COEP Technological University, Pune – 411005

Project Report on

"Cyberbullying Analysis of Tweets"

Submitted by

Janhavi Thikekar 142203009
Rewa Saykar 112103126

In completion of
(CT21015) Mini project : Software Engineering:Mini project Stage II

In partial fulfillment for the award of the degree
of
B. TECH (Computer Engineering)
COEP Technological University, Pune

Under the guidance of
Dr.JibiAbraham
COEP Technological University, Pune

1 Introduction

In recent years, social media platforms like Twitter have become integral parts of our daily lives, serving as powerful tools for communication, information dissemination, and community engagement. However, alongside their many benefits, these platforms also present significant challenges, particularly concerning the proliferation of malicious content and harmful activities.

In today's digital age, social media platforms have been providing avenues for communication, expression, and connectivity. However, alongside the benefits of these platforms, there exists a darker side: the prevalence of cyberbullying. Cyberbullying refers to the use of electronic communication to bully, harass, or intimidate others, typically through means such as social media, messaging apps, or online forums. Among various social media platforms, Twitter stands as one of the most popular and widely used, making it a focal point for understanding and addressing online harassment.

The "Cyberbullying Analysis of Tweets" project aims to delve deep into the dynamics of cyberbullying within the Twitter ecosystem. By employing advanced computational techniques and machine learning algorithms, this project seeks to analyze large datasets of Twitter interactions to identify patterns, trends, and factors contributing to online harassment.

In response to these challenges, there is a growing need for tools and technologies that can effectively identify and analyze malicious tweets, enabling prompt detection, moderation, and mitigation of harmful content. The development of a Twitter cyberbullying analyzer aims to address this need by leveraging advanced data mining, natural language processing (NLP), and machine learning techniques to systematically identify, classify, and analyze malicious tweets on the platform.

1.1 Objective

The objective of a cyberbullying analysis of tweets is to identify patterns, trends, and factors contributing to cyberbullying behaviour on platforms like Twitter. This analysis aims to develop insights into the nature and extent of cyberbullying incidents, identify vulnerable groups, understand the language and tactics used by cyberbullies, and ultimately develop strategies to prevent and address cyberbullying effectively.

1.2 Problem Statement

Cyberbullying poses a significant threat to the mental health and well-being of individuals, especially adolescents and young adults who are frequent users of social media platforms like Twitter. Despite efforts to combat cyberbullying, its prevalence continues to rise, causing psychological distress, social isolation, and in extreme cases, even leading to self-harm or suicide. The lack of comprehensive understanding of cyberbullying dynamics on Twitter hampers effective intervention strategies. Therefore, the problem statement of this analysis is to investigate and analyze tweets to identify instances of cyberbullying, understand its underlying causes and mechanisms, and propose data-driven solutions to prevent and mitigate cyberbullying on the platform.

1.2.1 Purpose

Cyberbullying remains a pervasive and distressing issue within online communities, with Twitter serving as a prominent platform where such behaviour often occurs. Despite efforts to combat this phenomenon, there remains a lack of comprehensive understanding regarding the underlying dynamics, contributing factors, and effective mitigation strategies specific to Twitter.

1.3 Scope

The scope of the proposed Twitter cyberbullying analyzer encompasses several key aspects related to the detection, analysis, and mitigation of harmful content and malicious activities on the Twitter platform.

1.3.1 Definitions, Acronyms and Abbreviations

- RF Classifier
- TF – IDF
- API

1.3.2 References

- RF Classifier
- TF-IDF Vectorizer
- Random Forest Classifier
- Term Frequency-Inverse Document Frequency
- Application Programming Interface

1.3.3 Overview

In today's digital age, social media platforms like Twitter have become indispensable tools for communication, information dissemination, and community engagement. However, alongside their many benefits, these platforms also present significant challenges, including the proliferation of malicious content and harmful activities.

2 Specific Requirements

2.1 Functionality

2.1.1 Real-time Tweet Monitoring

The analyzer continuously monitors Twitter streams in real-time to detect newly posted tweets. It utilizes Twitter's API or other authorized sources to access the latest tweet data.

2.1.2 Text Preprocessing

Before analysis, tweets undergo preprocessing steps such as tokenization, stemming, and stop-word removal to enhance analysis accuracy. Preprocessing ensures that tweets are in a suitable format for subsequent analysis.

2.1.3 Sentiment Analysis

The analyzer assesses the sentiment of tweets to determine their emotional tone, such as positive, negative, or neutral. Sentiment analysis helps identify tweets containing potentially harmful or abusive language.

2.1.4 Content Analysis

Tweets are analyzed for content to identify specific indicators of malicious intent, including hate speech, false claims, threats, or harassment. Content analysis involves keyword matching, linguistic analysis, and pattern recognition techniques.

2.1.5 Machine Learning Classification

Machine learning models classify tweets into categories such as fake news, hate speech, cyberbullying, propaganda, or spam. Classification models are trained on labeled data and continuously updated to improve accuracy.

2.1.6 Anomaly Detection

The analyzer employs anomaly detection techniques to identify tweets that deviate significantly from normal behaviour or language patterns. Anomalies may indicate potential instances of misinformation, coordinated campaigns, or unusual user activity.

2.1.7 User Profiling

Users posting malicious tweets are profiled based on their behaviour, engagement patterns, and network connections. User profiling helps identify suspicious accounts and detect coordinated efforts to spread harmful content.

2.1.8 Network Analysis

The analyzer examines network structures to identify clusters of users engaged in coordinated malicious activities. Network analysis provides insights into the dissemination and amplification of malicious content.

2.1.9 Alerting and Reporting

The analyzer implements alerting mechanisms to notify users, moderators, or platform administrators about the presence of malicious tweets. Users can report suspicious tweets, triggering further investigation and moderation actions by platform administrators.

2.1.10 Scalability and Performance

The analyzer is designed to handle large volumes of Twitter data efficiently, ensuring scalability and performance. Optimization techniques such as parallel processing and distributed computing are employed to manage computational resources effectively.

2.2 Usability

2.2.1 Graphical User Interface

- Dashboard Overview : Upon logging in, users are presented with a dashboard providing an overview of recent activities and trends on Twitter. The dashboard includes visualizations such as charts, graphs, and statistics summarizing tweet volumes, sentiment analysis results, and classification breakdowns.
- Tweet Feed : A centralized feed displays a continuous stream of tweets fetched in real-time from Twitter's API. Each tweet is accompanied by relevant metadata, including timestamp, user handle, tweet content, and engagement metrics (e.g., likes, retweets).
- Tweet Analysis Panel : Users can select individual tweets from the feed to view detailed analysis results. The analysis panel provides insights into sentiment analysis, content classification, and anomaly detection for the selected tweet. Visual indicators highlight specific elements of concern, such as offensive language or misleading information.
- Classification Filters : Users can filter tweets based on predefined classification categories (e.g., fake news, hate speech, cyberbullying). Filters allow users to focus on specific types of malicious content for targeted analysis and moderation.
- User Profiling Module : A dedicated section provides insights into user profiles associated with malicious tweets. Users can explore profiles of suspicious accounts, view engagement patterns, and assess the credibility of individual users.
- Reporting Mechanism : Users have the option to report suspicious tweets directly from the GUI. A simple reporting form allows users to provide additional context or details regarding the reported tweet for further investigation.
- Alerting System : An alerting system notifies users, moderators, or administrators about the presence of potentially harmful tweets in real-time. Alerts are displayed prominently on the GUI, ensuring timely attention and response to emerging threats.
- Customization Options : Users can customize the GUI layout, preferences, and display settings to suit their specific needs and preferences. Customization options may include theme selection, font size adjustments, and widget placement.
- Accessibility Features : The GUI incorporates accessibility features to ensure usability for users with diverse needs. Features such as screen reader compatibility, keyboard navigation shortcuts, and text-to- speech functionality enhance accessibility for all users.
- Help and Documentation : Comprehensive help resources, including user guides, tooltips, and FAQs, are accessible directly from the GUI. Users can quickly find answers to common questions and troubleshoot issues without leaving the application.

2.2.2 Accessibility

- The system shall provide regular access.
- The system shall provide English language support.

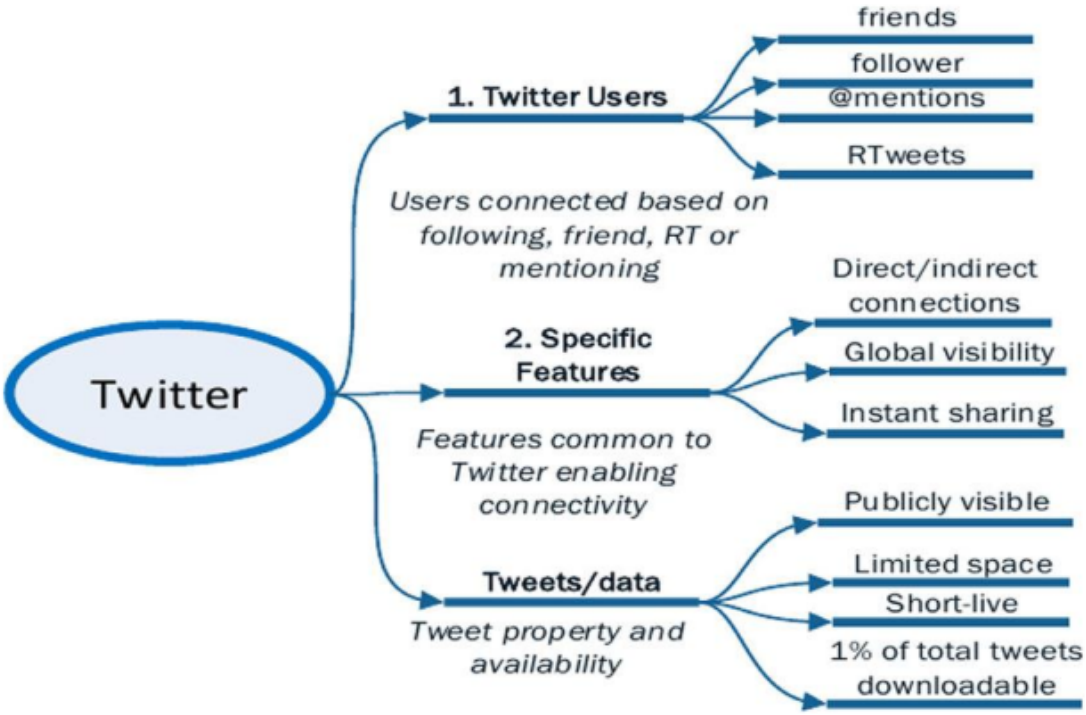

Figure 1: Accessibility

2.3 Reliability & Availability

2.3.1 Error Handling

The analyzer incorporates robust error handling mechanisms to gracefully manage unexpected errors or exceptions. Comprehensive error logs are maintained to track errors and facilitate debugging and troubleshooting by system administrators.

2.3.2 Data Integrity

Measures are implemented to ensure the integrity and consistency of data processed by the analyzer. Data validation checks are performed at various stages to detect and prevent data corruption or inaccuracies.

2.3.3 Fault Tolerance

The analyzer is designed to withstand system failures or disruptions without compromising functionality. Redundancy and failover mechanisms are employed to maintain uninterrupted operation in the event of hardware or software failures.

2.3.4 Performance Monitoring

Performance monitoring tools are utilized to continuously monitor the analyzer's performance metrics, such as response time and resource utilization. Thresholds and alerts are configured to notify system administrators of performance degradation or bottlenecks.

2.3.5 Automated Testing

Automated testing suites are employed to validate the reliability and correctness of analyzer functionalities. Unit tests, integration tests, and end-to-end tests are conducted regularly to verify the behaviour of critical components and workflows.

2.4 Performance

2.4.1 Based on Web

The product shall be based on web and has to be run from a web server. The product shall take initial load time depending on internet connection strength which also depends on the media from which the product is run. The performance shall depend upon hardware components of the client/customer.

2.5 Security

2.5.1 Authentication and Authorization

Users are required to authenticate themselves using secure login credentials (e.g., username and password) before accessing the analyzer. Role-based access control (RBAC) mechanisms are implemented to ensure that users are granted appropriate permissions based on their roles and responsibilities.

