Analyzing Social Media Posts for Mental Health Disorder Detection

Submitted by

SOUMYADEEP NANDY (13000121033)

PRITHWISH SARKAR (13000121037)

SAGNIK MUKHOPADHYAY (13000121040)

ARKAPRATIM GHOSH (13000121058)

⟨ Group 29 ⟩

Final Year 7th Semester

⟨ September, 2024 ⟩

Submitted for the partial fulfillment for the degree of
Bachelor of Technology in
Computer Science and Engineering

Techno Main Salt Lake,
EM 4/1, Salt lake, Sector - V, Kolkata - 700091

Department of Computer Science and Engineering
Techno Main Salt Lake
Kolkata - 700 091
West Bengal, India

APPROVAL

This is to certify that the project entitled **"Analyzing Social Media Posts for Mental Health Disorder Detection"** prepared by **SOUMYADEEP NANDY** *(13000121033)*, **PRITHWISH SARKAR** *(13000121037)*, **SAGNIK MUKHOPADHYAY** *(13000121040)* and **ARKAPRATIM GHOSH** *(13000121058)* be accepted in partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering.

It is to be understood that by this approval, the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn thereof, but approves the report only for the purpose for which it has been submitted.

. . . . . .
(Signature of the Internal Guide)

. . . . . .
(Signature of the HOD)

. . . . . .
(Signature of the External Examiner)

. . . . . .

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

. . . . . .

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project guide in the department of Computer Science and Engineering. We are extremely thankful for the keen interest our guide took in advising us, for the books, reference materials and support extended to us.

Last but not the least we convey our gratitude to all the teachers for providing us the technical skill that will always remain as our asset and to all non-teaching staffs for the gracious hospitality they offered us.

Place: Techno Main Salt Lake
Date: October 5, 2024

SOUMYADEEP NANDY (13000121033)

PRITHWISH SARKAR (13000121037)

SAGNIK MUKHOPADHYAY (13000121040)

ARKAPRATIM GHOSH (13000121058)

Table of Content

List of Figures

Abstract

This project aims to analyze social media posts for early detection of mental health disorders. Specifically, it focuses on using machine learning techniques to classify social media text data based on potential mental health issues. The problem lies in efficiently detecting patterns that indicate mental health conditions within large, unstructured datasets. Using methods like Support Vector Machines (SVM) and k-Nearest Neighbors (k-NN), the project seeks to enhance the accuracy of sentiment analysis and mental health issue prediction. Expected results include a robust classifier capable of distinguishing between different mental health concerns with high accuracy. This project could provide a valuable tool for mental health monitoring on social platforms.

1 Introduction

1.1 Project Overview

Mental health has become a critical global issue, with millions of people affected by various mental disorders, including depression, anxiety, and others. With the increasing use of social media, these platforms have emerged as spaces where people often express their emotions and struggles, sometimes unknowingly revealing signs of mental health challenges. This project focuses on analyzing social media posts to detect mental health disorders using advanced machine learning techniques. By examining language patterns, sentiment, and contextual usage in text data, this project aims to classify posts that potentially indicate mental health issues. Such detection can facilitate early intervention and help direct individuals to appropriate mental health services.

1.2 Project Purpose

The main goal of this project is to leverage machine learning to create a predictive model capable of identifying signs of mental health disorders from social media posts. This aligns with a broader goal of using technology to address public health concerns by enabling early detection through data analysis. Specifically, we use classification models such as Support Vector Machines (SVM) and k-Nearest Neighbors (k-NN) to predict mental health issues based on text patterns and sentiments. The project also addresses the technical challenges of processing large datasets and optimizing algorithms for accurate classification.

1.3 Technical Domain Specifications

This project falls within the intersection of natural language processing (NLP) and machine learning (ML), leveraging techniques such as sentiment analysis, text vectorization, and classification algorithms. Here are the key technical domain specifications:

- **Hardware** : The project does not require specialized hardware beyond a standard machine with adequate processing power. However, for larger datasets or complex model training, a machine equipped with a GPU (Graphics Processing Unit) could significantly reduce processing time. The project can be run on any system with at least 8GB of RAM and a multi-core processor.
- **Operating System** : The project is cross-platform and can be developed and executed on any modern operating system, including:
  - Windows 10/11
  - macOS
  - Linux distributions (Ubuntu, Linux Mint, etc.) A Linux-based system is often preferred in machine learning projects due to its stability and support for tools like TensorFlow, PyTorch, and other libraries used for model training.
- **Software** :
  - **Programming Languages** : Python 3.x will be the primary programming language, given its extensive libraries for machine learning, data analysis, and NLP.
  - **Libraries / Frameworks** :
    - **Scikit-learn** : Used for machine learning algorithms (k-NN, SVM) and model evaluation.
    - **Pandas** : For data manipulation and preprocessing.
    - **NumPy** : To handle large arrays and matrices, which are crucial for efficient numerical computations.
    - **NLTK and spaCy** : For text preprocessing and natural language understanding.
    - **Matplotlib and Seaborn** : For data visualization.
  - **Development Environment** :
    - **Jupyter Notebook** : For interactive development, experimentation, and visualization.
    - **Anaconda** : A distribution that simplifies package management and deployment.

