Brand Reputation Analysis based on YouTube Comments

Project Team

S. No.	Reg. No.	Student Name
1	20ETCS002152	Varun Raj B
2	20ETCS002055	H N Chetangowda
3	20ETCS00117	Saniya K R
4	20ETCS002151	V Sri Venkat

Supervisor: Dr Nayana B R
Dec – 2023

B. Tech. in Computer Science and Engineering
FACULTY OF ENGINEERING AND TECHNOLOGY
M. S. RAMAIAH UNIVERSITY OF APPLIED SCIENCES
BENGALURU -560 054

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

This is to certify that the Project titled "Brand Reputation Analysis based on YouTube Comments" is a bonafide work carried out in the Department of Computer Science and Engineering by Mr. Varun Raj B bearing Reg. No. 20ETCS002152 in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.

Dec – 2023

Dr Nayana B R
Associate Professor – Dept. of CSE

Dr. Rinki Sharma Desai	**Dr. Jagannathrao Venkatarao**
Head – Dept. of CSE	**Professor and Dean-FET**

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

This is to certify that the Project titled “Brand Reputation Analysis based on YouTube Comments” is a bonafide work carried out in the Department of Computer Science and Engineering by Mr. H N Chetangowda bearing Reg. No. 20ETCS002055 in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.

Dec – 2023

Dr Nayana B R
Associate Professor – Dept. of CSE

Dr. Rinki Sharma Desai	**Dr. Jagannathrao Venkatarao**
Head – Dept. of CSE	**Professor and Dean-FET**

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

This is to certify that the Project titled "Brand Reputation Analysis based on YouTube Comments is a bonafide work carried out in the Department of Computer Science and Engineering by Ms. Saniya K R bearing Reg. No. 20ETCS002117 in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.

Dec – 2023

Dr Nayana B R
Associate Professor – Dept. of CSE

Dr. Rinki Sharma Desai	**Dr. Jagannathrao Venkatarao**
Head – Dept. of CSE	**Professor and Dean-FET**

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

This is to certify that the Project titled " Brand Reputation Analysis based on YouTube Comments" is a bonafide work carried out in the Department of Computer Science and Engineering by Mr. V Sri Venkat bearing Reg. No. 20ETCS002151 in partial fulfilment of requirements for the award of B. Tech. Deg in Computer Science and Engineering of Ramaiah University of Applied Sciences.

Dec – 2023

Dr Nayana B R

Associate Professor – Dept. of CSE

Dr. Rinki Sharma Desai
Head – Dept. of CSE

Dr. Jagannathrao Venkatarao
Professor and Dean-FET

Declaration

Brand Reputation Analysis based on YouTube Comments

The project work is submitted in partial fulfilment of academic requirements for the award of B. Tech. Degree in the Department of Computer Science and Engineering of the Faculty of Engineering and Technology of Ramaiah University of Applied Sciences. The project report submitted herewith is a result of our own work and in conformance to the guidelines on plagiarism as laid out in the University Student Handbook. All sections of the text and results which have been obtained from other sources are fully referenced. We understand that cheating and plagiarism constitute a breach of University regulations, hence this project report has been passed through plagiarism check and the report has been submitted to the supervisor.

Sl. No.	**Reg. No.**	**Student Name**	**Signature**
1	**20ETCS002152**	**Varun Raj B**	
2	**20ETCS002055**	**H N Chetangowda**	
3	**20ETCS00117**	**Saniya K R**	
4	**20ETCS002151**	**V Sri Venkat**	

Date: 06 December 2023

Acknowledgements

We, the team members, express heartfelt gratitude to all the faculty members and the people who helped us in whatever way possible in completing this project.

We thank Dr Nayana B R, for guiding us through every step of the way, focusing on each and every detail, ensuring that we put our best effort forward.

We also thank our honourable Dean (FET), **Dr. Jagannathrao Venkatarao Desai** and Head of Department (CSE), **Dr. Rinki Sharma**, for providing the opportunity to work on this project and for being pillars of support and inspiration.

Summary

In the contemporary digital landscape, online platforms have become crucial arenas for brand interactions, with YouTube standing out as a prominent space for user engagement. This report explores the realm of brand reputational analysis by delving into the vast pool of YouTube comments, utilizing advanced computational techniques and natural language processing (NLP) methodologies.

The primary objective of this research is to develop a robust framework that extracts meaningful insights from the extensive and diverse dataset of YouTube comments. By applying NLP algorithms, the project aims to discern sentiments, opinions, and trends expressed by users in response to brand-related content on the platform. The methodology involves the development of a custom-built system capable of efficiently processing and analyzing large volumes of YouTube comments. Leveraging NLP algorithm decodes the textual comments to unveil the sentiments and themes associated with a brand.

To ensure the validity and reliability of the results, the study incorporates a diverse set of brands spanning various industries. The analysis encompasses both quantitative metrics, such as sentiment scores and comment frequency, and qualitative aspects, including identifying prevalent themes and sentiments within the user comments.

The findings of this research not only contribute to a deeper understanding of how brands are perceived in the digital space but also provide valuable insights for businesses to enhance their online presence and reputation management strategies. The implications extend to marketing, public relations, and social media management, offering actionable intelligence for fostering positive brand perception.

This report contributes to the evolving field of brand management by bridging the gap between traditional market research and the burgeoning realm of online user-generated content. By harnessing the power of YouTube comments, this project exemplifies an innovative approach to brand reputational analysis, shedding light on the dynamic and ever-changing landscape of digital brand perception.

Table of Contents

List of Tables

List of Figures

List of Abbreviations and Acronyms

VADER – Valence Aware Dictionary and sEntiment Reasoner
CNN – Convolutional Neural Networks
RNN – Recurrent Neural Networks
LSTM – Long Short-Term memory
MPLNN – Multilayer Perceptron Neural Networks

1. Introduction

This chapter sets the stage for the project by highlighting its motivation, context, and scope. It articulates the domain of application, justifying the significance of brand reputational analysis using YouTube comments and its potential benefits for both society and organizations. The introduction outlines the essential requirements and anticipated objectives of the system.

1.1 Motivation and Context

Understanding and managing brand reputation in the digital age poses significant challenges. Traditional tools for brand management fall short due to their reliance on manual input and the inherent subjectivity of human analysis. In the realm of online brand presence, particularly on platforms like YouTube, user-generated comments offer a wealth of information that can be harnessed for comprehensive brand reputational analysis.

YouTube, being a leading video-sharing platform, serves as a rich source of user opinions, sentiments, and feedback. Leveraging an automated approach, grounded in Natural Language Processing (NLP) and sentiment analysis, can provide invaluable insights into how a brand is perceived. This project proposes a Brand Reputational Analysis System that systematically processes YouTube comments to extract meaningful patterns and sentiments.

1.2 Scope

This project aims to develop a prototype Brand Reputational Analysis System based on YouTube comments, assessing the technical feasibility of such a solution. The scope involves an extensive survey of literature on NLP techniques, sentiment analysis algorithms, and methodologies for analyzing user-generated content on YouTube. The project will follow a structured development approach, combining aspects of both Prototype and V process models.

Objectives will be formulated to guide the development process, specifying methods and methodologies. The system's artifacts will be visualized and documented using Unified Modeling Language (UML). Requirements will be gathered and formulated, and a use case diagram will depict the system analysis. High-level and low-level sequence diagrams will detail the system design, while a class diagram will organize the executable code.

The system will rely on NLP algorithms to process YouTube comments, extracting sentiments and themes related to a brand. An interactive graphical user interface will be implemented for user management and activity review, allowing administrators to track and analyze brand sentiment over time.

1.3 Organization of the Report

The subsequent chapters will unfold as follows:

Chapter 2: Background Theory: Provides insights into the concepts and technologies underpinning the development, discussing the rationale behind the chosen NLP and sentiment analysis techniques.
Chapter 3: Project Objectives: Outlines the aim, objectives, and methodologies of the project, detailing the steps of progress and resource requirements.
Chapter 4: Methodology and Implementation: Describes the approach used for problem-solving, detailing functional and non-functional requirements, system design, data collection methods, and the implementation process.
Chapter 5: Results: Presents the findings of the implemented system, discussing how objectives were achieved and providing a detailed explanation of the output.
Chapter 6: Project Costing: Reports the expenses incurred, including hardware and software costs, design and development tools, and labor.
Chapter 7: Conclusion and Future Work: Concludes the project, summarizing the work done, addressing challenges faced, highlighting the benefits of the system, and discussing future enhancements and applications.

2. Background Theory

In this section, all theory related to proposed project including technical aspect and resources are explained. This section, it starts with discussion of Programming Languages and frameworks used, IDEs on which the proposed project is built, in the end, the entire theory is summarized via summary.

Brand Reputation:
Brand reputation is a critical asset for any business. It represents the perception and image of a brand in the eyes of its customers, potential customers, and the general public. A positive brand reputation is associated with trust, credibility, and customer loyalty, while a negative reputation can lead to decreased sales and trust issues.

Social Media and Brand Perception:
Social media platforms like YouTube have become central in shaping brand perception. These platforms provide a space for users to express their opinions and experiences with brands in a public forum. The feedback and discussions on social media play a significant role in influencing how the brand is perceived.

User-Generated Content (UGC):
YouTube comments are a prime example of user-generated content. Users voluntarily contribute content in the form of comments, reviews, and discussions related to brands.
UGC is considered a valuable source of data for understanding public sentiment.

Lexicon-Based and Machine Learning Approaches:
Sentiment analysis can be performed using lexicon-based approaches (where predefined sentiment dictionaries are used) and machine learning approaches (where algorithms are trained on labeled data to recognize sentiment patterns). Understanding these methods is critical to accurately gauge sentiment in comments.

Feedback Loop and Continuous Monitoring:
Brand reputation analysis is not a one-time effort. It involves continuous monitoring of social media and YouTube comments to track changes in sentiment over time. This feedback loop enables businesses to make timely adjustments to protect or enhance their brand reputation.

Competitive Analysis:
It's important to consider competitive analysis when evaluating brand reputation. Analyzing comments related to competitors can provide insights into how a brand compares in terms of sentiment, strengths, and weaknesses.

Actionable Insights:
The goal of brand reputation analysis is not just to gather data but to derive actionable insights. Understanding sentiment trends and specific issues raised in comments can guide strategic decisions, such as improving products, addressing customer concerns, or refining marketing strategies.

Ethical Considerations:
When analyzing user-generated content, ethical considerations related to privacy and data protection must be taken into account. Adherence to ethical guidelines and legal regulations is crucial in handling user data and comments.

In conclusion, the theory behind a brand reputation analysis project using YouTube comments is rooted in the understanding of brand reputation, social media's impact on it, sentiment analysis, data mining, and the application of NLP and machine learning techniques. This analysis provides valuable insights that can be used for strategic decision-making and maintaining a positive brand image in the digital age.

2.1 Python:

Python is a dynamic, high level, free open source and interpreted programming language. It supports object-oriented programming as well as procedural oriented programming. In Python, we don't need to declare the type of variable because it is a dynamically typed language. Some of the feature of python are:

- Free and Open Source
- High-Level Language
- Extensible feature
- Portable language
- Integrated language
- Can act as Object-Oriented Language
- GUI Programming Support
- Interpreted Language
- Large Standard Library
- Dynamically Typed Library

2.2 Jupyter Notebook:

Jupyter Notebooks basically provide an interative computational environment for developing Python based Data Science applications. They are formerly known as ipython notebooks. The following are some of features of the Jupyter notebooks are:
Jupyter notebooks can illustrate the analysis process step by step by arranging the stuff like code, images, text, output etc. in a step-by-step manner.
It helps a data scientist to document the thought process while developing the analysis process.
One can also capture the result as the part of the notebook.
With the help of Jupyter notebooks, we can share our work with a peer also.

2.3 Google Colab:

Google Colab provides tons of exciting features that any modern IDE offers, and much more. Some of the most exciting features are:

- Interactive tutorials to learn machine learning and neural networks.
- Write and execute Python 3 code without having a local setup.
- Execute terminal commands from the Notebook.
- Import datasets from external sources such as Kaggle.
- Save your Notebooks to Google Drive.
- Import Notebooks from Google Drive.
- Free cloud service, GPUs and TPUs.
- Integrate with PyTorch, Tensorflow, Open CV.
- Import or publish directly from/to GitHub.

2.4 Background of the used process:

Sentimental Analysis:

Brands must use data to understand their market's perception of their offerings. Organizations should use data across customer feedback channels to see their true impact on branding. Quantitative feedback like net promoter scores may only provide limited information about the performance of a brand but qualitative answers such as customer reviews and comments offer more insight into how customers feel about your brand. It is impossible to find all the data requested through manual processes as automated processes can help with finding brand sentiment through text. Sifting through this data is time consuming for humans and technology can help sift through that text of unsolicited feedback to find information on brand sentiment. Sentiment analysis is the process of detecting positive or negative sentiment in text. It can be used by businesses to detect sentiment in

social data, gauge brand reputation, and understand how customers feel about a company. Sentiment analysis is the process of organizing text and classifying whether it is positive/negative or neutral. Sentiment analysis is contextual mining of words which helps businesses determine whether the product they are manufacturing will be in demand or not. Sentiment analysis is a useful tool to help you understand the public opinion of your brand. It does this by identifying and extracting subjective information
from social media posts. Analysis of social media streams is usually restricted to simply basic sentiment analysis, but if you use more sophisticated methods on these streams, you will discover count-based metrics. Sentiment analysis is all about recognizing people's opinions and helping the companies to make their product or service more appealing for the customer. Sentiment analysis is not just about polarity (positive, negative), but also emotions (happy, sad, etc.). Some of the algorithms used in sentiment analysis are Rule-based and Automatic.

2.4.1. Types of Sentimental Analysis:

The following are some types of sentiment analysis which are observed on a general basis.
• Fine-Grained: With sentiment analysis, you can study reviews and ratings with the precision of polarity. The sentiment analysis uses categories like very positive, positive, neutral, negative, or very negative to identify potential sentiments in a review. For a rating scale, 1 is very negative and 5 is very positive. For a rating scale, 1-2 is very negative and 9-10 is very positive in some cases.
• Aspect-Based: Fine-grained analysis helps you determine the overall polarity of your customer reviews, but aspect-based analysis dives deeper.
• Emotion Detection: Emotion detection helps you express emotion, such as anger, sadness, happiness, frustration, fear, worry and panic. This is typically done through lexicons – a collection of words that convey certain emotions – or advanced machine learning algorithms to detect with greater accuracy.
• Intent Analysis: When businesses can accurately understand their consumers' intent with machines, they can save time, money, and effort. With a machine's ability to read and process data faster than a human, companies can find the leads that customers who want to buy at a certain time or need your product. Intent analysis informs the company and empowers them to provide the customer with what they want. There is a divide between
potential customers who might not be ready to buy, and those that are willing to purchase. It is helpful for your business goals to have knowledge of when each person might be ready for purchase. Then you can target said customer with advertisements to increase the chance of converting them into a customer.

2.4.2. Sentimental Analysis Using VADER:

VADER is a sentiment analysis that is sensitive to both polarity and intensity of emotion. Texts are analyzed with VADER before they are attached to a sentiment label. VADER is a lexicon and rule-based instrument that analyzes the feelings expressed in web content.
VADER uses a mix of lexical highlights, with most designating the direction of the feeling expressed as positive or negative. VADER relies on a sentiment score for each word, with words having fewer positive sentiments associated with negative numbers and more positive sentiments associated with positive numbers. The sentiment scores can be summed up to give the overall sentiment towards the sentence. For example, words like "love‟ and "enjoy‟ in the context of "did not love" carry a negative sentiment. VADER takes into consideration the context behind these words as well, such as emphasizing a word with capitalization and punctuation to express emotion.
Step of Analysis using VADER:
• A model trained on long text may not be effective for shorter texts. Make sure to use the appropriate model when analyzing a text, depending on its length.
• To continue, select the type of analysis you want to perform. To analyze sentence-length texts, we are going to use an NLTK lexicon called VADER which has been trained in sentiment analysis.
VADER can understand the sentiment of different messages and can extract meaningful pieces from it, such as emoticons, conjunctions, sentences, and more. VADER performs sentiment analysis very well and it is easy to use. It consists of a ready-made model which can be used across multiple domains, social media texts, reviewing, etc. The cherry on the cake with VADER is that you don't need any training data for it to work.

2.5 Machine Learning:

Learning involves observing data for changes and detection of patterns, based on which decisions are made for the future. Some of the learning techniques are:

- Supervised Machine Learning Algorithm
- Unsupervised Machine Learning Algorithm
- Semi-Supervised Machine Learning Algorithm
- Reinforced Machine Learning Algorithm

2.6 Neural Networks:

Neural networks are a class of machine learning models inspired by the structure and functioning of the human brain. They consist of interconnected nodes, or artificial neurons, organized into layers. Neural networks have become a fundamental building block in various artificial intelligence (AI) applications, and they are particularly powerful for tasks involving pattern recognition, classification, regression, and decision-making.

Types of Neural Networks:

Convolutional Neural Networks(CNN):

Convolutional Neural Networks (CNNs) are a class of deep neural networks designed for tasks involving grid-structured data, such as images and video. They are particularly effective in capturing hierarchical patterns and spatial dependencies in data.

Recurrent Neural Network(RNN):

Recurrent Neural Networks (RNNs) are a type of neural network architecture designed to handle sequential data by introducing connections that form loops. These loops allow information to persist, enabling RNNs to capture dependencies and patterns in sequences.Some of machine learning algorithms which can be used for sentimental analysis are:

- Naive Bayes:
- Support Vector Machines (SVM):
- Logistic Regression:
- Random Forest:
- Gradient Boosting:
- Long Short-Term Memory (LSTM) Networks:
- Bidirectional Encoder Representations from Transformers (BERT):

Long Short-Term Memory (LSTM) Networks:

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) architecture that excels in capturing long-term dependencies in sequential data. They have been widely adopted for sentiment analysis tasks, including the analysis of sentiments in textual data like YouTube comments. Here's an overview of how LSTMs can be applied to sentiment analysis:

- Sequential Data Handling
- Memory Cells
- Forget, Input, Output Gates
- Vanishing Gradient Problem

LSTMs are computationally intensive, and training on large datasets may require substantial resources.

Hyperparameter tuning and experimentation with different architectures can significantly impact model performance.

Pre-trained embeddings or models can boost performance, especially when training data is limited.

In summary, LSTM networks are a powerful tool for sentiment analysis in sequential data like YouTube comments, thanks to their ability to capture contextual information and dependencies over extended sequences.

Visual Studio IDE:

Visual Studio is a popular integrated development environment (IDE) developed by Microsoft. It provides a comprehensive set of tools and features for software development across various platforms, programming languages, and application types.

Key **Features:**

- Multi-Language Support
- Cross-Platform Development
- Integrated Debugger
- Code Navigation and IntelliSense
- Visual Designers
- NuGet Package Manager
- Version Control Integration
- Extensibility
- Azure Integration
- Performance Profiling

3. Aim and Objectives

3.1 Title

Brand Reputation Analysis Based on YouTube Comments

3.2 Aim

To develop a system for brand reputation analysis by extracting insights from user comments on YouTube videos related to a specific brand and provide actionable feedback.

3.3 Objectives

Conduct a Literature Review:
In this phase, a comprehensive literature review will be conducted to gain insights into sentiment analysis, natural language processing (NLP) techniques, and brand reputation analysis, specifically focusing on YouTube comments. By analyzing existing studies, papers, and articles, we aim to understand the latest advancements, methodologies, and challenges in these domains. This literature review will serve as the foundation for subsequent project phases, guiding decision-making and methodology development.

Define requirements, collect and preprocess data:
The next step involves defining clear requirements for the NLP model. This includes specifying the functionalities, features, and performance expectations of the model. Simultaneously, relevant data will be collected, focusing on YouTube comments. The collected data will undergo thorough preprocessing, involving cleaning and preparation.

Data preprocessing is crucial to ensure the data's quality, which directly impacts the effectiveness of the subsequent NLP model.

Obtain Labeled Data Using a Pretrained NLP Model:

To facilitate model training, labeled data is required. Leveraging a pretrained NLP model capable of sentiment classification, we aim to label the collected data efficiently. This step streamlines the process, as the pretrained model can discern sentiment nuances, providing a foundation for training a specialized model for the specific requirements of YouTube comment sentiment analysis.

Train and Fine-Tune the Model:

With labeled data in hand, the NLP model will undergo training using appropriate machine learning or deep learning algorithms. The training process involves optimizing the model's parameters to achieve accurate sentiment analysis and brand reputation assessment. Fine-tuning ensures the model adapts well to the unique characteristics of YouTube comments, enhancing its overall performance.

Evaluate Model Performance and Real-World Testing:

Once the NLP model is trained and fine-tuned, its performance will be rigorously evaluated using relevant evaluation metrics. This phase involves assessing accuracy, precision, recall, and other metrics to ensure the model meets the defined requirements. Subsequently, real-world testing using diverse YouTube comments will be conducted to evaluate the model's ability to handle varying comment types and expressions. The findings will be communicated through insightful visualizations, providing a comprehensive understanding of the model's capabilities.

These objectives collectively form a structured approach to developing an effective NLP model for sentiment analysis and brand reputation assessment in the context of YouTube comments. Each step contributes to the overall success of the project, from understanding the existing literature to implementing a functional and robust NLP solution.

3.4 Methods and Methodology/Approach to Attain Each Objective:

Objective No.	**Statement of the objective**	**Method/ methodology**	**Resources utilized**
1.	Conduct a literature review on sentiment analysis, NLP, and brand reputation	Literature Review: 1. Survey existing literature on sentiment analysis and NLP techniques 2. Identify Insights: Gather insights specific to YouTube comments 3. Focus on Brand Reputation: Explore literature on brand reputation analysis in online comments	Academic journals and papers Online articles and publications Research databases
2.	Define requirements, collect and preprocess data	1. Requirement Specification: Clearly define requirements for the NLP model 2. Data Collection: Collect relevant YouTube comments data 3. Data Preprocessing: Clean and prepare the collected data	Document for requirements Web scraping tools Data cleaning software/tools

3.	Obtain labeled data using a pretrained NLP model	1. Utilize Pretrained Model: Employ a pretrained NLP model for sentiment classification 2. Labelling: Obtain labelled data using the pretrained model	Pretrained NLP model Data labeling tools/software Relevant datasets for training
4.	Train and fine-tune the NLP model for accurate sentiment and brand analysis	1. Model Training: Train the NLP model using machine learning or deep learning algorithms 2. Fine-Tuning: Optimize the model for accurate sentiment and brand reputation analysis	Machine learning or deep learning frameworks (e.g., TensorFlow, PyTorch) Relevant algorithms for sentiment analysis
5.	Evaluate model performance and test on real-world YouTube comments	1. Performance Evaluation: Assess model performance using appropriate metrics 2. Real-world Testing: Test the model on diverse YouTube comments 3. Visualization: Communicate insights through visualizations	Evaluation metrics (e.g., accuracy, precision, recall) Real-world YouTube comments dataset Data visualization tools/software

Table 3.4: Methods and Methodology

3.5 Summary

The chapter aims to conduct a comprehensive literature review on sentiment analysis, NLP techniques, and brand reputation analysis specific to YouTube comments. Through this review, insights will be gathered to inform subsequent stages of the project. Clear requirements will be defined for the NLP model, followed by the collection and preprocessing of relevant YouTube comment data. Labeled data will be obtained using a pretrained NLP model proficient in sentiment classification. The model will then undergo training and fine-tuning with appropriate machine learning or deep learning algorithms, optimizing it for accurate sentiment and brand reputation analysis. Performance evaluation metrics will be employed to assess the model's effectiveness. Real-world testing will involve assessing the model's ability to handle diverse comment types and variations, with valuable insights communicated through visualizations. The project will culminate in the creation of an appropriate user interface and comprehensive documentation in the project report.

