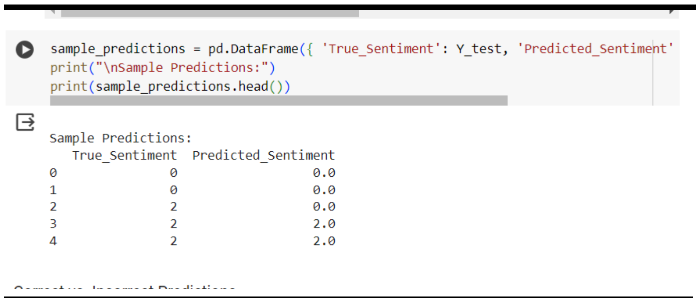
Confusion Matrix



```
from sklearn.metrics import confusion_matrix

# Generate confusion matrix
conf_matrix = confusion_matrix(Y_test, X_test_prediction)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



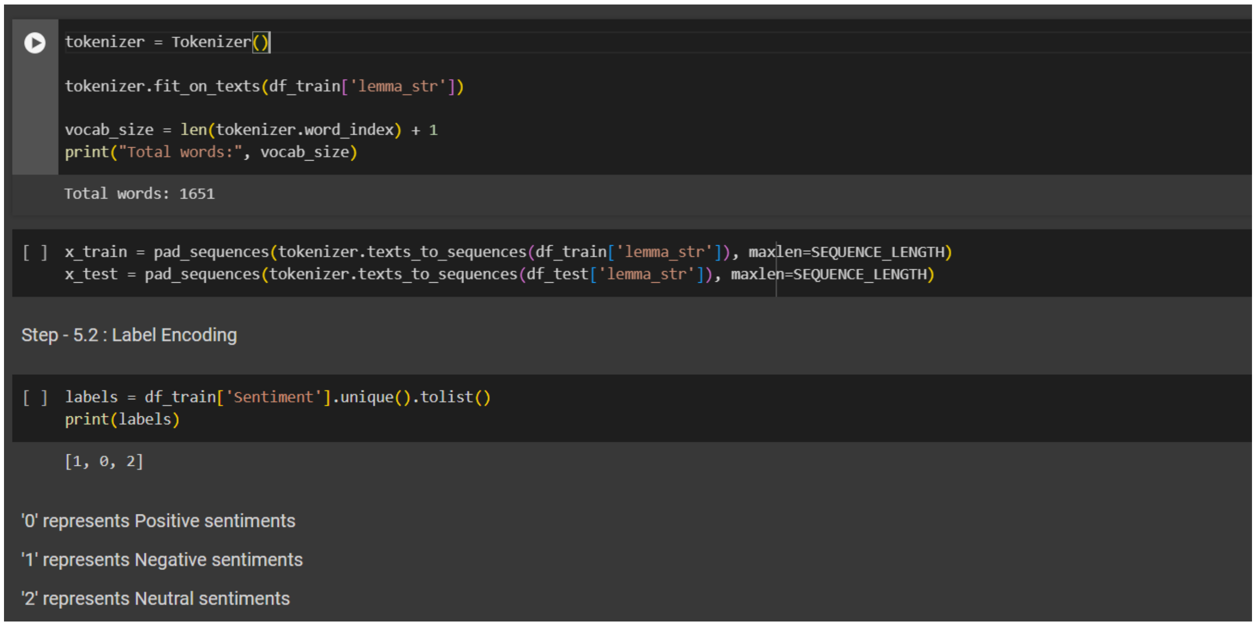
```
[ ] from sklearn.metrics import classification_report

    # Generate classification report
    class_report = classification_report(Y_test, X_test_prediction)

    # Print classification report
    print("Classification Report:\n", class_report)
```



```
sample_predictions = pd.DataFrame({ 'True_Sentiment': Y_test, 'Predicted_Sentiment'
print("\nSample Predictions:")
print(sample_predictions.head())
```

```
Sample Predictions:
   True_Sentiment  Predicted_Sentiment
0               0                  0.0
1               0                  0.0
2               2                  0.0
3               2                  2.0
4               2                  2.0
```

LSTM:



```
tokenizer = Tokenizer()

tokenizer.fit_on_texts(df_train['lemma_str'])

vocab_size = len(tokenizer.word_index) + 1
print("Total words:", vocab_size)
```

```
Total words: 1651
```

```
x_train = pad_sequences(tokenizer.texts_to_sequences(df_train['lemma_str']), maxlen=SEQUENCE_LENGTH)
x_test = pad_sequences(tokenizer.texts_to_sequences(df_test['lemma_str']), maxlen=SEQUENCE_LENGTH)
```

Step - 5.2 : Label Encoding

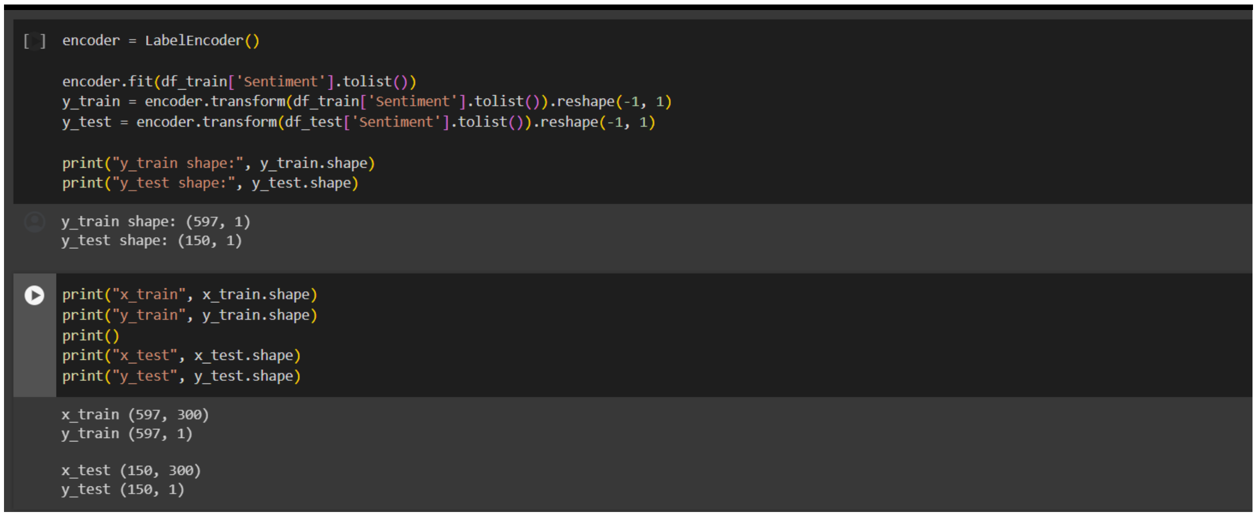
```
labels = df_train['Sentiment'].unique().tolist()
print(labels)
```