2.5.2 Data Encryption

Sensitive data, including user credentials, API keys, and analysis results, are encrypted both in transit and at rest using strong encryption algorithms (e.g., AES-256). Transport Layer Security (TLS) protocols are enforced to encrypt data exchanged between clients and servers over the network.

2.5.3 Secure Data Storage

Data storage systems, such as databases and file systems, employ robust security measures to protect against unauthorized access and data breaches. Access controls, encryption, and auditing mechanisms are implemented to enforce data confidentiality, integrity, and availability.

2.5.4 API Security

Access to the Twitter API or other data sources is restricted using API keys or tokens, and access controls are enforced to limit the scope of API usage. Rate limiting and throttling mechanisms are applied to prevent abuse and protect against denial-of-service (DoS) attacks.

2.5.5 Input Validation and Sanitization

Input validation and sanitization techniques are employed to mitigate common security vulnerabilities, such as SQL injection, cross-site scripting (XSS), and command injection. User inputs, including search queries and filtering parameters, are validated and sanitized to prevent malicious input from compromising the system.

2.5.6 Logging and Auditing

Comprehensive logging and auditing mechanisms are implemented to track user activities, system events, and security-related incidents. Audit logs capture details such as user logins, access attempts, data modifications, and security policy violations for forensic analysis and compliance auditing.

2.5.7 Security Patching and Updates

Regular security patching and updates are applied to all software components, including operating systems, libraries, frameworks, and third-party dependencies. Vulnerability scans and penetration tests are conducted periodically to identify and remediate security vulnerabilities proactively.

2.5.8 Incident Response Plan

An incident response plan is in place to guide the response to security incidents, such as data breaches, unauthorized access, or system compromises. The plan includes procedures for incident detection, containment, eradication, recovery, and post-incident analysis to minimize the impact of security breaches.

2.6 Design Constraints

2.6.1 Twitter API Limitations

The analyzer must adhere to rate limits and usage restrictions imposed by the Twitter API to prevent API abuse and ensure compliance with Twitter's terms of service.

2.6.2 Data Privacy Regulations

The system must comply with data privacy regulations such as GDPR (General Data Protection Regulation) and CCPA (California Consumer Privacy Act) when collecting, processing, and storing user data.

2.6.3 Resource Constraints

The system's performance may be limited by available hardware resources (e.g., CPU, memory, storage) and network bandwidth, requiring optimization for efficient resource utilization.

2.6.4 Scalability Requirements

The system's design must accommodate future scalability requirements to handle increasing volumes of Twitter data and user traffic without sacrificing performance or reliability.

2.6.5 Third-Party Dependencies

The analyzer relies on third-party APIs, libraries, or services for functionalities such as sentiment analysis, machine learning models, and data storage, which may introduce dependencies and compatibility issues.

2.6.6 Budgetary Constraints

Development and maintenance costs must be within budgetary constraints, requiring careful consideration of resource allocation, licensing fees, and operational expenses.

2.6.7 User Interface Considerations

The user interface design must be intuitive, accessible, and responsive across different devices and screen sizes, considering user preferences and accessibility requirements.

2.6.8 Compliance Requirements

The system must comply with industry standards, regulations, and legal requirements related to data security, privacy, and user consent, necessitating adherence to best practices and auditability.

2.6.9 Performance Requirements

The system's performance must meet defined service level agreements (SLAs) and performance targets, ensuring timely response times and availability of services under normal and peak load conditions.

2.6.10 Maintenance and Support

Adequate provisions must be made for system maintenance, updates, and ongoing support to address bug fixes, security patches, and feature enhancements over the system's lifecycle.

2.7 On-line User Documentation and Help System Requirements

2.7.1 User Manual

Provide a comprehensive user manual detailing the functionality, features, and usage instructions of the analyzer. Organize the manual into sections covering different aspectsof the system, including setup, configuration, analysis workflows, and troubleshooting.

2.7.2 Interactive Tutorials

Develop interactive tutorials or walkthroughs to guide users through common tasks and workflows within the analyzer. Include step-by-step instructions, screenshots, and interactive elements to enhance user understanding and engagement.

2.7.3 Context-Sensitive Help

Implement context-sensitive help features within the analyzer's user interface to provide relevant assistance based on user actions and queries. Display tooltips, pop-up messages, or inline help text to clarify interface elements, functions, and parameters.

2.7.4 Searchable Knowledge Base

Establish a searchable knowledge base or FAQ (Frequently Asked Questions) repository containing answers to common user queries, issues, and troubleshooting tips. Organize the knowledge base into categories and tags for easy navigation and retrieval of relevant information.

2.7.5 Video Tutorials and Demos

Create video tutorials and demos illustrating key functionalities, best practices, and use cases of the analyzer. Host video content on a dedicated platform or integrate it within the analyzer's documentation for easy access.

2.7.6 Community Forums or Discussion Boards

Foster a community-driven support environment by providing forums or discussion boards where users can ask questions, share experiences, and seek assistance from peers and experts. Moderate the forums to ensure the quality of discussions and provide timely responses to user inquiries.

2.7.7 Feedback Mechanism

Incorporate a feedback mechanism within the analyzer to collect user feedback, suggestions, and enhancement requests. Provide users with a means to submit feedback directly from the user interface or documentation portal.

2.7.8 Version History and Release Notes

Maintain version history and release notes documenting changes, updates, and new features introduced in each version of the analyzer. Include information on bug fixes, performance improvements, and security patches to keep users informed about the latest developments.

2.7.9 Localization and Multilingual Support

Offer localization options and multilingual support for user documentation to cater to diverse user populations and language preferences. Translate documentation materials into multiple languages to ensure accessibility and usability for global users.

2.7.10 Accessibility Compliance

Ensure that user documentation and help system comply with accessibility standards (e.g., WCAG) to accommodate users with disabilities or special needs. Provide alternative formats, such as screen reader compatibility and text-to-speech functionality, to enhance accessibility for all users.

2.8 Interfaces

2.8.1 User Interface (UI)

Design a user-friendly interface for the analyzer, allowing users to interact with the system easily. Include intuitive controls, informative visualizations, and interactive elements to facilitate data analysis and manipulation.

2.8.2 API Interface

Implement an API interface to allow external systems or applications to interact with the analyzer programmatically. Define clear endpoints, request parameters, and response formats for seamless integration with third-party tools or services.

2.8.3 Twitter API Integration

Integrate with the Twitter API to fetch real-time tweet data for analysis. Authenticate with Twitter's OAuth mechanism and adhere to rate limits and usage policies to ensure compliance and reliability.

2.8.4 Database Interface

Establish interfaces for interacting with the underlying database system used to store tweet data, analysis results, and system configurations. Implement CRUD (Create, Read, Update, Delete) operations to manipulate data securely and efficiently.

2.8.5 External Data Sources

Define interfaces for accessing external data sources or datasets that complement the tweet data for analysis. Ensure compatibility with various data formats and protocols for seamless data ingestion and integration.

2.8.6 User Authentication and Authorization Interface

Implement interfaces for user authentication and authorization to manage user access and permissions within the system. Support various authentication methods (e.g., username/password, OAuth) and role-based access control (RBAC) mechanisms.

2.8.7 Reporting Interface

Design interfaces for generating and exporting analysis reports, visualizations, and insightsderived from tweet data. Provide options for customizing report formats, parameters, and delivery methods (e.g., PDF, CSV, email).

2.8.8 Logging and Monitoring Interface

Define interfaces for logging system events, errors, and user activities for monitoring and auditing purposes. Integrate with logging frameworks or services to store logs centrally and analyze system performance and usage patterns.

2.8.9 Feedback and Support Interface

Implement interfaces for collecting user feedback, support requests, and bug reports. Provide channels for users to submit feedback, suggestions, or issues directly from the analyzer's interface or documentation portal.

2.8.10 External Notification Interfaces

Integrate with external notification services or platforms to send alerts, notifications, or updates to users or administrators. Support various notification channels, such as email, SMS, or push notifications, based on user preferences.

2.9 Legal, Copyright, and Other Notices

2.9.1 Copyright Notice

Include a copyright notice specifying the ownership of the analyzer's source code and associated materials. Format: "Copyright [Year] [Copyright Holder]. All rights reserved."

2.9.2 License Information

Display information about the software's licensing terms and conditions, including the specific open-source license under which it is distributed. Provide a summary of the license terms and a link to the full text of the license agreement.

2.9.3 Disclaimer of Warranty

Include a disclaimer of warranty stating that the software is provided "as is" without any warranties or guarantees of performance, reliability, or suitability for specific purposes. Clarify that the software may contain bugs, errors, or limitations, and users use it at their own risk.

2.9.4 Limitation of Liability

Limit liability for damages arising from the use, inability to use, or reliance on the software, including direct, indirect, incidental, or consequential damages. Specify the maximum extent of liability permitted under applicable law.

2.9.5 Attribution Requirements

Specify any attribution requirements or acknowledgments that users must include when redistributing or modifying the software. Clarify the form and placement of attribution notices, such as in source code comments, documentation, or user interfaces.

2.9.6 Trademark Notice

Include a trademark notice identifying any trademarks or service marks associated with the software or its brand. Specify proper usage guidelines for trademarks and prohibit unauthorized use or modification.

2.9.7 Third-Party Components

List third-party components, libraries, or dependencies used in the analyzer and provide attribution notices for their respective authors or copyright holders. Include license information and disclaimer notices for third-party components as required by their respective licenses.

2.9.8 Compliance Statements

Include statements affirming compliance with legal and regulatory requirements, such as data privacy laws, export control regulations, and intellectual property rights. Ensure that the software complies with applicable laws and regulations in all jurisdictions where it is distributed or used.

2.9.9 Modification and Redistribution Policy

Specify the conditions under which users are allowed to modify, redistribute, or sublicense the software, including any requirements for preserving copyright notices and license terms. Clarify whether modifications must be contributed back to the community or can be kept proprietary.

2.9.10 Contact Information

Provide contact information for inquiries, feedback, and legal notices related to the software, including email addresses, mailing addresses, and website URLs. Ensure that users have a means of contacting the copyright holder or designated representatives for legal or licensing issues.

3 Diagrams

3.1 DFD Level 0

The Data Flow Diagram (DFD) level 0 (Figure 2) depicts a system designed for text data analysis, with a specific focus on integrating with Twitter as a data source. At the core of this system is the analyzer process, which serves as the central component responsible for processing and analyzing the text data provided by the user. The user interacts with the system by inputting text data, which serves as the initial trigger for the analysis process.

Upon receiving the input text, the Analyser process swings into action, utilizing its algorithms and logic to delve into the text and extract meaningful insights. To enhance the analysis, the system is designed to fetch additional data from Twitter, leveraging the platforms vast repository of real-time information. This interaction with Twitter allows the system to access a wealth of data such as tweets, trends, and other relevant information that can enrich the analysis process.

The connection with Twitter serves as a valuable source of external data that complements the user-provided text, enabling a more comprehensive and insightful analysis. By tapping into this external data source, the system can validate findings, uncover trends, or provide context to the userapos;s input text, ultimately enhancing the quality and depth of the analysis.

As the Analyser process completes its analysis, the results are channeled to the Output component, where the analyzed text is stored or presented in a format that is accessible to the user. This final output, referred to as the Analysis Results, encapsulates the findings and insights derived from the text analysis process. It represents the culmination of the system's efforts, delivering valuable information and interpretations to the user based on the input text and the supplementary data fetched from Twitter. In essence, the DFD level 0 diagram showcases a system that seamlessly integrates user-provided text data with external data from Twitter to perform in-depth analysis and deliver meaningful results to the user. Through its structured processes and data flow, the system aims to empower users with valuable insights and interpretations derived from the fusion of user-generated content and external data sources.