4. Design and Implementation

Based on the methodology discussed in the previous chapter, this chapter details the development process of the prototype, following the methodologies to accomplish each objective as described in the previous chapter. It provides the deliverables obtained at the end of each stage in the process. The chapter describes the analysis and design models obtained which explains the implemented prototype system with code listings.

4.1 Design:

Design is necessary when it comes to development since it acts as a blueprint for entire process from requirement making to finished (final product). Hence, in this section designs specific to this project like block diagram is attached.

A block diagram is a visual representation of a system or process that uses blocks to represent different components or stages of the system, and lines or arrows to indicate the flow or sequence of steps. It is a widely used tool in engineering, science, and various other fields to illustrate the structure, composition, and interactions within a complex system. Block diagrams are particularly useful for providing a high-level overview and helping in the understanding and communication of complex systems.

4.1.1Block Diagram:

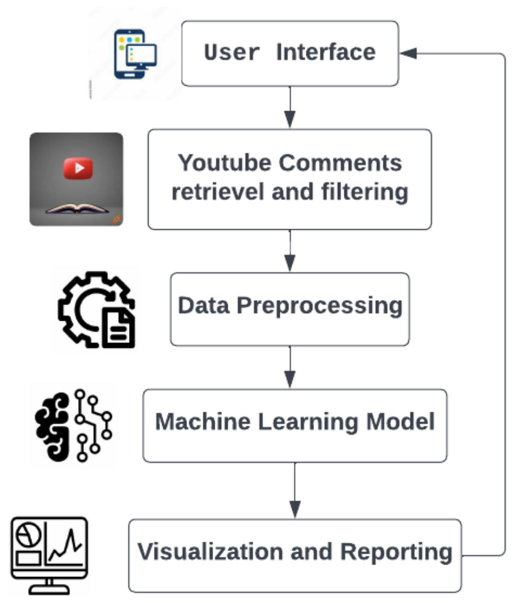

User Interface
Youtube Comments retrievel and filtering
Data Preprocessing
Machine Learning Model
Visualization and Reporting

Figure 4.1 System Interaction Block Diagram

4.1.2 Block diagram explanation:

1. User Interface:
The User Interface (UI) serves as the entry point for users to interact with the brand reputation analysis system. It provides a user-friendly platform where users can input parameters, initiate queries, and visualize the results. The UI acts as the bridge between the end user and the underlying components of the system, offering a seamless and intuitive experience.

2. YouTube Comments Retrieval and Filtering:
This component is responsible for fetching comments related to a specific brand from YouTube. It involves using APIs or web scraping techniques to extract comments from relevant videos. Filtering mechanisms are applied to refine the dataset, removing irrelevant or spam comments. This stage ensures that the subsequent analysis is conducted on a clean and meaningful dataset.

3. Data Preprocessing:
Once the comments are retrieved, the data preprocessing stage comes into play. This involves cleaning and structuring the raw text data. Tasks such as removing stop words, handling missing data, and converting text into a format suitable for machine learning models are performed. Data preprocessing is crucial for enhancing the quality of the input data, leading to more accurate and reliable results from the subsequent analysis.

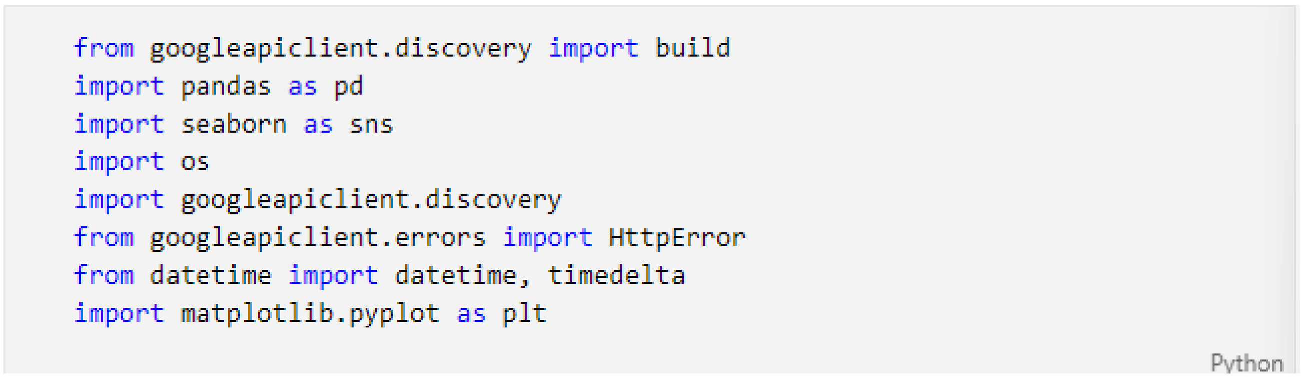
4. Machine Learning Model:
The heart of the brand reputation analysis system lies in the machine learning model. This component leverages natural language processing (NLP) techniques and sentiment analysis algorithms to assess the sentiment of the YouTube comments. The model is trained on labeled data to recognize patterns and sentiments expressed in the text. It classifies comments into positive, negative, or neutral sentiments, contributing to the overall understanding of brand perception.

5. Visualization and Reporting:
The final stage involves presenting the results in a comprehensible manner. Visualization tools and techniques are employed to display the sentiment analysis outcomes. This could include charts, graphs, or dashboards that highlight trends and patterns in brand sentiment over time. The reporting component also generates detailed summaries or reports that can be shared with stakeholders, providing valuable insights into the brand's reputation based on YouTube comments.

In summary, each component in the block diagram plays a crucial role in the brand reputation analysis process, from user interaction to data retrieval, preprocessing, analysis, and finally, presenting the results in an understandable format.

4.2 Implementation:

4.2.1 Importing all the necessary libraries:



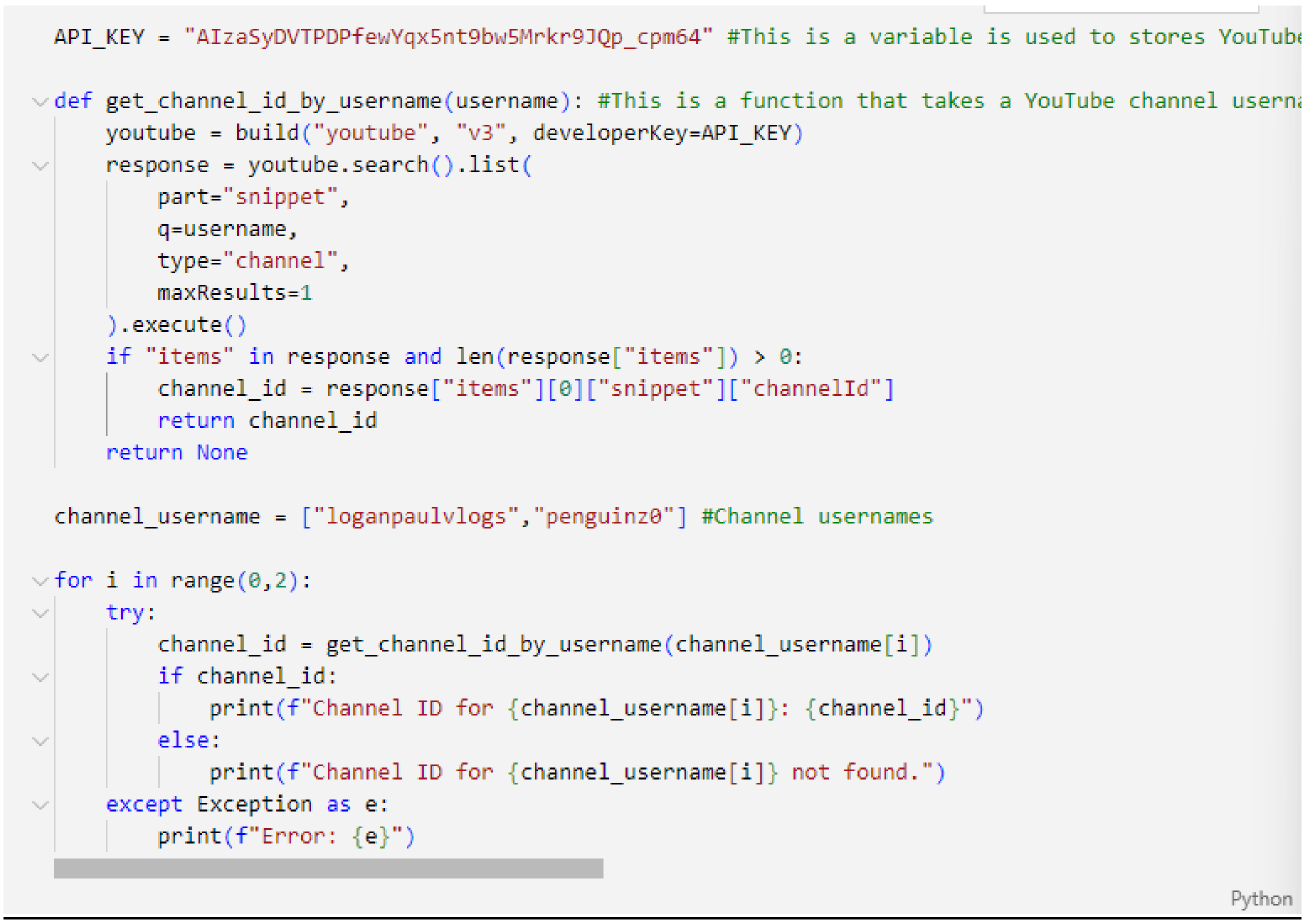
```python
from googleapiclient.discovery import build
import pandas as pd
import seaborn as sns
import os
import googleapiclient.discovery
from googleapiclient.errors import HttpError
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
```

Python

Fig 4.2.1: Represents Importing libraries

- The script employs the googleapiclient library to interact with Google APIs, indicating a focus on data retrieval from Google services.
- Utilizes the pandas library for efficient data processing, suggesting the manipulation and analysis of retrieved data through Pandas DataFrames.
- Incorporates seaborn for statistical data visualization, enhancing the presentation of insights through attractive and informative graphics.
- Imports matplotlib.pyplot to provide additional plotting capabilities, enhancing the script's visualization capabilities.
- Includes the os module, implying potential interactions with the operating system, likely for tasks such as file or directory management.
- Imports datetime and timedelta modules, indicating the involvement of date and time operations within the script.

4.2.2 Retrieving unique YouTube channel ID’s:



```python
API_KEY = "AIzaSyDVTPDPfewYqx5nt9bw5Mrkr9JQp_cpm64" #This is a variable is used to stores YouTube

def get_channel_id_by_username(username): #This is a function that takes a YouTube channel userna
    youtube = build("youtube", "v3", developerKey=API_KEY)
    response = youtube.search().list(
        part="snippet",
        q=username,
        type="channel",
        maxResults=1
    ).execute()
    if "items" in response and len(response["items"]) > 0:
        channel_id = response["items"][0]["snippet"]["channelId"]
        return channel_id
    return None

channel_username = ["loganpaulvlogs","penguinz0"] #Channel usernames

for i in range(0,2):
    try:
        channel_id = get_channel_id_by_username(channel_username[i])
        if channel_id:
            print(f"Channel ID for {channel_username[i]}: {channel_id}")
        else:
            print(f"Channel ID for {channel_username[i]} not found.")
    except Exception as e:
        print(f"Error: {e}")
```

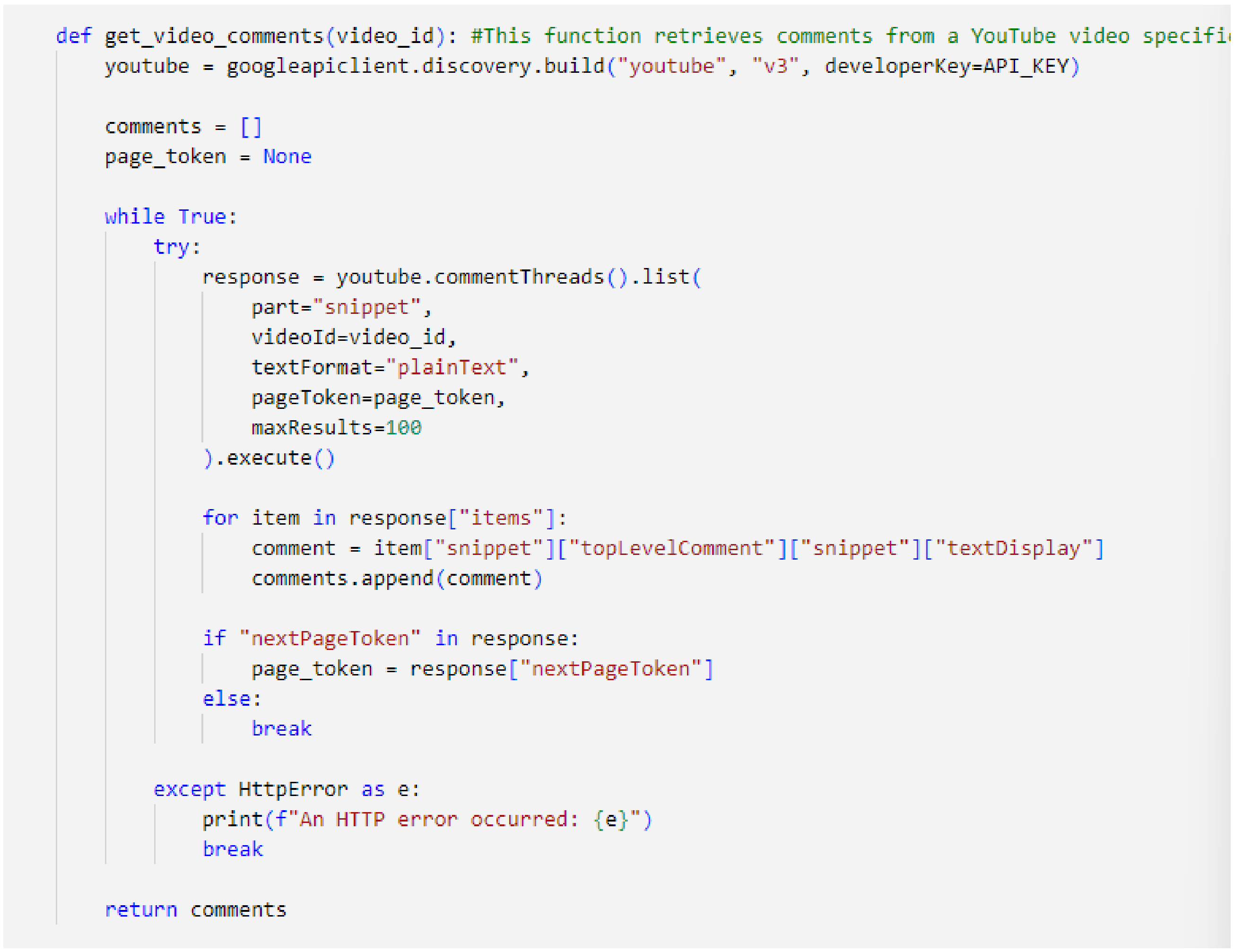
Python

Fig 4.2.2: Represents retrieving unique YouTube channel ID's associated with specific channel usernames("loganpaulvlogs","penguinz0").

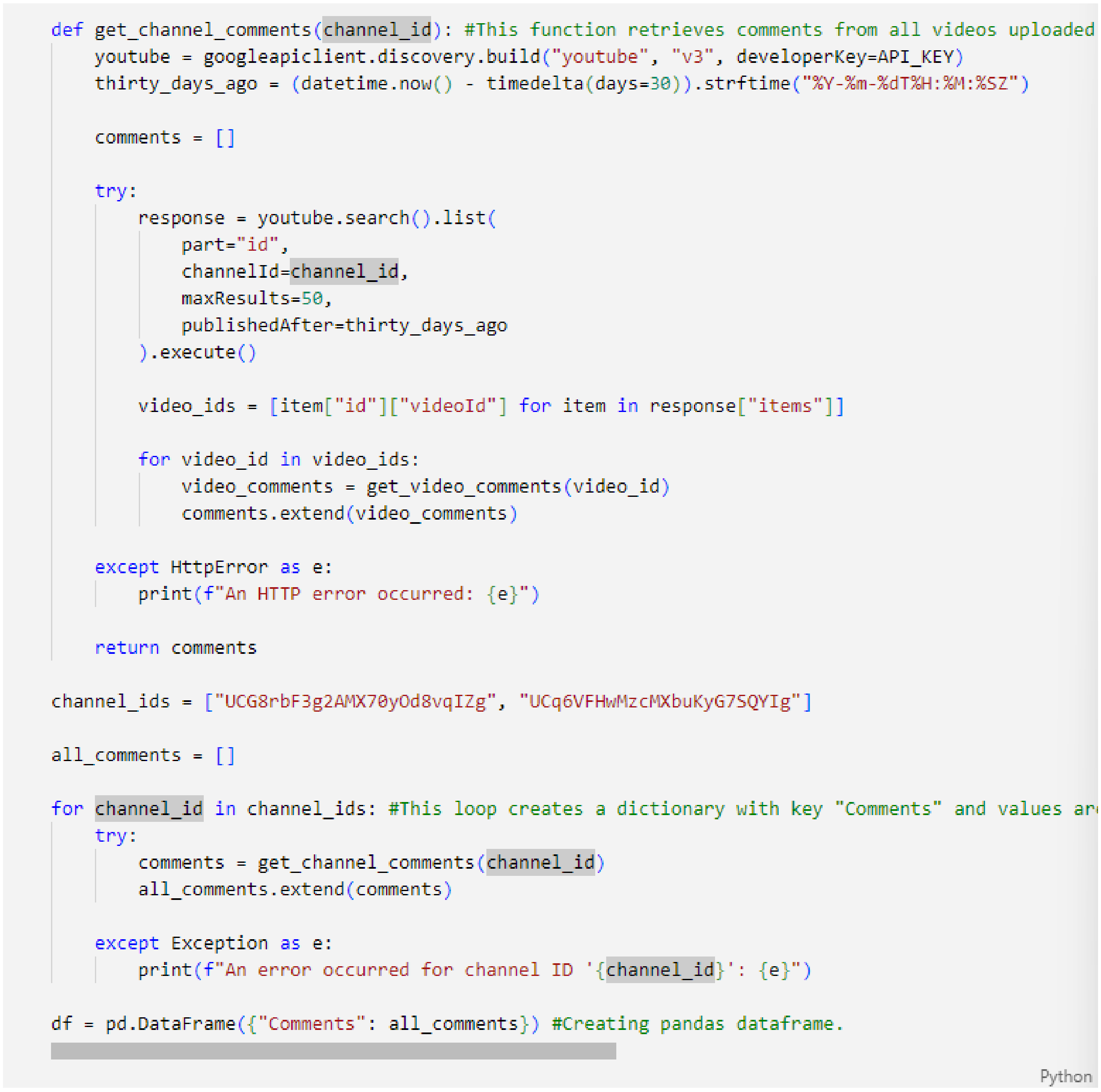
- API_KEY: It defines a variable API_KEY that stores the YouTube Data API key. This key is necessary to access the YouTube Data API.
- get_channel_id_by_username Function: It is a function that takes a YouTube channel username as input and uses the YouTube Data API to retrieve the associated YouTube Channel ID. The function returns None if the channel ID is not found.
- Loop through Usernames: The script defines a list channel_usernames containing the specific channel usernames for which it wants to retrieve the Channel IDs. It then loops through each username and calls the get_channel_id_by_username function.
- Try-Except Block: It includes a try-except block to handle potential errors during the API request. If an error occurs, it prints an error message.

- Print Results: After obtaining the Channel ID or indicating that it was not found, the script prints the results for each channel username.

4.2.3 Retrieving comments from YouTube Videos:



```python
def get_video_comments(video_id): #This function retrieves comments from a YouTube video specifi
    youtube = googleapiclient.discovery.build("youtube", "v3", developerKey=API_KEY)

    comments = []
    page_token = None

    while True:
        try:
            response = youtube.commentThreads().list(
                part="snippet",
                videoId=video_id,
                textFormat="plainText",
                pageToken=page_token,
                maxResults=100
            ).execute()

            for item in response["items"]:
                comment = item["snippet"]["topLevelComment"]["snippet"]["textDisplay"]
                comments.append(comment)

            if "nextPageToken" in response:
                page_token = response["nextPageToken"]
            else:
                break

        except HttpError as e:
            print(f"An HTTP error occurred: {e}")
            break

    return comments
```



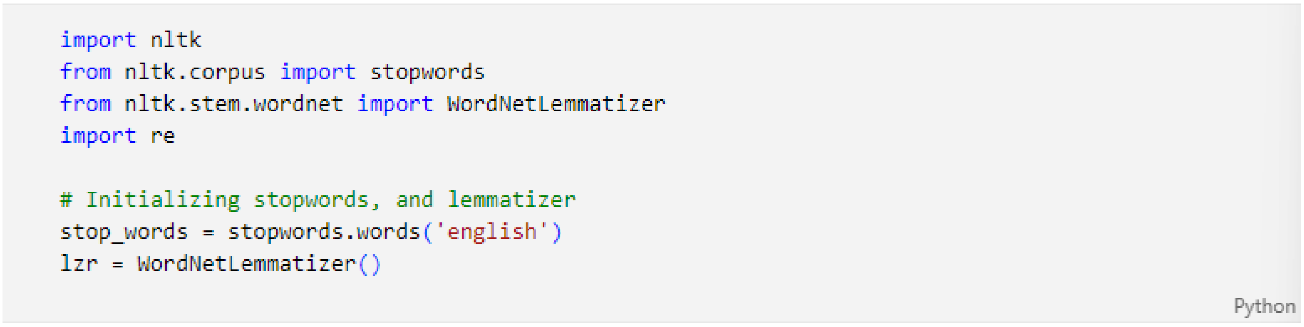
```python
def get_channel_comments(channel_id): #This function retrieves comments from all videos uploaded
    youtube = googleapiclient.discovery.build("youtube", "v3", developerKey=API_KEY)
    thirty_days_ago = (datetime.now() - timedelta(days=30)).strftime("%Y-%m-%dT%H:%M:%SZ")

    comments = []

    try:
        response = youtube.search().list(
            part="id",
            channelId=channel_id,
            maxResults=50,
            publishedAfter=thirty_days_ago
        ).execute()

        video_ids = [item["id"]["videoId"] for item in response["items"]]

        for video_id in video_ids:
            video_comments = get_video_comments(video_id)
            comments.extend(video_comments)

    except HttpError as e:
        print(f"An HTTP error occurred: {e}")

    return comments

channel_ids = ["UCG8rbF3g2AMX70yOd8vqIZg", "UCq6VFHwMzcMXbuKyG7SQYIg"]

all_comments = []

for channel_id in channel_ids: #This loop creates a dictionary with key "Comments" and values ar
    try:
        comments = get_channel_comments(channel_id)
        all_comments.extend(comments)

    except Exception as e:
        print(f"An error occurred for channel ID '{channel_id}': {e}")

df = pd.DataFrame({"Comments": all_comments}) #Creating pandas dataframe.
```

Python

Fig 4.2.3: Retrieving comments from YouTube videos associated with specific YouTube channels and storing them in a DataFrame.

- **get_video_comments Function:** This function takes a YouTube video ID as input and retrieves comments associated with that video using the YouTube Data API. It paginates through comments using a while loop to handle cases where there are more than 100 comments.
- **get_channel_comments Function:** This function takes a YouTube channel ID as input, retrieves video IDs uploaded by the channel in the last 30 days, and then calls get_video_comments to retrieve comments for each video.
- **Loop through Channel IDs:** The script defines a list channel_ids containing the specific YouTube channel IDs for which it wants to retrieve comments. It then loops through each channel ID and calls the get_channel_comments function.
- **Try-Except Block**: It includes a try-except block to handle potential errors during the API requests. If an error occurs, it prints an error message.
- **Pandas DataFrame Creation:** After obtaining all the comments, the script creates a pandas DataFrame with a column named "Comments" to store the comment text.