```
[1, 0, 2]
```

'0' represents Positive sentiments

'1' represents Negative sentiments

'2' represents Neutral sentiments

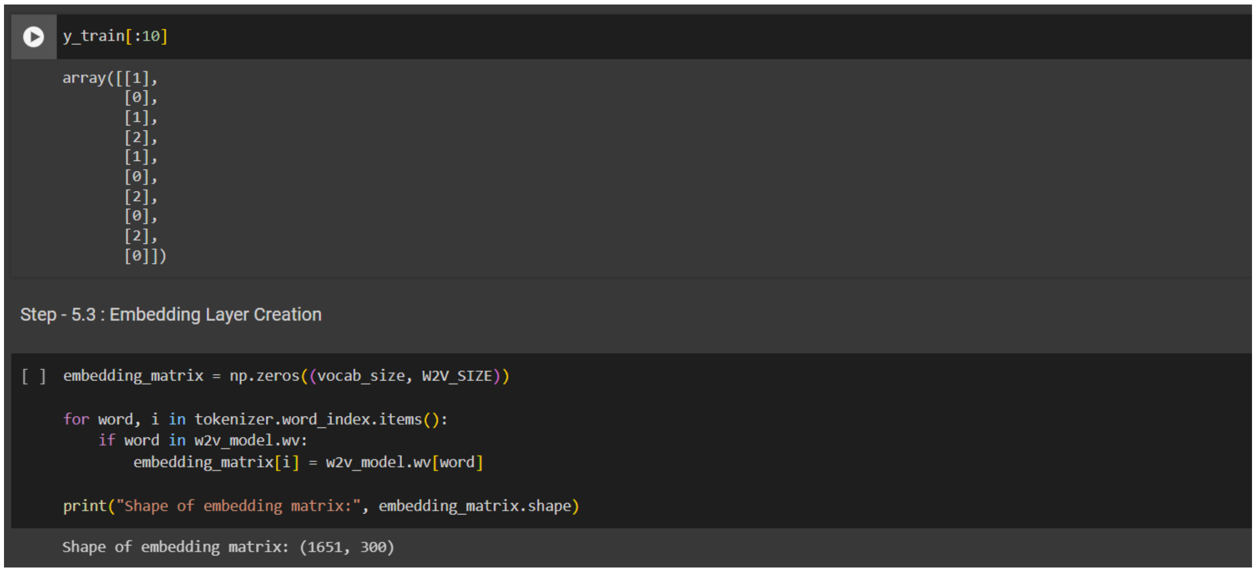


```
encoder = LabelEncoder()

encoder.fit(df_train['Sentiment'].tolist())
y_train = encoder.transform(df_train['Sentiment'].tolist()).reshape(-1, 1)
y_test = encoder.transform(df_test['Sentiment'].tolist()).reshape(-1, 1)

print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
y_train shape: (597, 1)
y_test shape: (150, 1)
```

```
print("x_train", x_train.shape)
print("y_train", y_train.shape)
print()
print("x_test", x_test.shape)
print("y_test", y_test.shape)
```

```
x_train (597, 300)
y_train (597, 1)

x_test (150, 300)
y_test (150, 1)
```



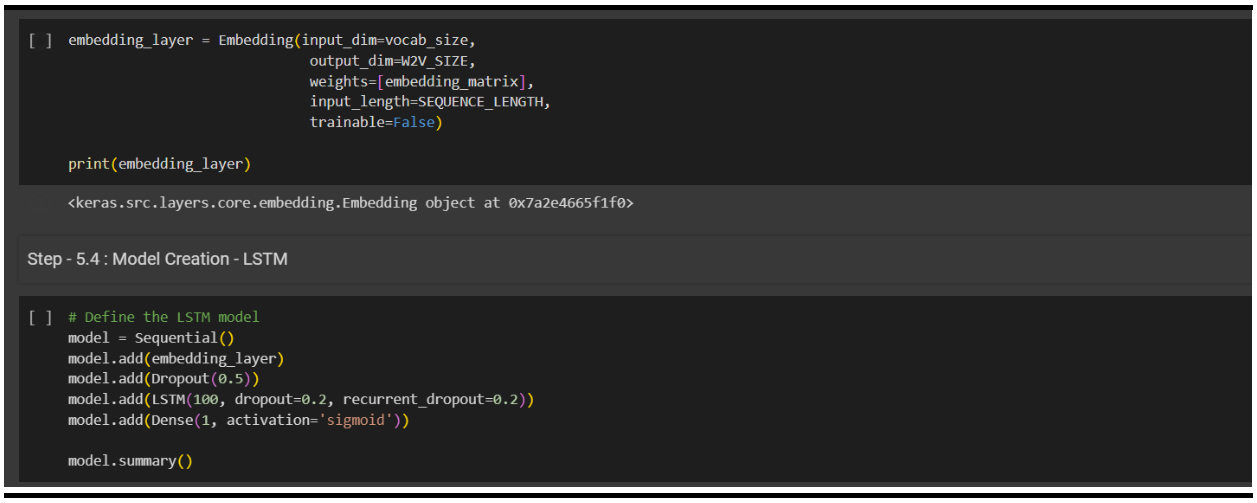
```
y_train[:10]
```

```
array([[1],
       [0],
       [1],
       [2],
       [1],
       [0],
       [2],
       [0],
       [2],
       [0]])
```

Step - 5.3 : Embedding Layer Creation

```
[ ] embedding_matrix = np.zeros((vocab_size, W2V_SIZE))

    for word, i in tokenizer.word_index.items():
        if word in w2v_model.wv:
            embedding_matrix[i] = w2v_model.wv[word]

    print("Shape of embedding matrix:", embedding_matrix.shape)
```

```
Shape of embedding matrix: (1651, 300)
```

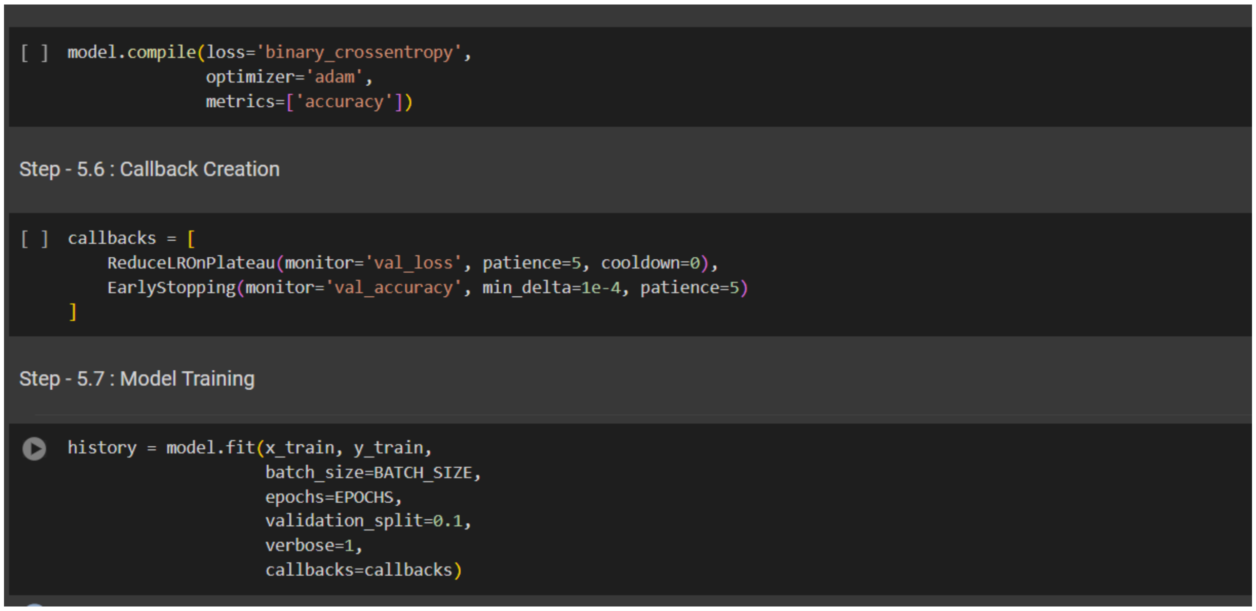


```
[ ] embedding_layer = Embedding(input_dim=vocab_size,
                                output_dim=W2V_SIZE,
                                weights=[embedding_matrix],
                                input_length=SEQUENCE_LENGTH,
                                trainable=False)

    print(embedding_layer)
```

```
<keras.src.layers.core.embedding.Embedding object at 0x7a2e4665f1f0>
```

Step - 5.4 : Model Creation - LSTM