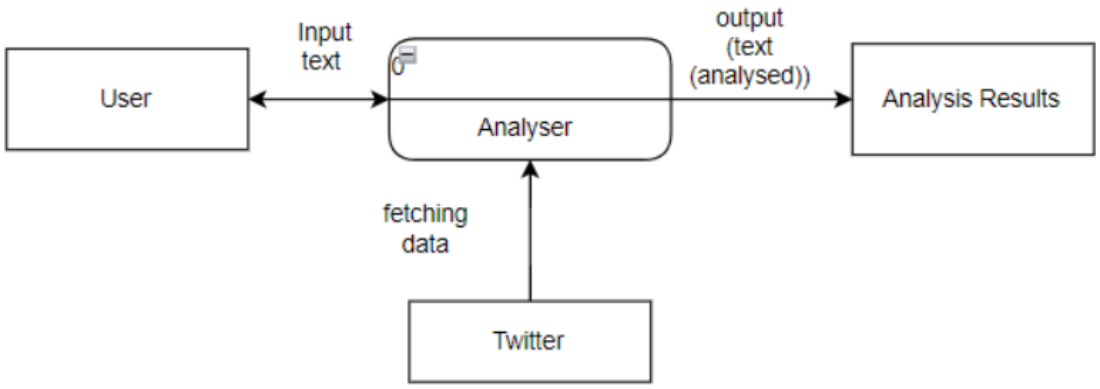

Figure 2: DFD level 0

3.2 DFD Level 1

The Data Flow Diagram (DFD) level 1 (Figure 3) offers a more detailed view of the system for analyzing Twitter data. This diagram breaks down the main process into sub-processes, showing the flow of data between them.

The system begins with the "User" as the external entity that provides "Input Data" to the system. This input initiates the process of data retrieval and analysis.

The "Tweet Retrieval" process is responsible for obtaining live data scraped from Twitter. This data is fetched through the "Twitter Data" process, which likely involves interacting with the Twitter API

to collect tweets relevant to the user's input.

Once the data is fetched, it moves to the "Preprocess Tweets" process. Here, the data is preprocessed to ensure quality and consistency. This typically involves cleaning the data by removing missing values, checking for redundancy, and possibly normalizing the data.

The next step is "Tokenization," where the preprocessed tweet data is broken down into tokens. Tokenization is a crucial step in text analysis as it converts the text into smaller units (like words) that can be analyzed.

Following tokenization, the "Feature Selection" process takes place. This step involves selecting the most relevant attributes or features from the tokenized data that will be used for classification. Feature selection is important for improving the performance of the classifier by reducing the dimensionality of the data.

The "Negate Sequences" process may involve identifying and handling negations in the text, which can significantly alter the sentiment or meaning of phrases within the tweets.

The "Train Classifier" process involves using a "Trained Dataset" to train a machine learning model that can classify new data. This trained classifier is then used in the "Test Classifier" process to analyze new tweet data and produce "Analysed Data."

Finally, the "Results" process outputs the analyzed data, which represents the insights gained from the classification of the tweets. This could include sentiment analysis, trend detection, or other forms of interpretation based on the user's requirements.

In summary, this DFD level 1 diagram provides a granular view of the system's architecture, illustrating the detailed processes involved in retrieving, preprocessing, and analyzing Twitter data to produce actionable insights for the user.

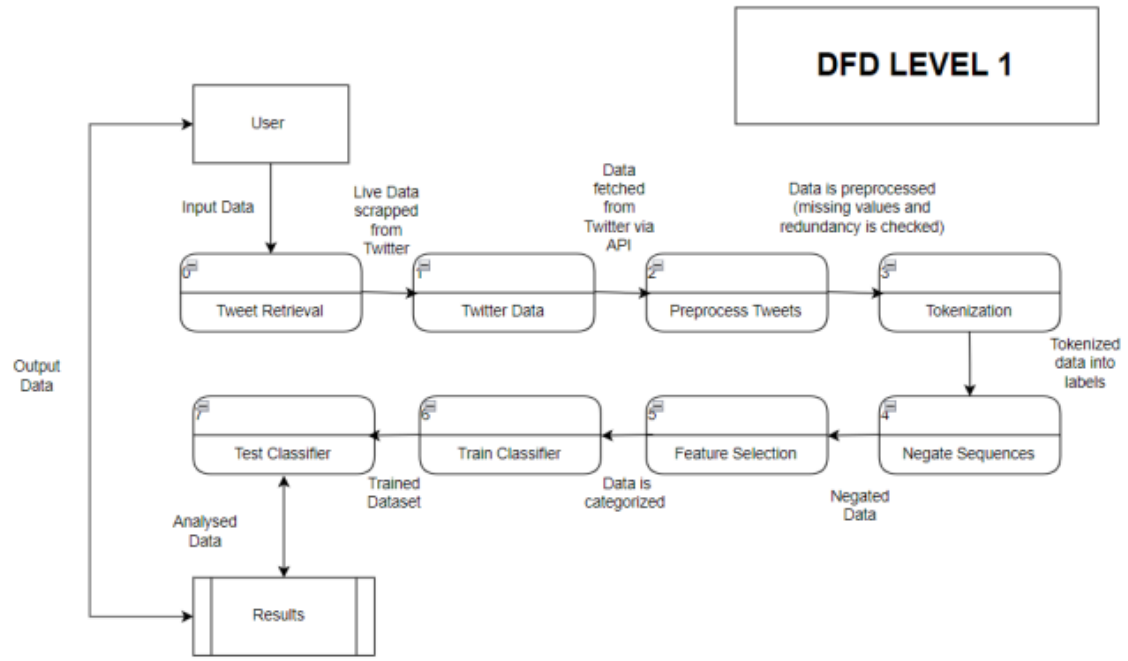

Figure 3: DFD level 1

3.3 UML Diagram

The UML (Unified Modeling Language) flowchart (Figure 4) outlines the process for building, testing, and selecting a Natural Language Processing (NLP) model specifically designed for detecting cyberbullying in tweets. The process begins with the initial step marked as "Start."

The first action in the workflow is to build, test, and select an NLP model for cyberbullying tweets. This involves Exploratory Data Analysis (EDA) and preprocessing of the data to ensure it is clean and suitable for model training. EDA is a critical step that allows for understanding the underlying patterns and characteristics of the data, while preprocessing involves tasks such as tokenization, stemming, lemmatization, and removal of stop words and noise.

Upon preparing the data, four different models are built. These models are based on two distinct types of recurrent neural network architectures: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). The LSTM model is known for its ability to capture long-term dependencies, while the GRU is a variation that can achieve similar performance with fewer parameters and potentially faster training times.

In addition to the standard LSTM and GRU models, two other variations are explored. One is an improved version of the LSTM model, which likely includes optimizations or additional layers to enhance its performance. The other is a GRU model that utilizes transfer learning, indicated as "GRU Transfer." Transfer learning allows the model to leverage knowledge from a related task that has already been learned, potentially improving its ability to generalize and detect cyberbullying.

After constructing the four models, the next step is to choose the best model with relatively the best accuracy. This involves evaluating the performance of each model on a validation set or through cross-validation to determine which model most effectively identifies cyberbullying content.

Once the best model is selected, it undergoes further testing to validate its performance. The final step in the process is calculating the accuracy of the chosen model, which quantifies how well the model performs on unseen data, presumably a test set.

The process concludes with the "end" step, signifying the completion of the model selection and evaluation phase for the Cyber-Bullying Detection App project. The outcome is a trained and tested NLP model ready for deployment in detecting cyberbullying instances in tweets.

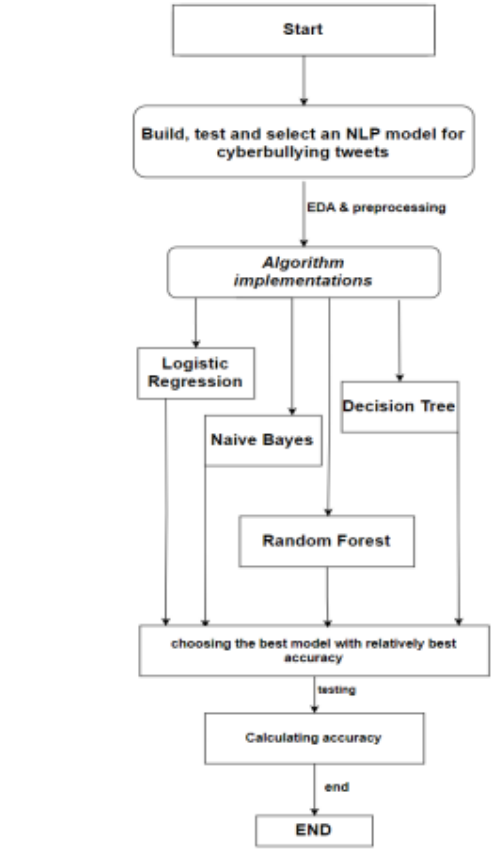

Figure 4: UML Diagram

3.4 System Architecture Diagram

The system architecture diagram (Figure ??) illustrates the structured development process of the Cyber-Bullying Detection App (CBDA), showcasing the integration of machine learning, web development, version control, and deployment stages. At the core of the project lies the creation of a machine learning model, denoted as CBD.ML, is designed to identify patterns indicative of cyber-bullying within data sets. This machine learning model is developed using Python, a versatile programming language known for its effectiveness in data analysis and machine learning applications. Following the development of the machine learning model, the project progresses to the web development phase, where a user interface is crafted to enable interaction with the CBD.ML model. Utilizing Streamlit, a Python library, the data scripts are transformed into a user-friendly web application with minimal coding requirements. Complementary technologies such as HTML, CSS, and JavaScript are incorporated to enhance the visual appeal and user experience of the application. To facilitate collaboration and streamline the development process, version control is implemented using Git. This system enables tracking of code changes, facilitates concurrent development by multiple team members, and maintains a comprehensive history of the project's evolution. The deployment stage, represented by the Git platform icon, signifies the final step of making the CBDA accessible to users online. This process involves setting up the necessary infrastructure to ensure the application's availability, scalability, and security for end-users. In essence, the CBDA project embodies a harmonious fusion of machine learning and web development technologies aimed at combating cyber-bullying. By incorporating robust version control practices and efficient deployment strategies, the project ensures a collaborative and systematic approach to software development, culminating in a high-quality application that addresses the critical issue of cyber-bullying in the digital realm.

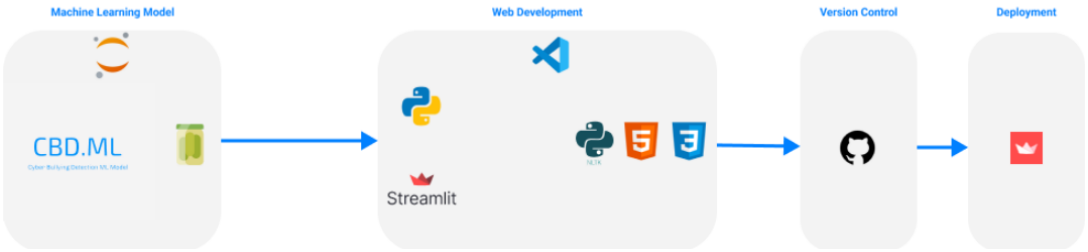

Figure 5: System Architecture Diagram

4 Key Files and Functionalities

Cyberbullying has become a significant concern in today's digital age, affecting individuals across various demographics. To address this issue, our project aims to develop a comprehensive cyberbullying detection system that leverages machine learning techniques and web technologies. The system is designed to analyze social media data, detect instances of cyberbullying, and provide valuable insights for stakeholders. This documentation provides an overview of the key files and functionalities of our cyberbullying detection system.

4.1 `1_CBDA.py`

In the initial phase of the project, the script named `1_CBDA.py` sets the foundation by importing essential libraries. These libraries serve as the toolkit for creating a dynamic and interactive web application. The script then delves into the realm of aesthetics, defining CSS code to hide the menu and footer elements. This CSS code snippet, when applied, enhances the visual appeal of the application by focusing the user's attention on the main content.

Furthermore, the script introduces a variable named `showWarningOnDirectExecution`, which plays a role in controlling the behavior of the application. This variable, when set to False, ensures that warning messages are not displayed upon direct execution of the script.

Moving on, the script utilizes the capabilities of the Python Imaging Library (PIL) to load an image, which serves as a visual element within the application. This image, carefully chosen and incorporated, enhances the overall user experience and adds a touch of creativity to the interface.

The script also configures the page layout using Streamlit's `set_page_config()` method. This configuration includes setting the page title and icon, providing a cohesive and branded appearance to the application.

To further refine the user interface, the script applies the previously defined CSS code to hide the menu, contributing to a clutter-free and immersive user experience. The script then proceeds to display the loaded image in the sidebar, creating a visually engaging sidebar that complements the main content area.