4.2.4 Data Preprocessing:



```python
import nltk
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import re

# Initializing stopwords, and lemmatizer
stop_words = stopwords.words('english')
lzr = WordNetLemmatizer()
```

Python

Fig 4.2.4: Represents Data Preprocessing step

NLTK Import: The import nltk statement imports the entire NLTK library. NLTK is a powerful library for natural language processing (NLP) tasks in Python.

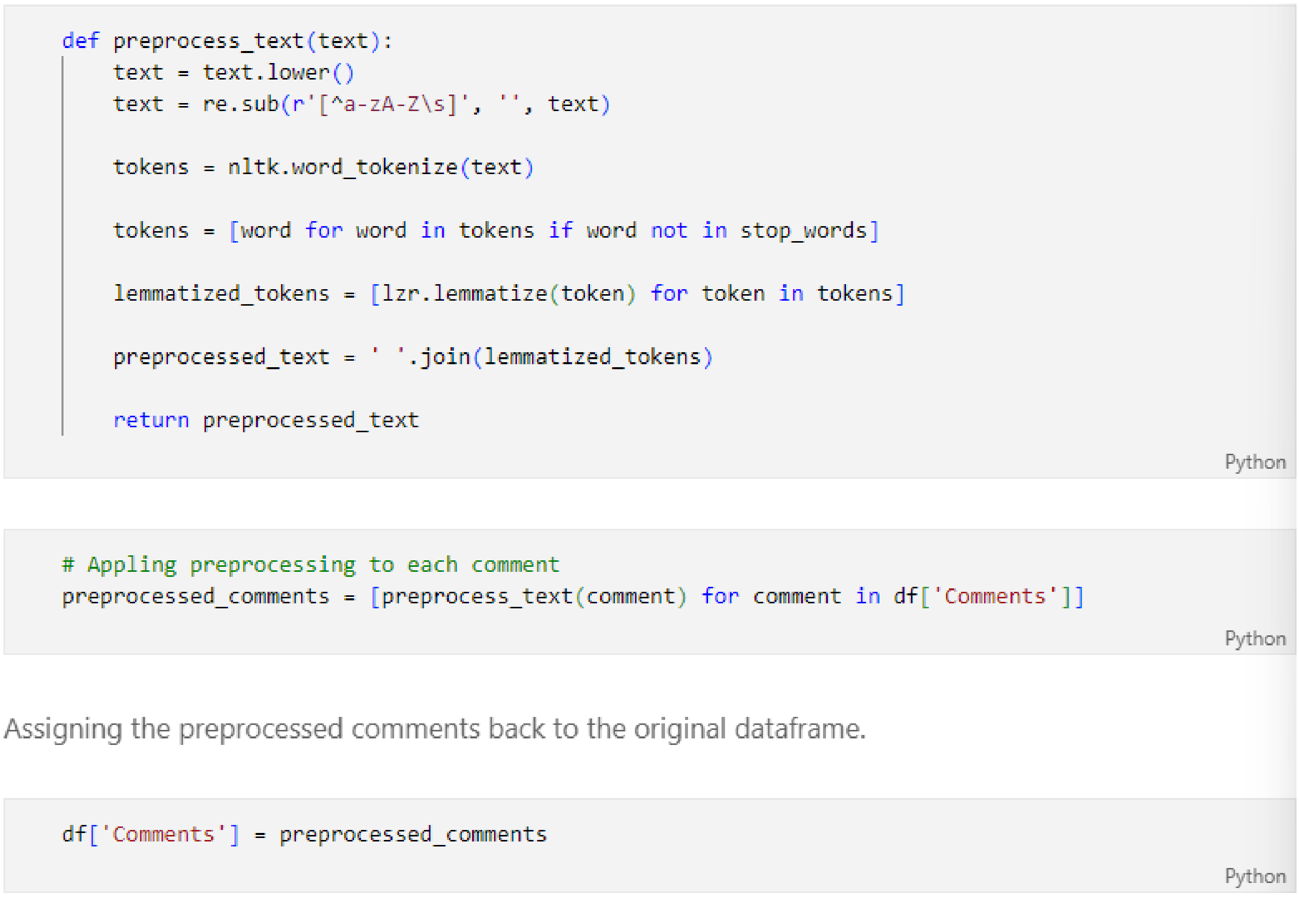
Stopwords and WordNetLemmatizer Import: The from nltk.corpus import stopwords and from nltk.stem.wordnet import WordNetLemmatizer statements import specific modules from NLTK.

Regular Expression (re) Import: The import re statement imports the regular expression module, which is commonly used for string manipulation and pattern matching.

Stopwords Initialization: stopwords.words('english') initializes a list of English stopwords. Stopwords are commonly used words (e.g., "the", "and", "is") that are often removed during text preprocessing because they don't carry much meaning in many NLP tasks.

Lemmatizer Initialization: WordNetLemmatizer() initializes an instance of the WordNet Lemmatizer from NLTK. Lemmatization is a text normalization process that reduces words to their base or root form (e.g., "running" to "run"). WordNet is a lexical database of the English language that NLTK uses for lemmatization.

4.2.5 Data cleaning:



```python
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    tokens = nltk.word_tokenize(text)

    tokens = [word for word in tokens if word not in stop_words]

    lemmatized_tokens = [lzr.lemmatize(token) for token in tokens]

    preprocessed_text = ' '.join(lemmatized_tokens)

    return preprocessed_text
```

Python

```python
# Appling preprocessing to each comment
preprocessed_comments = [preprocess_text(comment) for comment in df['Comments']]
```

Python

Assigning the preprocessed comments back to the original dataframe.

```python
df['Comments'] = preprocessed_comments
```

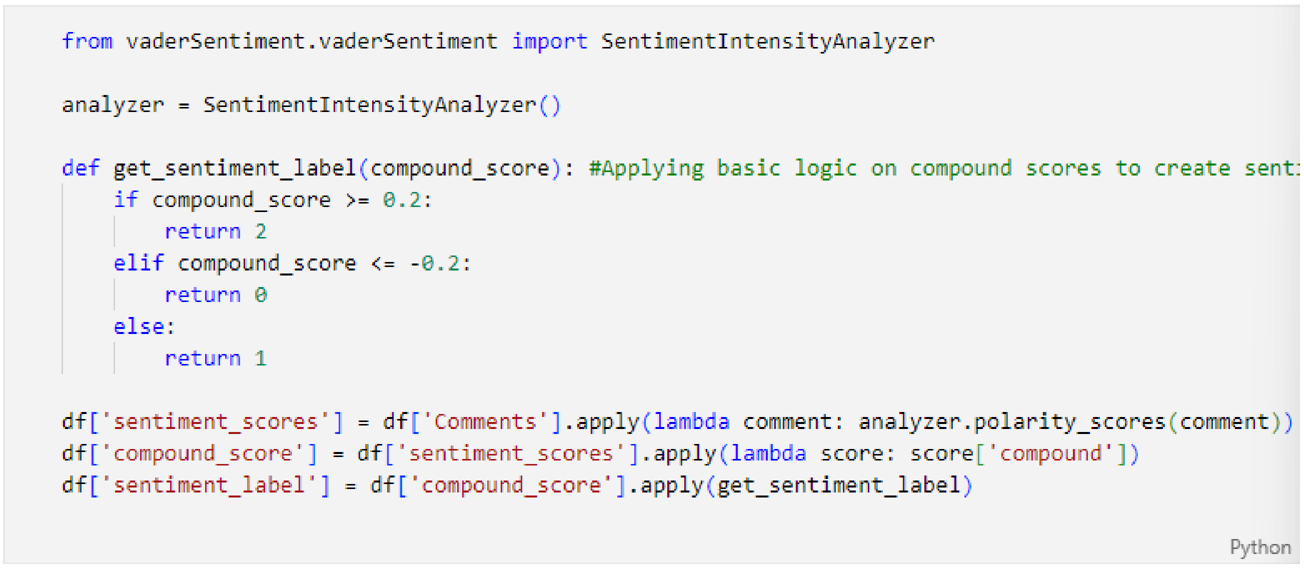
Python

Fig 4.2.5: Represents the data cleaning

A function **preprocess_text** that takes a text as input and performs several text preprocessing steps. It appears to be part of a natural language processing (NLP) pipeline, commonly used for tasks such as text classification or sentiment analysis. Let's break down the code step by step:

- **text = text.lower()**: Converts all characters in the text to lowercase. This step is commonly done to ensure consistency and avoid treating the same word differently based on its case.
- **text = re.sub(r'[^a-zA-Z\s]', '', text)**: Uses a regular expression to remove non-alphabetic characters (characters other than letters) from the text. This step eliminates numbers, punctuation, and other non-alphabetic symbols.
- **tokens = nltk.word_tokenize(text)**: Tokenizes the text into a list of words using the Natural Language Toolkit (nltk) **word_tokenize** function. Tokenization is the process of breaking a text into words or other meaningful elements.
- **tokens = [word for word in tokens if word not in stop_words]**: Removes common English stop words from the list of tokens. Stop words are frequently used words like "the," "and," or "is" that are often removed in text analysis because they may not contribute much to the meaning of the text.
- **lemmatized_tokens = [lzr.lemmatize(token) for token in tokens]**: Lemmatizes each token using a lemmatizer. Lemmatization reduces words to their base or root form. For example, "running" becomes "run."
- **preprocessed_text = ' '.join(lemmatized_tokens)**: Joins the lemmatized tokens back into a single string, separating each token with a space.
- The function returns the **preprocessed_text**.

4.2.6 Sentiment labelling using VADER:



```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

analyzer = SentimentIntensityAnalyzer()

def get_sentiment_label(compound_score): #Applying basic logic on compound scores to create sent
    if compound_score >= 0.2:
        return 2
    elif compound_score <= -0.2:
        return 0
    else:
        return 1

df['sentiment_scores'] = df['Comments'].apply(lambda comment: analyzer.polarity_scores(comment))
df['compound_score'] = df['sentiment_scores'].apply(lambda score: score['compound'])
df['sentiment_label'] = df['compound_score'].apply(get_sentiment_label)
```

Fig 4.2.6: Represents sentiment labelling using VADER

This code performs sentiment analysis on the 'Comments' column of a DataFrame (df) using the VADER sentiment analysis tool. The goal is to create a labeled dataset for a machine learning model based on the sentiment scores generated by VADER.

Import the necessary library:
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
This line imports the SentimentIntensityAnalyzer class from the VADER sentiment analysis tool.

Initialize the sentiment analyzer:
analyzer = SentimentIntensityAnalyzer()
Creates an instance of the SentimentIntensityAnalyzer class, which will be used to obtain sentiment scores for each comment.

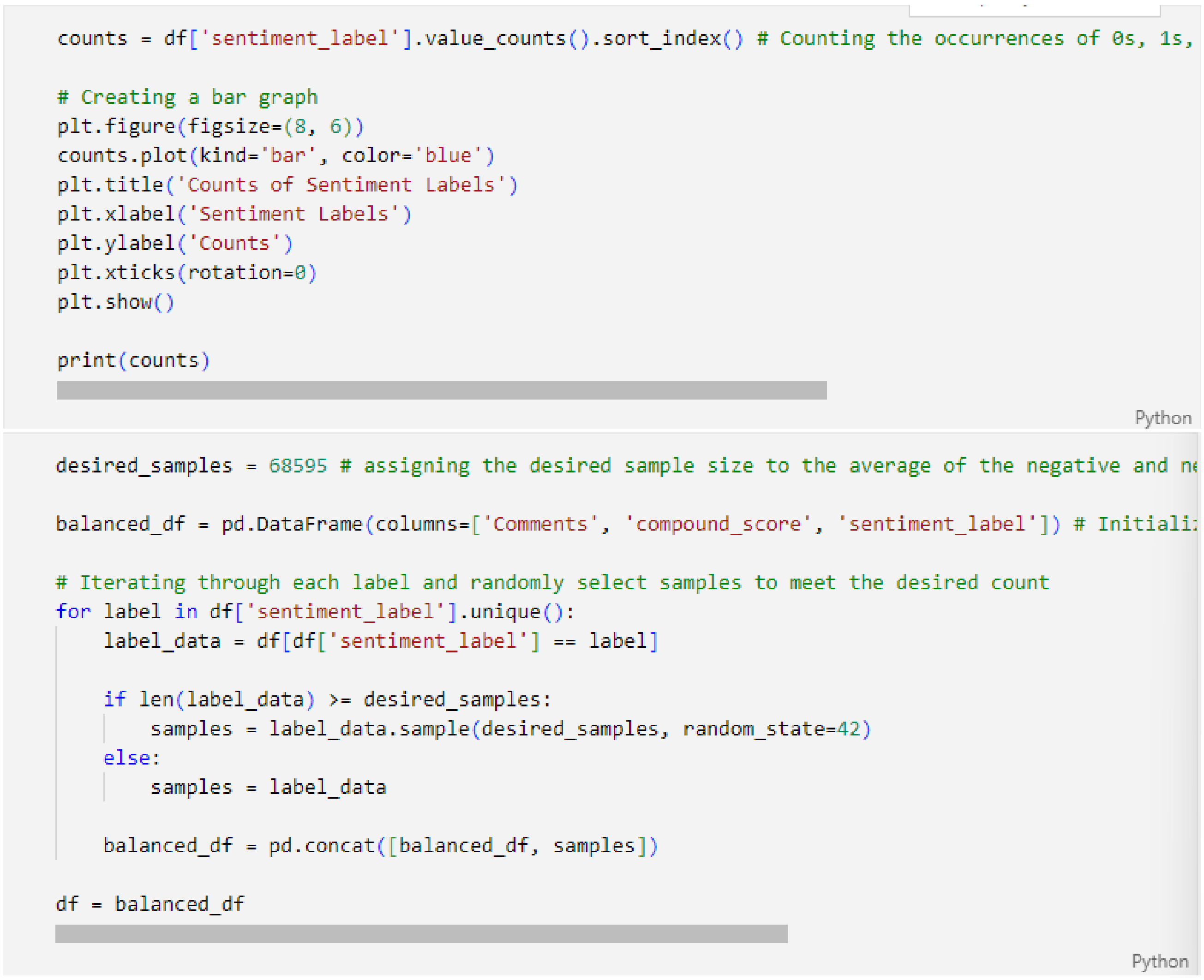
Define a function for sentiment label based on compound scores:
def get_sentiment_label(compound_score): if compound_score >= 0.2: return 2 elif compound_score <= -0.2: return 0 else: return 1
This function takes a compound score as input and returns a sentiment label based on basic logic:
If the compound score is greater than or equal to 0.2, the sentiment label is set to 2 (positive).
If the compound score is less than or equal to -0.2, the sentiment label is set to 0 (negative).
Otherwise, the sentiment label is set to 1 (neutral).

Apply sentiment analysis to each comment in the DataFrame:
df['sentiment_scores']=df['Comments'].apply(lambda
comment: analyzer.polarity_scores(comment))
Applies the VADER sentiment analyzer to each comment in the 'Comments' column, generating a dictionary of sentiment scores (positive, neutral, negative, and compound) for each comment. The results are stored in a new column called 'sentiment_scores'.

Extract the compound score:
df['compound_score']=df['sentiment_scores'].apply(lambdascore:score['compound']) Extracts the compound score from the 'sentiment_scores' column and stores it in a new column called 'compound_score'.

Create sentiment labels based on compound scores:
df['sentiment_label'] = df['compound_score'].apply(get_sentiment_label)
Applies the get_sentiment_label function to each compound score in the 'compound_score' column, creating a new column 'sentiment_label' that contains the sentiment label for each comment.
After running this code, the DataFrame (df) will have three additional columns: 'sentiment_scores', 'compound_score', and 'sentiment_label', providing sentiment information for each comment in the 'Comments' column based on the VADER sentiment analysis. The 'sentiment_label' column can be used as labeled data for training a machine learning model on sentiment analysis.

4.2.7 Data balancing:



```
counts = df['sentiment_label'].value_counts().sort_index() # Counting the occurrences of 0s, 1s,

# Creating a bar graph
plt.figure(figsize=(8, 6))
counts.plot(kind='bar', color='blue')
plt.title('Counts of Sentiment Labels')
plt.xlabel('Sentiment Labels')
plt.ylabel('Counts')
plt.xticks(rotation=0)
plt.show()

print(counts)
```

Python

```
desired_samples = 68595 # assigning the desired sample size to the average of the negative and ne

balanced_df = pd.DataFrame(columns=['Comments', 'compound_score', 'sentiment_label']) # Initiali

# Iterating through each label and randomly select samples to meet the desired count
for label in df['sentiment_label'].unique():
    label_data = df[df['sentiment_label'] == label]

    if len(label_data) >= desired_samples:
        samples = label_data.sample(desired_samples, random_state=42)
    else:
        samples = label_data

    balanced_df = pd.concat([balanced_df, samples])

df = balanced_df
```

Python

Fig 4.2.7: Represents Data balancing

Checking Data Imbalance:

1. **Counting Sentiment Labels:**

   - The code calculates the number of occurrences for each sentiment label (positive, neutral, and negative) in the 'sentiment_label' column.

2. **Bar Graph Visualization:**
   - A bar graph is created to visually represent the distribution of sentiment labels in the original dataset. This helps to understand the balance or imbalance in the data.

Balancing the Dataset:

1. **Setting Desired Sample Size:**
   - The desired sample size is determined, set as the average count of negative, neutral, and positive comments.
2. **Initializing Empty DataFrame:**
   - An empty DataFrame named **balanced_df** is created with the same structure as the original DataFrame. It will be used to store the balanced data.
3. **Sampling Data for Balancing:**
   - For each unique sentiment label in the original dataset:
     - If the count of data points for a label is greater than or equal to the desired sample size, a random sample of data points is selected to match the desired size.
     - If there are fewer data points than the desired size, all available data points for that label are included.
4. **Concatenating Samples:**
   - The selected samples for each label are concatenated into the **balanced_df** DataFrame.

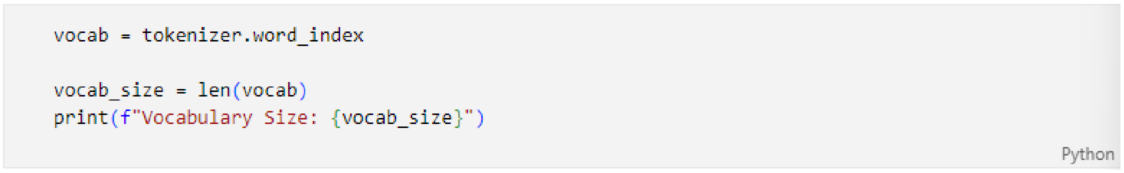
5. **Updating the Original DataFrame:**
    - The original DataFrame (**df**) is replaced with the balanced DataFrame (**balanced_df**).

Checking Data Balance After Balancing:

1. **Counting Sentiment Labels in Balanced Dataset:**
    - The code calculates the number of occurrences for each sentiment label in the 'sentiment_label' column of the balanced dataset.
2. **Bar Graph Visualization After Balancing:**
    - Another bar graph is created to visually represent the distribution of sentiment labels in the balanced dataset. This allows for a quick comparison with the initial distribution.
3. **Print Counts After Balancing:**
    - The count of each sentiment label in the balanced dataset is printed for verification.

By performing these steps, the dataset is transformed to be more balanced, with each sentiment label having approximately the same number of data points. This balancing process helps prevent a machine learning model from being biased towards the majority class and can lead to better generalization performance.

4.2.8 Finding the Vocabulary Size:



```python
vocab = tokenizer.word_index

vocab_size = len(vocab)
print(f"Vocabulary Size: {vocab_size}")
```

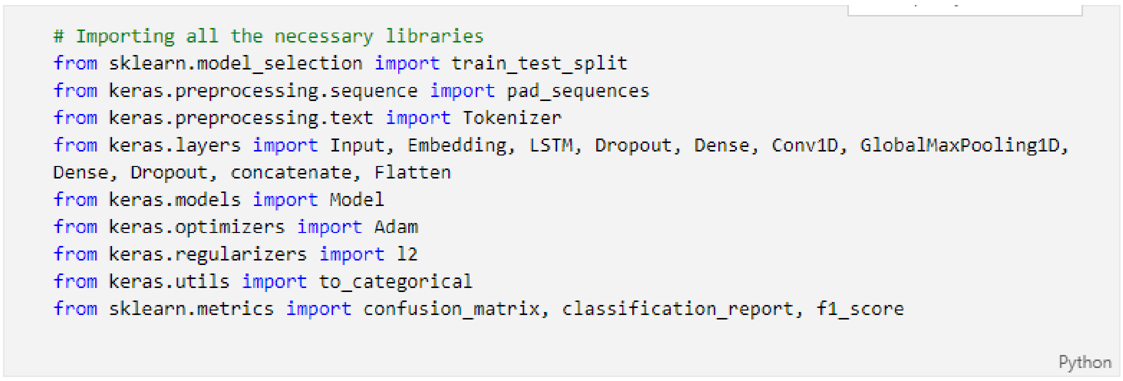
Fig 4.2.8: Represents finding the vocabulary size

Vocabulary Size Calculation:

1. **Tokenization:**
    - Tokenization is the process of converting text into individual tokens or words. In this context, a tokenizer is used to break down the text into its constituent words.
2. **Word Index:**
    - The **tokenizer.word_index** operation generates a dictionary where each unique word in the dataset is a key, and the corresponding value is the token assigned to that word during tokenization.
3. **Calculating Vocabulary Size:**
    - The vocabulary size is determined by finding the number of unique words in the dataset. It is essentially the count of distinct tokens assigned to words during tokenization.
4. **Understanding Vocabulary Size:**
    - The vocabulary size represents the diversity of words in the dataset. A larger vocabulary size indicates a more varied set of words used in the text.
5. **Print Result:**
    - The final step is to print the calculated vocabulary size. This provides a quantitative measure of the richness and complexity of the language used in the dataset.

The printed result "Vocabulary Size: 61452" suggests that there are 61,452 unique words or tokens present in the dataset after tokenization. This information is valuable in various natural language processing tasks, especially when designing and configuring neural network models for text analysis.