    * **Google Colab** : For cloud-based execution when working with larger datasets or GPU-based model training.

1.4 Business Domain Specifications

From a business perspective, this project holds significant value across various sectors, particularly those that intersect with mental health monitoring, public health awareness, and social media governance. With the increasing prevalence of mental health issues globally, organizations within these industries are searching for innovative solutions to mitigate the growing mental health crisis. Leveraging machine learning for early detection of mental health disorders from social media data can revolutionize how mental health is addressed at both individual and societal levels. Below is a detailed exploration of how this project can impact different business domains:

- **Mental Health Services** : Mental health service providers—such as hospitals, therapy centers, and private practices—can greatly benefit from machine learning models capable of identifying early signs of mental health issues from social media data. In the traditional mental health setting, early detection of disorders like depression or anxiety often relies on self-reporting or clinical assessments, which may come too late in the progression of the disorder. By analyzing patterns in social media posts, these services can adopt a more proactive approach, reaching out to potential patients earlier in their mental health journey.

- **Social Media Platforms** : Social media platforms like Twitter, Facebook, Instagram, and others play an integral role in the public's expression of thoughts and feelings, including mental health struggles. These platforms face increasing pressure to safeguard the well-being of their users. This project's machine learning models can enable these companies to offer valuable services to users while adhering to ethical standards.

- **Public Health Organizations** : Public health organizations are tasked with monitoring and improving the mental well-being of the population on a large scale. For these organizations, access to real-time data from social media can provide a comprehensive view of the mental health landscape, identifying emerging trends and enabling data-driven interventions. Understanding how mental health is being discussed online can help public health organizations create more effective mental health awareness campaigns. Tailored messaging based on the language patterns identified by the model can lead to better engagement with individuals suffering from mental health issues.

1.5 Glossary / Keywords

Term	Definition
Machine Learning (ML)	A subset of artificial intelligence (AI) that enables computers to learn from data and make predictions or decisions without explicit programming.
Natural Language Processing (NLP)	A branch of artificial intelligence focused on the interaction between computers and humans through natural language, including tasks like text analysis.
Support Vector Machines (SVM)	A supervised learning algorithm used for classification or regression tasks, focusing on finding a hyperplane that best separates different classes.
k-Nearest Neighbors (k-NN)	A simple, non-parametric classification algorithm that assigns a class to a point based on the majority class of its 'k' nearest neighbors in the dataset.
Sentiment Analysis	A technique in NLP that analyzes the emotional tone behind a body of text, typically categorizing it as positive, negative, or neutral.
Bag of Words (BoW)	A text representation technique in NLP where text is represented as a collection of words, disregarding grammar and word order but keeping multiplicity.
Vectorization	The process of converting textual data into numerical form (such as a vector) so that it can be used as input for machine learning models.
Classifier	A machine learning model or algorithm that categorizes or labels data points into predefined classes.
Mental Health Disorder	A wide range of conditions that affect mood, thinking, and behavior, including depression, anxiety, schizophrenia, etc.
Data Preprocessing	The process of preparing raw data for analysis by cleaning, normalizing, and transforming it into a usable format for machine learning models.
Cross-validation	A model validation technique used to assess how well a model performs by dividing data into training and testing sets multiple times for better accuracy.
Precision	In the context of classification, precision refers to the accuracy of positive predictions, calculated as the ratio of true positives to the sum of true and false positives.
Recall	In classification, recall measures the ability of a model to identify all relevant instances within a dataset, calculated as the ratio of true positives to the sum of true positives and false negatives.

Term	Definition
Depression	There is a difference between depression and mood swings or short-lived emotional reactions to daily experiments; A mental state causing painful symptoms adversely disrupts normal activities (e.g., sleeping).
Anxiety	Several behavioral disturbances are associated with anxiety disorders, including excessive fear and worry. Severe symptoms cause significant impairment in functioning cause considerable distress. Anxiety disorders come in many forms, such as social anxiety, generalized anxiety, panic, etc.
Bipolar Disorder	An alternating pattern of depression and manic symptoms is associated with bipolar disorder. An individual experiencing a depressive episode may feel sad, irritable, empty, or lose interest in daily activities. Emotions of euphoria or irritability, excessive energy, and increased talkativeness can all be signs of manic depression. Increased self-esteem, decreased sleep need, disorientation, and reckless behavior may also be signs of manic depression.
Post-Traumatic Stress Disorder (PTSD)	In PTSD, persistent mental and emotional stress can occur after an injury or severe psychological shock, characterized by sleep disturbances, constant vivid memories, and dulled response to others and the outside world. People who re-experience symptoms may have difficulties with their everyday routines and experience significant impairment in their performance.