Additionally, the script adds a horizontal rule in the sidebar and displays a copyright notice. These elements not only add structure and professionalism to the sidebar but also convey ownership and acknowledgment of creative contributions.

Finally, the script displays the loaded image in the main content area, completing the setup phase and paving the way for the subsequent functionalities and features of the web application.

Pseudo Code

```
Import necessary libraries
Define hide_menu CSS code
Define showWarningOnDirectExecution variable
Load an image using PIL.Image.open()
Set page configuration using st.set_page_config()
Hide the menu using the hide_menu CSS code
Display the image in the sidebar using st.sidebar.image()
Display a horizontal rule in the sidebar using st.sidebar.markdown()
Display a copyright notice in the sidebar using st.sidebar.markdown()
```

```
Display the image in the main content area using st.image()
```

4.2 2_Cyberbullying_Detection.py

The script `2_Cyberbullying_Detection.py` plays a pivotal role in the cyberbullying detection project, focusing on data preprocessing, model development, and user interaction. It begins by importing necessary libraries and modules, setting the stage for advanced data analysis and machine learning tasks.

The script defines CSS styles to hide the menu and footer elements, enhancing the visual appeal and focusing the user's attention on the core functionalities of the application. Additionally, it initializes a variable named `hide_menu`, encapsulating the CSS code for hiding these elements.

Next, the script loads and preprocesses models essential for cyberbullying detection. This step involves tasks such as data cleaning, text transformation, and feature extraction, all crucial for preparing the data for machine learning algorithms.

The script further enhances the user interface by setting page configurations, defining sidebar content, and creating functions for text cleaning and transformation. These functions play a crucial role in ensuring the quality and relevance of the data fed into the machine-learning models.

The main function of the script orchestrate the user interaction aspect of the application. It creates a title for the cyberbullying detection module, displays input text areas for user input, and provides buttons for prediction and clearing input.

Upon user interaction, the script handles the prediction process, including text preprocessing, transformation, vectorization, model prediction, and result display. It also incorporates error handling and warning messages to guide users and ensure a smooth user experience.

Overall, the script serves as the backbone of the cyberbullying detection module, encompassing data preprocessing, model development, user interaction, and result visualization.

Pseudo Code

```
import necessary_libraries
import models_and_transformers

hide_menu = define_css_to_hide_menu_and_footer()

load_and_preprocess_models()

set_page_configuration()

define_sidebar_content()

define_clean_text_function(tweet):
    remove_URL(tweet)
    remove_usernames(tweet)
    remove_hashtags(tweet)
    remove_emojis(tweet)
    remove_special_characters(tweet)
    remove_RT(tweet)
    remove_numbers(tweet)
    return cleaned_tweet

define_transform_text_function(text):
    convert_to_lowercase(text)
    tokenize_text(text)
    filter_non_alphanumeric_tokens(text)
    filter_stopwords_and_punctuation(text)
    apply_stemming(text)
    return transformed_text

define_clear_text_function():
    clear_input_text()
```

```
main():
    create_title("Cyber-Bullying Detection")
    display_input_text_area()
    display_predict_and_clear_buttons()

    if predict_button_clicked():
        if input_text_is_empty():
            show_warning_message("Please provide some text!")
        else:
            display_prediction_in_progress_message()

            clean_text = clean_text(input_text)
            transformed_text = transform_text(clean_text)
            vector_input = vectorize_text(transformed_text)
            result = predict(vector_input)

            display_result(result)
            display_original_text(input_text)
            display_cleaned_text(clean_text)
            display_transformed_text(transformed_text)
            display_binary_prediction(result)
            display_model_accuracy()

main()
```

4.3 3_Datasets.py

In the context of the project, the script `3_Datasets.py` focuses on data exploration and visualization, providing users with insights into different datasets related to cyberbullying. The script begins by importing necessary libraries such as Streamlit for web interface development, Pandas for data manipulation, Matplotlib for charting capabilities, and PIL for image processing.

The script defines CSS styles to hide the menu and footer elements, creating a clean and focused interface for data exploration. It also initializes a variable named showWarningOnDirectExecution, which serves as a flag for handling user interactions and warnings.

One of the key functionalities of the script is the loading and display of an image, which adds visual context to the application and enhances user engagement. This image is configured using Streamlit's `set_page_config()` method, ensuring a cohesive visual identity for the application.

The script further enriches the sidebar with visual elements such as an image and a copyright notice, adding professionalism and branding to the interface. It also displays the loaded image in the main content area, contributing to a visually appealing and cohesive user experience.

Moving on to the core functionality, the script provides a dropdown menu for selecting datasets related to cyberbullying. This dropdown menu allows users to choose from a variety of datasets, each offering unique insights and data points relevant to cyberbullying analysis.

Upon selecting a dataset, the script dynamically loads and displays relevant information about the chosen dataset. This includes general information, data shape, preview, head and tail of the data, columns, and summary statistics. Additionally, the script visualizes the dataset using charts such as bar charts and pie charts, offering users a graphical representation of the data for better understanding and analysis.

Overall, the script serves as a gateway to exploring and understanding different datasets related to cyberbullying, providing users with tools and visualizations to gain insights and make informed decisions.

Pseudo Code

```
Import streamlit as st
Import pandas as pd
Import matplotlib.pyplot as plt
Import io
Import Image from PIL
```

```
Define hide_menu as a multi-line string containing CSS code to hide menu and footer

Define showWarningOnDirectExecution as False

Open an image file using Image.open() from PIL and store it in the variable image

Set page configuration using st.set_page_config(), providing page title and icon as parameters

Hide the menu using st.markdown(), passing hide_menu as the argument and allowing unsafe HTML

Display the image in the sidebar using st.sidebar.image(), with use_column_width=True and output_for

Display a horizontal rule in the sidebar using st.sidebar.markdown()

Display a copyright notice in the sidebar using st.sidebar.markdown(), with HTML formatting

Display the title "Datasets" using st.title()

Display a dropdown menu for dataset selection using st.selectbox(), providing dataset options as a l

Display a horizontal rule using st.markdown()

Check the selected dataset choice using if-elif statements:

    If "Cyber Bullying Types Dataset" is selected:
        Read the dataset using pd.read_csv()
        Display dataset information if requested using st.checkbox() and df.info()
        Display dataset shape information if requested using st.checkbox() and df.shape
        Display dataset preview, head, tail, selected columns, and summary if requested using st.che
        Display dataset plots (bar chart and pie chart) if requested using st.checkbox(), plt.subplo

    If "Cyber Troll Dataset" is selected:
        (Repeat the same steps as above for this dataset)

    If "Classified Tweets Dataset" is selected:
        (Repeat the same steps as above for this dataset)

    If "Cyberbulling Classification Dataset" is selected:
        (Repeat the same steps as above for this dataset)

    If "Cyber Bullying Types Dataset + Cyber Troll Dataset" is selected:
        (Repeat the same steps as above for this dataset)

    If "Cyber Bullying Types Dataset + Cyber Troll Dataset + Classified Tweets Dataset + Cyberbullin
        (Repeat the same steps as above for this dataset)
```

4.4 4_Algorithms.py

The script `4_Algorithms.py` is the engine room of the cyberbullying detection system, responsible for implementing machine learning algorithms, handling user interactions, and displaying evaluation metrics. It begins by importing necessary libraries and modules, setting the stage for advanced data analysis and model evaluation.

The script defines CSS styles to hide the menu and footer elements, ensuring a clean and focused interface for algorithm selection and evaluation. It also initializes a variable named `showWarningOnDirectExecution`, which serves as a flag for handling user interactions and warnings.

One of the key functionalities of the script is the creation of a sidebar with visual elements such as an image and a copyright notice, adding professionalism and branding to the interface. This sidebar also contains dropdown menus for selecting datasets, vectorizers, and machine learning models, providing

users with a range of options for analysis.

Upon selecting valid options for dataset, vectorizer, and machine learning model, the script triggers the evaluation process. It preprocesses the data, applies the selected vectorization technique, trains the machine learning model, and evaluates its performance using metrics such as accuracy.

The script also incorporates error handling and warning messages to guide users and ensure a smooth user experience. For instance, if a user forgets to select a dataset, vectorizer, or machine learning model, the script displays a warning message prompting the user to make the necessary selections.

Overall, the script serves as the analytical backbone of the cyberbullying detection system, enabling users to explore different algorithms, evaluate their performance, and gain insights into cyberbullying detection capabilities.

Pseudo Code

```
Import necessary libraries

Define CSS styles to hide menu and footer

Set configuration for Streamlit page with title and icon

Create sidebar with image and footer

Display main title and divider

Define available datasets, vectorizers, and ML models as dropdown options

Get user selections for dataset, vectorizer, and ML model

If user selects valid options for all three dropdowns:
    Display evaluation metrics based on the selected dataset, vectorizer, and ML model
Else if user misses selecting a dataset, vectorizer, or ML model:
    Show warning messages to prompt user to select the missing options

Handle different dataset options with corresponding evaluation metrics based on vectorizer and ML mo

Display success message with accuracy percentage based on the selected dataset, vectorizer, and ML m
```

4.5 5_About_us.py

The script 5_About_us.py is a pivotal part of the cyberbullying detection project, focusing on providing valuable information about the project team, authors, and affiliated institutions. It begins by importing necessary libraries, namely Streamlit for web interface development and PIL.Image for image handling capabilities.

The script then dives into the realm of aesthetics, defining a CSS style to hide the menu and footer elements. This CSS style snippet, when applied, ensures a clean and focused interface for presenting information about the project team and affiliated institutions.

Next, the script initializes a variable named `showWarningOnDirectExecution`, serving as a flag for handling user interactions and warnings within the application. This flag contributes to the overall functionality and user experience by guiding users and providing informative messages when necessary.

Moving on to visual elements, the script loads an image for the page icon and sidebar, adding visual context and branding to the interface. This image, carefully chosen and integrated, enhances the overall user experience and adds a touch of creativity to the presentation.

The script proceeds to set the page configuration, including the title and icon, and applies the previously defined CSS style to hide the menu. This configuration step ensures a cohesive visual identity and a clutter-free interface for presenting information.

One of the key features of the script is the creation of a sidebar with a logo image. This sidebar serves as a navigation hub and visual anchor, providing users with easy access to project-related information and enhancing the overall aesthetics of the application.

In the main content area, the script displays information about the authors, project details, and affiliated institutions. This content is curated to provide users with insights into the background, expertise,

and contributions of the project team members. Additionally, the script may include university logos and information in columns, showcasing affiliations and collaborations that enrich the project.

Overall, the script `5_About_us.py` plays a vital role in fostering transparency, credibility, and engagement within the user community. It serves as a bridge between the application and its users, offering valuable context, information, and visual elements that enhance the user experience and strengthen the project's impact.

Pseudo Code

```
Import necessary libraries: streamlit, PIL.Image

Define CSS style to hide menu and footer
Define showWarningOnDirectExecution flag

Load image for page icon and sidebar

Set page configuration and apply CSS style to hide menu

Create sidebar with logo image

Display authors and project information in main content

Display university logo and information in columns
```

4.6 app.py

The script app.py plays a crucial role in the project, serving as the entry point for the web application and handling user interactions for social media analytics. It begins by importing necessary libraries and modules, setting the stage for creating an interactive and insightful dashboard.

The script defines the app function, which configures the Streamlit page with a title, icon, layout, and initial sidebar state. This configuration ensures a cohesive and user-friendly interface for social media analytics.

The app function displays the header "Social Media Analytics Dashboard," setting the tone for the user's exploration journey. It creates a sidebar select box for choosing the platform (Twitter, Facebook, Instagram), providing users with options for data analysis.

Upon selecting a platform, the script dynamically adjusts the interface to include relevant input fields and analysis options. For instance, if Twitter is selected, the script adds checkboxes for including retweets, text input for entering hashtags or words, and a slider for choosing the number of tweets to analyze.

The script also incorporates interactive elements such as buttons for triggering sentiment analysis and downloading preprocessed data. Upon clicking the sentiment analysis button, the script preprocesses the data, performs sentiment analysis, and visualizes the results through graphs and data tables.