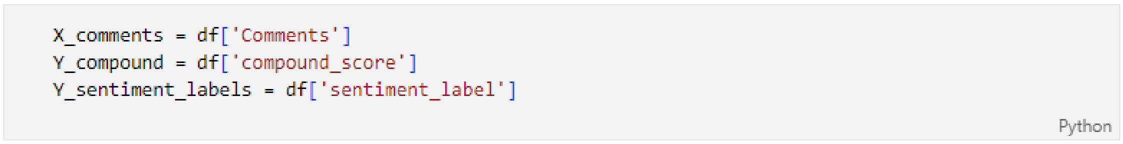
4.2.9 Model Building:



```python
# Importing all the necessary libraries
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.layers import Input, Embedding, LSTM, Dropout, Dense, Conv1D, GlobalMaxPooling1D,
Dense, Dropout, concatenate, Flatten
from keras.models import Model
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, classification_report, f1_score
```

Python

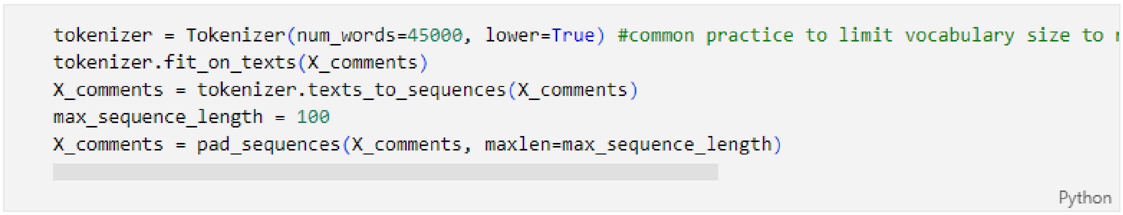
Creating input and output variables.



```python
X_comments = df['Comments']
Y_compound = df['compound_score']
Y_sentiment_labels = df['sentiment_label']
```

Python

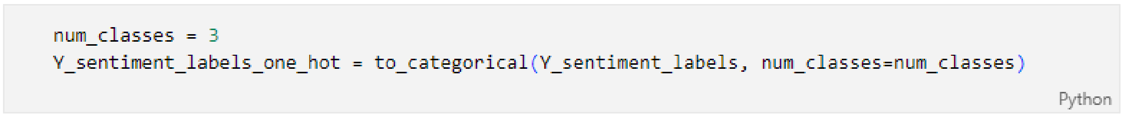
Tokenization and padding the comment sequences.



```python
tokenizer = Tokenizer(num_words=45000, lower=True) #common practice to limit vocabulary size to
tokenizer.fit_on_texts(X_comments)
X_comments = tokenizer.texts_to_sequences(X_comments)
max_sequence_length = 100
X_comments = pad_sequences(X_comments, maxlen=max_sequence_length)
```

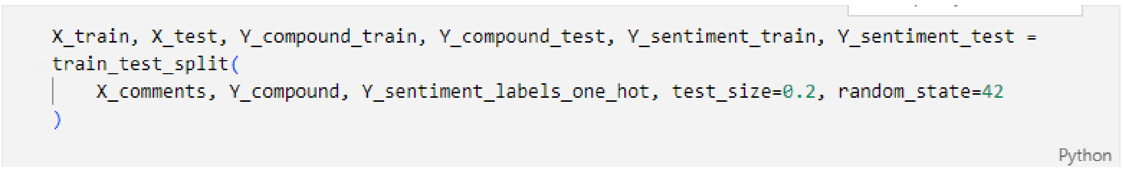
Python

One-hot Encoding Y_sentiment_labels



```python
num_classes = 3
Y_sentiment_labels_one_hot = to_categorical(Y_sentiment_labels, num_classes=num_classes)
```

Python



```python
X_train, X_test, Y_compound_train, Y_compound_test, Y_sentiment_train, Y_sentiment_test =
train_test_split(
    X_comments, Y_compound, Y_sentiment_labels_one_hot, test_size=0.2, random_state=42
)
```

Python

Fig 4.2.9: Represents model building

1. Input and Output Variables:

- **Input (X_comments):**
    - **X_comments** represents the text comments from the dataset.
- **Output (Y_compound, Y_sentiment_labels):**
    - **Y_compound** represents the compound sentiment scores for each comment.
    - **Y_sentiment_labels** represents the categorical sentiment labels for each comment (positive, neutral, or negative).

2. Tokenization and Padding:

- **Tokenizer:**
    - A **Tokenizer** is used to convert text comments into sequences of integers (tokens).
    - The **num_words** parameter is set to limit the vocabulary size, reducing memory usage and improving model performance.
- **Fit on Texts:**
    - The tokenizer is fitted on the text comments to build a vocabulary.
- **Texts to Sequences:**
    - Text comments are then converted to sequences of integers using the tokenizer.
- **Padding Sequences:**
    - Sequences are padded to ensure a consistent length (max_sequence_length). Padding is essential for handling variable-length input in neural networks.

3. One-Hot Encoding:

- **Y_sentiment_labels_one_hot:**
    - **Y_sentiment_labels** (categorical sentiment labels) are one-hot encoded using **to_categorical**. This is a common practice in classification tasks, representing each class as a binary vector.

4. Train-Test Split:

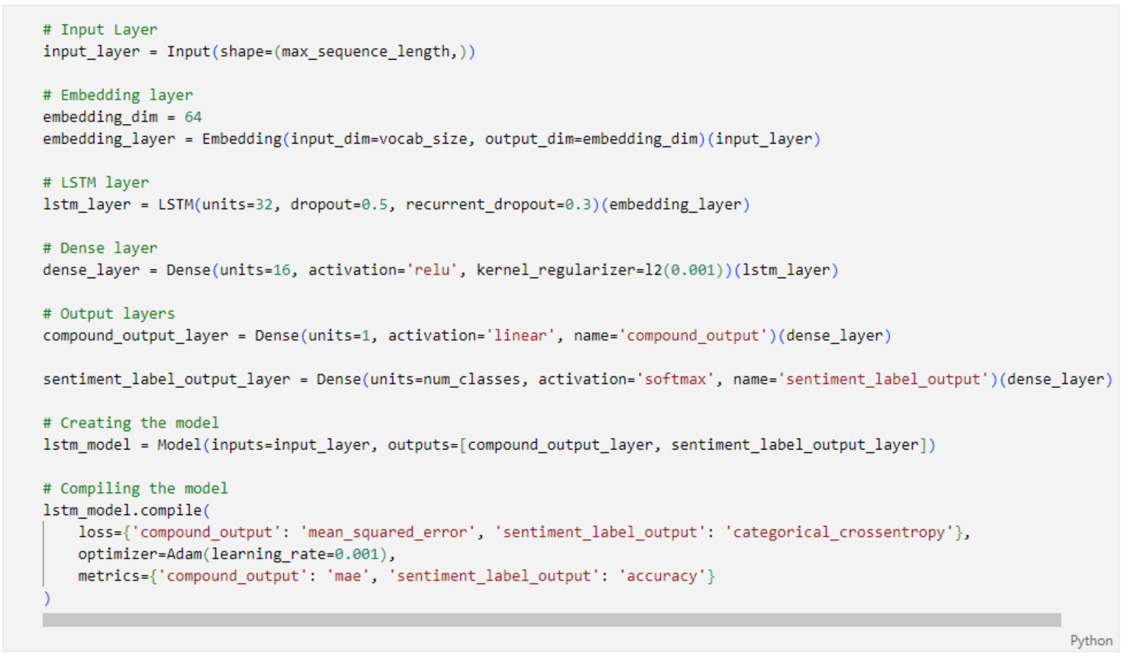
- **Splitting Data:**
    - The dataset is split into training and testing sets using **train_test_split**.
    - **X_comments** is split into **X_train** and **X_test**.
    - **Y_compound** is split into **Y_compound_train** and **Y_compound_test**.

        - **Y_sentiment_labels_one_hot** is split into **Y_sentiment_train** and **Y_sentiment_test**.
    - **Parameters:**
        - **test_size=0.2**: 20% of the data is used for testing, and 80% for training.
        - **random_state=42**: A seed is set for reproducibility.
- The input variables (text comments) are processed using tokenization and padding to prepare them for input into a neural network.
- Output variables are prepared by one-hot encoding the categorical sentiment labels.
- The dataset is split into training and testing sets, a standard practice in machine learning, to train the model on one subset and evaluate its performance on another.

This preparation sets the stage for building a neural network model that can learn patterns from the text data and predict both the compound sentiment scores and the categorical sentiment labels.

4.2.10 Model Architechture:

LSTM [Long Short-Term Memory]:



```python
# Input Layer
input_layer = Input(shape=(max_sequence_length,))

# Embedding layer
embedding_dim = 64
embedding_layer = Embedding(input_dim=vocab_size, output_dim=embedding_dim)(input_layer)

# LSTM layer
lstm_layer = LSTM(units=32, dropout=0.5, recurrent_dropout=0.3)(embedding_layer)

# Dense layer
dense_layer = Dense(units=16, activation='relu', kernel_regularizer=l2(0.001))(lstm_layer)

# Output layers
compound_output_layer = Dense(units=1, activation='linear', name='compound_output')(dense_layer)

sentiment_label_output_layer = Dense(units=num_classes, activation='softmax', name='sentiment_label_output')(dense_layer)

# Creating the model
lstm_model = Model(inputs=input_layer, outputs=[compound_output_layer, sentiment_label_output_layer])

# Compiling the model
lstm_model.compile(
    loss={'compound_output': 'mean_squared_error', 'sentiment_label_output': 'categorical_crossentropy'},
    optimizer=Adam(learning_rate=0.001),
    metrics={'compound_output': 'mae', 'sentiment_label_output': 'accuracy'}
)
```

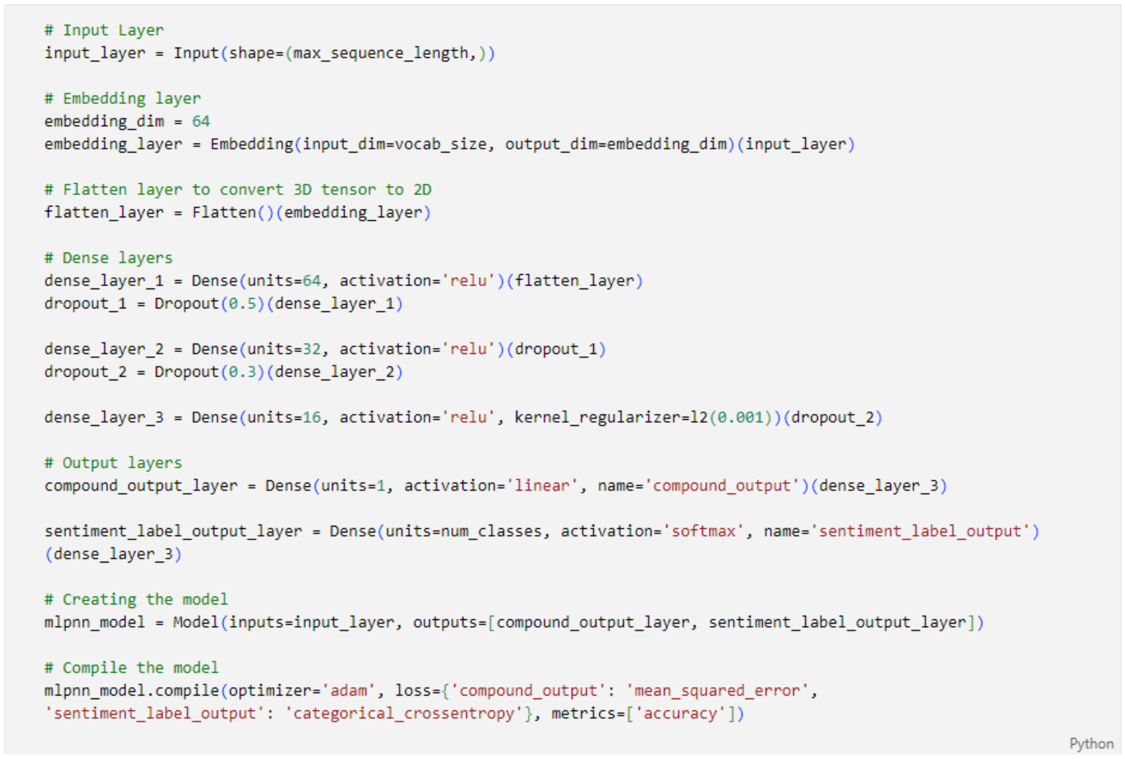
Fig 4.2.10(1): Represents Model Architecture[LSTM]

This code defines and compiles a neural network model for sentiment analysis using a combination of Long Short-Term Memory (LSTM) and dense layers. Here's a concise explanation:

1. **Input Layer:**
    - The model's input layer is defined with a shape corresponding to the maximum sequence length of the tokenized and padded text comments.
2. **Embedding Layer:**
    - An embedding layer converts integer-encoded words into dense vectors of fixed size (embedding_dim). It captures semantic relationships between words.
3. **LSTM Layer:**
    - The LSTM layer, a type of recurrent neural network layer, processes sequential data, capturing long-term dependencies and patterns in the text.
4. **Dense Layer:**
    - A dense layer follows the LSTM layer, providing non-linear transformations and reducing the dimensionality of the data. It uses rectified linear unit (ReLU) activation and includes regularization (kernel_regularizer).
5. **Output Layers:**
    - Two output layers are defined: one for predicting the compound sentiment score (regression task) and another for predicting the categorical sentiment label (classification task) using softmax activation.
6. **Model Creation:**
    - The Keras Model class is used to create the overall neural network model, specifying the input and output layers.
7. **Compiling the Model:**
    - The model is compiled with different loss functions and metrics for each output layer. The compound sentiment score is treated as a regression task with mean squared error loss, while the sentiment label prediction is a classification task with categorical cross-entropy loss. Adam optimizer is used with a specific learning rate. Metrics such as mean absolute error (MAE) and accuracy are tracked during training and evaluation.

This model architecture combines LSTM for sequence processing, dense layers for feature extraction, and dual output layers for predicting both compound sentiment scores and sentiment labels. It is compiled with suitable loss functions and metrics for the respective tasks.

MLPNN [Multilayer Perceptron Neural Network]:



```python
# Input Layer
input_layer = Input(shape=(max_sequence_length,))

# Embedding layer
embedding_dim = 64
embedding_layer = Embedding(input_dim=vocab_size, output_dim=embedding_dim)(input_layer)

# Flatten layer to convert 3D tensor to 2D
flatten_layer = Flatten()(embedding_layer)

# Dense layers
dense_layer_1 = Dense(units=64, activation='relu')(flatten_layer)
dropout_1 = Dropout(0.5)(dense_layer_1)

dense_layer_2 = Dense(units=32, activation='relu')(dropout_1)
dropout_2 = Dropout(0.3)(dense_layer_2)

dense_layer_3 = Dense(units=16, activation='relu', kernel_regularizer=l2(0.001))(dropout_2)

# Output layers
compound_output_layer = Dense(units=1, activation='linear', name='compound_output')(dense_layer_3)

sentiment_label_output_layer = Dense(units=num_classes, activation='softmax', name='sentiment_label_output')
(dense_layer_3)

# Creating the model
mlpnn_model = Model(inputs=input_layer, outputs=[compound_output_layer, sentiment_label_output_layer])

# Compile the model
mlpnn_model.compile(optimizer='adam', loss={'compound_output': 'mean_squared_error',
'sentiment_label_output': 'categorical_crossentropy'}, metrics=['accuracy'])
```

Fig 4.2.10(2): Represents Model Architecture[MLPNN]

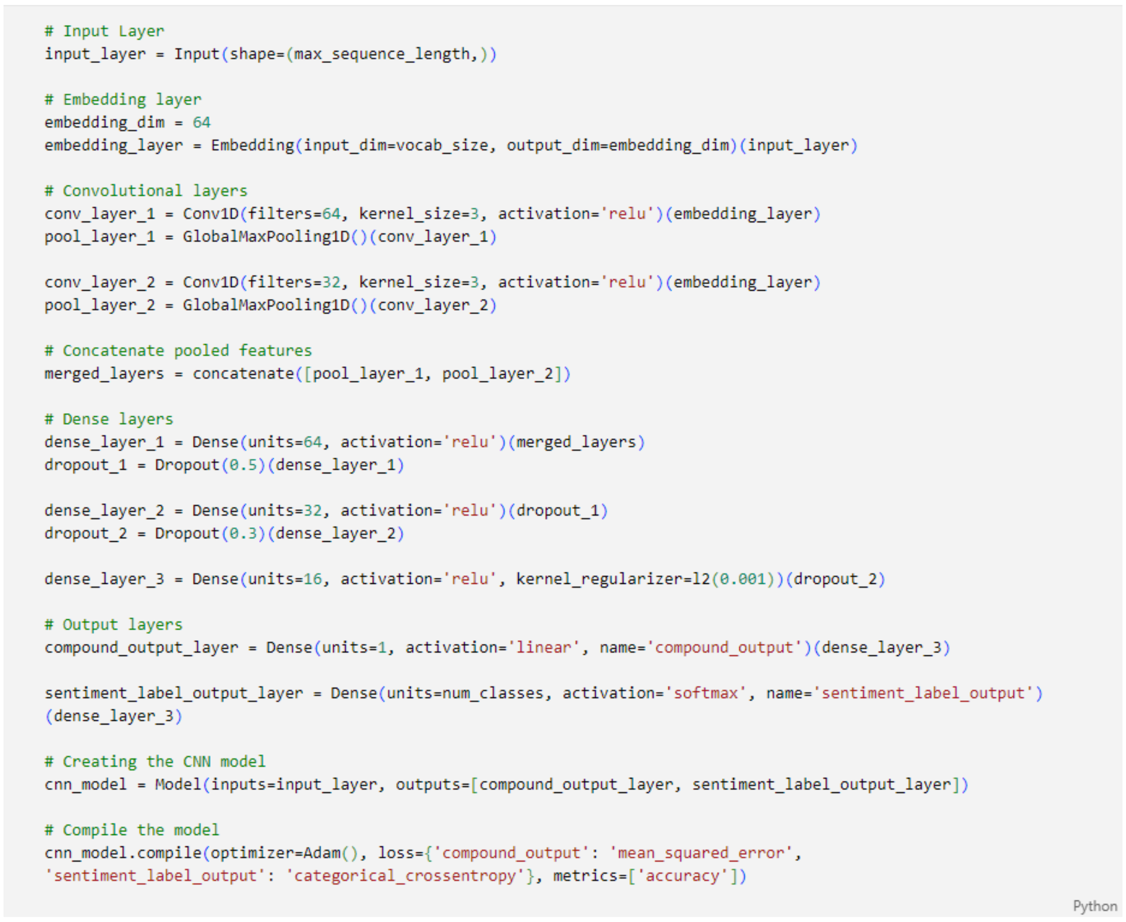
This code defines and compiles a Multi-Layer Perceptron Neural Network (MLPNN) model for sentiment analysis.

1. **Input Layer:**
    - The model starts with an input layer specifying the shape of the input data, corresponding to the maximum sequence length of tokenized and padded text comments.
2. **Embedding Layer:**
    - An embedding layer converts integer-encoded words into dense vectors, capturing semantic relationships. In this case, the embeddings are flattened to convert the 3D tensor to a 2D tensor.
    -

3. **Flatten Layer:**
   - A flatten layer is employed to convert the 3D tensor output from the embedding layer into a 2D tensor, preparing the data for the subsequent dense layers.
4. **Dense Layers with Dropout:**
   - Multiple dense layers with rectified linear unit (ReLU) activation are used for feature extraction and transformation. Dropout layers are incorporated to mitigate overfitting by randomly dropping a fraction of neurons during training.
5. **Output Layers:**
   - Two output layers are defined: one for predicting the compound sentiment score (regression task) and another for predicting the categorical sentiment label (classification task) using softmax activation.
6. **Model Creation:**
   - The Keras Model class is employed to define the overall neural network architecture, specifying the input and output layers.
7. **Model Compilation:**
   - The model is compiled using the Adam optimizer, with distinct loss functions for each output layer. Mean squared error loss is used for the regression task (compound sentiment score), and categorical cross.
   - entropy loss is employed for the classification task (sentiment label). Accuracy is tracked as the evaluation metric.

This MLPNN architecture comprises dense layers with dropout for feature transformation and is designed to predict both compound sentiment scores and sentiment labels. It is compiled with appropriate loss functions and accuracy as the metric for evaluation.

CNN [Convolutional Neural Network]:



```python
# Input Layer
input_layer = Input(shape=(max_sequence_length,))

# Embedding layer
embedding_dim = 64
embedding_layer = Embedding(input_dim=vocab_size, output_dim=embedding_dim)(input_layer)

# Convolutional layers
conv_layer_1 = Conv1D(filters=64, kernel_size=3, activation='relu')(embedding_layer)
pool_layer_1 = GlobalMaxPooling1D()(conv_layer_1)

conv_layer_2 = Conv1D(filters=32, kernel_size=3, activation='relu')(embedding_layer)
pool_layer_2 = GlobalMaxPooling1D()(conv_layer_2)

# Concatenate pooled features
merged_layers = concatenate([pool_layer_1, pool_layer_2])

# Dense layers
dense_layer_1 = Dense(units=64, activation='relu')(merged_layers)
dropout_1 = Dropout(0.5)(dense_layer_1)

dense_layer_2 = Dense(units=32, activation='relu')(dropout_1)
dropout_2 = Dropout(0.3)(dense_layer_2)

dense_layer_3 = Dense(units=16, activation='relu', kernel_regularizer=l2(0.001))(dropout_2)

# Output layers
compound_output_layer = Dense(units=1, activation='linear', name='compound_output')(dense_layer_3)

sentiment_label_output_layer = Dense(units=num_classes, activation='softmax', name='sentiment_label_output')
(dense_layer_3)

# Creating the CNN model
cnn_model = Model(inputs=input_layer, outputs=[compound_output_layer, sentiment_label_output_layer])

# Compile the model
cnn_model.compile(optimizer=Adam(), loss={'compound_output': 'mean_squared_error',
'sentiment_label_output': 'categorical_crossentropy'}, metrics=['accuracy'])
```

Fig 4.2.10(3): This code defines and compiles a Convolutional Neural Network (CNN) model for sentiment analysis.

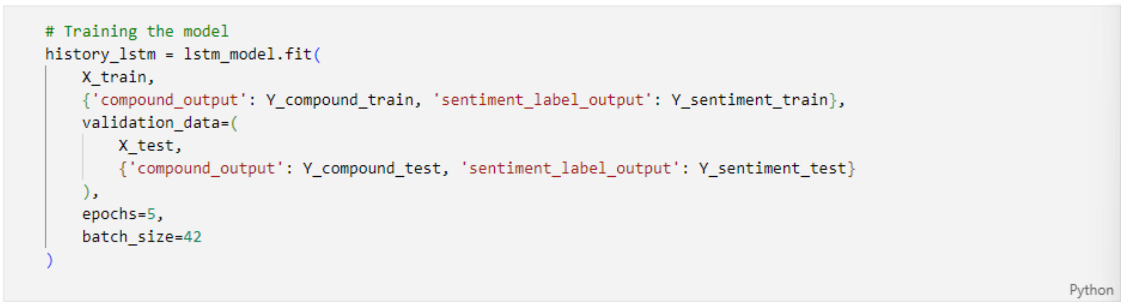
1. **Input Layer:**
    - The model begins with an input layer specifying the shape of the input data, which corresponds to the maximum sequence length of tokenized and padded text comments.
2. **Embedding Layer:**
    - An embedding layer converts integer-encoded words into dense vectors, capturing semantic relationships.

3. **Convolutional Layers:**
    - Two convolutional layers with different filter sizes (3) are applied to the embedding layer. Each convolutional layer captures local patterns in the text.
4. **Pooling Layers:**
    - Global max-pooling layers extract the most important features from each convolutional layer, reducing the dimensionality of the data.
5. **Concatenation:**
    - Pooled features from both convolutional layers are concatenated to capture a broader range of features.
6. **Dense Layers with Dropout:**
    - Multiple dense layers with rectified linear unit (ReLU) activation are employed for feature transformation. Dropout layers are used for regularization to prevent overfitting.
7. **Output Layers:**
    - Two output layers are defined: one for predicting the compound sentiment score (regression task) and another for predicting the categorical sentiment label (classification task) using softmax activation.
8. **Model Creation:**
    - The Keras Model class is used to define the overall neural network architecture, specifying the input and output layers.
9. **Model Compilation:**
    - The model is compiled using the Adam optimizer, with distinct loss functions for each output layer. Mean squared error loss is used for the regression task (compound sentiment score), and categorical cross-entropy loss is employed for the classification task (sentiment label). Accuracy is tracked as the evaluation metric.

This CNN architecture leverages convolutional and pooling layers for local feature extraction, followed by dense layers for higher-level feature transformation. It is designed to predict both compound sentiment scores and sentiment labels.