2 Related Studies

The intersection of social media analytics and mental health research has received increasing attention in recent years, leading to several important studies that highlight the potential for early detection and intervention. This section reviews key findings from various studies, emphasizing the relevance and applicability of social media data for identifying mental health disorders.

One of the seminal works in this domain is by Choudhury et al. (2013), who explored the predictive capabilities of social media content in identifying depression. They analyzed Twitter data and discovered that specific linguistic patterns, such as the use of negative emotion words, correlated strongly with self-reported depressive symptoms. This study demonstrated that social media could serve as a valuable resource for predicting mental health conditions, offering a potential tool for clinicians and researchers alike [2].

Similarly, Guntuku et al. (2017) conducted an integrative review that focused on detecting mental illness through social media. Their work synthesized various approaches and method-

ologies used in the field, providing insights into the effectiveness of different machine learning algorithms and sentiment analysis techniques. They found that social media platforms are rich sources of data that can reveal critical information about users' mental health, advocating for the development of robust systems to analyze this data effectively [3].

A systematic review by Mathur et al. (2022) further emphasized the significance of mental health classification on social media. They examined various studies that utilized machine learning techniques for mental health detection, highlighting the success of these models in identifying depression and anxiety based on user-generated content. Their findings reinforced the notion that social media can be leveraged not only for individual assessments but also for broader epidemiological studies to understand population mental health trends [4].

In addition, Nadeem (2016) contributed to the discussion by investigating depression identification on Twitter. The study focused on developing algorithms that could discern emotional cues in tweets, indicating a user's mental state. The findings revealed that simple text analysis could lead to significant improvements in identifying at-risk individuals, further validating the potential of social media data in mental health monitoring [5].

Research by AlSagri and Ykhlef (2020) introduced a machine learning-based approach specifically for depression detection on Twitter. Their study incorporated both content and activity features, demonstrating that a combination of linguistic and behavioral analysis could enhance the accuracy of depression identification. This work illustrated the multifaceted nature of social media data and its ability to capture not just what users say but also how they interact online [1].

In a more recent study, Vaishnavi et al. (2022) investigated the application of various machine learning algorithms for predicting mental health illnesses. They found that certain algorithms outperformed others in classifying mental health conditions based on social media posts. This study provided a comparative analysis that could inform future research directions, emphasizing the importance of algorithm selection in the context of mental health detection [7].

Lastly, Safa et al. (2023) presented a roadmap for future development in predicting mental health using social media. Their work highlighted the ongoing challenges in the field, including ethical considerations and the need for improved data privacy measures. They emphasized that while social media offers rich data for mental health analysis, researchers must approach this opportunity with a strong ethical framework to ensure user safety and data security [6].

These studies collectively underscore the growing body of evidence supporting the integration of social media analytics and machine learning for mental health detection. They provide a solid foundation for the current project, which aims to enhance existing methodologies and develop a predictive model for identifying mental health disorders from social media posts.

3 Problem Definition and Preliminaries

3.1 Context and Background

Mental health disorders have become a significant public health concern worldwide. The World Health Organization (WHO) estimates that approximately 1 in 8 people globally experience mental health disorders, which encompass conditions such as depression, anxiety, bipolar disorder, and post-traumatic stress disorder (PTSD). The rise of social media platforms has changed how individuals express their mental health struggles, share experiences, and seek support. Posts on platforms like Reddit and Twitter provide a wealth of data reflecting real-time sentiments, issues, and conversations surrounding mental health. However, this vast amount of unstructured textual data presents challenges in effectively identifying and categorizing specific mental health disorders.

3.2 Objective

The primary objective of this project is to develop a robust system that can automatically analyze social media posts, specifically from Reddit and Twitter, to detect various mental health disorders. By leveraging Natural Language Processing (NLP) techniques and machine learning algorithms, this project aims to:

- **Classify Posts** : Accurately classify social media posts based on the type of mental health disorder mentioned, including but not limited to depression, anxiety, bipolar disorder, and PTSD.
- **Sentiment Analysis** : Assess the sentiment expressed in each post, categorizing it as positive, negative, or neutral to gain insights into the emotional state of the users.
- **Data Driven Insights** : Provide valuable insights into the prevalence and expression of mental health issues on social media, helping researchers, mental health professionals, and policymakers understand trends and patterns.