```
[ ] # Define the LSTM model
    model = Sequential()
    model.add(embedding_layer)
    model.add(Dropout(0.5))
    model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(1, activation='sigmoid'))

    model.summary()
```



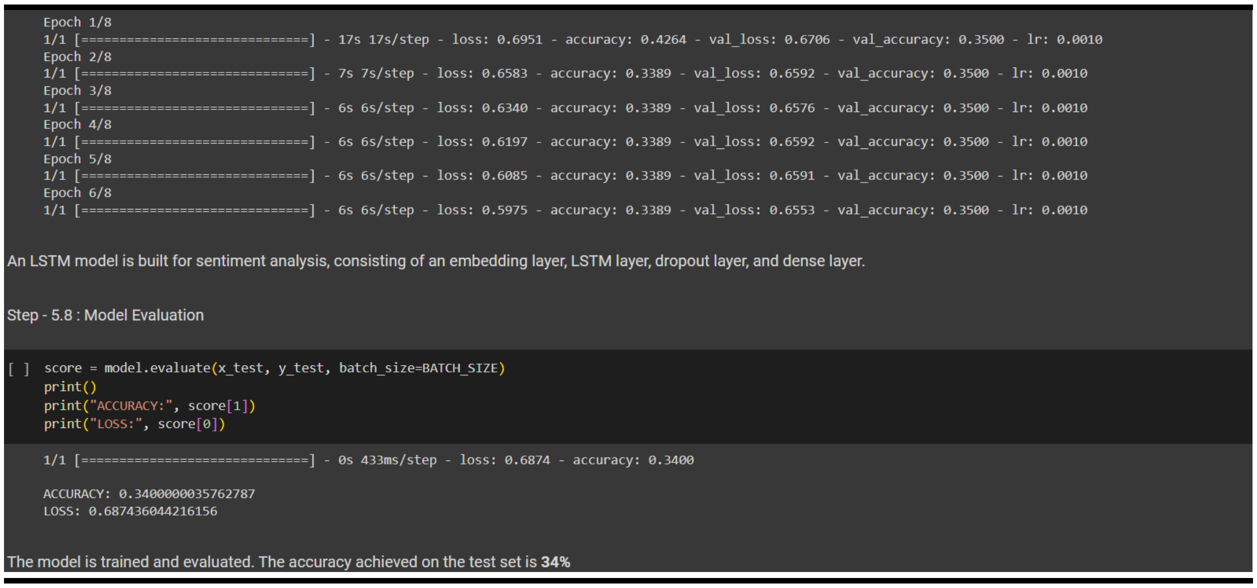
```
[ ] model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

Step - 5.6 : Callback Creation

```
[ ] callbacks = [
        ReduceLROnPlateau(monitor='val_loss', patience=5, cooldown=0),
        EarlyStopping(monitor='val_accuracy', min_delta=1e-4, patience=5)
    ]
```

Step - 5.7 : Model Training

```
history = model.fit(x_train, y_train,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_split=0.1,
                    verbose=1,
                    callbacks=callbacks)
```



```
Epoch 1/8
1/1 [==============================] - 17s 17s/step - loss: 0.6951 - accuracy: 0.4264 - val_loss: 0.6706 - val_accuracy: 0.3500 - lr: 0.0010
Epoch 2/8
1/1 [==============================] - 7s 7s/step - loss: 0.6583 - accuracy: 0.3389 - val_loss: 0.6592 - val_accuracy: 0.3500 - lr: 0.0010
Epoch 3/8
1/1 [==============================] - 6s 6s/step - loss: 0.6340 - accuracy: 0.3389 - val_loss: 0.6576 - val_accuracy: 0.3500 - lr: 0.0010
Epoch 4/8
1/1 [==============================] - 6s 6s/step - loss: 0.6197 - accuracy: 0.3389 - val_loss: 0.6592 - val_accuracy: 0.3500 - lr: 0.0010
Epoch 5/8
1/1 [==============================] - 6s 6s/step - loss: 0.6085 - accuracy: 0.3389 - val_loss: 0.6591 - val_accuracy: 0.3500 - lr: 0.0010
Epoch 6/8
1/1 [==============================] - 6s 6s/step - loss: 0.5975 - accuracy: 0.3389 - val_loss: 0.6553 - val_accuracy: 0.3500 - lr: 0.0010
```

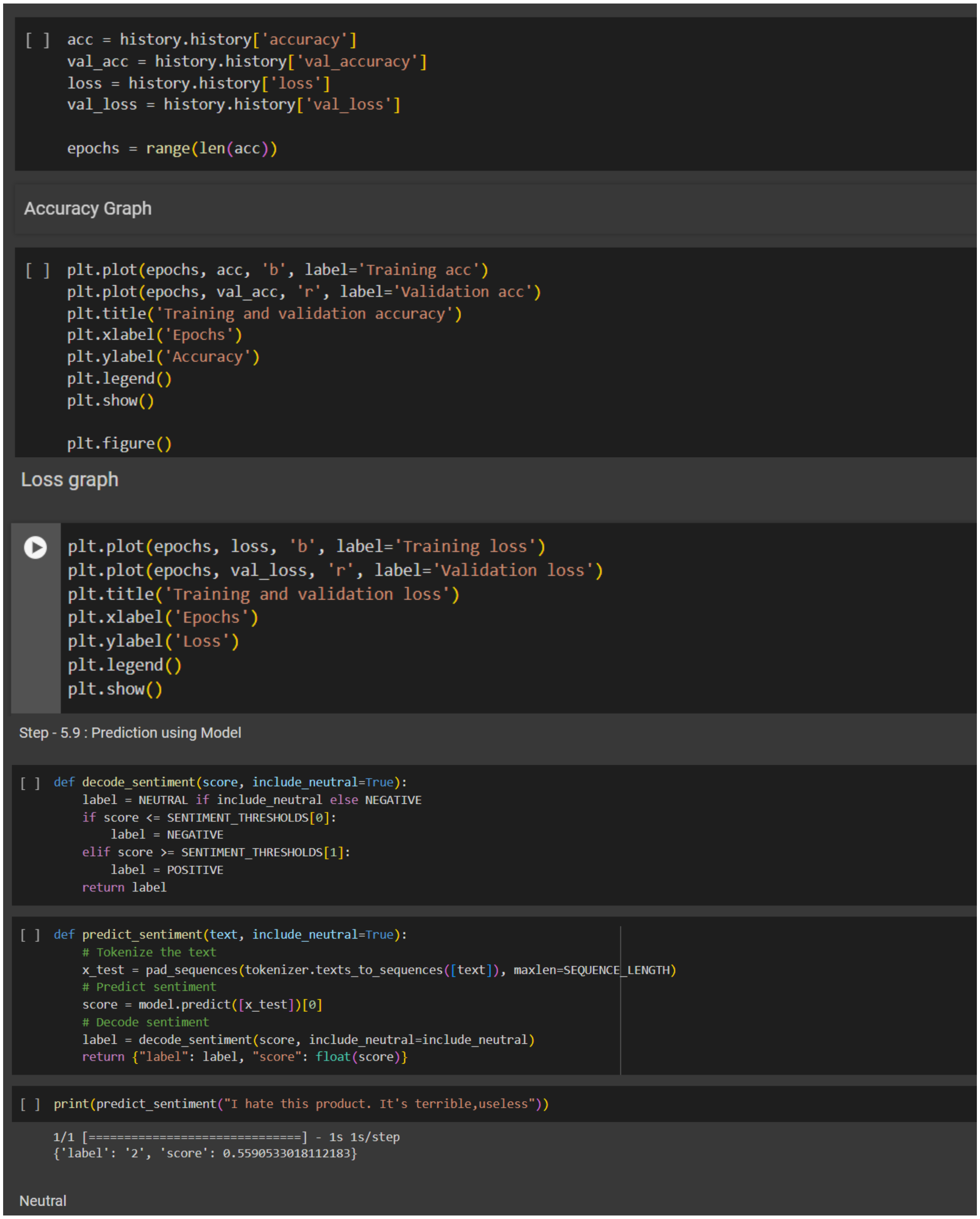
An LSTM model is built for sentiment analysis, consisting of an embedding layer, LSTM layer, dropout layer, and dense layer.