4.2.11 Model Training:

LSTM:



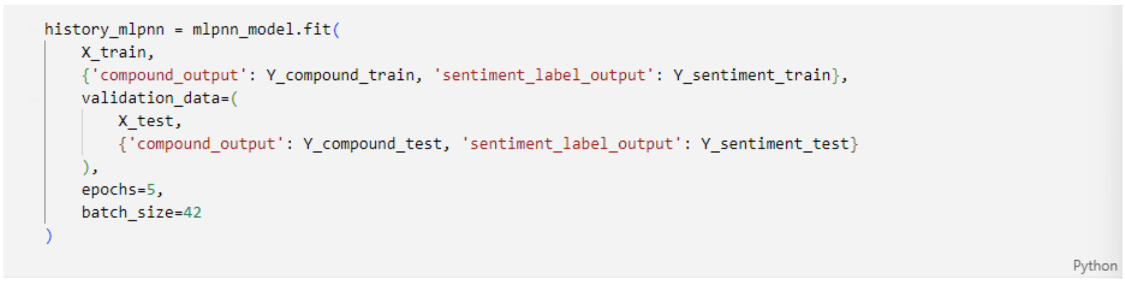
```python
# Training the model
history_lstm = lstm_model.fit(
    X_train,
    {'compound_output': Y_compound_train, 'sentiment_label_output': Y_sentiment_train},
    validation_data=(
        X_test,
        {'compound_output': Y_compound_test, 'sentiment_label_output': Y_sentiment_test}
    ),
    epochs=5,
    batch_size=42
)
```

Python

Fig 4.2.11(1): Represents Model Training [LSTM]

The code trains a sentiment analysis model (**lstm_model**) using the **fit** function. It utilizes the training data (**X_train**, **Y_compound_train**, and **Y_sentiment_train**) and validates the model on the test data (**X_test**, **Y_compound_test**, and **Y_sentiment_test**). The training is performed over five epochs with a batch size of 42. The model is designed to predict both compound sentiment scores (**compound_output**) and sentiment labels (**sentiment_label_output**). The training process updates the model's parameters to minimize the specified loss functions for both regression (mean squared error) and classification (categorical cross-entropy) tasks. Validation results are provided for monitoring performance on unseen data.

MLPNN:



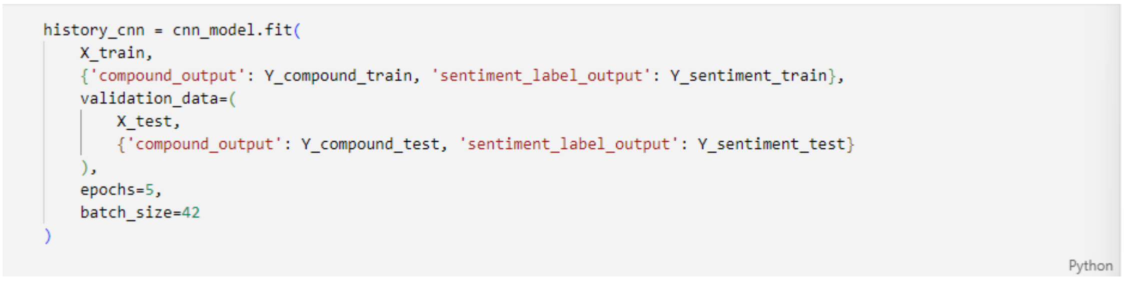
```python
history_mlpnn = mlpnn_model.fit(
    X_train,
    {'compound_output': Y_compound_train, 'sentiment_label_output': Y_sentiment_train},
    validation_data=(
        X_test,
        {'compound_output': Y_compound_test, 'sentiment_label_output': Y_sentiment_test}
    ),
    epochs=5,
    batch_size=42
)
```

Python

Fig 4.2.11(2): Represents Model Training [MLPN]

Similarly training MLPNN.

CNN:

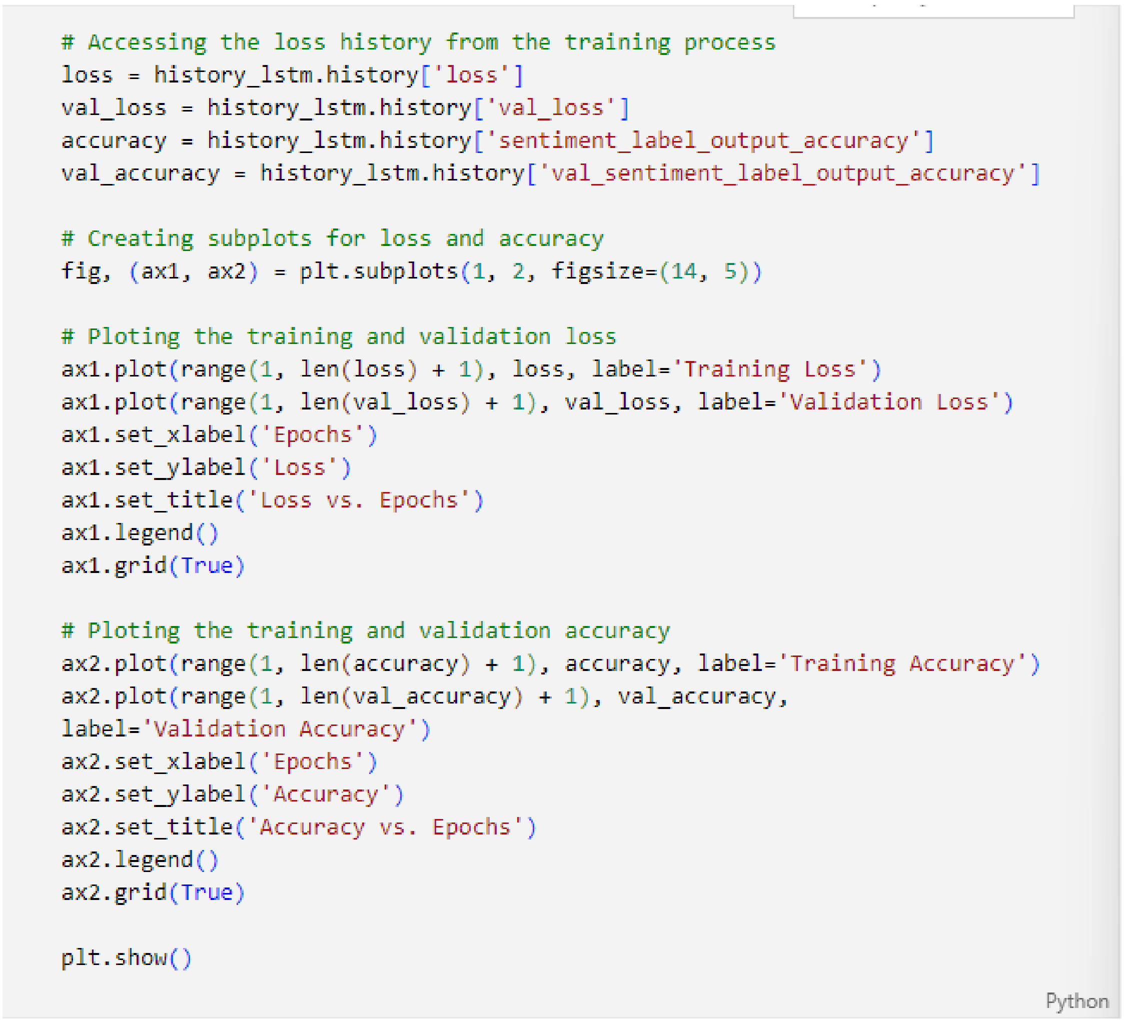


```python
history_cnn = cnn_model.fit(
    X_train,
    {'compound_output': Y_compound_train, 'sentiment_label_output': Y_sentiment_train},
    validation_data=(
        X_test,
        {'compound_output': Y_compound_test, 'sentiment_label_output': Y_sentiment_test}
    ),
    epochs=5,
    batch_size=42
)
```

Fig 4.2.11(3): Represents Model Training [CNN]

Similarly training CNN.

4.2.12. Model Evaluation:



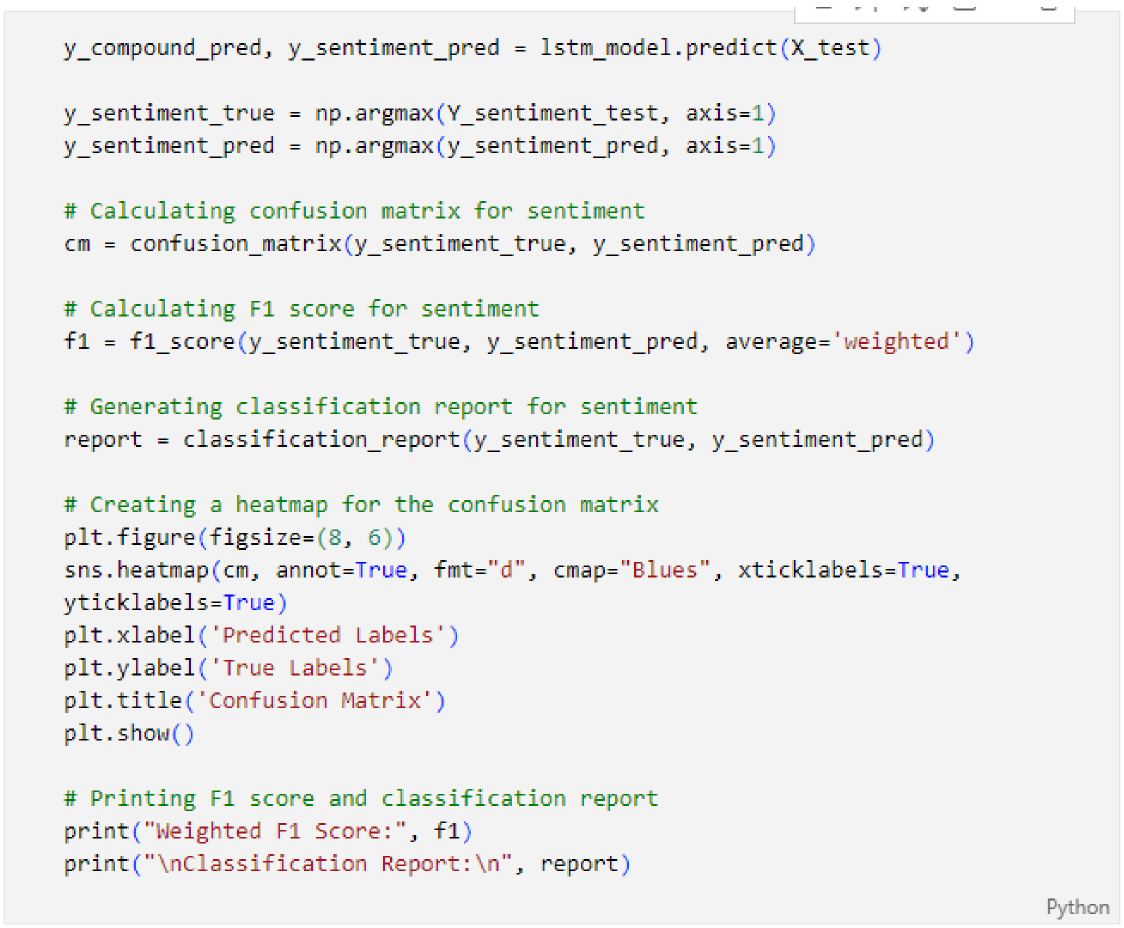
```python
# Accessing the loss history from the training process
loss = history_lstm.history['loss']
val_loss = history_lstm.history['val_loss']
accuracy = history_lstm.history['sentiment_label_output_accuracy']
val_accuracy = history_lstm.history['val_sentiment_label_output_accuracy']

# Creating subplots for loss and accuracy
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Ploting the training and validation loss
ax1.plot(range(1, len(loss) + 1), loss, label='Training Loss')
ax1.plot(range(1, len(val_loss) + 1), val_loss, label='Validation Loss')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Loss')
ax1.set_title('Loss vs. Epochs')
ax1.legend()
ax1.grid(True)

# Ploting the training and validation accuracy
ax2.plot(range(1, len(accuracy) + 1), accuracy, label='Training Accuracy')
ax2.plot(range(1, len(val_accuracy) + 1), val_accuracy,
label='Validation Accuracy')
ax2.set_xlabel('Epochs')
ax2.set_ylabel('Accuracy')
ax2.set_title('Accuracy vs. Epochs')
ax2.legend()
ax2.grid(True)

plt.show()
```

Fig 4.2.12: Represents code to evaluate training history

This code is for visualizing the training history of the neural networks with respect to loss and accuracy.

1. Data Extraction:
    - The code first extracts relevant information from the training history (any of the models trained). It retrieves lists of training loss (loss), validation loss (val_loss), training accuracy (accuracy), and validation accuracy (val_accuracy).
2. Plotting Setup:
    - It creates a subplot with two panels (ax1 and ax2) to visualize loss and accuracy separately.
    - The figure size is set to (14, 5) for better visibility.
3. Loss Visualization:
    - Panel 1 (ax1) is dedicated to plotting training and validation loss over epochs.
    - It plots the training loss and validation loss against the number of epochs.
    - X-axis represents the epochs, and Y-axis represents the loss values.
    - It includes labels, title, and a legend for clarity.
    - A grid is added to aid visualization.
4. Accuracy Visualization:
    - Panel 2 (ax2) is dedicated to plotting training and validation accuracy over epochs.
    - It plots the training accuracy and validation accuracy against the number of epochs.
    - X-axis represents the epochs, and Y-axis represents the accuracy values.
    - It includes labels, title, and a legend for clarity.
    - A grid is added to aid visualization.
5. Displaying the Plot:
    - The plt.show() command displays the entire plot with both panels.

4.2.13. Evaluating the model's performance on a sentiment classification task, including the calculation of a confusion matrix, F1 score, and a classification report.



```
y_compound_pred, y_sentiment_pred = lstm_model.predict(X_test)

y_sentiment_true = np.argmax(Y_sentiment_test, axis=1)
y_sentiment_pred = np.argmax(y_sentiment_pred, axis=1)

# Calculating confusion matrix for sentiment
cm = confusion_matrix(y_sentiment_true, y_sentiment_pred)

# Calculating F1 score for sentiment
f1 = f1_score(y_sentiment_true, y_sentiment_pred, average='weighted')

# Generating classification report for sentiment
report = classification_report(y_sentiment_true, y_sentiment_pred)

# Creating a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=True,
yticklabels=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

# Printing F1 score and classification report
print("Weighted F1 Score:", f1)
print("\nClassification Report:\n", report)
```

Fig 4.2.13: Represents Code for Evaluation of model performance

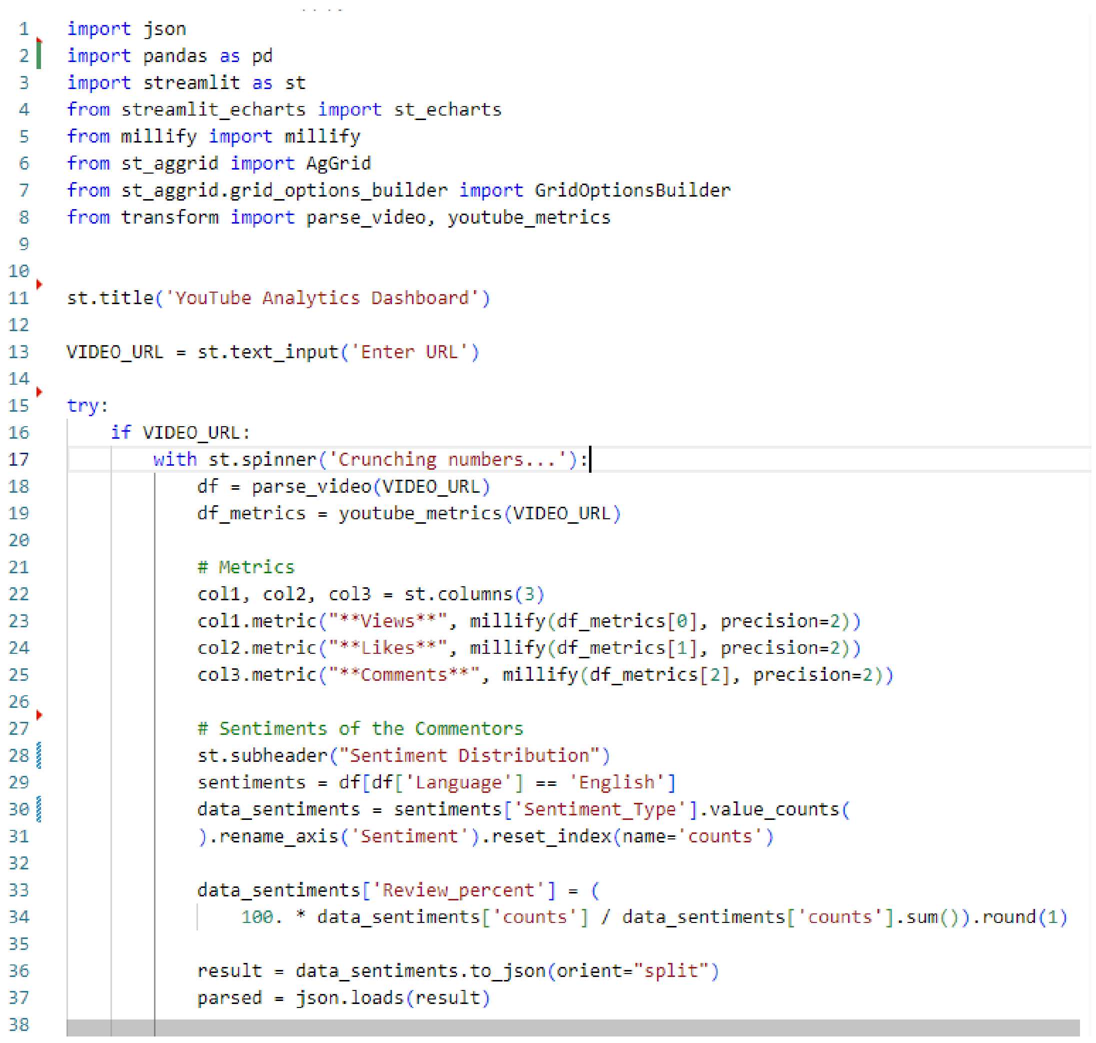
This code evaluates the performance of a sentiment analysis model on a test dataset. Here's a breakdown:

1. **Model Prediction:**
   - **model.predict(X_test)**: The model makes predictions on the test data (**X_test**), resulting in predicted values for both compound sentiment (**y_compound_pred**) and sentiment labels (**y_sentiment_pred**).
2. **True Sentiment Labels:**
   - **y_sentiment_true = np.argmax(Y_sentiment_test, axis=1)**: Converts one-hot encoded true sentiment labels (**Y_sentiment_test**) to their original integer representation.

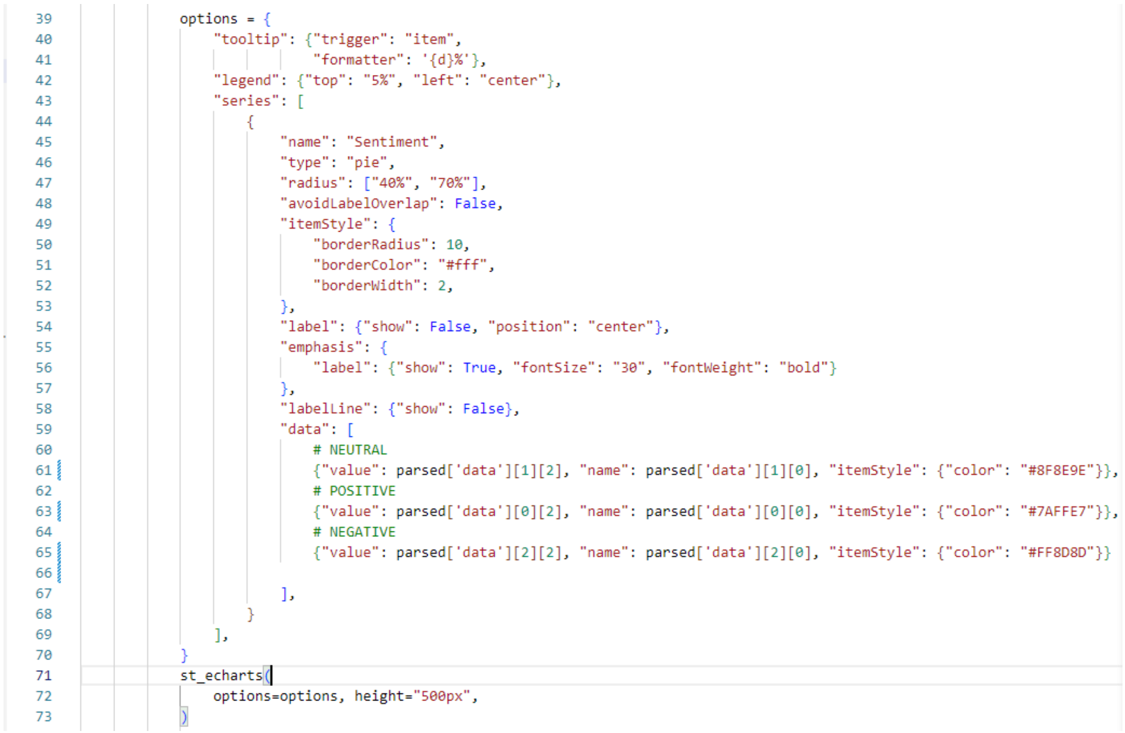
3. **Confusion Matrix:**
   - **confusion_matrix(y_sentiment_true, y_sentiment_pred)**: Computes a confusion matrix, a table showing the count of true positive, true negative, false positive, and false negative predictions for each sentiment class.
4. **F1 Score:**
   - **f1_score(y_sentiment_true, y_sentiment_pred, average='weighted')**: Calculates the weighted F1 score, which is a metric that combines precision and recall. It is computed for each sentiment class and then averaged using weights.
5. **Classification Report:**
   - **classification_report(y_sentiment_true, y_sentiment_pred)**: Generates a detailed classification report, including precision, recall, and F1 score, for each sentiment class.
6. **Confusion Matrix Heatmap:**
   - The code creates a heatmap using Seaborn (**sns**) to visually represent the confusion matrix. It includes annotations with the count values and uses a blue color map.
7. **Displaying Results:**
   - The heatmap is displayed with axis labels and a title to provide a visual representation of the model's performance.
8. **Printing Metrics:**
   - The weighted F1 score and the classification report are printed to provide a summary of the sentiment analysis model's evaluation on the test dataset.