3.3 Challenges

- **Data Variability** : Social media posts can vary significantly in structure, style, and length. Users may employ slang, abbreviations, and informal language, making it difficult for algorithms to accurately interpret and classify posts.
- **Sentiment Complexity** : Mental health discourse often includes complex emotional expressions that may not conform to standard sentiment analysis frameworks. For example, a post might convey feelings of hope while simultaneously expressing despair, complicating sentiment classification.
- **Imbalanced Data** : Certain mental health issues may be underrepresented in social media discussions, leading to an imbalanced dataset. This imbalance can adversely affect model training and performance, making it harder to detect less frequent disorders.
- **Cultural and Contextual Nuances** : Mental health perceptions and discussions can vary across different cultures and contexts. The model needs to account for these nuances to avoid misclassification and provide accurate insights.
- **Privacy and Ethical Considerations** : Analyzing social media data raises ethical concerns regarding user privacy. It is crucial to handle sensitive information responsibly and comply with data protection regulations.

3.4 Scope

The scope of the project ”Analyzing Social Media Posts for Mental Health Disorder Detection” delineates the specific aspects that will be covered, the methodologies employed, and the boundaries within which the research and analysis will occur. The project aims to harness the potential of machine learning and natural language processing (NLP) techniques to analyze social media sentiment and its correlation with mental health disorders, focusing specifically on a Twitter sentiment dataset sourced from Kaggle. This dataset contains user-generated content that reflects various sentiments, opinions, and emotional states, making it a valuable resource for this analysis.

- **Dataset Selection and Characteristics**
  The primary data source for this project is the Twitter sentiment dataset from Kaggle. This dataset includes tweets that are labeled with sentiments, such as positive, negative, and neutral, along with other relevant metadata. The selection of this dataset is pivotal, as it encapsulates a wide range of mental health-related discussions and sentiments expressed by individuals on social media. Key characteristics of the dataset include:

  - **Diversity of Sentiment** : The dataset encompasses various sentiments that users express regarding mental health topics, providing a rich foundation for sentiment analysis.
  - **User Anonymity** : To respect user privacy and adhere to ethical standards, the dataset does not contain personally identifiable information (PII) about the Twitter users. This ensures compliance with data protection regulations while allowing for robust analysis.

- **Analysis Objectives**
  The project will focus on several key objectives:

  - **Sentiment Classification** : Using machine learning algorithms, the project aims to classify tweets into predefined sentiment categories. This classification will serve as a basis for understanding public perceptions of mental health.
  - **Correlation with Mental Health Issues** : The analysis will explore the correlation between identified sentiments and specific mental health disorders, thereby contributing to the understanding of how social media discourse reflects mental health challenges.
  - **Trend Analysis** : By analyzing sentiment trends over time, the project seeks to identify patterns in public discourse surrounding mental health, including any potential spikes in negative sentiments during particular events or crises.

- **Methodologies** : The project will employ various methodologies to achieve its objectives, including:

  - **Data Processing** : Cleaning and preparing the dataset to ensure that it is suitable for analysis. This includes tasks such as removing noise (e.g., URLs, hashtags), tokenization, and normalization of text.
  - **Feature Extraction** : Utilizing techniques like Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) to convert textual data into numerical representations suitable for machine learning algorithms.
  - **Machine Learning Techniques** : Implementing various machine learning algorithms, including k-Nearest Neighbors (k-NN) and Support Vector Machines (SVM), to classify tweet sentiments and evaluate their performance based on accuracy, precision, recall, and F1 score.
  - **Data Visualizations** : Employing visualization tools to present findings clearly, including sentiment distribution graphs, trend lines, and correlation heatmaps.

3.5 Exclusions

In delineating the boundaries of the project ”Analyzing Social Media Posts for Mental Health Disorder Detection,” it is crucial to specify what is excluded from the scope of this research to maintain a clear focus on the primary objectives. This project will not encompass the direct collection or real-time monitoring of Twitter data via the Twitter API, as it is solely reliant on the pre-existing Twitter sentiment dataset obtained from Kaggle. Therefore, any analysis involving the dynamic aspects of social media engagement, such as real-time sentiment shifts in response to current events or trending topics, will be outside the project's purview. Furthermore, the study will not address the technicalities of Twitter's platform-specific features, such as hashtags, user mentions, or retweets, in detail, as these elements are not central to the primary research focus on sentiment classification and its correlation with mental health issues. While the project aims to analyze sentiments surrounding mental health, it will deliberately exclude in-depth examinations of individual mental health cases or diagnoses, opting instead for a more generalized approach that focuses on sentiment trends in the broader population. Additionally, the research will not explore the ethical implications of data ownership or the responsibilities of social media platforms regarding user-generated content, as the focus will be primarily on data analysis techniques and outcomes rather than the broader ethical landscape. Finally, the project will not extend to investigating the efficacy of social media interventions or support systems for mental health, as it seeks to analyze sentiments rather than evaluate mental health services or outreach initiatives. These exclusions are designed to ensure that the project remains manageable, focused, and aligned with its core objectives of sentiment analysis and correlation with mental health disorder detection.