Step - 5.8 : Model Evaluation

```
[ ] score = model.evaluate(x_test, y_test, batch_size=BATCH_SIZE)
    print()
    print("ACCURACY:", score[1])
    print("LOSS:", score[0])
```

```
1/1 [==============================] - 0s 433ms/step - loss: 0.6874 - accuracy: 0.3400

ACCURACY: 0.3400000035762787
LOSS: 0.687436044216156
```

The model is trained and evaluated. The accuracy achieved on the test set is **34%**



```
[ ] acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(len(acc))
```

Accuracy Graph

```
[ ] plt.plot(epochs, acc, 'b', label='Training acc')
    plt.plot(epochs, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

    plt.figure()
```

Loss graph

```
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

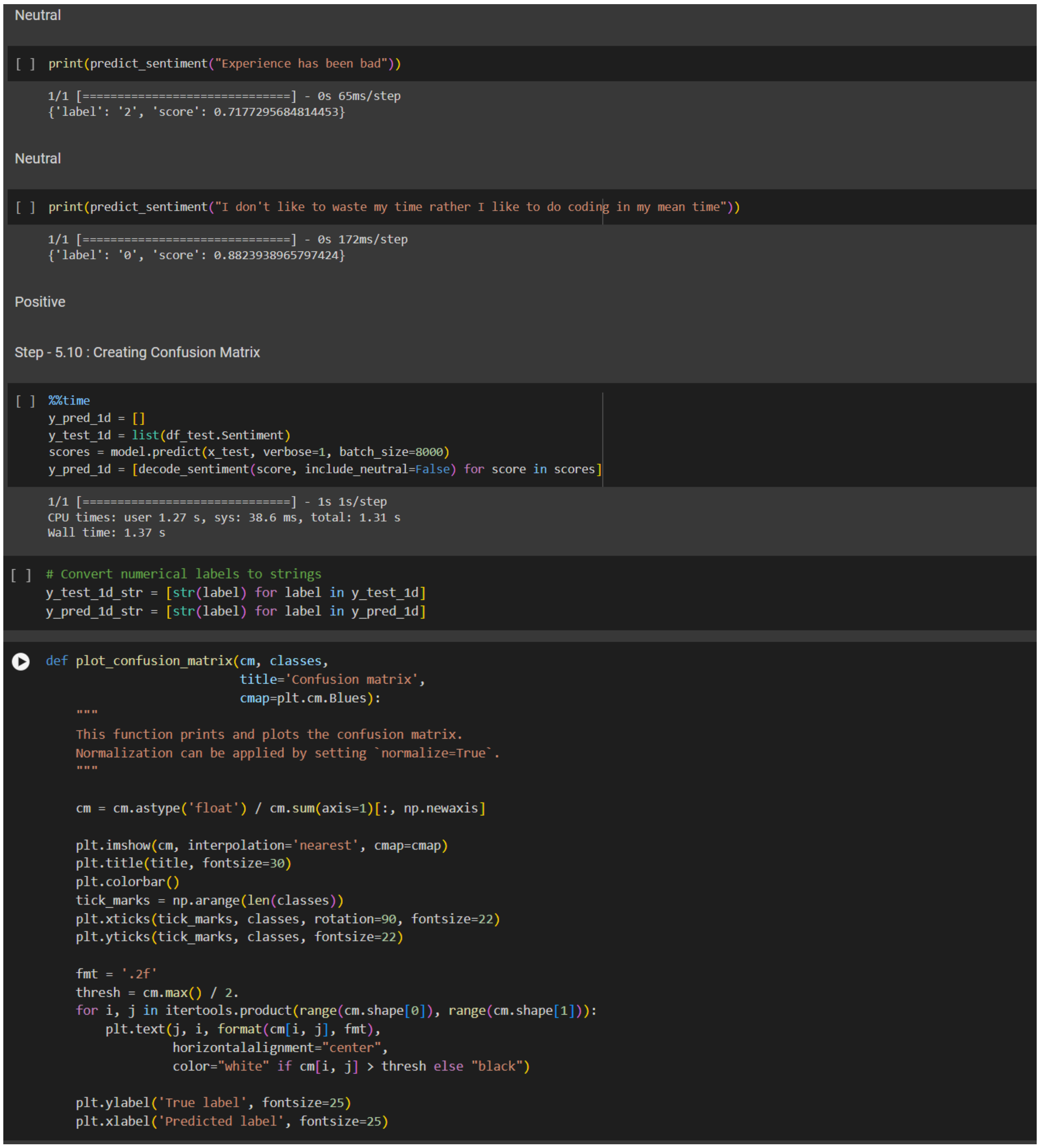
Step - 5.9 : Prediction using Model

```
[ ] def decode_sentiment(score, include_neutral=True):
        label = NEUTRAL if include_neutral else NEGATIVE
        if score <= SENTIMENT_THRESHOLDS[0]:
            label = NEGATIVE
        elif score >= SENTIMENT_THRESHOLDS[1]:
            label = POSITIVE
        return label
```

```
[ ] def predict_sentiment(text, include_neutral=True):
        # Tokenize the text
        x_test = pad_sequences(tokenizer.texts_to_sequences([text]), maxlen=SEQUENCE_LENGTH)
        # Predict sentiment
        score = model.predict([x_test])[0]
        # Decode sentiment
        label = decode_sentiment(score, include_neutral=include_neutral)
        return {"label": label, "score": float(score)}
```

```
[ ] print(predict_sentiment("I hate this product. It's terrible,useless"))

    1/1 [==============================] - 1s 1s/step
    {'label': '2', 'score': 0.5590533018112183}
```

Neutral



Neutral

```
[ ] print(predict_sentiment("Experience has been bad"))

    1/1 [==============================] - 0s 65ms/step
    {'label': '2', 'score': 0.7177295684814453}
```

Neutral

```
[ ] print(predict_sentiment("I don't like to waste my time rather I like to do coding in my mean time"))

    1/1 [==============================] - 0s 172ms/step
    {'label': '0', 'score': 0.8823938965797424}
```

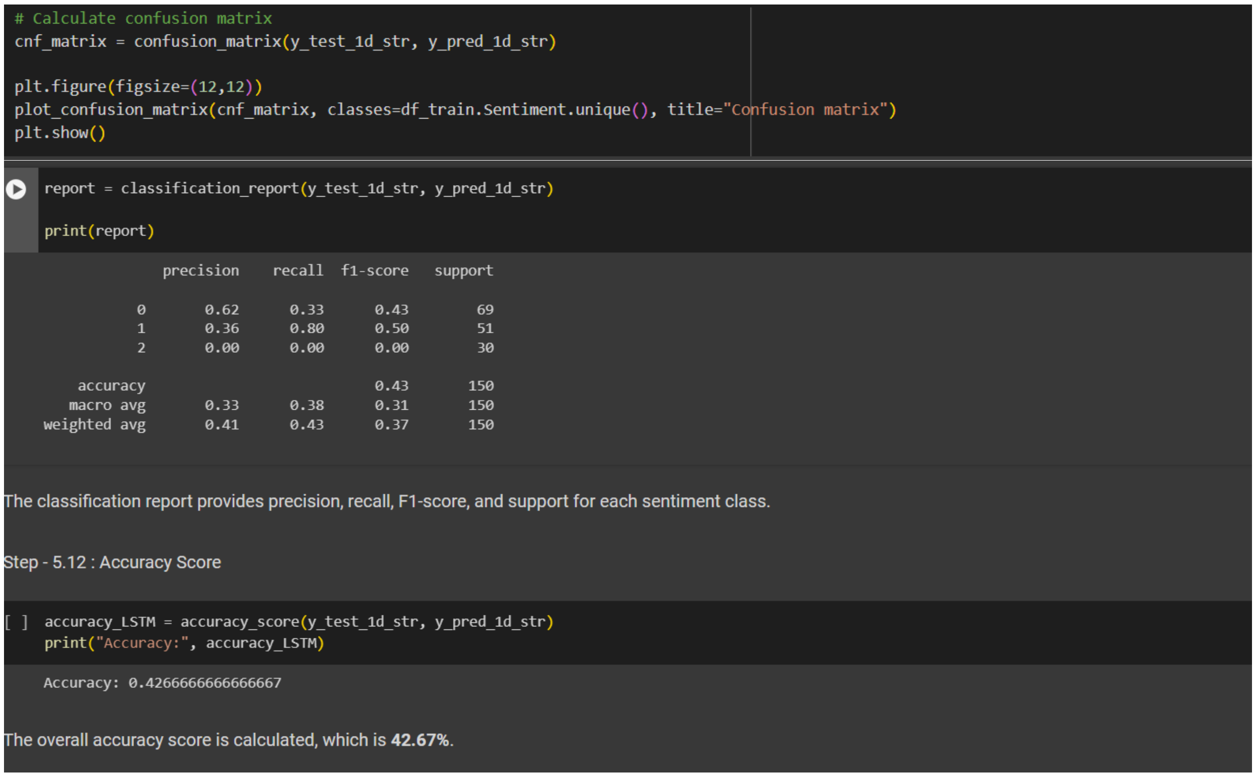
Positive

Step - 5.10 : Creating Confusion Matrix