4.2.14 User Interface:

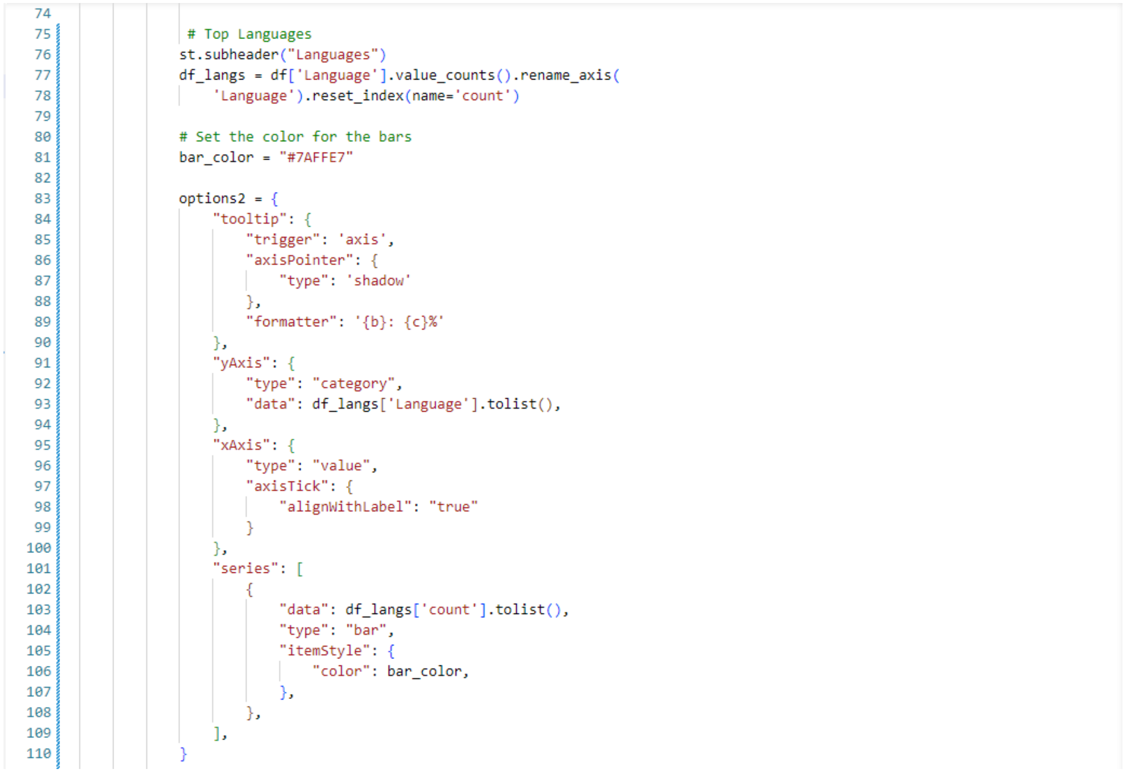
app.py:



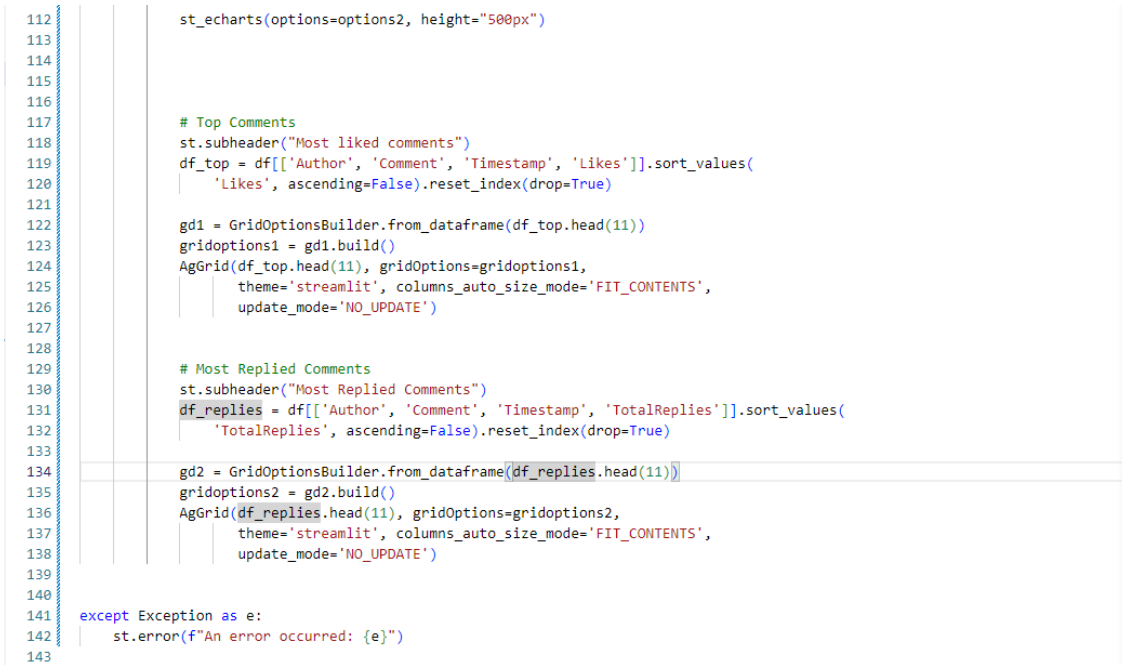
```python
import json
import pandas as pd
import streamlit as st
from streamlit_echarts import st_echarts
from millify import millify
from st_aggrid import AgGrid
from st_aggrid.grid_options_builder import GridOptionsBuilder
from transform import parse_video, youtube_metrics


st.title('YouTube Analytics Dashboard')

VIDEO_URL = st.text_input('Enter URL')

try:
    if VIDEO_URL:
        with st.spinner('Crunching numbers...'):
            df = parse_video(VIDEO_URL)
            df_metrics = youtube_metrics(VIDEO_URL)

            # Metrics
            col1, col2, col3 = st.columns(3)
            col1.metric("**Views**", millify(df_metrics[0], precision=2))
            col2.metric("**Likes**", millify(df_metrics[1], precision=2))
            col3.metric("**Comments**", millify(df_metrics[2], precision=2))

            # Sentiments of the Commentors
            st.subheader("Sentiment Distribution")
            sentiments = df[df['Language'] == 'English']
            data_sentiments = sentiments['Sentiment_Type'].value_counts(
            ).rename_axis('Sentiment').reset_index(name='counts')

            data_sentiments['Review_percent'] = (
                100. * data_sentiments['counts'] / data_sentiments['counts'].sum()).round(1)

            result = data_sentiments.to_json(orient="split")
            parsed = json.loads(result)

```



```python
options = {
    "tooltip": {"trigger": "item",
                "formatter": '{d}%'},
    "legend": {"top": "5%", "left": "center"},
    "series": [
        {
            "name": "Sentiment",
            "type": "pie",
            "radius": ["40%", "70%"],
            "avoidLabelOverlap": False,
            "itemStyle": {
                "borderRadius": 10,
                "borderColor": "#fff",
                "borderWidth": 2,
            },
            "label": {"show": False, "position": "center"},
            "emphasis": {
                "label": {"show": True, "fontSize": "30", "fontWeight": "bold"}
            },
            "labelLine": {"show": False},
            "data": [
                # NEUTRAL
                {"value": parsed['data'][1][2], "name": parsed['data'][1][0], "itemStyle": {"color": "#8F8E9E"}},
                # POSITIVE
                {"value": parsed['data'][0][2], "name": parsed['data'][0][0], "itemStyle": {"color": "#7AFFE7"}},
                # NEGATIVE
                {"value": parsed['data'][2][2], "name": parsed['data'][2][0], "itemStyle": {"color": "#FF8D8D"}}

            ],
        }
    ],
}
st_echarts(
    options=options, height="500px",
)
```

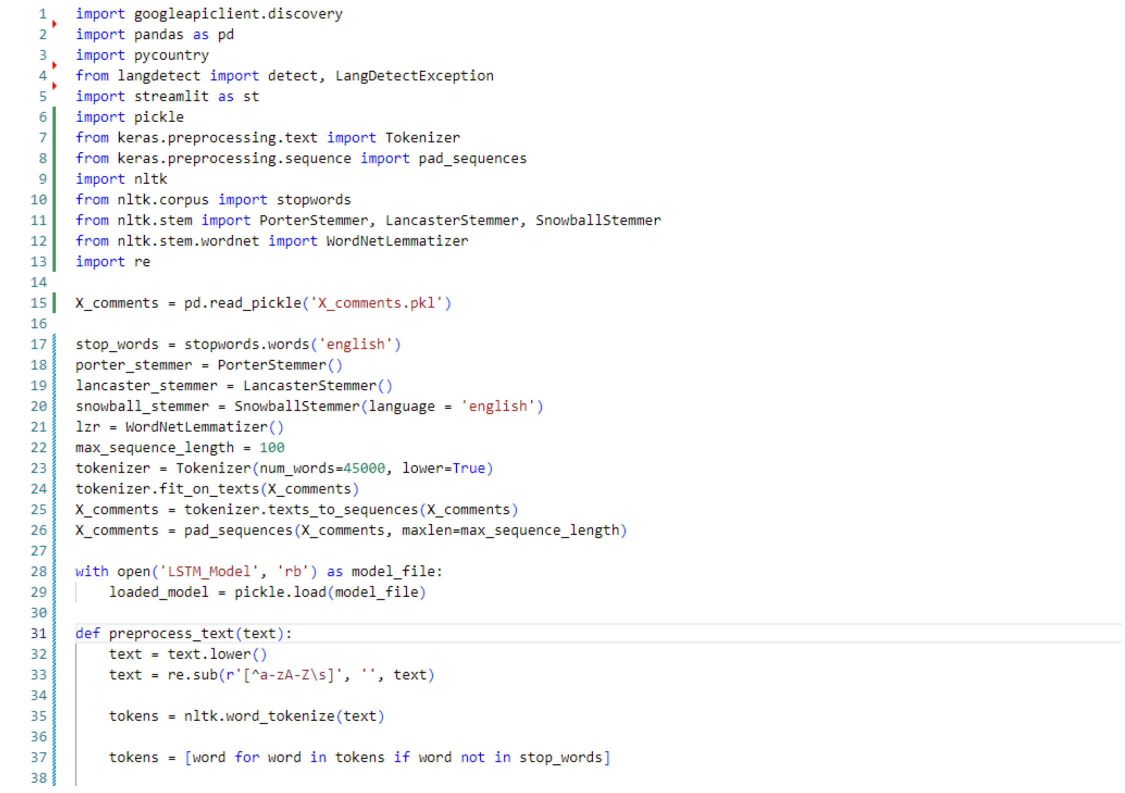


```python

 # Top Languages
st.subheader("Languages")
df_langs = df['Language'].value_counts().rename_axis(
    'Language').reset_index(name='count')

# Set the color for the bars
bar_color = "#7AFFE7"

options2 = {
    "tooltip": {
        "trigger": 'axis',
        "axisPointer": {
            "type": 'shadow'
        },
        "formatter": '{b}: {c}%'
    },
    "yAxis": {
        "type": "category",
        "data": df_langs['Language'].tolist(),
    },
    "xAxis": {
        "type": "value",
        "axisTick": {
            "alignWithLabel": "true"
        }
    },
    "series": [
        {
            "data": df_langs['count'].tolist(),
            "type": "bar",
            "itemStyle": {
                "color": bar_color,
            },
        },
    ],
}
```

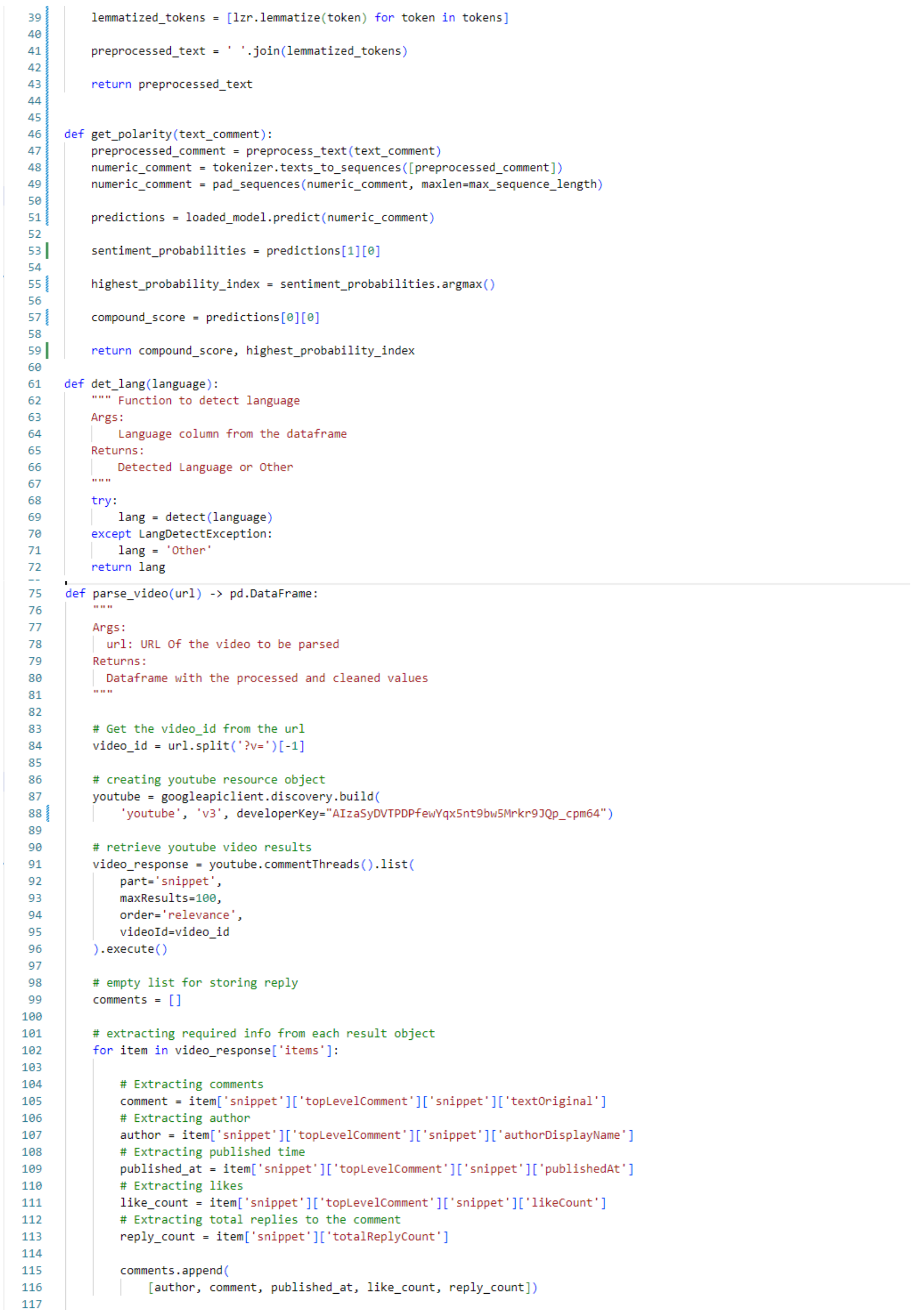


```python
            st_echarts(options=options2, height="500px")




            # Top Comments
            st.subheader("Most liked comments")
            df_top = df[['Author', 'Comment', 'Timestamp', 'Likes']].sort_values(
                'Likes', ascending=False).reset_index(drop=True)

            gd1 = GridOptionsBuilder.from_dataframe(df_top.head(11))
            gridoptions1 = gd1.build()
            AgGrid(df_top.head(11), gridOptions=gridoptions1,
                   theme='streamlit', columns_auto_size_mode='FIT_CONTENTS',
                   update_mode='NO_UPDATE')


            # Most Replied Comments
            st.subheader("Most Replied Comments")
            df_replies = df[['Author', 'Comment', 'Timestamp', 'TotalReplies']].sort_values(
                'TotalReplies', ascending=False).reset_index(drop=True)

            gd2 = GridOptionsBuilder.from_dataframe(df_replies.head(11))
            gridoptions2 = gd2.build()
            AgGrid(df_replies.head(11), gridOptions=gridoptions2,
                   theme='streamlit', columns_auto_size_mode='FIT_CONTENTS',
                   update_mode='NO_UPDATE')


except Exception as e:
    st.error(f"An error occurred: {e}")

```

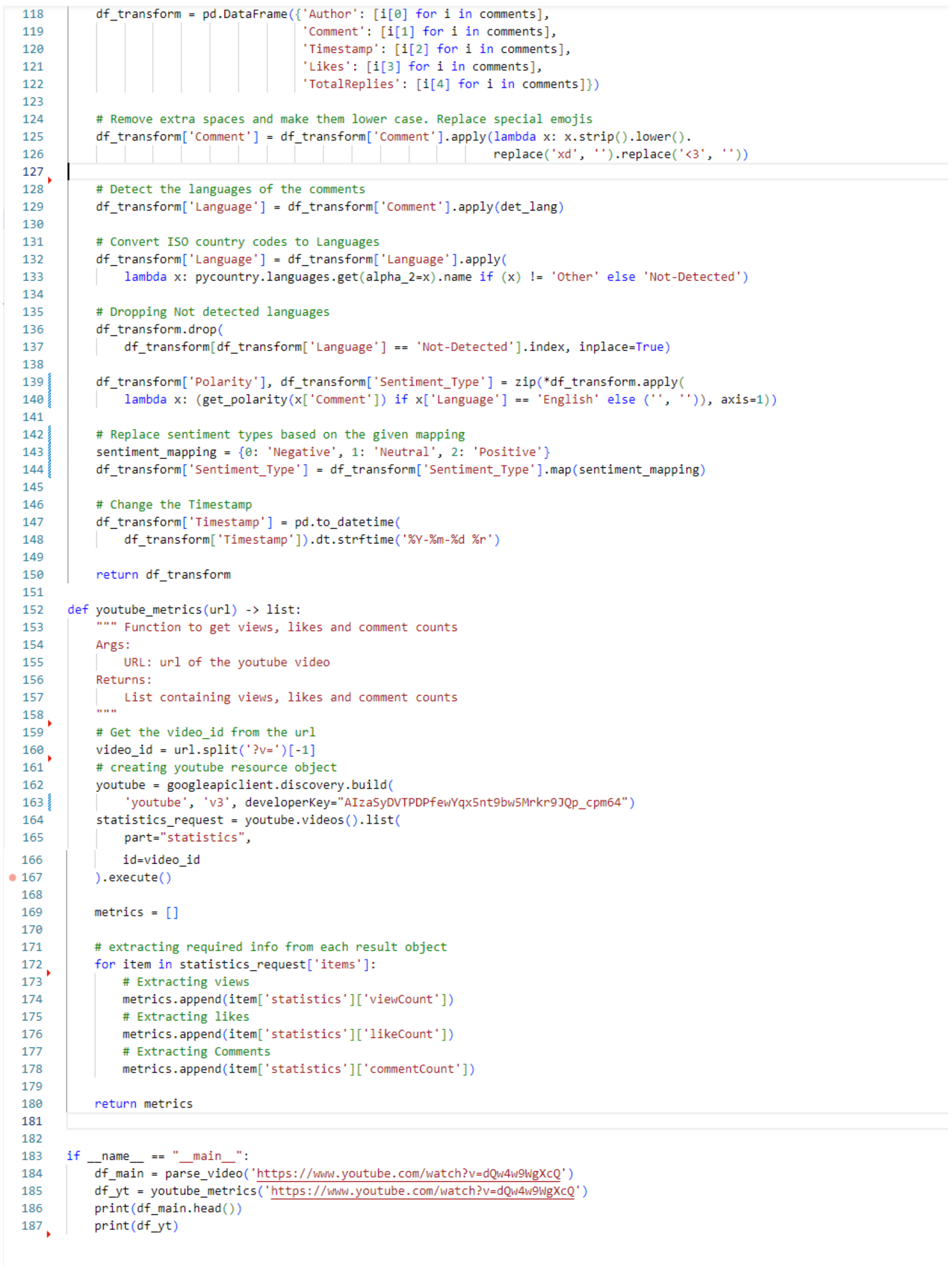
transform.py:



```python
import googleapiclient.discovery
import pandas as pd
import pycountry
from langdetect import detect, LangDetectException
import streamlit as st
import pickle
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
import re

X_comments = pd.read_pickle('X_comments.pkl')

stop_words = stopwords.words('english')
porter_stemmer = PorterStemmer()
lancaster_stemmer = LancasterStemmer()
snowball_stemmer = SnowballStemmer(language = 'english')
lzr = WordNetLemmatizer()
max_sequence_length = 100
tokenizer = Tokenizer(num_words=45000, lower=True)
tokenizer.fit_on_texts(X_comments)
X_comments = tokenizer.texts_to_sequences(X_comments)
X_comments = pad_sequences(X_comments, maxlen=max_sequence_length)

with open('LSTM_Model', 'rb') as model_file:
    loaded_model = pickle.load(model_file)

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    tokens = nltk.word_tokenize(text)

    tokens = [word for word in tokens if word not in stop_words]

```



```python
    lemmatized_tokens = [lzr.lemmatize(token) for token in tokens]

    preprocessed_text = ' '.join(lemmatized_tokens)

    return preprocessed_text


def get_polarity(text_comment):
    preprocessed_comment = preprocess_text(text_comment)
    numeric_comment = tokenizer.texts_to_sequences([preprocessed_comment])
    numeric_comment = pad_sequences(numeric_comment, maxlen=max_sequence_length)

    predictions = loaded_model.predict(numeric_comment)

    sentiment_probabilities = predictions[1][0]

    highest_probability_index = sentiment_probabilities.argmax()

    compound_score = predictions[0][0]

    return compound_score, highest_probability_index

def det_lang(language):
    """ Function to detect language
    Args:
        Language column from the dataframe
    Returns:
        Detected Language or Other
    """
    try:
        lang = detect(language)
    except LangDetectException:
        lang = 'Other'
    return lang

def parse_video(url) -> pd.DataFrame:
    """
    Args:
      url: URL Of the video to be parsed
    Returns:
      Dataframe with the processed and cleaned values
    """

    # Get the video_id from the url
    video_id = url.split('?v=')[-1]

    # creating youtube resource object
    youtube = googleapiclient.discovery.build(
        'youtube', 'v3', developerKey="AIzaSyDVTPDPfewYqx5nt9bw5Mrkr9JQp_cpm64")

    # retrieve youtube video results
    video_response = youtube.commentThreads().list(
        part='snippet',
        maxResults=100,
        order='relevance',
        videoId=video_id
    ).execute()

    # empty list for storing reply
    comments = []

    # extracting required info from each result object
    for item in video_response['items']:

        # Extracting comments
        comment = item['snippet']['topLevelComment']['snippet']['textOriginal']
        # Extracting author
        author = item['snippet']['topLevelComment']['snippet']['authorDisplayName']
        # Extracting published time
        published_at = item['snippet']['topLevelComment']['snippet']['publishedAt']
        # Extracting likes
        like_count = item['snippet']['topLevelComment']['snippet']['likeCount']
        # Extracting total replies to the comment
        reply_count = item['snippet']['totalReplyCount']

        comments.append(
            [author, comment, published_at, like_count, reply_count])

```



```
    df_transform = pd.DataFrame({'Author': [i[0] for i in comments],
                                 'Comment': [i[1] for i in comments],
                                 'Timestamp': [i[2] for i in comments],
                                 'Likes': [i[3] for i in comments],
                                 'TotalReplies': [i[4] for i in comments]})

    # Remove extra spaces and make them lower case. Replace special emojis
    df_transform['Comment'] = df_transform['Comment'].apply(lambda x: x.strip().lower().
                                                            replace('xd', '').replace('<3', ''))

    # Detect the languages of the comments
    df_transform['Language'] = df_transform['Comment'].apply(det_lang)

    # Convert ISO country codes to Languages
    df_transform['Language'] = df_transform['Language'].apply(
        lambda x: pycountry.languages.get(alpha_2=x).name if (x) != 'Other' else 'Not-Detected')

    # Dropping Not detected languages
    df_transform.drop(
        df_transform[df_transform['Language'] == 'Not-Detected'].index, inplace=True)

    df_transform['Polarity'], df_transform['Sentiment_Type'] = zip(*df_transform.apply(
        lambda x: (get_polarity(x['Comment']) if x['Language'] == 'English' else ('', '')), axis=1))

    # Replace sentiment types based on the given mapping
    sentiment_mapping = {0: 'Negative', 1: 'Neutral', 2: 'Positive'}
    df_transform['Sentiment_Type'] = df_transform['Sentiment_Type'].map(sentiment_mapping)

    # Change the Timestamp
    df_transform['Timestamp'] = pd.to_datetime(
        df_transform['Timestamp']).dt.strftime('%Y-%m-%d %r')

    return df_transform

def youtube_metrics(url) -> list:
    """ Function to get views, likes and comment counts
    Args:
        URL: url of the youtube video
    Returns:
        List containing views, likes and comment counts
    """
    # Get the video_id from the url
    video_id = url.split('?v=')[-1]
    # creating youtube resource object
    youtube = googleapiclient.discovery.build(
        'youtube', 'v3', developerKey="AIzaSyDVTPDPfewYqx5nt9bw5Mrkr9JQp_cpm64")
    statistics_request = youtube.videos().list(
        part="statistics",
        id=video_id
    ).execute()

    metrics = []

    # extracting required info from each result object
    for item in statistics_request['items']:
        # Extracting views
        metrics.append(item['statistics']['viewCount'])
        # Extracting likes
        metrics.append(item['statistics']['likeCount'])
        # Extracting Comments
        metrics.append(item['statistics']['commentCount'])

    return metrics


if __name__ == "__main__":
    df_main = parse_video('https://www.youtube.com/watch?v=dQw4w9WgXcQ')
    df_yt = youtube_metrics('https://www.youtube.com/watch?v=dQw4w9WgXcQ')
    print(df_main.head())
    print(df_yt)
```

Fig 4.2.14 : Code for User Interface

This creates a YouTube Analytics Dashboard for a given YouTube video URL. Here's an explanation of the key components:

1. **User Input:**
   - The user inputs a YouTube video URL using **st.text_input('Enter URL')**.
2. **Data Processing:**
   - If a video URL is provided, the script uses the **parse_video** and **youtube_metrics** functions to retrieve information about the video's comments and metrics, respectively.
3. **Display Metrics:**
   - Metrics such as views, likes, and comments are displayed in three columns using **st.metric**.
4. **Sentiment Distribution Pie Chart:**
   - The sentiment distribution of English comments is visualized using a pie chart created with ECharts and **st_echarts**. The sentiment categories include Positive (green), Neutral (gray), and Negative (red).
5. **Language Distribution Bar Chart:**
   - The distribution of languages in the comments is displayed as a horizontal bar chart. The color of the bars is set to a teal shade.
6. **Top Liked Comments:**
   - The most liked comments are displayed in an AgGrid table, showing details such as author, comment, timestamp, and likes.
7. **Most Replied Comments:**
   - The most replied comments are displayed in another AgGrid table, showing details such as author, comment, timestamp, and total replies.
8. **Error Handling:**
   - Exception handling is implemented to catch errors and display error messages using **st.error**.
9. **Streamlit App Execution:**
   - The script is designed to be run with Streamlit, and it updates the UI dynamically based on user input.