3.6 Assumptions

In the context of the project ”Analyzing Social Media Posts for Mental Health Disorder Detection,” several key assumptions underpin the research framework and methodologies employed. Firstly, it is assumed that the Twitter sentiment dataset obtained from Kaggle is representative of broader social media discourse regarding mental health issues, capturing a diverse range of sentiments expressed by users on the platform. This assumption is critical as it establishes the foundation for analyzing sentiment trends and their potential correlations with various mental health disorders. Secondly, it is presumed that the sentiments recorded in the dataset accurately reflect the users' true emotions and perspectives at the time of posting, thereby providing valid data for analysis. Furthermore, it is assumed that the textual data within the dataset can be effectively processed and interpreted through natural language processing (NLP) techniques, allowing for the accurate classification of sentiments and identification of patterns. Another assumption is that the selected machine learning algorithms, including k-Nearest Neighbors

(k-NN) and Support Vector Machines (SVM), will perform optimally with the provided data, leading to reliable and interpretable results regarding sentiment analysis and mental health correlations. Additionally, it is assumed that the sentiments expressed in social media posts can serve as a valid proxy for understanding public perceptions of mental health, enabling insights into societal attitudes and the potential stigmatization associated with these disorders. The project also assumes that the cleaning and preprocessing steps applied to the data will sufficiently prepare the dataset for analysis, minimizing noise and irrelevant information that could skew the results. Lastly, it is presumed that the ethical considerations surrounding the use of publicly available social media data have been adequately addressed, ensuring that the research adheres to relevant ethical standards and does not compromise user privacy or data integrity. These assumptions serve as the bedrock for the project's analytical framework, guiding the research processes and interpretations that follow.

4 Proposed Solution

The proposed work centers around "Analyzing Social Media Posts for Mental Health Disorder Detection," leveraging advanced data analytics and machine learning techniques to provide insights into the sentiments expressed in social media discussions related to mental health issues. This research is significant due to the increasing prevalence of mental health disorders globally and the role social media plays in shaping public perception and discourse around these issues. The core objective of this project is to develop a systematic approach to classify sentiments in tweets, thereby enabling better understanding and awareness of mental health conditions through social media analysis. Below, I outline the specific contributions and reusable components deployed in this project.

4.1 Special Contributions

- **Dataset Acquisition and Preparation** : The initial step involved sourcing a high-quality dataset from Kaggle, specifically the Twitter sentiment dataset. This dataset comprises user-generated tweets containing sentiments related to various mental health issues, serving as the primary data source for analysis. The preparation phase included extensive data cleaning and preprocessing, wherein missing values were handled, duplicate entries removed, and text normalization performed. This crucial step ensured the dataset's integrity and suitability for subsequent analysis, allowing for a more accurate representation of sentiments.

- **Text Vectorization** : To enable machine learning models to interpret textual data, a Bag of Words (BoW) model was implemented. This approach involved converting tweets into a numerical format by creating a matrix representation of word frequencies across the

dataset. The Scikit-learn library was instrumental in this process, offering functions for text vectorization and feature extraction. The reusable components for text preprocessing and vectorization were packaged into functions, allowing for easy application to future datasets or similar projects.

- **Implementation of Machine Learning Algorithms** : The project employed several machine learning algorithms, focusing primarily on k-Nearest Neighbors (k-NN) and Support Vector Machines (SVM) for sentiment classification. The k-NN algorithm was chosen for its simplicity and effectiveness in classifying data points based on proximity in the feature space. Hyperparameter tuning was conducted using GridSearchCV, which allowed for the optimization of parameters to enhance model performance. In addition to k-NN, SVM was implemented due to its robustness in handling high-dimensional data and binary classification tasks. SVM's ability to find the optimal hyperplane for separating different classes made it particularly suitable for the sentiment analysis task. The SVM implementation utilized Scikit-learn's built-in functionalities, enabling efficient model training and evaluation. Both k-NN and SVM models were developed as reusable components, encapsulated in functions that facilitate easy retraining and evaluation on new data.

- **Model Evaluation** : Comprehensive evaluation metrics were employed to assess the performance of the machine learning models. Metrics such as accuracy, precision, recall, and F1-score were calculated to provide a holistic view of model effectiveness in classifying sentiments related to mental health. This evaluation process not only demonstrated the models' capabilities but also highlighted areas for improvement, providing a foundation for future iterations of the project. The evaluation framework, including metrics calculations and visualization, was designed as reusable components to streamline future model assessments.