Error handling is implemented to guide users and provide informative messages if input parameters are missing or invalid. For instance, if a user forgets to enter a hashtag or word for analysis, the script displays a warning message prompting the user to provide the necessary input.

Overall, the script app.py serves as the gateway to the social media analytics dashboard, offering users a platform to explore and analyze social media data in a user-friendly and interactive manner.

Pseudo Code

```
Import necessary libraries: streamlit, preprocessing_data, graph_sentiment, analyse_mention, analyse

Define app function:
    Configure Streamlit page:
        Set page title to "Social Dashboard"
        Set page icon to "icon.png"
        Set layout to "wide"
        Set initial sidebar state to "expanded"
```

```
    Display header "Social Media Analytics Dashboard"

    Create a sidebar selectbox to choose the platform (Twitter, Facebook, Instagram)

    If platform selected is Twitter:
        Create checkbox to include retweets in analysis
        Create text input for entering hashtag or word
        Create slider to choose number of tweets to analyze (range: 100 to 10,000)
        Display info message indicating processing time for tweets
        Create button "Analyse Sentiment" to trigger analysis

        On button click:
            Preprocess data using preprocessing_data function
            Analyze sentiment and generate sentiment graph using graph_sentiment function
            Analyze @mentions and generate bar chart using analyse_mention function
            Analyze hashtags and generate bar chart using analyse_hashtag function
            Display extracted and preprocessed dataset
            Provide option to download preprocessed data
            Display top 10 @mentions, top 10 hashtags, and top 10 used links in separate columns
            Display all tweets containing top 10 used links
            Display sentiment analysis graph

    Else if platform selected is Facebook or Instagram:
        Display message "Coming Soon" as these features are not yet implemented

Run the app function if the script is executed directly.
```

4.7 helper.py

The script helper.py is a utility module that plays a crucial role in supporting various functionalities of the cyberbullying detection project. It begins by importing necessary libraries and modules, including Tweepy for Twitter API access, Pandas for data manipulation, and TextBlob for text analysis.

The script defines utility functions such as emoji_pattern to remove emojis from text, twitter_connection to establish a connection with the Twitter API, and cleanTxt to clean text by removing mentions, hashtags, hyperlinks, and emojis.

Other functions in the script include extract_mentions and extract_hashtag to extract mentions and hashtags from text, getSubjectivity and getPolarity to calculate subjectivity and polarity scores using TextBlob, and getAnalysis to classify polarity scores into sentiment labels.

The script also defines a caching function using st.cache decorator to preprocess data efficiently and handle data caching within the Streamlit application. This caching mechanism optimizes data processing and improves application performance.

Overall, the script helper.py serves as a valuable asset in data preprocessing, text analysis, and utility functions required for social media analytics and cyberbullying detection. It encapsulates essential functionalities and supports the seamless operation of the main application modules.

Pseudo Code

```
Import necessary libraries: tweepy, pandas, configparser, re, TextBlob, WordCloud, streamlit, dateti

Define emoji_pattern using regex to remove emojis from text

Define twitter_connection function to establish a connection with Twitter API:
    Read API keys and access tokens from config.ini using configparser
    Initialize OAuthHandler with API keys and access tokens
    Create an API object using OAuthHandler
    Return the API object

Create API object by calling twitter_connection()
```

```
Define cleanTxt function to clean text:
    Remove @mentions, #hashtags, RT, hyperlinks, and emojis from text
    Return cleaned text

Define extract_mentions function to extract @mentions from text:
    Use regex to find @mentions in text
    Return list of @mentions

Define extract_hashtag function to extract hashtags from text:
    Use regex to find hashtags in text
    Return list of hashtags

Define getSubjectivity function to calculate subjectivity of text using TextBlob:
    Return subjectivity score

Define getPolarity function to calculate polarity of text using TextBlob:
    Return polarity score

Define getAnalysis function to classify polarity score into Negative, Neutral, or Positive:
    Return sentiment analysis label based on polarity score

Create a caching function using st.cache decorator to preprocess data:
    If platform is Twitter:
        Use Tweepy Cursor to fetch tweets based on query and number of tweets
        Extract tweets, mentions, hashtags, links, retweets from fetched data
        Clean tweets using cleanTxt function
        Remove specific keywords from tweets
        Calculate subjectivity and polarity of tweets
        Classify polarity into sentiment analysis labels
        Return preprocessed data

Define download_data function to download data as CSV:
    Generate a download button using st.download_button
    Return download button

Define analyse_mention function to analyze @mentions in data:
    Extract @mentions from data
    Count occurrences of @mentions and select top 10
    Return top 10 @mentions

Define analyse_hashtag function to analyze hashtags in data:
    Extract hashtags from data
    Count occurrences of hashtags and select top 10
    Return top 10 hashtags

Define graph_sentiment function to graph sentiment analysis results:
    Count sentiment analysis labels in data
    Sort sentiment analysis counts
    Return sorted sentiment analysis counts

# Main code:
Initialize API object
Define functions for data preprocessing, analysis, and visualization
```

5 Functional Testing (Black Box)

Test ID	Test Condition	Test Steps	Test Inputs	Test Expected Results	Actual Result	Status	Result
TC_01	Empty input text	1. Click on the predict button	(empty string)	A warning message is displayed prompting the user to provide some text.	(Empty string)	pass	User prompted to input text before prediction

Table 1: Functional Testing (Black Box)

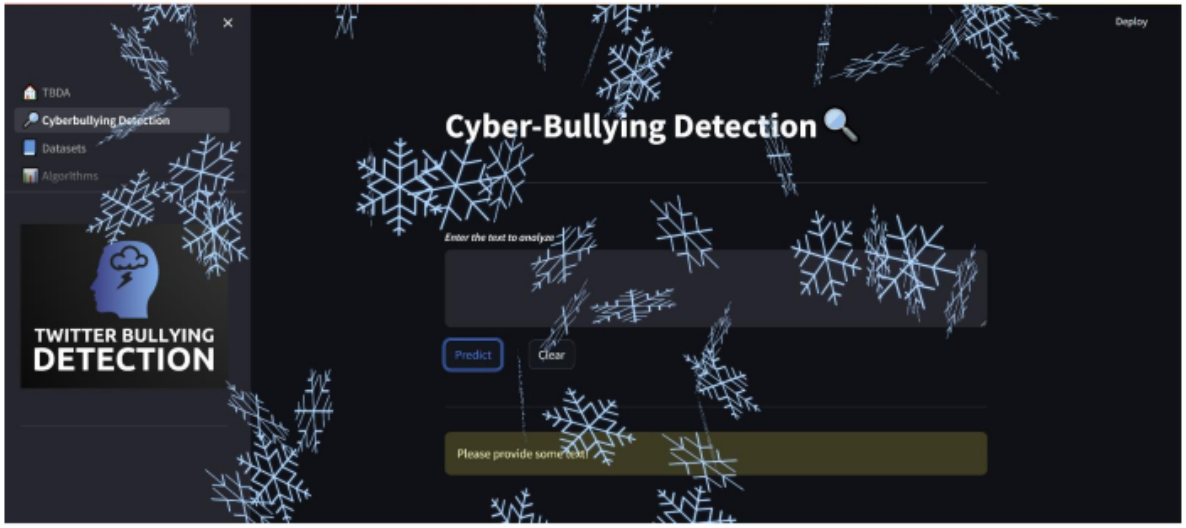

Figure 6: Test 1

TC_02	Non-Empty Input Text	1. Enter a sample text in the text area. 2. Click on the Predict button.	"This is a test text."	Display the prediction result as "Not cyberbullying" or "Cyberbullying"	"Not Cyberbullying"	Pass	Correct prediction result displayed

Table 2: Functional Testing (Black Box)

Figure 7: Test 2

TC_03	Clear Button Function-ality	1. Enter a sample text in the text area. 2. Click on the Clear button.	"This is a test text."	Text area is cleared.	(Empty string)	Pass	Text area cleared success-fully.

Table 3: Functional Testing (Black Box)

Figure 8: Test 3

TC_04	Preprocessing - Clean Text	1. Enter a sample text containing URLs, usernames, hashtags, emojis, special characters, RT, and numbers. 2. Click on the Predict button.	”RT @user1: Hello! #testing :smile: 123 https://example.com”	Cleaned text should not contain URLs, usernames, hashtags, emojis, RT, etc.	”Hello testing”	Pass	Text cleaned successfully.

Table 4: Functional Testing (Black Box)

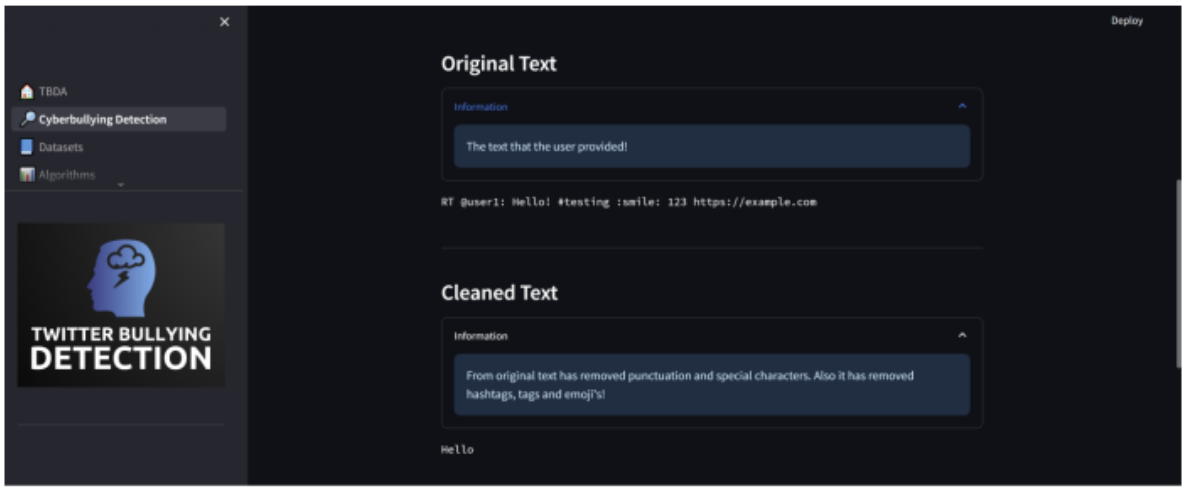

Figure 9: Test 4

TC_05	Preprocessing - Transform Text	1. Enter a sample text. 2. Click on the Predict button.	"This is a test text."	Transformed text should be in lowercase, without stopwords, and stemmed.	"test text"	Pass	Text transformed successfully.