5.Result

5.1 Model Evaluation:

5.1.1. Evaluating Training History:

5.1.1(1). LSTM:

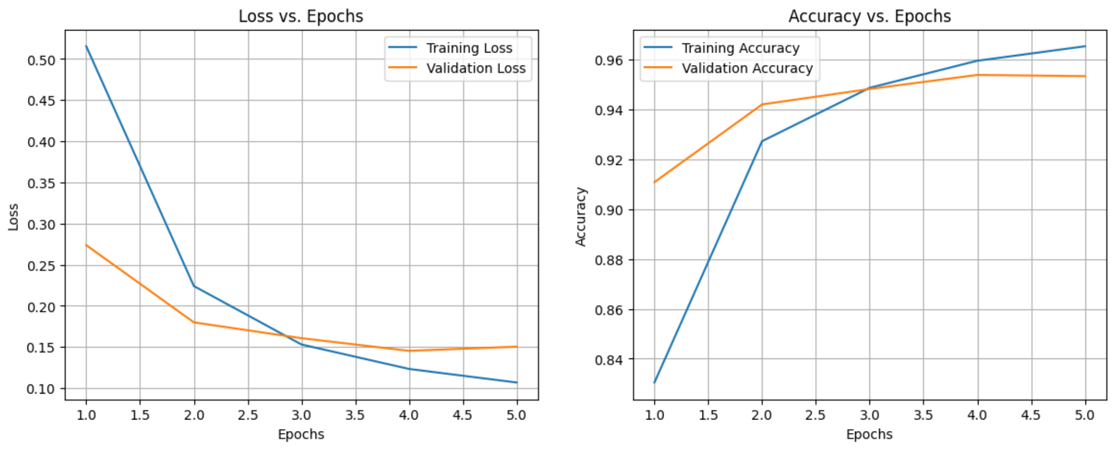


Fig5.1.1(1): Represents Training History of LSTM model

1. **Epochs:**
   - The model is trained for five epochs, meaning the entire training dataset is processed five times.
2. **Training Metrics (per epoch):**
   - Loss:
     - Epoch 1: 0.5154
     - Epoch 2: 0.2241
     - Epoch 3: 0.1531
     - Epoch 4: 0.1234
     - Epoch 5: 0.1069
   - Compound Output MAE (Mean Absolute Error):
     - Epoch 1: 0.1642

        - Epoch 2: 0.0914
        - Epoch 3: 0.0693
        - Epoch 4: 0.0622
        - Epoch 5: 0.0583
    - Sentiment Label Output Accuracy:
        - Epoch 1: 0.8304
        - Epoch 2: 0.9273
        - Epoch 3: 0.9488
        - Epoch 4: 0.9597
        - Epoch 5: 0.9655

3. **Observations**:
    - Both training and validation loss decrease over epochs, suggesting the model is improving.
    - The accuracy for sentiment label predictions increases, indicating improved classification performance.
    - Compound output MAE decreases, showing improved accuracy in predicting the compound sentiment score.

4. **Overall**:
    - The training history suggests that the model is learning effectively and generalizing well to unseen data, as indicated by the improvement in both training and validation metrics over the five epochs.

5.1.1(2). MLPNN:

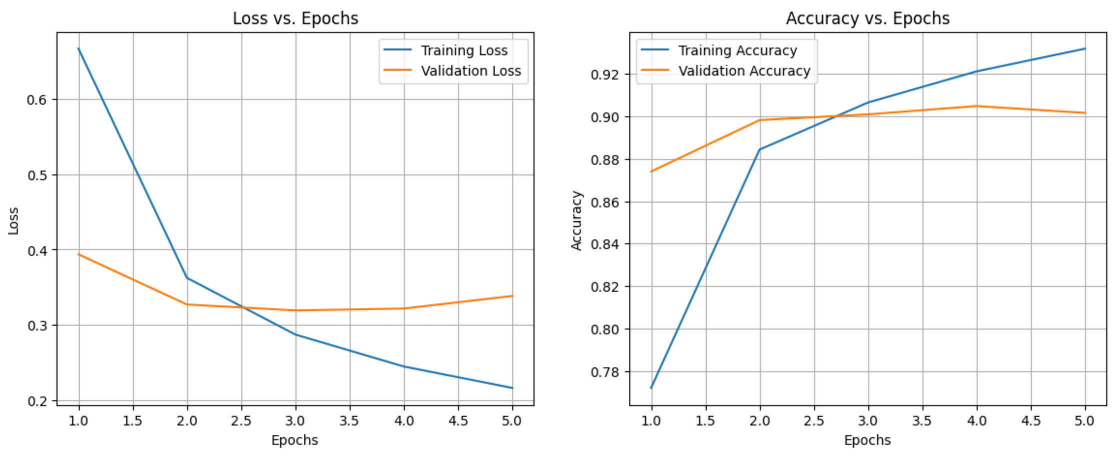


Fig 5.1.1(2): Represents Training History of MPLNN model

1. **Epochs:**
    - The model is trained for five epochs.
2. **Training Metrics (per epoch):**
    - **Loss:**
        - Epoch 1: 0.6668
        - Epoch 2: 0.3623
        - Epoch 3: 0.2870
        - Epoch 4: 0.2446
        - Epoch 5: 0.2161
3. **Validation Metrics (per epoch):**
    - **Validation Loss:**
        - Epoch 1: 0.3936
        - Epoch 2: 0.3269
        - Epoch 3: 0.3191
        - Epoch 4: 0.3216
        - Epoch 5: 0.3383

4. **Observations:**

   - Training loss and validation loss decrease over epochs, suggesting the model is improving.
   - The compound output accuracy remains relatively constant, indicating challenges in predicting the compound sentiment score.
   - Sentiment label output accuracy increases, suggesting improved performance in sentiment classification.

5. **Overall:**

   - The training history shows a potential challenge in predicting the compound sentiment score (indicated by low accuracy). However, the model demonstrates improvement in sentiment label classification accuracy over the five epochs. Further analysis may be needed to address issues with the compound sentiment score prediction.

5.1.1(3). CNN:

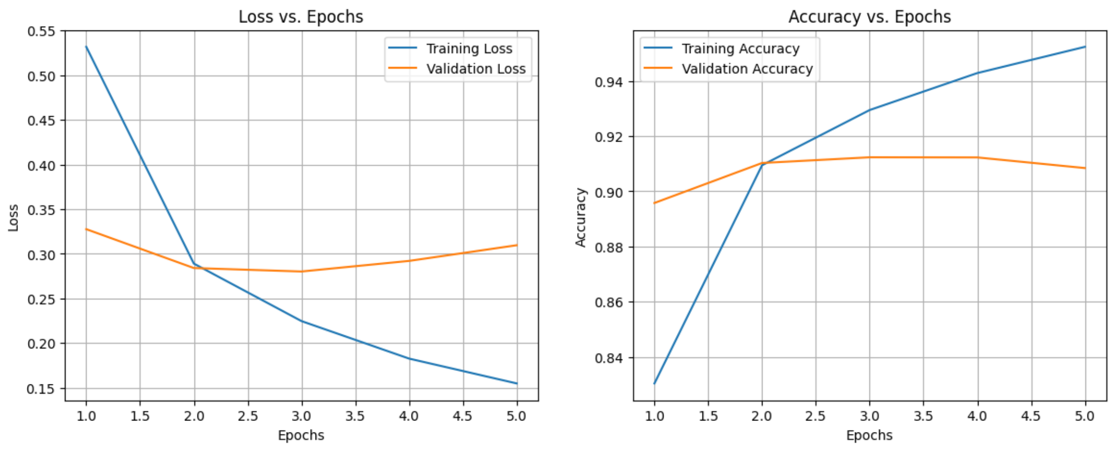


Fig 5.1.1(3): Represents Training History of CNN model

1. **Epochs:**

   - The model is trained for at least four epochs, and the information about the fifth epoch is not complete.

2. **Training Metrics (per epoch):**

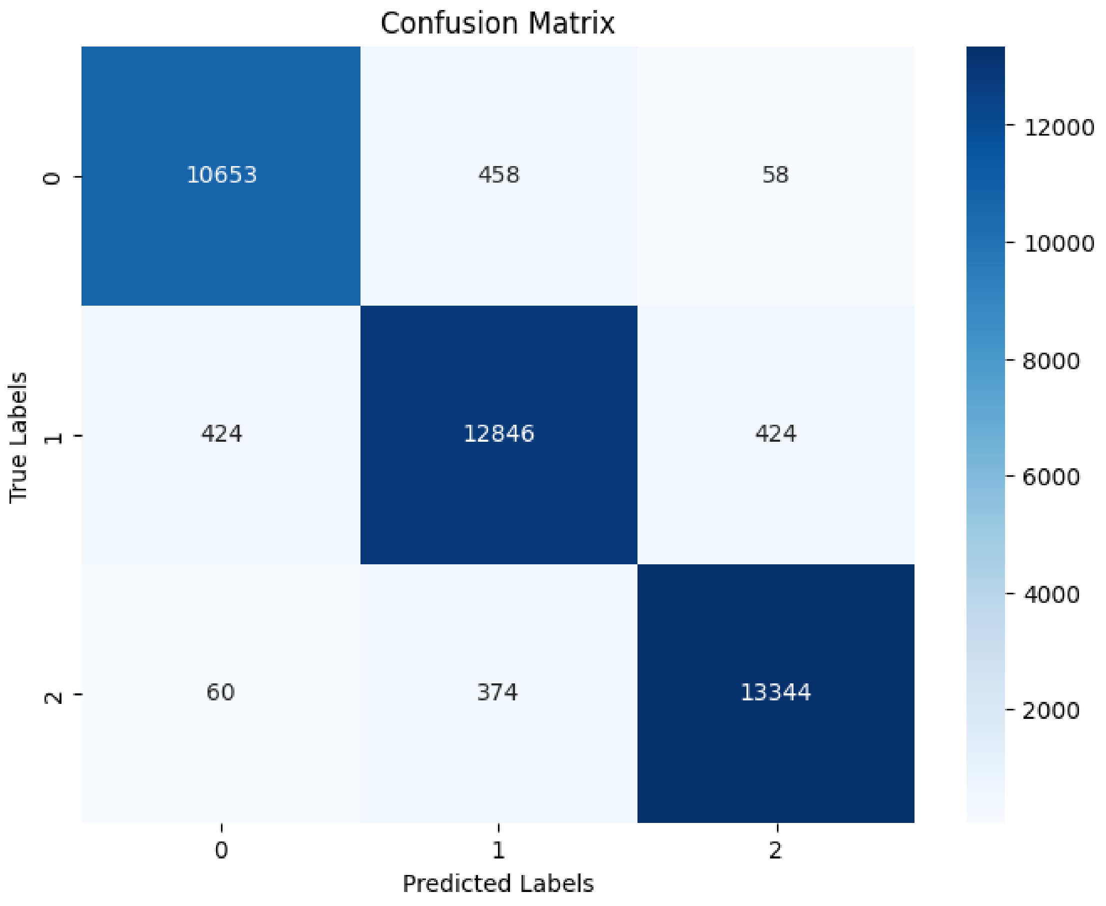
- **Loss:**
    - Epoch 1: 0.5318
    - Epoch 2: 0.2891
    - Epoch 3: 0.2247
    - Epoch 4: 0.1827
    - Epoch 5: 0.1623

3. **Validation Metrics (per epoch):**
    - **Validation Loss:**
        - Epoch 1: 0.3277
        - Epoch 2: 0.2841
        - Epoch 3: 0.2802
        - Epoch 4: 0.2921
        - Epoch 5: 0.3123

4. **Observations:**
    - Training loss and validation loss decrease over epochs, indicating improvement in the model's overall performance.
    - Compound output accuracy remains constant at 0.2695, suggesting challenges in predicting the compound sentiment score.
    - Sentiment label output accuracy increases over epochs, reaching 0.9128 in the fourth epoch.

5.1.2. Evaluations Using Confusion Matrix:

5.1.2(1). LSTM:



```
Weighted F1 Score: 0.9534584533994811

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.95      0.96     11169
           1       0.94      0.94      0.94     13694
           2       0.97      0.97      0.97     13778

    accuracy                           0.95     38641
   macro avg       0.95      0.95      0.95     38641
weighted avg       0.95      0.95      0.95     38641
```

Fig 5.1.2(1): Represents Confusion matrix and Classification Report of LSTM model

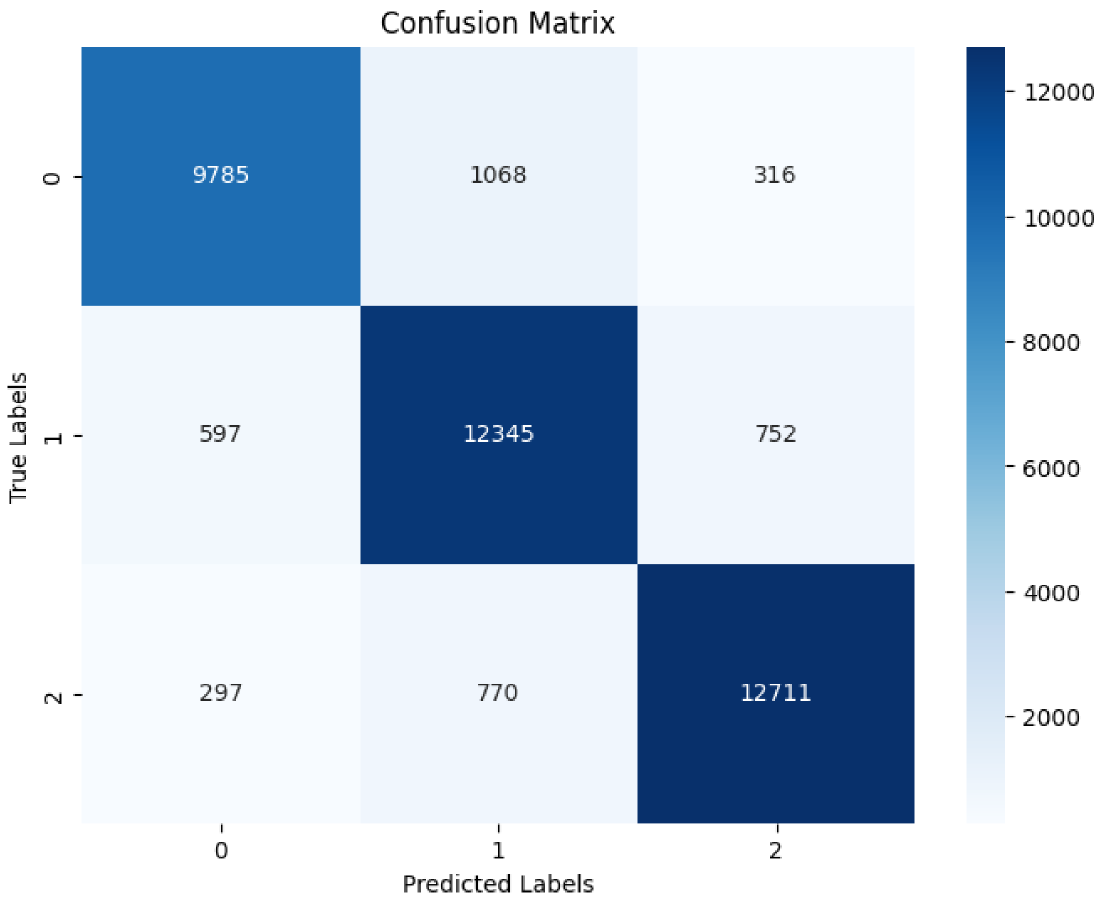
1. **Weighted F1 Score: 0.9534:**
    - The weighted F1 score is a metric that considers both precision and recall, providing a balanced measure of a model's performance across different classes. A score of 0.9534 indicates a high overall performance.
2. **Classification Report:**
    - The classification report provides detailed metrics for each class (0, 1, and 2) and overall performance.
    - **Precision:** The ability of the model not to mislabel a class. High precision indicates few false positives.
    - **Recall:** The ability of the model to capture all instances of a class. High recall indicates few false negatives.
    - **F1-Score:** The harmonic mean of precision and recall. It balances precision and recall, providing a single score.
    - **Support:** The number of actual occurrences of the class in the specified dataset.
3. **Per-Class Metrics:**
    - **Class 0:**
        - Precision: 0.96
        - Recall: 0.95
        - F1-Score: 0.96
        - Support: 11169
    - **Class 1:**
        - Precision: 0.94
        - Recall: 0.94
        - F1-Score: 0.94
        - Support: 13694
    - **Class 2:**
        - Precision: 0.97

- Recall: 0.97
- F1-Score: 0.97
- Support: 13778

4. **Overall Metrics:**
    - **Accuracy:** The proportion of correctly classified instances out of the total instances. In this case, the overall accuracy is 0.95, or 95%.
    - **Macro Avg:** The average of precision, recall, and F1-score across all classes. It provides equal weight to each class.
    - **Weighted Avg:** The average of precision, recall, and F1-score, with each class contribution weighted by its support. It is a useful metric when dealing with imbalanced datasets.
5. **Summary:**
    - The model performs well across all classes, with high precision, recall, and F1-scores.
    - The weighted F1 score of 0.9534 and overall accuracy of 95% suggest strong overall performance on the test dataset.
    - The support values indicate the distribution of instances across different classes.

In conclusion, the model demonstrates effective classification performance with high accuracy and balanced metrics across multiple classes.

5.1.2(2). MLPNN:



```
Weighted F1 Score: 0.9017221535906149

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.88      0.90     11169
           1       0.87      0.90      0.89     13694
           2       0.92      0.92      0.92     13778

    accuracy                           0.90     38641
   macro avg       0.90      0.90      0.90     38641
weighted avg       0.90      0.90      0.90     38641
```

Fig 5.1.2(2): Represents Confusion matrix and Classification Report of MLPNN model

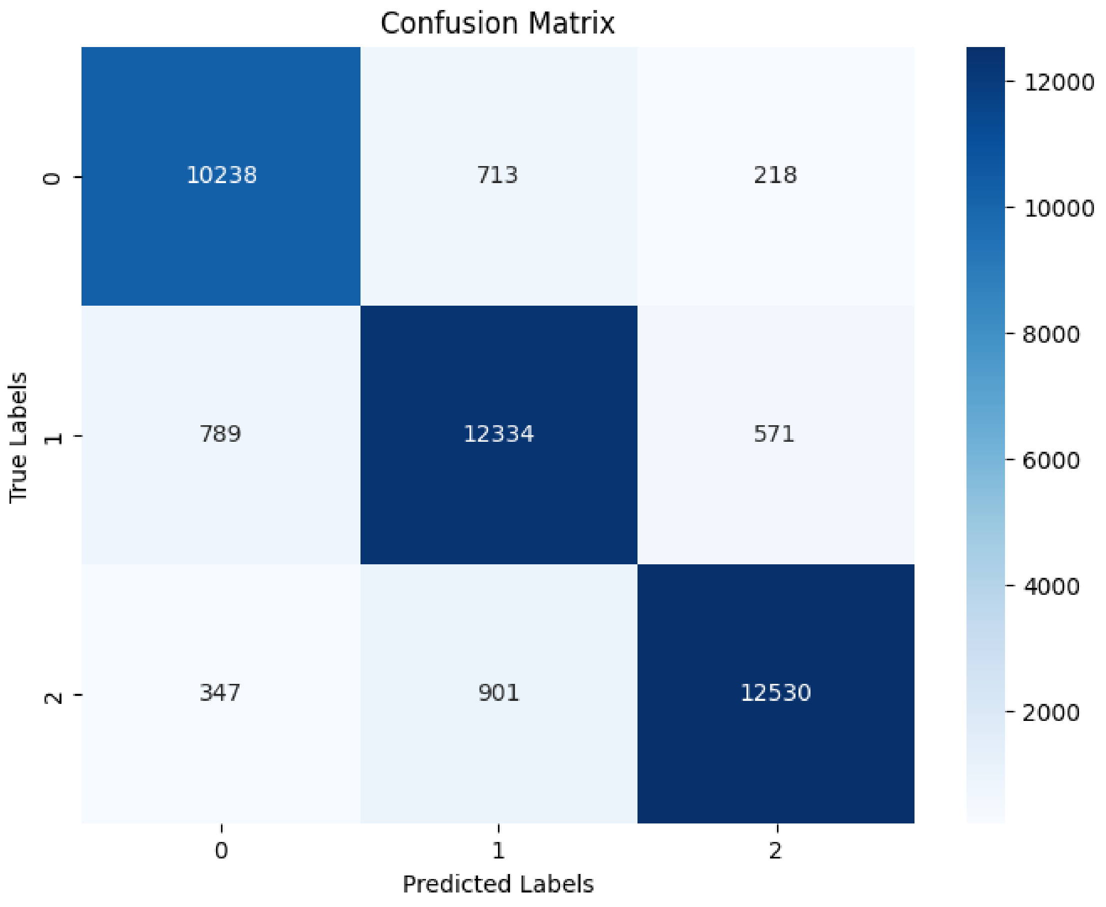
1. **Weighted F1 Score: 0.9017:**
    - The weighted F1 score is a composite metric that balances precision and recall, considering the class distribution. A score of 0.9017 indicates a strong overall performance.
2. **Classification Report:**
    - Detailed metrics for each class (0, 1, and 2) and overall performance.
    - **Precision:** The ability of the model not to mislabel a class.
    - **Recall:** The ability of the model to capture all instances of a class.
    - **F1-Score:** The harmonic mean of precision and recall, balancing both metrics.
    - **Support:** The number of actual occurrences of the class in the specified dataset.
3. **Per-Class Metrics:**
    - **Class 0:**
        - Precision: 0.92
        - Recall: 0.88
        - F1-Score: 0.90
        - Support: 11169
    - **Class 1:**
        - Precision: 0.87
        - Recall: 0.90
        - F1-Score: 0.89
        - Support: 13694
    - **Class 2:**

- Precision: 0.92
- Recall: 0.92
- F1-Score: 0.92
- Support: 13778

4. **Overall Metrics:**
   - **Accuracy:** The proportion of correctly classified instances out of the total instances. In this case, the overall accuracy is 0.90, or 90%.
   - **Macro Avg:** The average of precision, recall, and F1-score across all classes. It provides equal weight to each class.
   - **Weighted Avg:** The average of precision, recall, and F1-score, with each class contribution weighted by its support. It is useful for imbalanced datasets.
5. **Summary:**
   - The model performs well across all classes, with high precision, recall, and F1-scores.
   - The weighted F1 score of 0.9017 and overall accuracy of 90% indicate solid performance on the test dataset.
   - The support values give insight into the distribution of instances across different classes.

In conclusion, the model demonstrates effective classification performance with strong overall metrics and balanced performance across multiple classes.