- **Insights and Recommendations** : A critical aspect of this project is the generation of actionable insights based on the analysis of social media sentiments. The findings from the sentiment classification can inform mental health professionals, researchers, and policymakers about public sentiment trends, potential stigma associated with mental health issues, and the effectiveness of awareness campaigns. Recommendations for mental health awareness strategies can be derived from understanding how sentiments vary across different demographics and regions. This interpretative analysis, combined with quantitative results, contributes valuable knowledge to the ongoing conversation about mental health in society.

- **Documentation and Reproducibility** : To enhance the usability and impact of the project, thorough documentation was maintained throughout the research process. This documentation includes detailed explanations of the methodologies employed, code snippets, and instructions for reproducing the results. The aim is to ensure that the components developed in this project can be easily utilized by other researchers and practitioners in the field. By documenting the code and methodologies, I am contributing to the open-source community, allowing for collaborative improvements and innovations in mental health sentiment analysis.

4.2 Reusable Components

- **Data Preprocessing Functions** : Modular functions designed for data cleaning and preprocessing, which can be reused across different datasets.
- **Text Vectorization Module** : A component that encapsulates the Bag of Words transformation process, enabling quick adaptation to new textual data.
- **Machine Learning Model Functions** : Functions for implementing k-NN and SVM, along with hyperparameter tuning capabilities, allowing for easy retraining on varying datasets.
- **Evaluation Framework** : A set of functions for calculating and visualizing evaluation metrics, making it easy to assess different models' performances.

5 Project Planning

5.1 Software Life Cycle Model

In developing the project, I adopted an iterative approach to the Waterfall model, enabling a structured yet flexible framework for managing the various phases of the project. The project plan outlines specific tasks, dependencies, timelines, and milestones to ensure a systematic progression towards the final goal of analyzing social media posts for mental health disorder detection.

The Iterative Waterfall model was chosen for this project due to its structured approach while allowing for iterative revisions and refinements. Unlike the traditional Waterfall model, which emphasizes a linear progression through distinct phases, the iterative variant permits revisiting earlier stages based on findings and feedback. This flexibility is particularly beneficial in data-driven projects where insights gained during the analysis may necessitate adjustments to earlier

stages, such as refining requirements or enhancing data preparation techniques.

In this project, the iterative nature of the Waterfall model facilitated ongoing improvement and adaptation throughout the development process. For example, initial results from the model evaluation phase may prompt a revisit to data preprocessing to enhance data quality or to explore alternative modeling techniques. This approach ultimately fosters a more robust final product, ensuring that the developed system meets the dynamic needs of mental health disorder detection in social media posts.

The key feature of the iterative approach is the feedback loop that exists between the phases. For instance, after completing the testing phase, if certain models do not meet performance expectations, the project can loop back to the implementation phase. This allows for modifications to the models, preprocessing techniques, or even revisiting the requirements to ensure alignment with the project's objectives.

The project is divided into distinct phases:

- **Requirement Gathering and Analysis** : This initial phase involved understanding the project's goals, objectives, and stakeholder expectations. It spanned approximately two weeks, culminating in a detailed requirements document that guided the subsequent stages.
- **Data Collection and Preparation** : Utilizing a Twitter sentiment dataset from Kaggle and Reddit API, the data collection phase was executed over a week. This included downloading the dataset, examining its structure, and performing data cleaning and prepossessing to ensure its suitability for analysis.
- **Model Development** : This phase, lasting about three weeks, included the creation of a Bag of Words model, splitting the dataset into training and test sets, and implementing various machine learning algorithms such as k-Nearest Neighbors (k-NN) and Support Vector Machines (SVM) for sentiment classification.
- **Model Evaluation** : Following model development, a week was allocated for rigorous testing and validation of the models, ensuring they met the required accuracy benchmarks. This phase involved using performance metrics such as accuracy, precision, recall, and F1-score to evaluate the models' effectiveness.
- **Final Deployment and Documentation** : The last phase, spanning two weeks, focused on deploying the best-performing model and creating comprehensive documentation. This included user manuals and technical documentation to facilitate future maintenance and enhancements.