```
[ ] %%time
    y_pred_1d = []
    y_test_1d = list(df_test.Sentiment)
    scores = model.predict(x_test, verbose=1, batch_size=8000)
    y_pred_1d = [decode_sentiment(score, include_neutral=False) for score in scores]

    1/1 [==============================] - 1s 1s/step
    CPU times: user 1.27 s, sys: 38.6 ms, total: 1.31 s
    Wall time: 1.37 s
```

```
[ ] # Convert numerical labels to strings
    y_test_1d_str = [str(label) for label in y_test_1d]
    y_pred_1d_str = [str(label) for label in y_pred_1d]
```

```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """

    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=30)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90, fontsize=22)
    plt.yticks(tick_marks, classes, fontsize=22)

    fmt = '.2f'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label', fontsize=25)
    plt.xlabel('Predicted label', fontsize=25)
```



```
# Calculate confusion matrix
cnf_matrix = confusion_matrix(y_test_1d_str, y_pred_1d_str)

plt.figure(figsize=(12,12))
plot_confusion_matrix(cnf_matrix, classes=df_train.Sentiment.unique(), title="Confusion matrix")
plt.show()
```

```
report = classification_report(y_test_1d_str, y_pred_1d_str)

print(report)
```

```
              precision    recall  f1-score   support

           0       0.62      0.33      0.43        69
           1       0.36      0.80      0.50        51
           2       0.00      0.00      0.00        30

    accuracy                           0.43       150
   macro avg       0.33      0.38      0.31       150
weighted avg       0.41      0.43      0.37       150
```

The classification report provides precision, recall, F1-score, and support for each sentiment class.

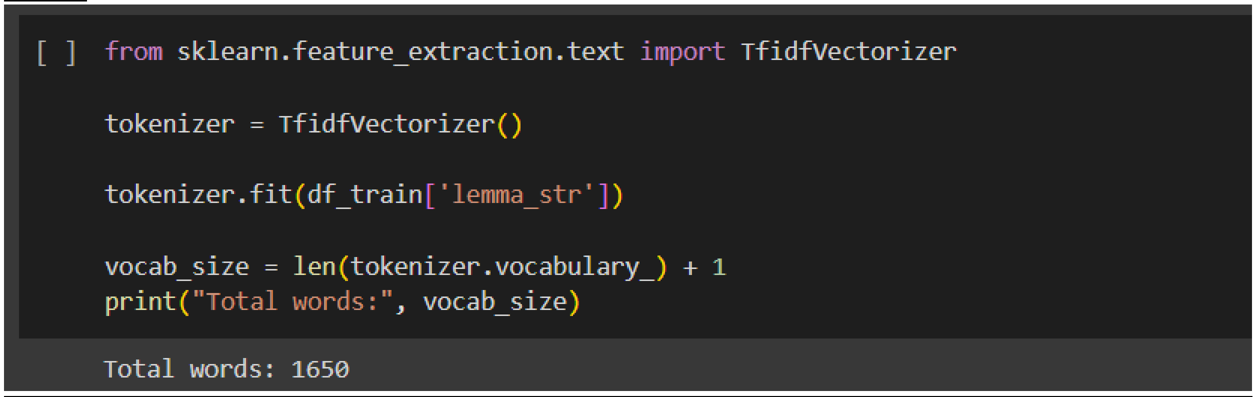
Step - 5.12 : Accuracy Score

```
[ ] accuracy_LSTM = accuracy_score(y_test_1d_str, y_pred_1d_str)
    print("Accuracy:", accuracy_LSTM)
```

```
Accuracy: 0.4266666666666667
```

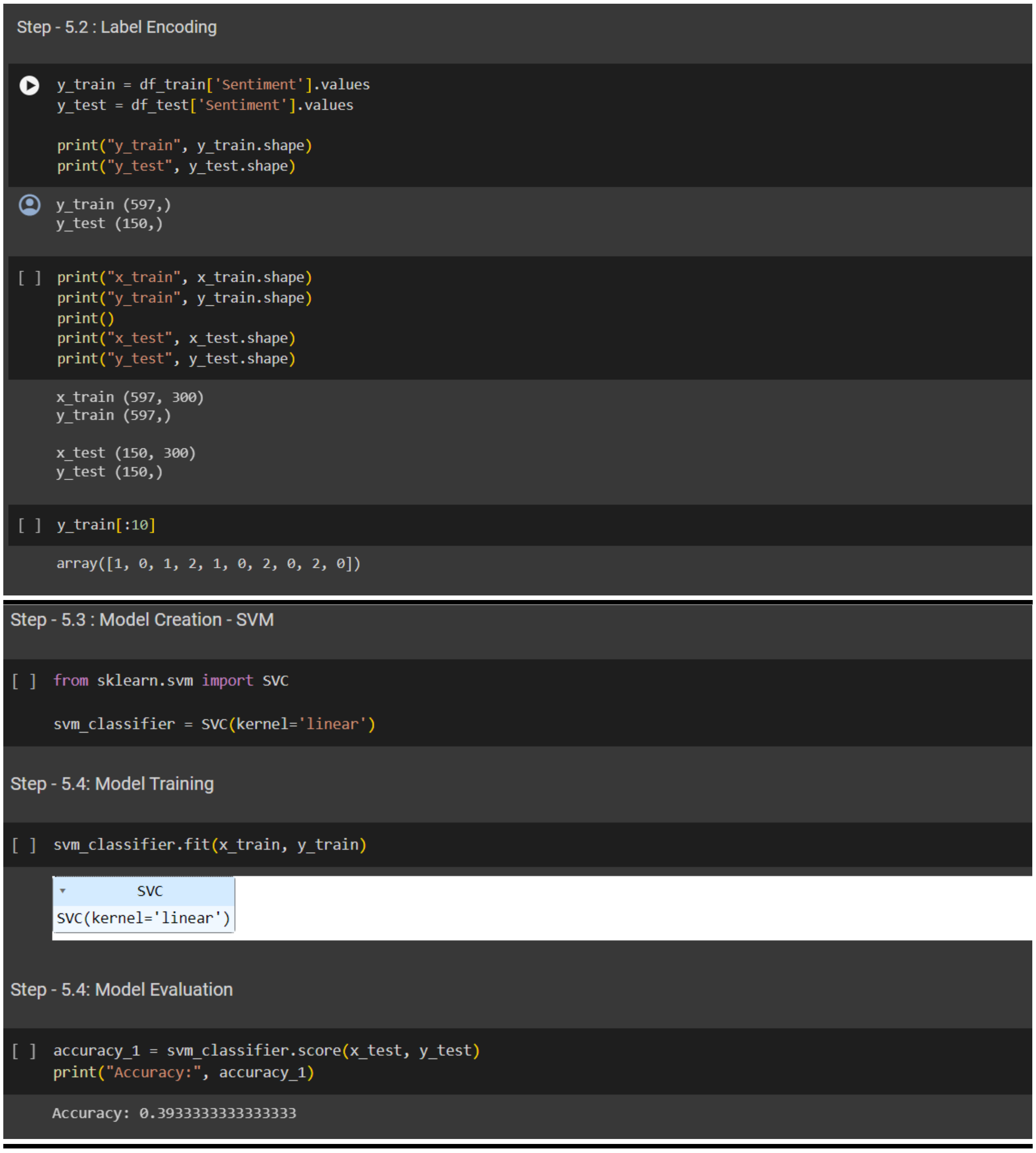
The overall accuracy score is calculated, which is **42.67%**.

SVM:



```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer

    tokenizer = TfidfVectorizer()

    tokenizer.fit(df_train['lemma_str'])

    vocab_size = len(tokenizer.vocabulary_) + 1
    print("Total words:", vocab_size)
```

```
Total words: 1650
```


Step - 5.2 : Label Encoding

```
y_train = df_train['Sentiment'].values
y_test = df_test['Sentiment'].values

print("y_train", y_train.shape)
print("y_test", y_test.shape)
```

```
y_train (597,)
y_test (150,)
```

```
[ ] print("x_train", x_train.shape)
    print("y_train", y_train.shape)
    print()
    print("x_test", x_test.shape)
    print("y_test", y_test.shape)
```

```
x_train (597, 300)
y_train (597,)

x_test (150, 300)
y_test (150,)
```

```
[ ] y_train[:10]
```

```
array([1, 0, 1, 2, 1, 0, 2, 0, 2, 0])
```

Step - 5.3 : Model Creation - SVM

```
[ ] from sklearn.svm import SVC

    svm_classifier = SVC(kernel='linear')
```

Step - 5.4: Model Training

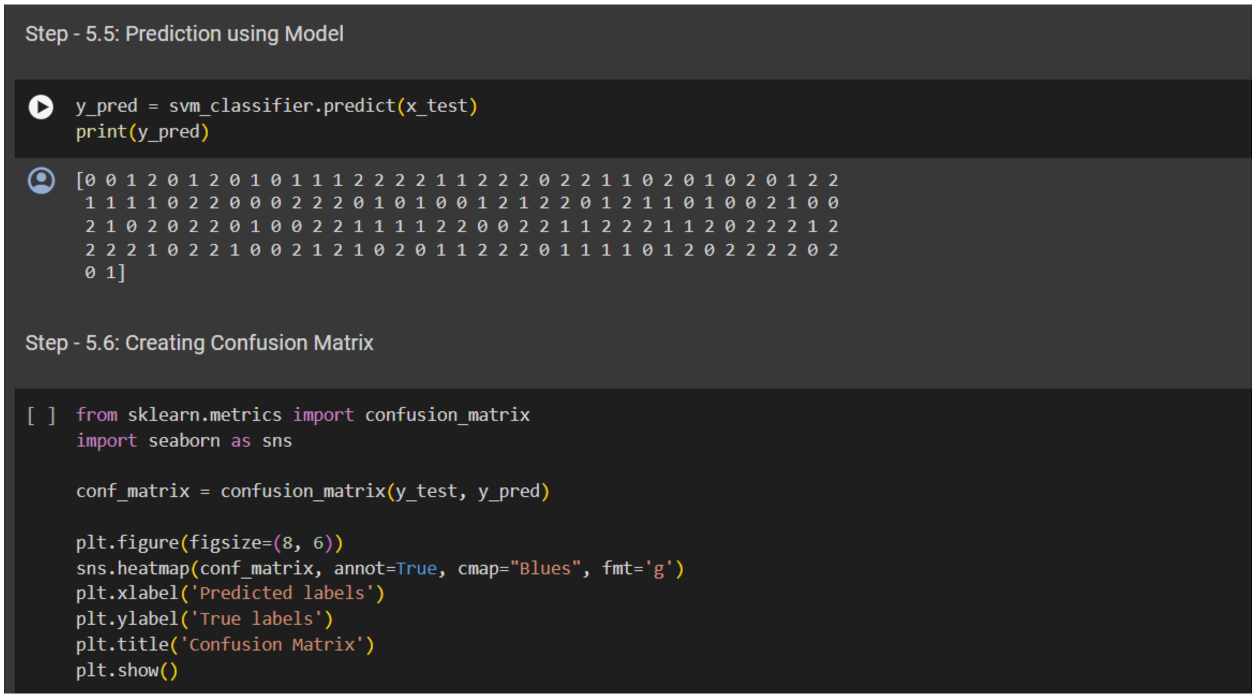
```
[ ] svm_classifier.fit(x_train, y_train)
```

```
SVC
SVC(kernel='linear')
```

Step - 5.4: Model Evaluation

```
[ ] accuracy_1 = svm_classifier.score(x_test, y_test)
    print("Accuracy:", accuracy_1)
```

```
Accuracy: 0.3933333333333333
```

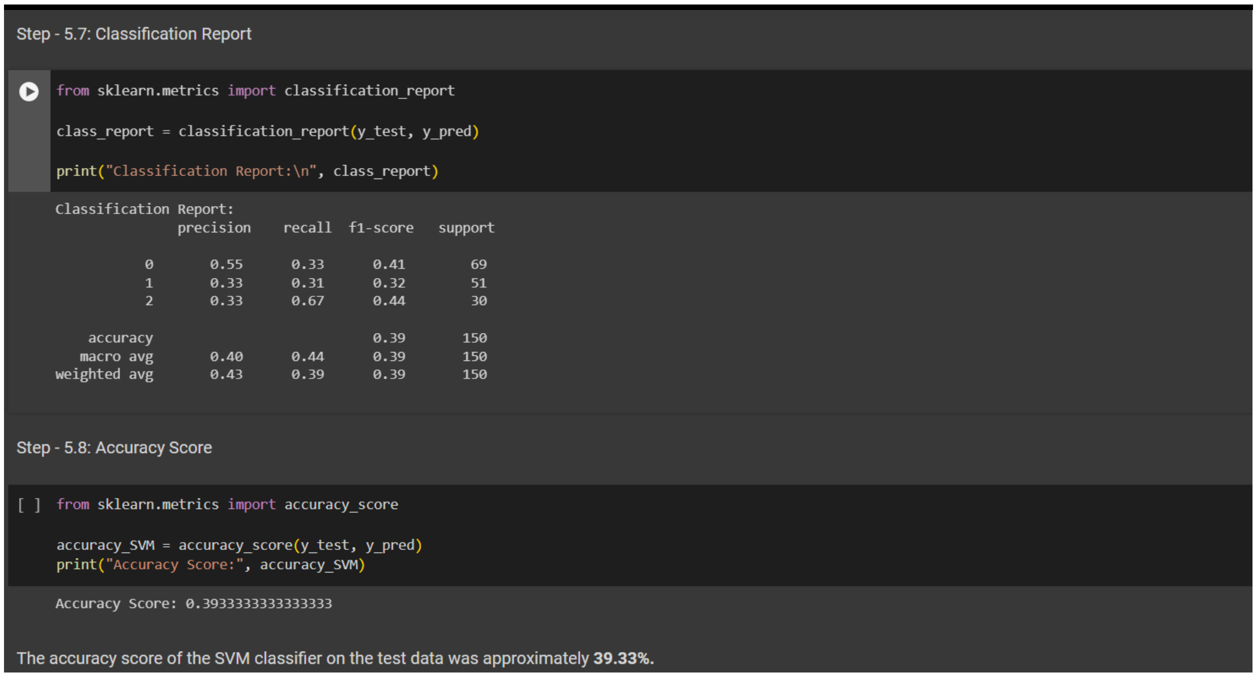

Step - 5.5: Prediction using Model

```
y_pred = svm_classifier.predict(x_test)
print(y_pred)
```

```
[0 0 1 2 0 1 2 0 1 0 1 1 1 2 2 2 2 1 1 2 2 2 0 2 2 1 1 0 2 0 1 0 2 0 1 2 2
 1 1 1 1 0 2 2 0 0 0 2 2 2 0 1 0 1 0 0 1 2 1 2 2 0 1 2 1 1 0 1 0 0 2 1 0 0
 2 1 0 2 0 2 2 0 1 0 0 2 2 1 1 1 1 2 2 0 0 2 2 1 1 2 2 2 1 1 2 0 2 2 2 1 2
 2 2 2 1 0 2 2 1 0 0 2 1 2 1 0 2 0 1 1 2 2 2 0 1 1 1 1 0 1 2 0 2 2 2 2 0 2
 0 1]
```

Step - 5.6: Creating Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```


Step - 5.7: Classification Report