5.1.2(3). CNN:



```
Weighted F1 Score: 0.9085636958282285

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.92      0.91     11169
           1       0.88      0.90      0.89     13694
           2       0.94      0.91      0.92     13778

    accuracy                           0.91     38641
   macro avg       0.91      0.91      0.91     38641
weighted avg       0.91      0.91      0.91     38641
```

Fig 5.1.2(3): Represents Confusion matrix and Classification Report of CNN model

1. **Weighted F1 Score: 0.9086:**
    - The weighted F1 score is a composite metric that balances precision and recall, considering the class distribution. A score of 0.9086 indicates a strong overall performance.
2. **Classification Report:**
    - Detailed metrics for each class (0, 1, and 2) and overall performance.
    - **Precision:** The ability of the model not to mislabel a class.
    - **Recall:** The ability of the model to capture all instances of a class.
    - **F1-Score:** The harmonic mean of precision and recall, balancing both metrics.
    - **Support:** The number of actual occurrences of the class in the specified dataset.
3. **Per-Class Metrics:**
    - **Class 0:**
        - Precision: 0.90
        - Recall: 0.92
        - F1-Score: 0.91
        - Support: 11169
    - **Class 1:**
        - Precision: 0.88
        - Recall: 0.90
        - F1-Score: 0.89
        - Support: 13694
    - **Class 2:**
        - Precision: 0.94
        - Recall: 0.91
        - F1-Score: 0.92

- Support: 13778

4. **Overall Metrics:**
    - **Accuracy:** The proportion of correctly classified instances out of the total instances. In this case, the overall accuracy is 0.91, or 91%.
    - **Macro Avg:** The average of precision, recall, and F1-score across all classes. It provides equal weight to each class.
    - **Weighted Avg:** The average of precision, recall, and F1-score, with each class contribution weighted by its support. It is useful for imbalanced datasets.
5. **Summary:**
    - The model performs well across all classes, with high precision, recall, and F1-scores.
    - The weighted F1 score of 0.9086 and overall accuracy of 91% indicate solid performance on the test dataset.
    - The support values give insight into the distribution of instances across different classes.

In conclusion, the model demonstrates effective classification performance with strong overall metrics and balanced performance across multiple classes, similar to the second model you previously provided.

Model	**Weighted F1 Score**	**Accuracy**	**Precision (Class 0, 1, 2)**	**Recall (Class 0, 1, 2)**	**F1-Score (Class 0, 1, 2)**
LSTM	0.9534	95%	0.96, 0.94, 0.97	0.95, 0.94, 0.97	0.96, 0.94, 0.97
MLPNN	0.9017	90%	0.92, 0.87, 0.92	0.88, 0.90, 0.92	0.90, 0.89, 0.92
CNN	0.9086	91%	0.90, 0.88, 0.94	0.92, 0.90, 0.91	0.91, 0.89, 0.92

Table5.1.2: Represents the comparison between the different models

- The LSTM model stands out as the top-performing model, providing the highest weighted F1 score and accuracy.
- The MLPNN and CNN models both demonstrate strong performance, with the CNN model slightly edging out the MLPNN model in terms of weighted F1 score and accuracy.
- The choice between these models would depend on specific considerations such as computational resources, training time, and interpretability, as each model shows competitive performance in different aspects.

In summary, the LSTM model appears to be the most robust and effective in this context, while the MLPNN and CNN models also offer viable alternatives with slightly different trade-offs.

5.2 USER INTERFACE:

5.2.1. Submit the YouTube video URL for analysis.

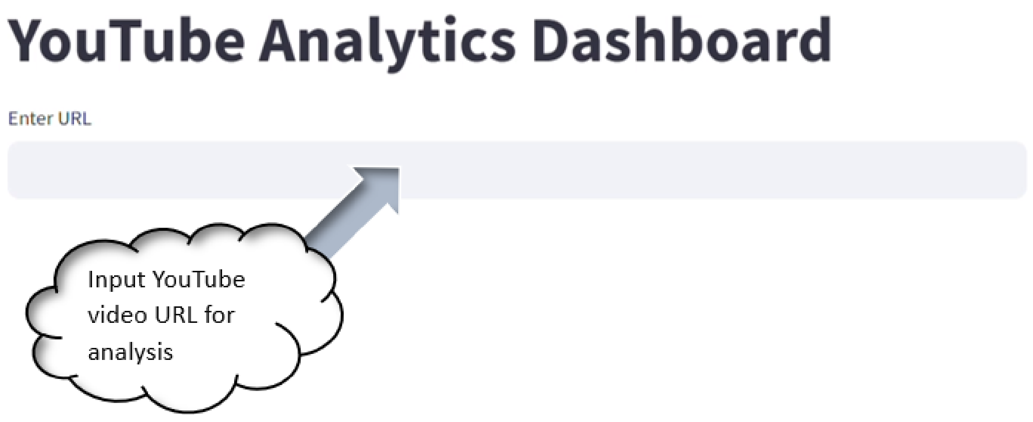
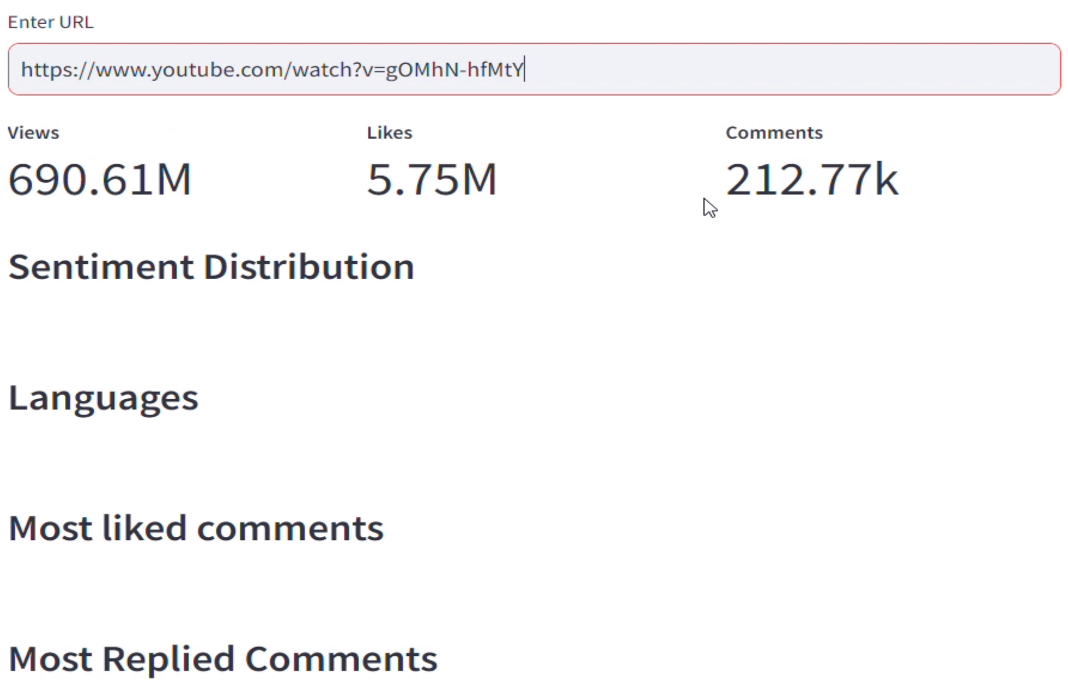

YouTube Analytics Dashboard
Enter URL
Input YouTube video URL for analysis

Fig 5.2.1: Represents User Interface which allows the user to submit the YouTube video URL for analysis

- Users are instructed to submit a YouTube video URL for analysis.
- **Input Section:** "Enter URL" indicates the dedicated space for users to input the YouTube video URL.
- **User Action:** The phrase "Submit the YouTube video URL for analysis" directs users to actively engage with the dashboard.
- **Progress Indicator:** "Crunching numbers..." communicates that the system is processing and analyzing the provided data, keeping users informed about the ongoing operation.

5.2.2. Information Displayed:


Enter URL

https://www.youtube.com/watch?v=gOMhN-hfMtY

Views 690.61M

Likes 5.75M

Comments 212.77k

Sentiment Distribution

Languages

Most liked comments

Most Replied Comments

Fig 5.2.2: Represents about the information displayed

Entered YouTube video URL: https://www.youtube.com/watch?v=gOMhN-hfMtY

Metrics:

- **Views:** 690.61M- Indicates the total number of times the video has been viewed.
- **Likes:** 5.75M- Represents the number of users who have liked the video.
- **Comments:** 212.77k- Shows the total count of comments on the video.

Analytics:

- **Sentiment Distribution:** Visualizes comments sentiment with a pie chart, showcasing the proportion of positive, negative, and neutral feedback.
- **Languages:** Presents a bar chart illustrating the distribution of comment languages, aiding in understanding the audience diversity.
- **Most Liked Comments:** Displays a ranked list of comments with the highest likes, spotlighting the community's favored contributions.

- **Most Replied Comments:** Features a ranked list of comments generating the most replies, offering insight into engaging discussions around the content.

5.2.3. Sentiment Distribution:

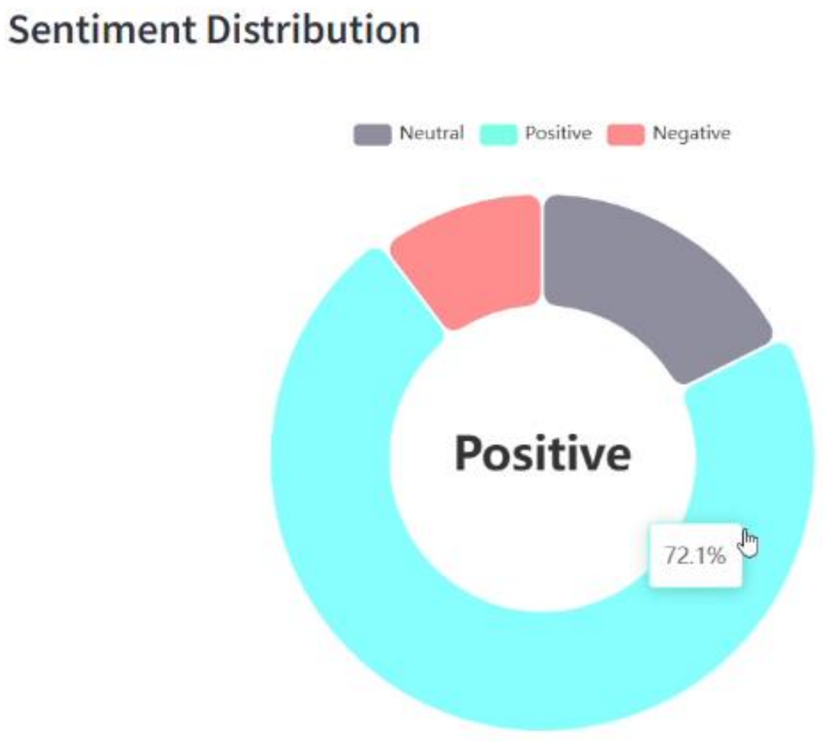


Fig 5.2.3: Represent pie chart displaying the Sentiment distribution of comments

- A pie chart visually representing the sentiment distribution of comments on the video.
- Sections of the pie chart denote different sentiment categories (positive, negative, neutral), with each section's size indicating the proportion of comments expressing that sentiment.
- **Neutral (17.6%):** The majority of comments (17.6%) express a neutral sentiment. These comments generally do not convey strong positive or negative emotions.
- **Positive (72.1%):** Approximately 72.1% of comments are positive in sentiment. These comments likely express favorable opinions or appreciation.
- **Negative (10.3%):** About 10.3% of comments convey a negative sentiment. These comments may express dissatisfaction, criticism, or disapproval.

5.2.4. <u>Languages:</u>

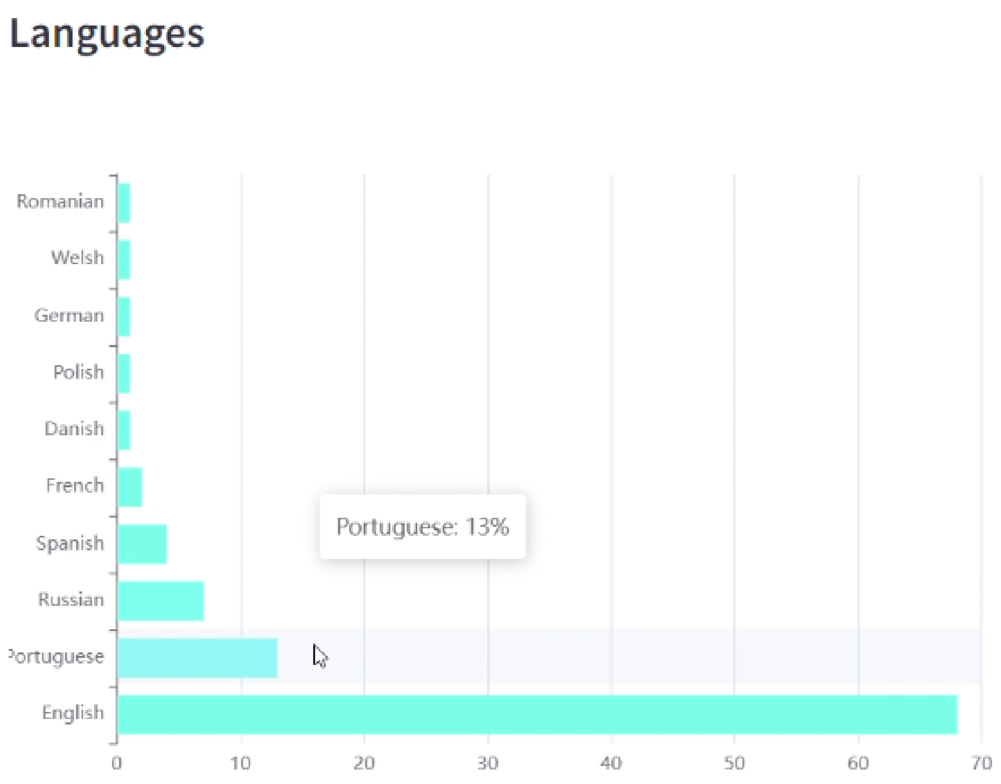
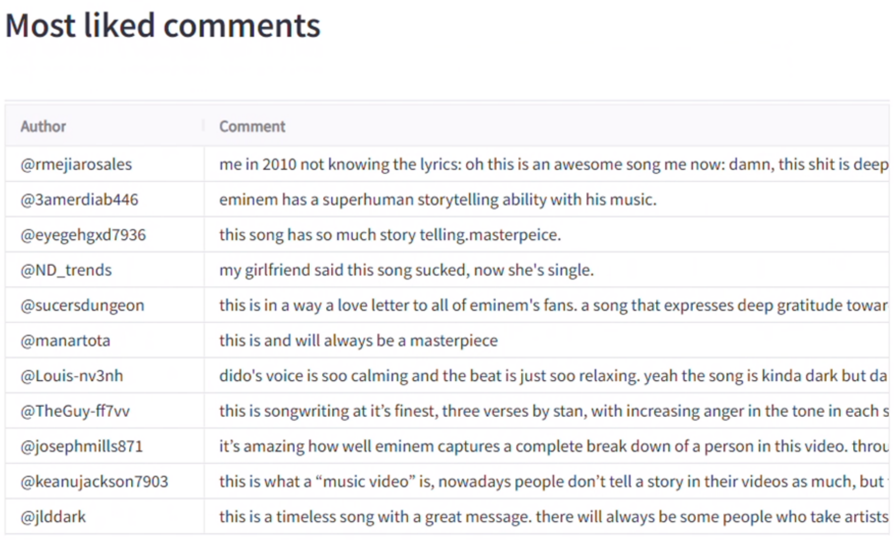


Fig 5.2.4: Represents bar chat showcasing the distribution of languages

- A bar chart showcasing the distribution of languages in which comments are written on the video.
- Each bar corresponds to a language, and the length of the bar represents the percentage or count of comments in that language.
- For instance, The majority of comments, 68%, are in English, 13% of the comments are written in Portuguese and 7% of the comments are written in Russian etc.

5.2.5. Most Liked Comments :



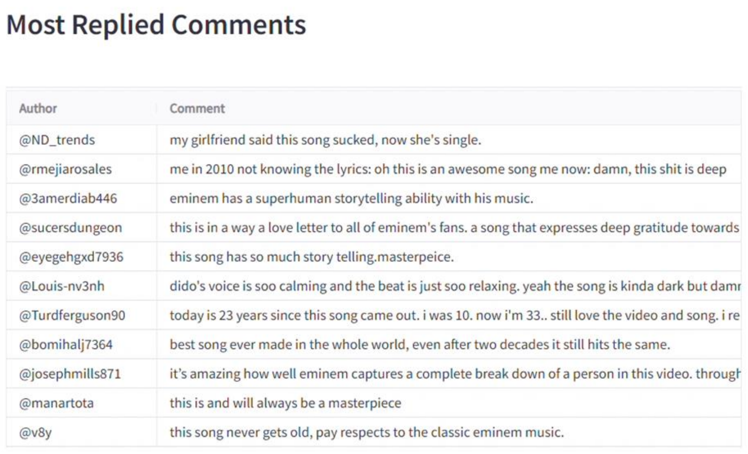
Most liked comments

Author	Comment
@rmejiarosales	me in 2010 not knowing the lyrics: oh this is an awesome song me now: damn, this shit is deep
@3amerdiab446	eminem has a superhuman storytelling ability with his music.
@eyegehgxd7936	this song has so much story telling.masterpeice.
@ND_trends	my girlfriend said this song sucked, now she's single.
@sucersdungeon	this is in a way a love letter to all of eminem's fans. a song that expresses deep gratitude towar
@manartota	this is and will always be a masterpiece
@Louis-nv3nh	dido's voice is soo calming and the beat is just soo relaxing. yeah the song is kinda dark but da
@TheGuy-ff7vv	this is songwriting at it's finest, three verses by stan, with increasing anger in the tone in each s
@josephmills871	it's amazing how well eminem captures a complete break down of a person in this video. throu
@keanujackson7903	this is what a "music video" is, nowadays people don't tell a story in their videos as much, but
@jlddark	this is a timeless song with a great message. there will always be some people who take artists

Fig 5.2.5 : Represents most liked comments

- **Comment Ranking:** The "Most Liked Comments" section presents a ranked list based on the number of likes each comment has received.
- **User Engagement:** Comments are sorted in descending order, showcasing the most popular or impactful comments at the top, creating a user-driven hierarchy.
- **Username Inclusion:** Each entry displays the commenter's username, providing recognition and attribution to the individuals sharing their thoughts.
- **Content Insight:** Users can quickly identify and engage with comments that resonate the most with the community, fostering a sense of community interaction and shared sentiments.
- **Visibility:** The feature enhances user experience by spotlighting the most liked comments, offering valuable insights and facilitating discussions around noteworthy contributions.

5.2.6. <u>Most Replied Comments:</u>


Most Replied Comments

Author	Comment
@ND_trends	my girlfriend said this song sucked, now she's single.
@rmejiarosales	me in 2010 not knowing the lyrics: oh this is an awesome song me now: damn, this shit is deep
@3amerdiab446	eminem has a superhuman storytelling ability with his music.
@sucersdungeon	this is in a way a love letter to all of eminem's fans. a song that expresses deep gratitude towards
@eyegehgxd7936	this song has so much story telling.masterpeice.
@Louis-nv3nh	dido's voice is soo calming and the beat is just soo relaxing. yeah the song is kinda dark but dam
@Turdferguson90	today is 23 years since this song came out. i was 10. now i'm 33.. still love the video and song. i re
@bomihalj7364	best song ever made in the whole world, even after two decades it still hits the same.
@josephmills871	it's amazing how well eminem captures a complete break down of a person in this video. throug
@manartota	this is and will always be a masterpiece
@v8y	this song never gets old, pay respects to the classic eminem music.

Fig 5.2.6: Represents most replied comments

- **Comment Ranking:** The "Most Replied Comments" section displays comments ordered by the number of replies, highlighting the most engaging or discussion-provoking entries.
- **Community Interaction:** Entries are arranged in descending order based on the quantity of replies, spotlighting comments that sparked extensive conversations within the community.
- **User Attribution:** Each entry features the commenter's username, providing credit to contributors and fostering a sense of authorship and community engagement.
- **Discussion Insights:** Users gain insights into the most conversational comments, promoting a dynamic and interactive environment where community members actively engage in discussions.
- **User-Generated Content:** This feature encourages users to contribute compelling comments that drive meaningful discussions, contributing to a vibrant and participatory online community.

6. Conclusions and Suggestions for Future Work

6.1 Conclusion

This project commenced with an extensive Literature Review, drawing insights from a diverse range of sources including academic publications, patented documents, well-regarded websites, and books published through oral transmission. Additionally, it entailed an examination of prominent applications within the realm of brand reputation analysis using YouTube comments, analysing their respective features. This comprehensive literature survey laid the foundation for a nuanced comparison and contrast between existing applications and the novel application proposed in this project.

Overall, this project signifies a significant advancement in the application of AI-driven technologies to brand reputation analysis, marking a pivotal moment in the field. It introduces an innovative and more tailored approach to understanding and managing brand perception in the digital era.

6.2 Suggestion for future work

Future work in brand reputation analysis should explore multimodal analysis, encompassing text, audio, and visual data from YouTube, as well as real-time monitoring

for immediate responses to sentiment changes. Emotion recognition, language localization, and the integration of insights with Customer Relationship Management (CRM) systems can provide a more comprehensive view of brand perception. Advanced machine learning models, contextual analysis, and the consideration of ethical and privacy concerns are essential for accurate and responsible analysis. Combining sentiment data with user surveys, benchmarking against industry standards, and assessing the impact on financial metrics can further refine reputation management. Lastly, investigations into user behaviour and the application of A/B testing can lead to more effective and data-driven strategies for brand reputation enhancement.

References

1. Jyoti Gautam, Mihir Atrey, Nitima Malsa, Abhishek Balyan,Rabindra Nath Shaw, and Ankush Ghosh(2021)YouTube Sentiment Analysis and Topic Modeling on Healthcare Products: https://repository.ihu.edu.gr/xmlui/bitstream/handle/11544/30285/Anastasiou_Pan agiota_Dissertation_3308200001.pdf?sequence=1
2. Comparison of Neural Network Models for Sentiment Analysis: https://dergipark.org.tr/en/download/article-file/1506505
3. A sentiment analysis of the Ukraine-Russia conflict tweets using Recurrent Neural Networks https://www.researchgate.net/publication/361275253_A_sentiment_analysis_of_th e_Ukraine-Russia_conflict_tweets_using_Recurrent_Neural_Networks
4. Twitter Data Sentiment Analysis Using Naive Bayes Classifier and Generation of Heat Map for Analyzing Intensity Geographically https://www.researchgate.net/publication/350932230_Twitter_Data_Sentiment_Analysis_Using_Naive_Bayes_Classifier_and_Generation_of_Heat_Map_for_Analyzing_Intensity_Geographically
5. Comparative Study of CNN and RNN for Natural Language Processing https://browse.arxiv.org/pdf/1702.01923.pdf
6. Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory.Neural computation, 9(8), 1735-1780 https://direct.mit.edu/neco/article-abstract/9/8/1735/6109/Long-Short-Termhttps://direct.mit.edu/neco/article-abstract/9/8/1735/6109/Long-Short-Term-Memory?redirectedFrom=fulltextMemory?redirectedFrom=fulltext

7. Benchmarking data-driven rainfall–runoff models in Great Britain: a comparison of long short-term memory (LSTM)-based models with four lumped conceptual models.

https://hess.copernicus.org/articles/25/5517/2021/

8. AfriSenti: A Twitter Sentiment Analysis Benchmark for African Languages

https://arxiv.org/pdf/2302.08956v4.pdf

9. Sentimental analysis of amazon customers based on their review comments
https://typeset.io/papers/sentimental-analysis-of-amazon-customers-based-on-their-2jb1il2e
‘

10. Sentimental Analysis of YouTube Videos by IJRET

IRJET-V7I12374.pdf