Table 5: Functional Testing (Black Box)

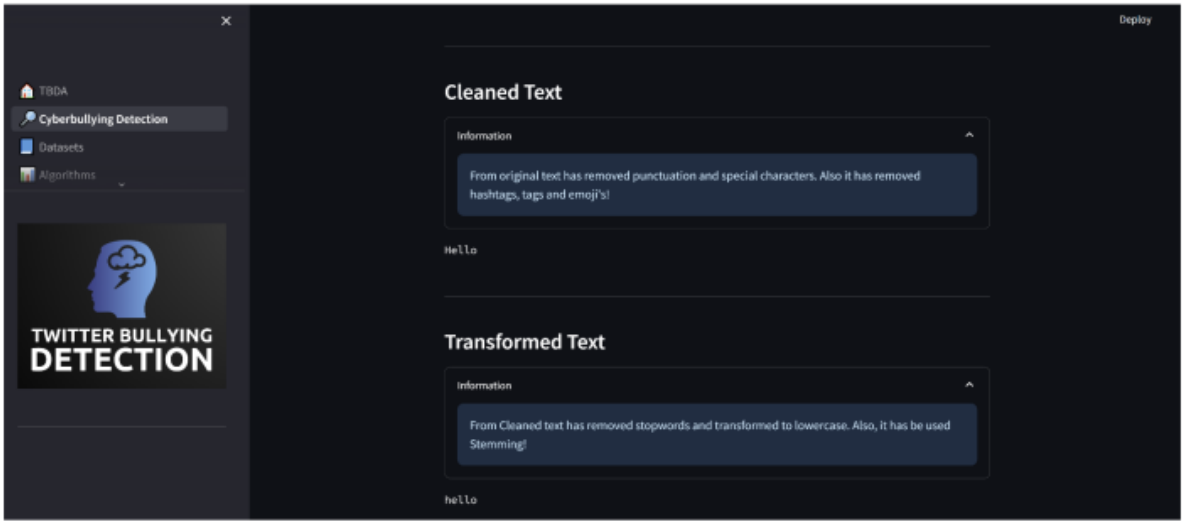

Figure 10: Test 5

TC_06	Binaru prediction - Cyber-bullting detected	1. Enter a sample text containing cyber-bullying content 2. Click on the predict button.	"You're a loser!"	Display the prediction result as "cyber-bullying"	Pass	Correctly predicts cyberbul-lying

Table 6: Functional Testing (Black Box)

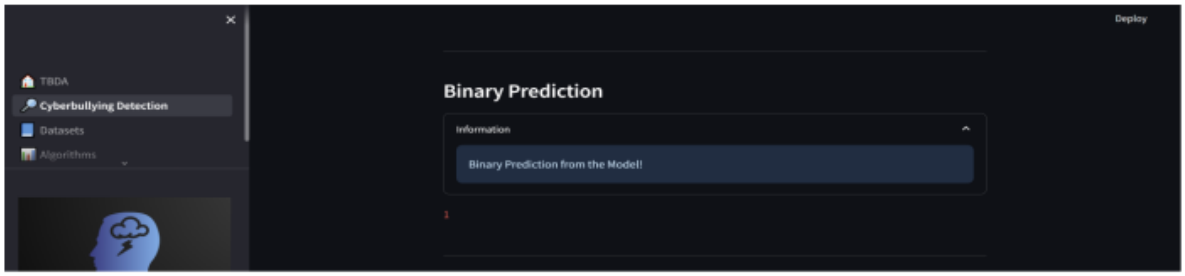

Figure 11: Test 6

TC_07	binary prediction - No cyberbullying	1. Enter a sample text without cyberbullying content 2. Click on the Predict button	"This is a normal text."	Display the prediction result as "Not Cyberbullying"	"Not Cyberbullying"	Pass	Correctly predicts no cyberbullying.

Table 7: Functional Testing (Black Box)

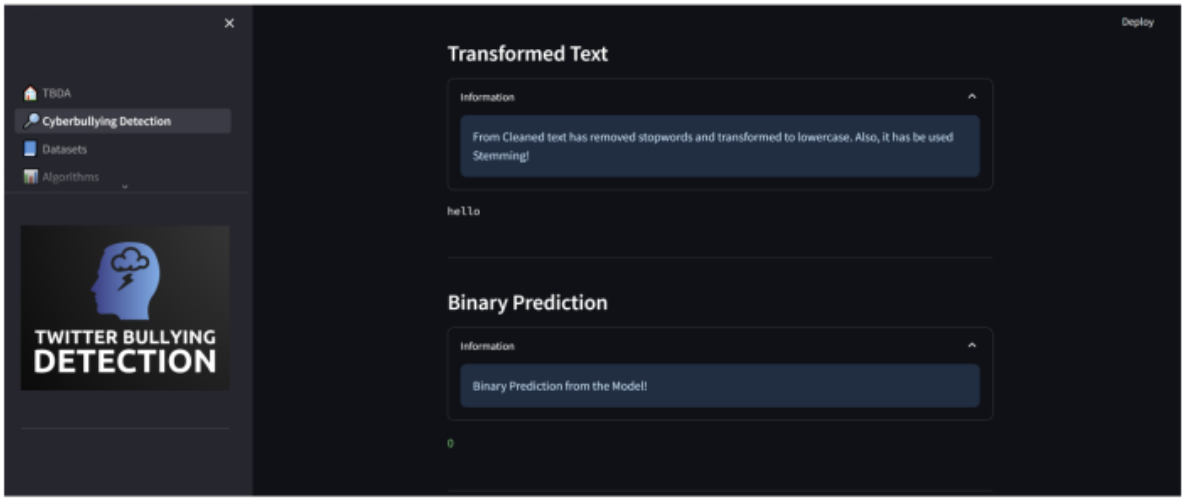

Figure 12: Test 7

TC_08	Model Accuracy Display	1. Click on the Predict button after entering any text.	(Any input)	Display the model accuracy.	"Accuracy: 91.70 percent"	Pass	Model accuracy displayed correctly

Table 8: Functional Testing (Black Box)

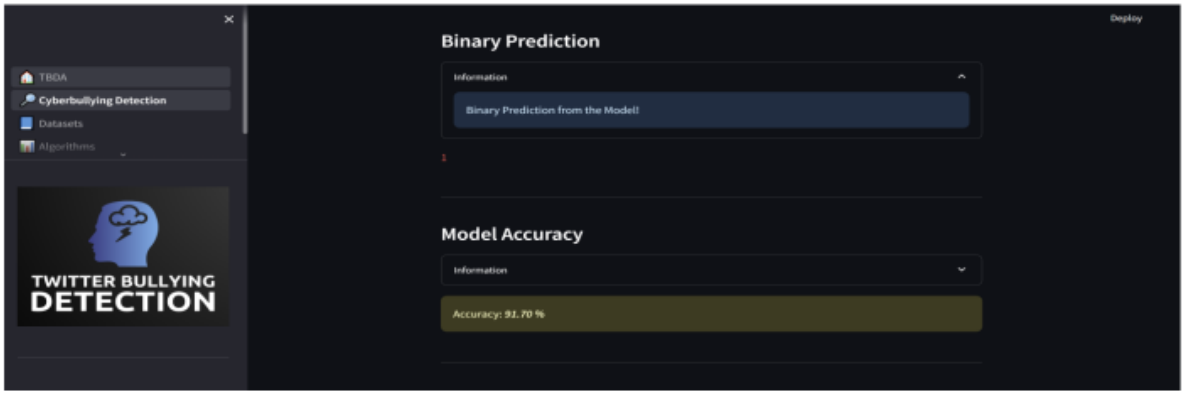

Figure 13: Test 8

TC_09	No selection made	1. Execute the code without selecting any dataset, vectorizer, and machine learning algorithm. 2. Check the warning message displayed.	Data Choice: "Select a Dataset" Vectorizer Choice: "Select a Vectorizer" Model Choice: "Select a Machine Learning Algorithm"	Warning message should be displayed asking to select dataset, vectorizer, and machine learning algorithm.	Warning message is displayed as expected.	Pass	.

Table 9: Functional Testing (Black Box)

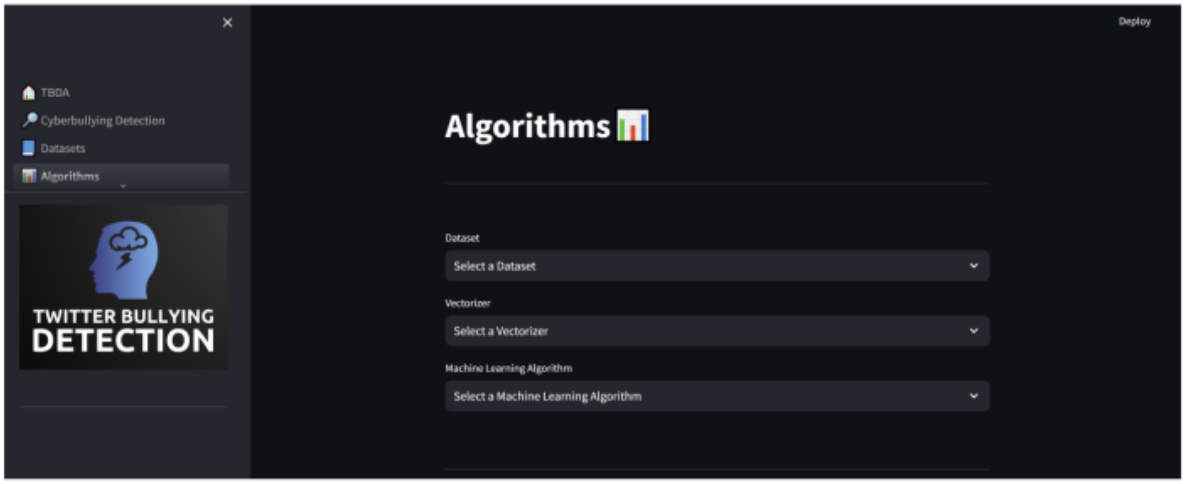

Figure 14: Test 9

TC_10	Dataset not selected	1. Execute the code without selecting a dataset. 2. Check the warning message displayed.	Data Choice: "Select a Dataset" Vectorizer Choice: "TF-IDF" Model Choice: "Logistic Regression"	Warning message should be displayed asking to select a dataset.	Warning message is displayed as expected.	Pass	-

Table 10: Functional Testing (Black Box)

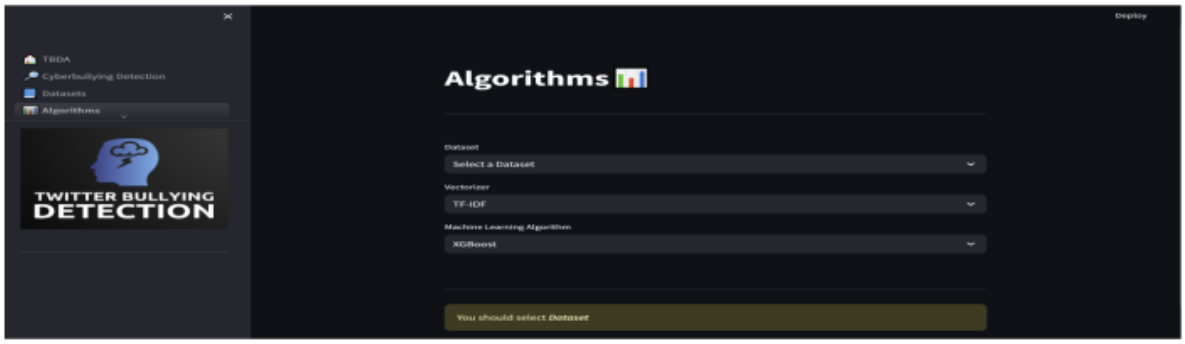

Figure 15: Test 10

TC_11	Vectorizer not selected	1. Execute the code without selecting a vectorizer. 2. Check the warning message displayed.	Data Choice: "Cyber Bullying Types Dataset" Vectorizer Choice: "Select a Vectorizer" Model Choice: "Logistic Regression"	Warning message should be displayed asking to select a vectorizer.	Warning message is displayed as expected.	Pass	.

Table 11: Functional Testing (Black Box)

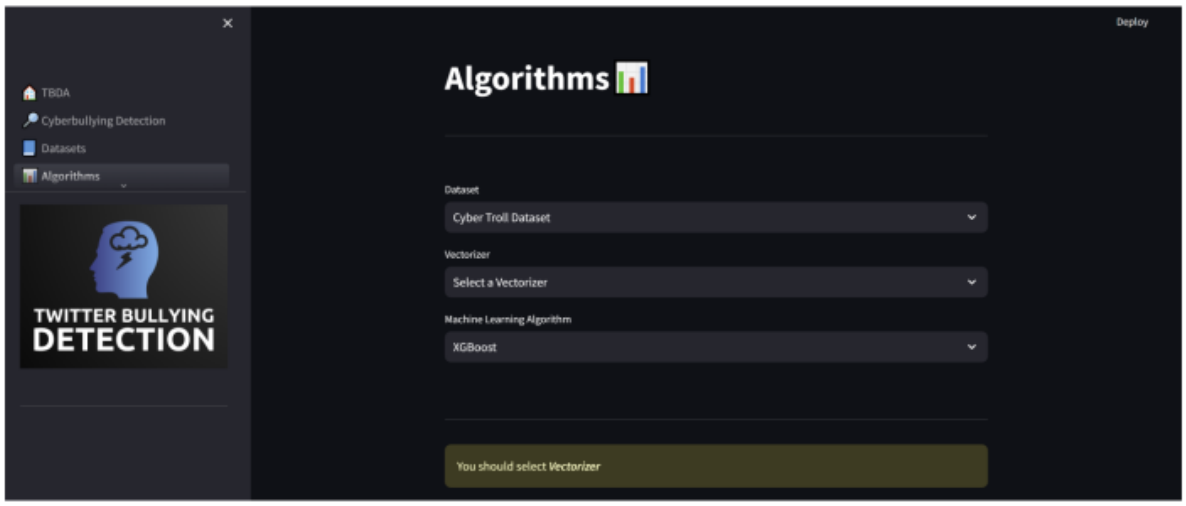

Figure 16: Test 11

TC_12	Model not selected	1. Execute the code without selecting a machine learning algorithm. 2. Check the warning message displayed.	Data Choice: "Cyber Bullying Types Dataset" Vectorizer Choice: "TF-IDF" Model Choice: "Select a Machine Learning Algorithm"	Warning message should be displayed asking to select a machine learning algorithm.	Warning message is displayed as expected.	Pass	.