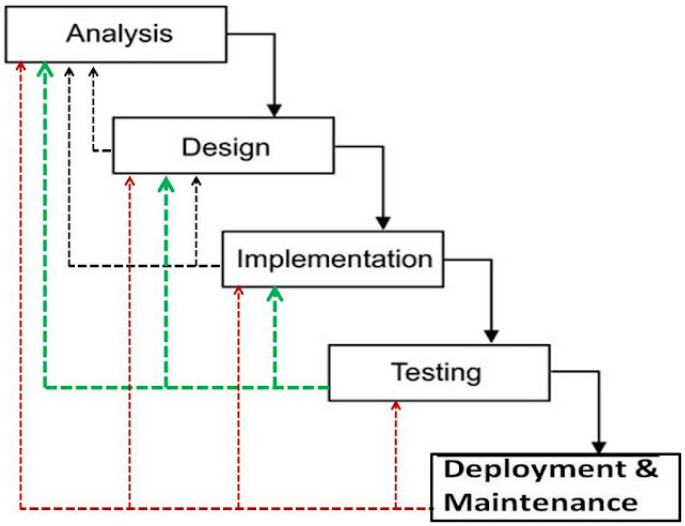


Figure 1: Iterative Waterfall Model

5.2 Dependencies and Milestones

Key dependencies were identified for successful project progression. For instance, completion of the data preparation phase was critical before proceeding to model development. Milestones were established at the end of each phase to ensure accountability and track progress. The successful completion of the requirement gathering phase marked the first milestone, followed by the data preparation phase, and so on.

5.3 Scheduling

Effective scheduling is crucial to the success of any project, as it establishes a clear timeline for tasks, milestones, and dependencies. In the context of our project on detecting mental health disorders through social media analysis, a detailed schedule has been developed to guide the project from inception to completion. This schedule includes specific tasks such as requirement gathering, data preprocessing, model implementation, testing, and deployment, each with clearly defined deadlines. The iterative nature of our chosen methodology allows for flexibility within the schedule, enabling adjustments based on testing outcomes and stakeholder feedback. Key milestones, such as the completion of data analysis, model validation, and user acceptance testing, have been identified to monitor progress and ensure timely delivery of the final product. By utilizing project management tools, such as Microsoft Project, we can visualize and track the progress of tasks, manage resources effectively, and maintain open communication among team members, ensuring that the project stays on schedule and meets its objectives. This proactive approach to scheduling enhances our ability to deliver a high-quality solution that aligns with our goals and stakeholder expectations.

Task Mode	Task Name	Duration	Start	Finish	Predecessors
	Analyzing Social Media Posts for Mental Health Disorder Detection	**109 days**	**Wed 03-07-24**	**Sat 30-11-24**	
	Feasibility Study	6 days	Wed 03-07-24	Wed 10-07-24	
	Requirements Analysis	**15 days**	**Thu 11-07-24**	**Wed 31-07-24**	
	Requirements Gathering	9 days	Thu 11-07-24	Tue 23-07-24	2
	Analysis Of Requirements	6 days	Wed 24-07-24	Wed 31-07-24	2
	Design	**15 days**	**Thu 01-08-24**	**Wed 21-08-24**	
	High Level Design	8 days	Thu 01-08-24	Mon 12-08-24	5
	Low Level Design	7 days	Tue 13-08-24	Wed 21-08-24	7
	Coding	**30 days**	**Thu 22-08-24**	**Wed 02-10-24**	
	Data Collection and Preprocessing	10 days	Thu 22-08-24	Wed 04-09-24	8
	Model Development	15 days	Thu 05-09-24	Wed 25-09-24	10
	Refining Models	1 day	Thu 26-09-24	Thu 26-09-24	11
	Testing	**25 days**	**Thu 03-10-24**	**Wed 06-11-24**	
	Unit Testing	5 days	Thu 03-10-24	Wed 09-10-24	12
	Integration Testing	5 days	Thu 10-10-24	Wed 16-10-24	14
	System Testing	9 days	Thu 17-10-24	Tue 29-10-24	15
	Acceptance Testing	6 days	Thu 17-10-24	Thu 24-10-24	15
	Deployment and Documentation	**13 days**	**Thu 07-11-24**	**Mon 25-11-24**	
	Deployment	8 days	Thu 07-11-24	Mon 18-11-24	17
	Documentation	5 days	Tue 19-11-24	Mon 25-11-24	19
	Delivery	1 day	Tue 26-11-24	Tue 26-11-24	20