```
from sklearn.metrics import classification_report

class_report = classification_report(y_test, y_pred)

print("Classification Report:\n", class_report)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.55      0.33      0.41        69
           1       0.33      0.31      0.32        51
           2       0.33      0.67      0.44        30

    accuracy                           0.39       150
   macro avg       0.40      0.44      0.39       150
weighted avg       0.43      0.39      0.39       150
```

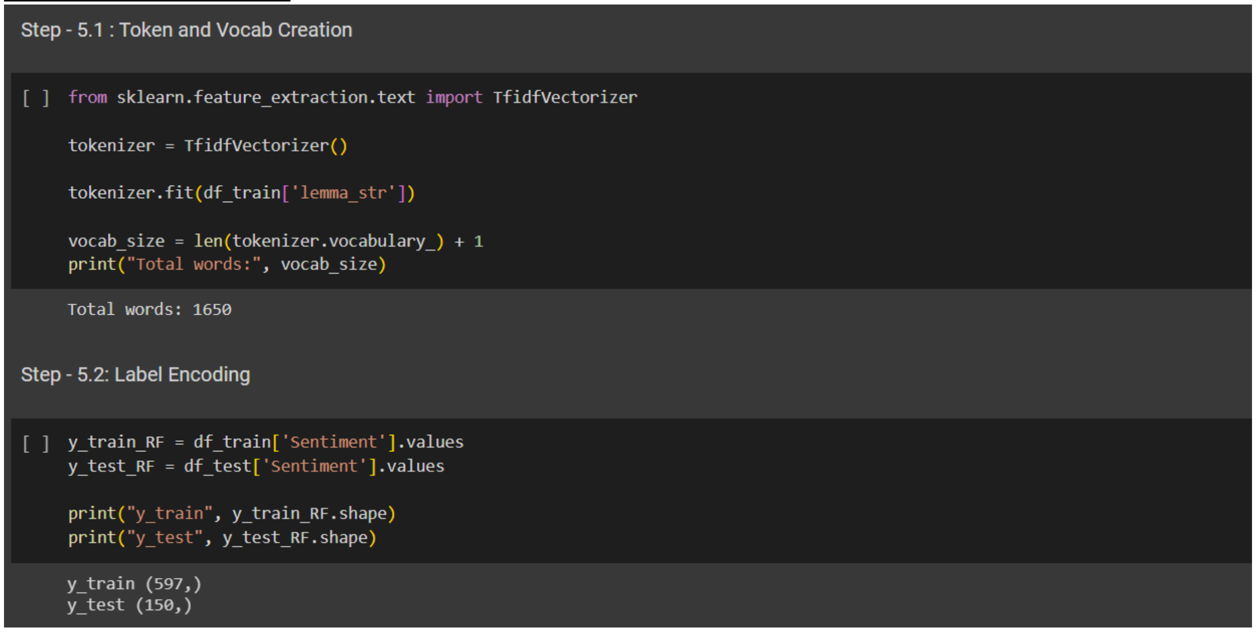
Step - 5.8: Accuracy Score

```
from sklearn.metrics import accuracy_score

accuracy_SVM = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy_SVM)
```

```
Accuracy Score: 0.3933333333333333
```

The accuracy score of the SVM classifier on the test data was approximately **39.33%.**

RANDOM FOREST:



Step - 5.1 : Token and Vocab Creation

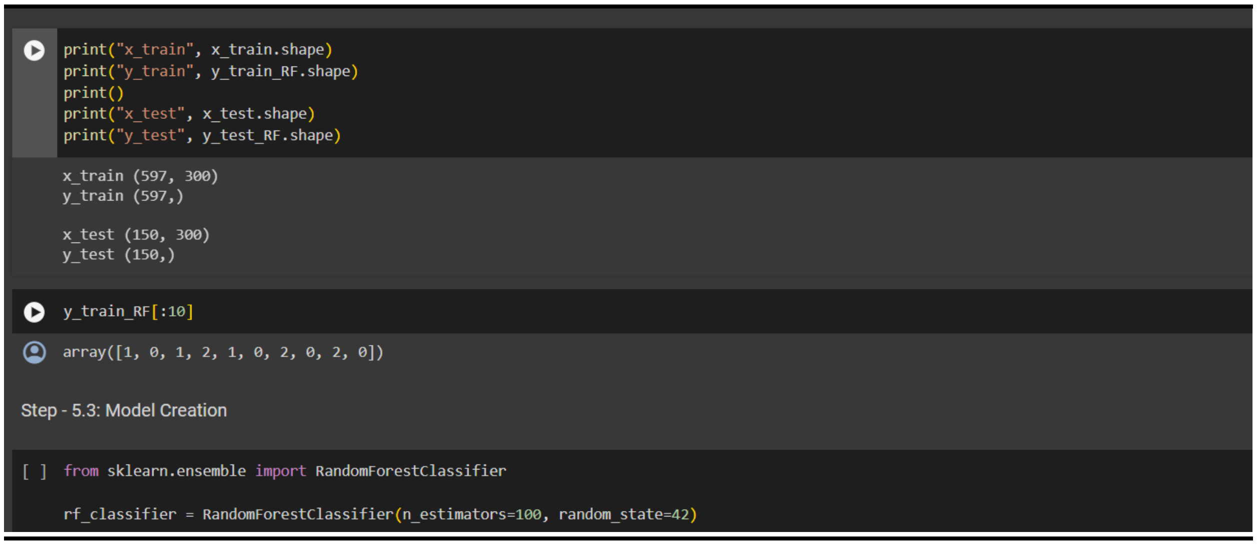
```
[ ]  from sklearn.feature_extraction.text import TfidfVectorizer

     tokenizer = TfidfVectorizer()

     tokenizer.fit(df_train['lemma_str'])

     vocab_size = len(tokenizer.vocabulary_) + 1
     print("Total words:", vocab_size)
```

```
     Total words: 1650
```

Step - 5.2: Label Encoding

```
[ ]  y_train_RF = df_train['Sentiment'].values
     y_test_RF = df_test['Sentiment'].values