Table 12: Functional Testing (Black Box)

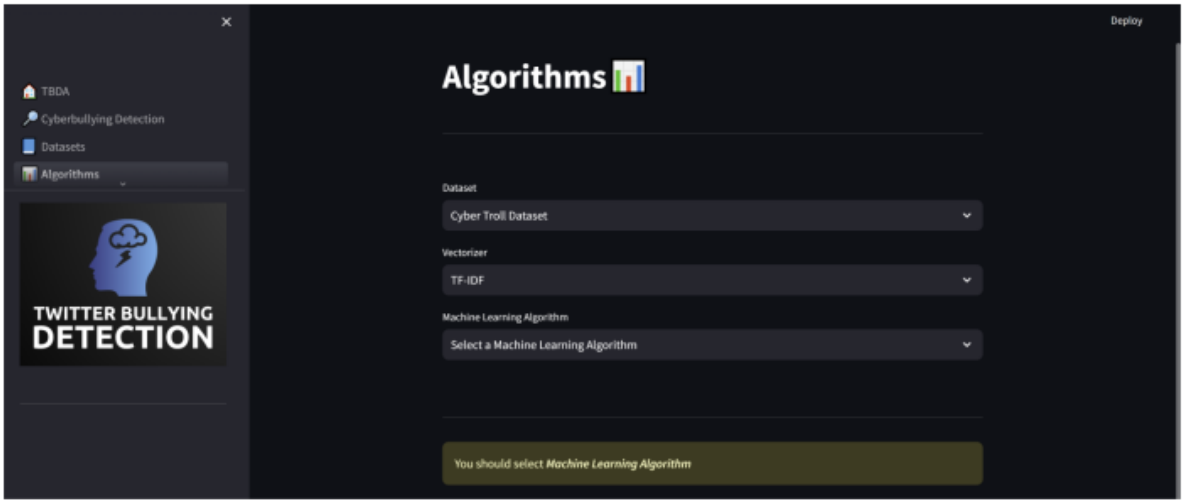

Figure 17: Test 12

TC_13	Dataset, Vectorizer, and Model selected	1. Execute the code by selecting the dataset, a vectorizer, and a machine learning algorithm. 2. Check the accuracy displayed based on the selected options.	Data choice by "Cyber Bullying Types Dataset" vectorizer choice: "TF-IDF" model Choice: "Logic=stic regression"	Accuracy should be displayed based on the selected options.	Accuracy is displayed as expected.	Pass	.

Table 13: Functional Testing (Black Box)

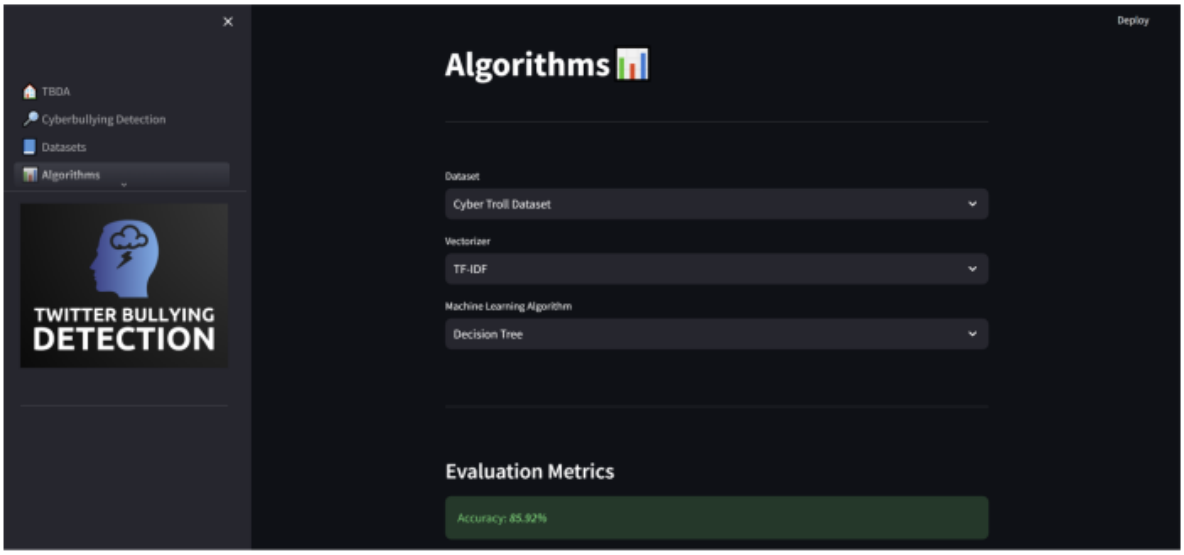

Figure 18: Test 13

By covering these edge cases in our testing, we ensured that the code functions correctly and provides a robust user experience across various scenarios.