     print("y_train", y_train_RF.shape)
     print("y_test", y_test_RF.shape)
```

```
     y_train (597,)
     y_test (150,)
```



```
print("x_train", x_train.shape)
print("y_train", y_train_RF.shape)
print()
print("x_test", x_test.shape)
print("y_test", y_test_RF.shape)
```

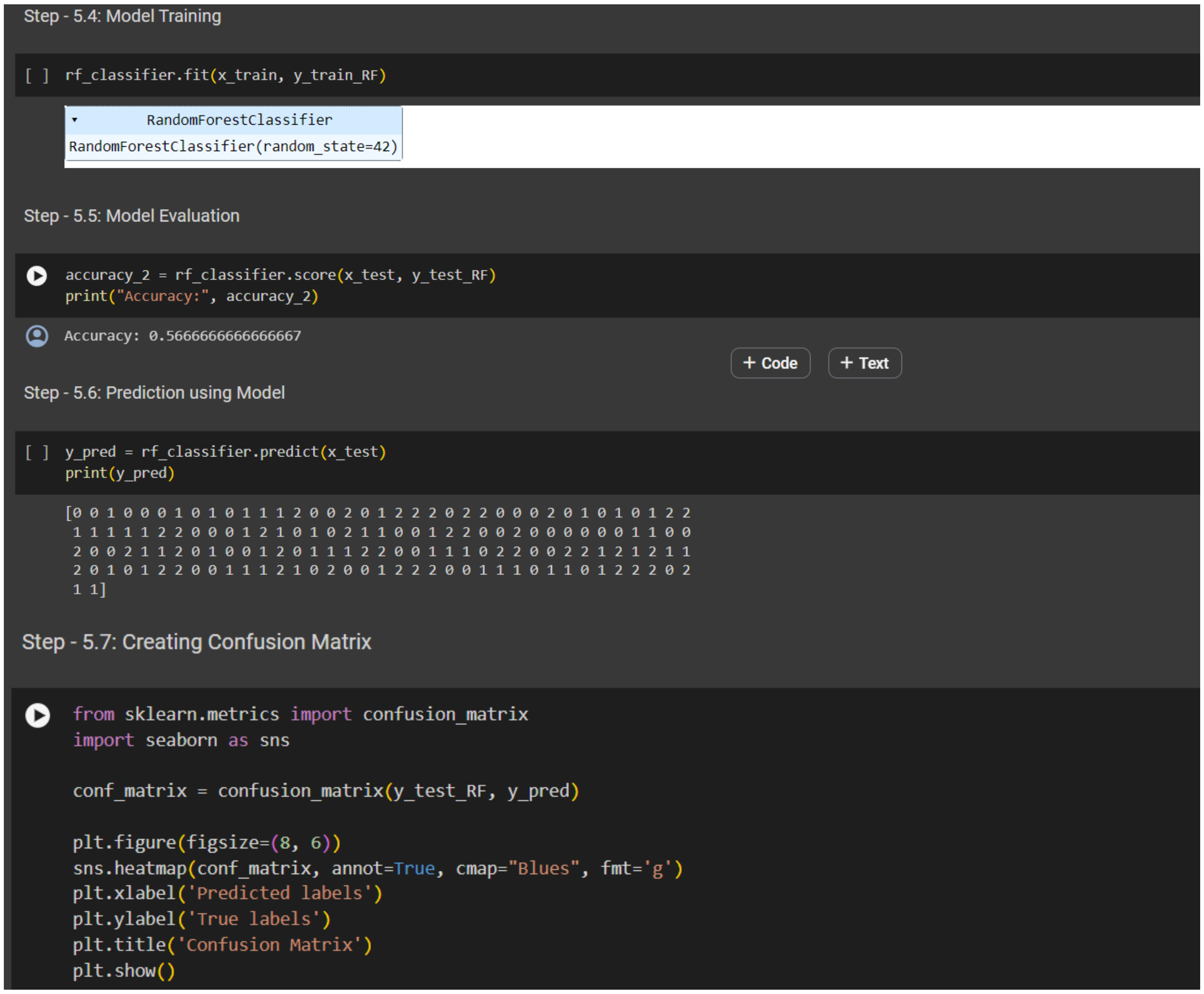
```
x_train (597, 300)
y_train (597,)

x_test (150, 300)
y_test (150,)
```

```
y_train_RF[:10]
```

```
array([1, 0, 1, 2, 1, 0, 2, 0, 2, 0])
```

Step - 5.3: Model Creation

```
[ ]  from sklearn.ensemble import RandomForestClassifier

     rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```



Step - 5.4: Model Training

```
[ ] rf_classifier.fit(x_train, y_train_RF)
```

```
▾        RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Step - 5.5: Model Evaluation

```
accuracy_2 = rf_classifier.score(x_test, y_test_RF)
print("Accuracy:", accuracy_2)
```

```
Accuracy: 0.5666666666666667
```

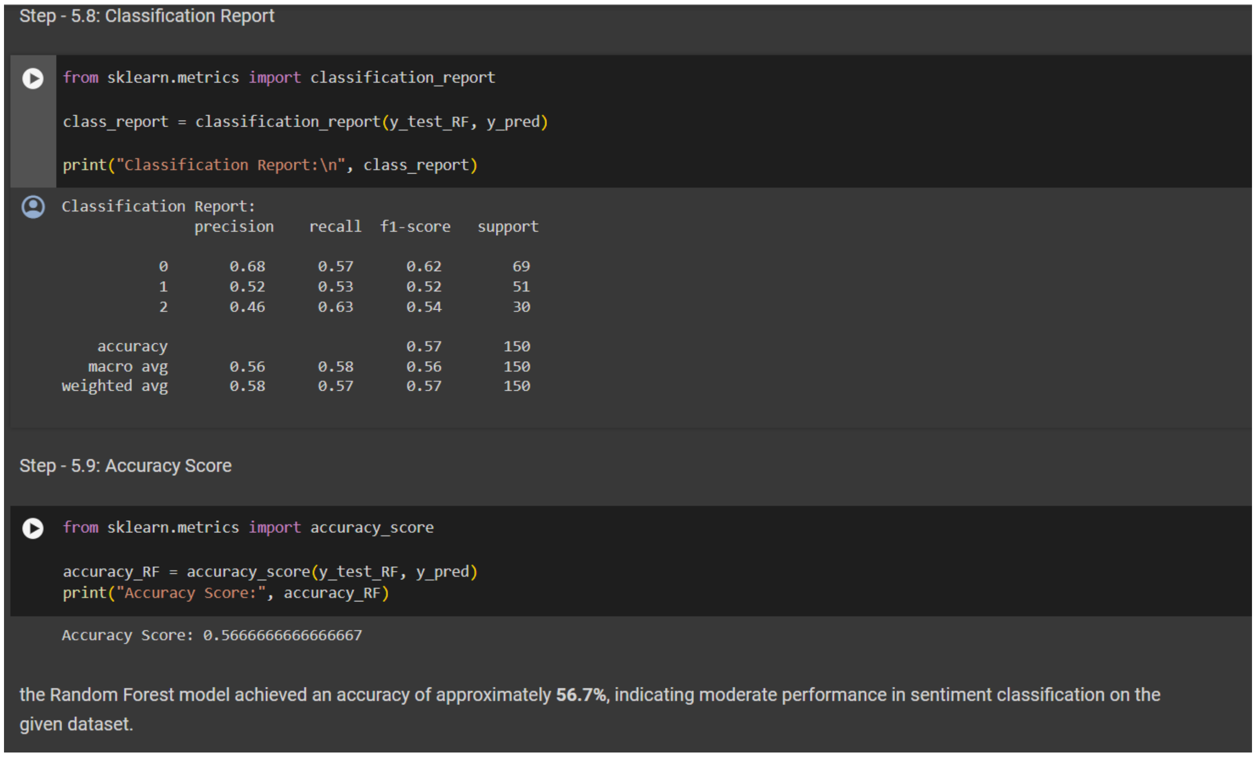
Step - 5.6: Prediction using Model

```
[ ] y_pred = rf_classifier.predict(x_test)
    print(y_pred)
```

```
[0 0 1 0 0 0 1 0 1 0 1 1 1 2 0 0 2 0 1 2 2 2 0 2 2 0 0 0 2 0 1 0 1 0 1 2 2
 1 1 1 1 1 2 2 0 0 0 1 2 1 0 1 0 2 1 1 0 0 1 2 2 0 0 2 0 0 0 0 0 0 1 1 0 0
 2 0 0 2 1 1 2 0 1 0 0 1 2 0 1 1 1 2 2 0 0 1 1 1 0 2 2 0 0 2 2 1 2 1 2 1 1
 2 0 1 0 1 2 2 0 0 1 1 1 2 1 0 2 0 0 1 2 2 2 0 0 1 1 1 0 1 1 0 1 2 2 2 0 2
 1 1]
```

Step - 5.7: Creating Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_matrix = confusion_matrix(y_test_RF, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```


Step - 5.8: Classification Report

```
from sklearn.metrics import classification_report

class_report = classification_report(y_test_RF, y_pred)

print("Classification Report:\n", class_report)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.68      0.57      0.62        69
           1       0.52      0.53      0.52        51
           2       0.46      0.63      0.54        30

    accuracy                           0.57       150
   macro avg       0.56      0.58      0.56       150
weighted avg       0.58      0.57      0.57       150
```

Step - 5.9: Accuracy Score

```
from sklearn.metrics import accuracy_score

accuracy_RF = accuracy_score(y_test_RF, y_pred)
print("Accuracy Score:", accuracy_RF)
```

```
Accuracy Score: 0.5666666666666667
```

the Random Forest model achieved an accuracy of approximately **56.7%**, indicating moderate performance in sentiment classification on the given dataset.