6 Overall Project Snippets

6.1 Homepage

Figure 19: Homepage

6.2 Tweet Analysis Page

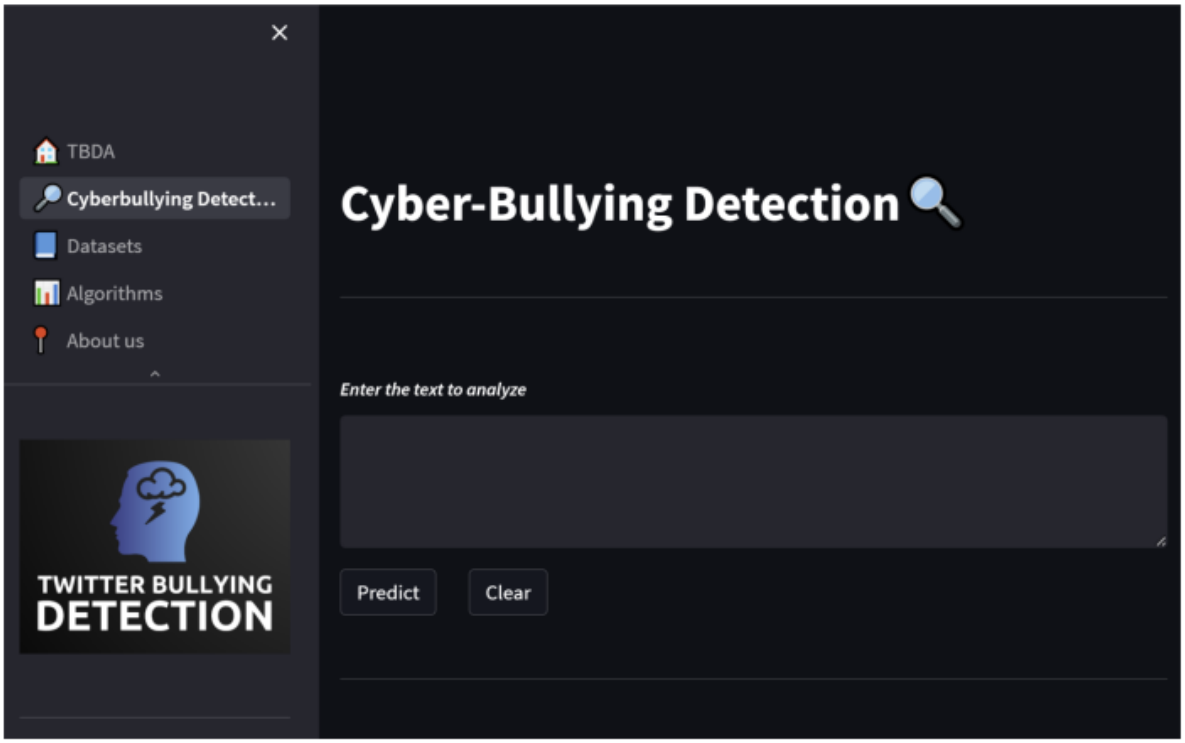

Figure 20: Tweet Analysis Page

6.3 Data Testing Page

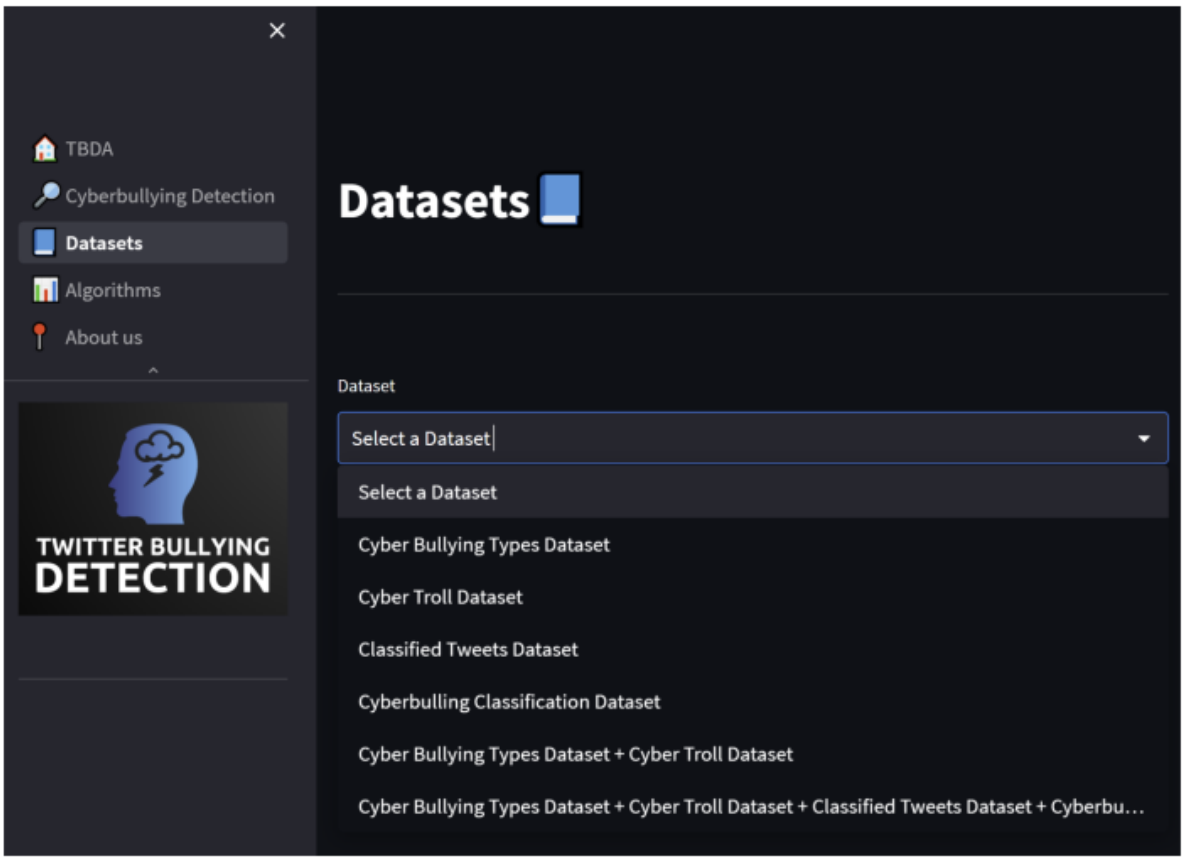

Figure 21: Dataset Testing Page

6.4 Data Testing Filter

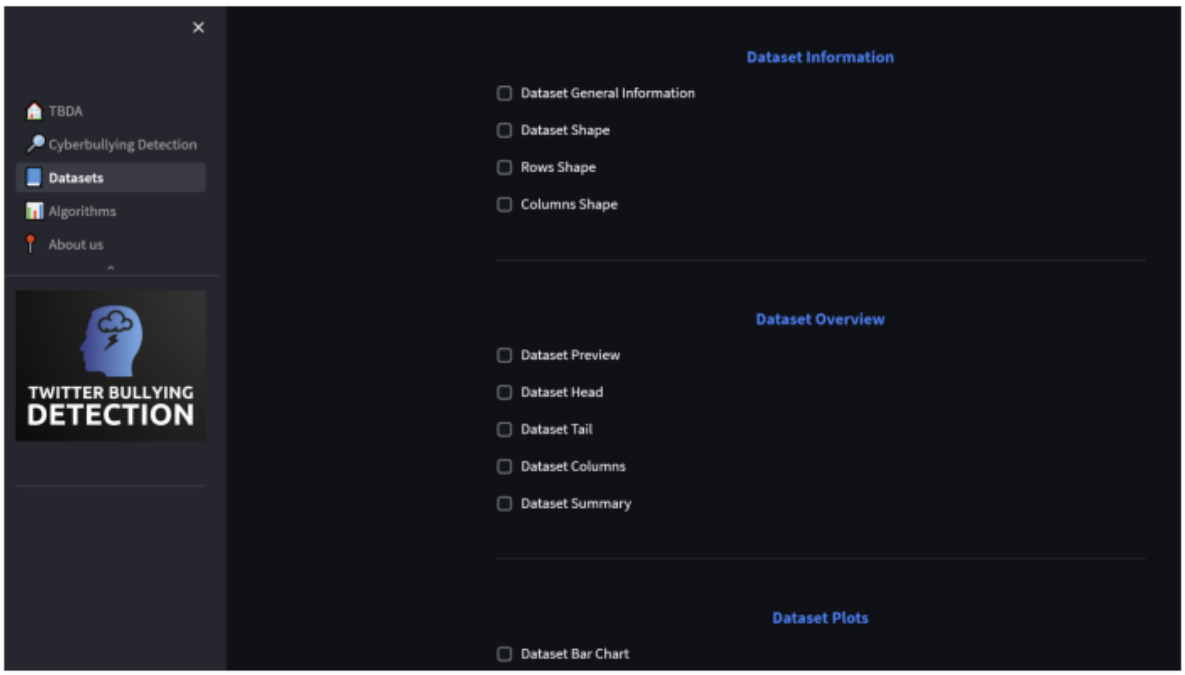

Figure 22: Dataset Testing Filter

6.5 Algorithm Testing Page

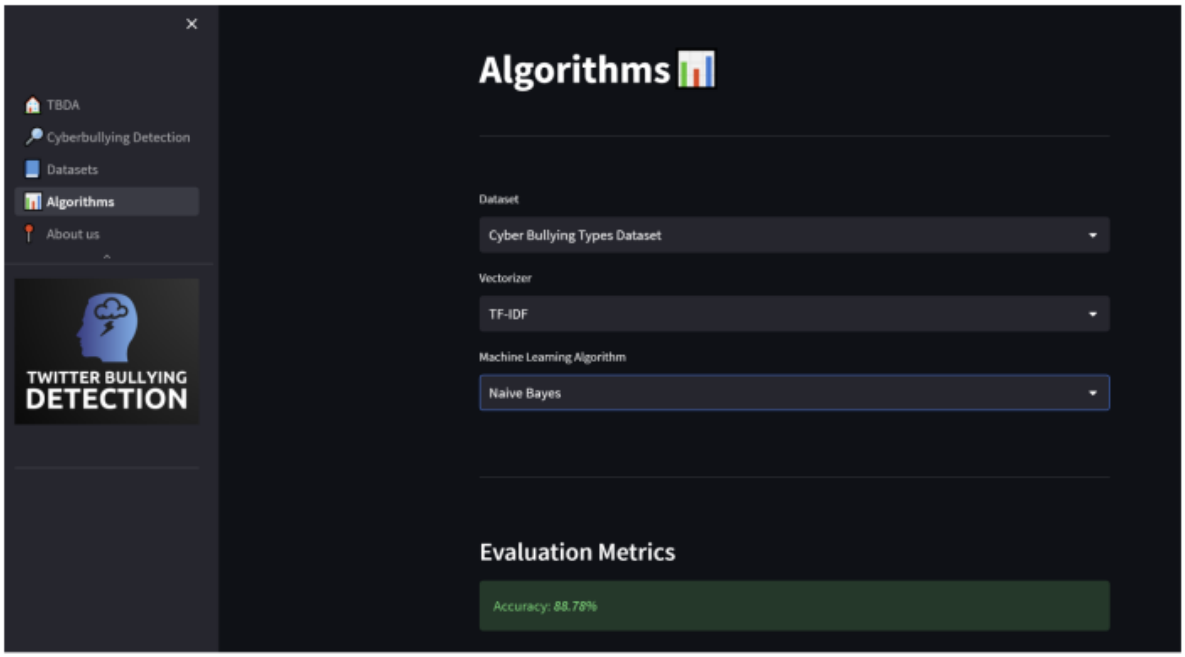

Figure 23: Algorithm Testing Page

6.6 About Us Page

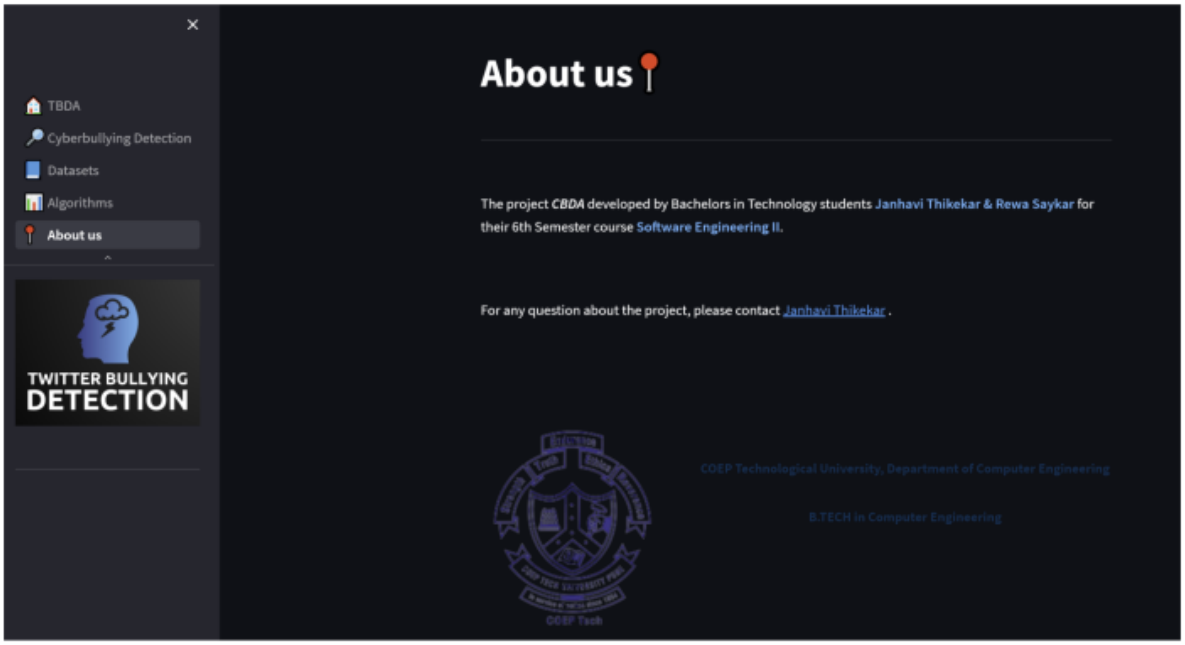

Figure 24: About Us Page

7 Conclusion

The conclusion of the cyberbullying analysis project emphasizes the user-centric approach taken to empower users in exploring and utilizing various tools and methodologies for text classification tasks related to cyberbullying. Here's an elaboration: In conclusion, our cyberbullying analysis project introduces an innovative application designed to facilitate user interaction and decision-making in tackling cyberbullying through text classification. By providing a user-friendly interface, we aim to democratize access to advanced machine-learning techniques for addressing this pressing societal issue. The application offers users the flexibility to experiment with different combinations of datasets, vectorization methods, and machine learning algorithms. This versatility allows users to tailor their approach to the specific nuances and challenges of cyberbullying detection and classification. Whether utilizing traditional algorithms or cutting-edge deep learning models, users have the tools at their disposal to explore and assess various strategies for combating cyberbullying effectively. One of the key strengths of the application is its emphasis on performance evaluation. By presenting accuracy metrics in a clear and accessible manner, users can quickly gauge the effectiveness of different models and approaches. This enables informed decision-making, empowering users to select the most suitable methods for their particular context and objectives. Overall, the application represents a significant step forward in leveraging machine learning for the prevention and mitigation of cyberbullying. By placing powerful tools in the hands of users and prioritizing usability and performance evaluation, we hope to foster a community- driven approach to combating cyberbullying and promoting a safer online environment for all.

7.1 Gnatt chart :

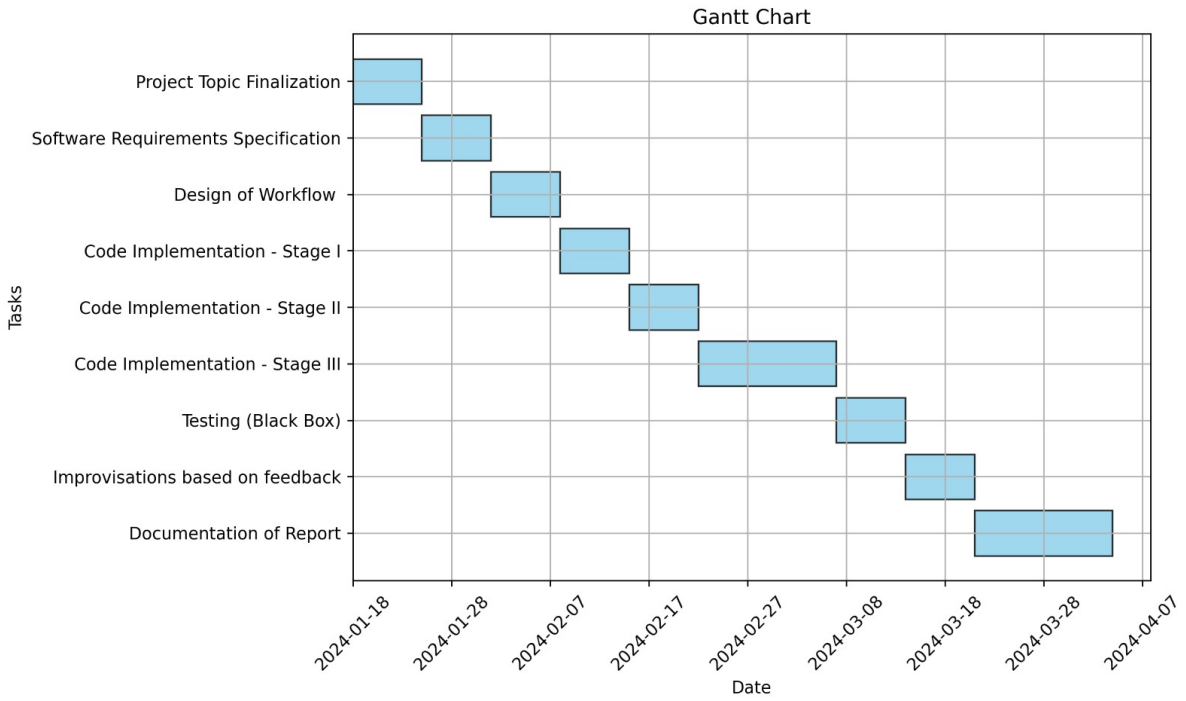

Figure 25: About Us Page

8 Future Work

The future work that we would like to put in efforts are:

1. Enhancing the visualizations: You can consider incorporating more visualizations such as confusion matrices, precision-recall curves, or ROC curves to provide users with a deeper understanding of model performance.

2. Model tuning: Implement hyperparameter tuning techniques such as grid search or random search to optimize the performance of the machine learning models further.

3. Feature engineering: Explore additional features or preprocessing techniques to improve model performance, such as word embeddings or text augmentation.

4. Deployment: Deploy the application on a web server to make it accessible to a broader audience. You can use platforms like Heroku or AWS for deployment.

5. User feedback: Collect feedback from users to understand their needs better and incorporate new features or improvements accordingly. This can help in making the application more robust and user-friendly.