Figure 2: Project Plan

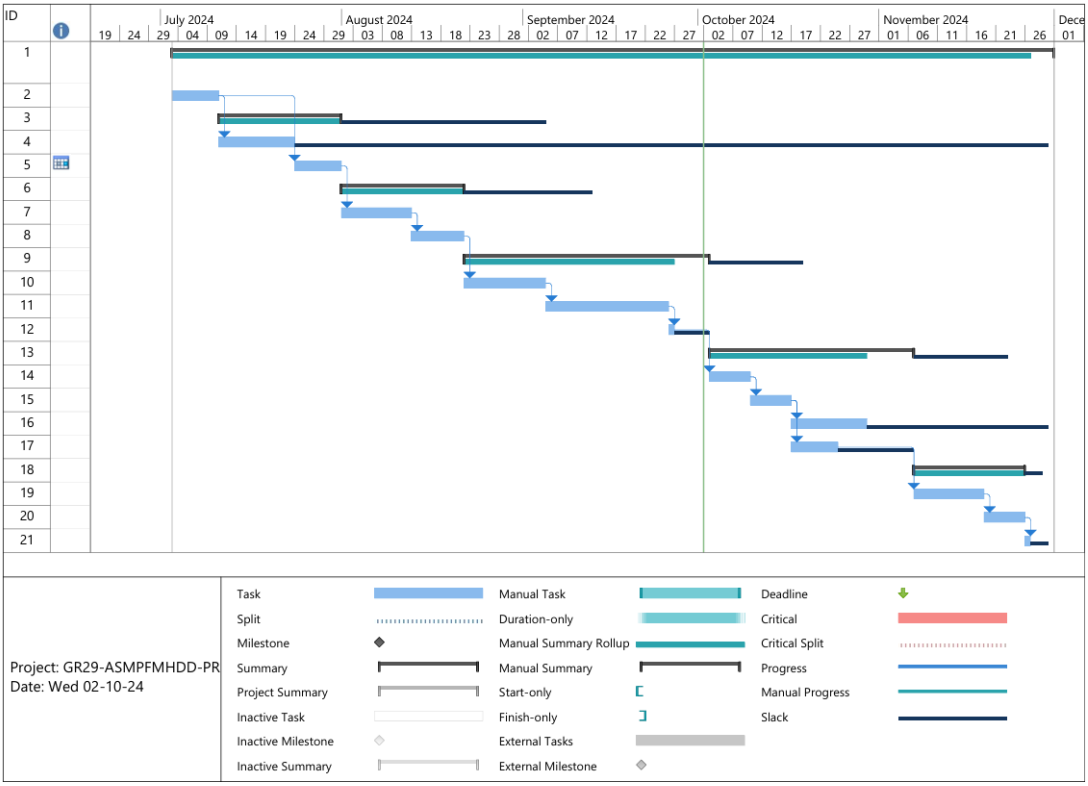


Figure 3: Gantt Chart

6 Requirement Analysis

6.1 Requirement Matrix

	A	B	C	D	E	F	G	H	I
1	**Requirement ID**	**Requirement Description**	**Priority**	**Category**	**Source**	**Status**	**Dependencies**	**Assigned To**	**Verification**
2	FR-001	Collect and preprocess social media data from Kaggle and Reddit.	High	Functional	Stakeholder	Completed	FR-002	Data Team	Data Analysis Review
3	FR-002	Implement data cleaning and feature extraction for NLP.	High	Functional	Project Docs	Completed	FR-001	Development Team	Feature Verification
4	FR-003	Train machine learning and deep learning models for mental health classification.	High	Functional	Stakeholder	In Progress	FR-001, FR-002	ML Team	Model Accuracy Testing
5	FR-004	Evaluate models using performance metrics (accuracy, recall, F1).	High	Functional	Stakeholder	In Progress	FR-003	ML Team	Performance Report
6	NFR-001	Document all technical and user aspects of the final solution.	Medium	Non Functional	Stakeholder	In Progress	FR-004	Documentation Team	Peer Review
7	NFR-002	Ensure model deployment is done on a scalable platform.	Medium	Non Functional	Project Docs	Not Started	FR-003, FR-004	Development Team	System Testing

Figure 4: Requirement Matrix

The Requirement Matrix is a comprehensive tool used to track and manage the key requirements of a project. It systematically organizes each requirement with a unique identifier, description, priority, and category (such as functional or non-functional). The matrix also records the source of the requirement, its current status (e.g., in progress, completed), any dependencies on other requirements, and the team or individual responsible for fulfilling it. Additionally, it includes a verification method to ensure the requirement is met, such as through testing or review. This structured format helps ensure that all requirements are clearly defined, prioritized, and tracked, enabling effective project management and ensuring alignment with stakeholder expectations.

6.2 Requirement Elaboration

6.2.1 Functional Requirements

Requirement ID: FR-001
Description: Data Collection
Priority: High
Category: Functional

The system requires an ability to collect and ingest a large dataset of Twitter sentiment data from Kaggle. The data should include text content from tweets, associated sentiment labels, and other metadata such as timestamp and user details. This will serve as the primary source of information for sentiment analysis and mental health disorder detection. The system must ensure that the dataset is loaded correctly into the machine learning environment, and any discrepancies in the structure should be handled with pre-processing steps like cleaning, normalization.

Requirement ID: FR-002
Description: Data Cleaning and Preprocessing
Priority: High
Category: Functional
The system must include modules to clean the raw data, such as removing irrelevant characters, handling missing data, and tokenizing text. For the social media posts, it is essential to remove URLs, stopwords, and unnecessary punctuation. The pre-processing pipeline should also convert the text into a suitable numerical format, such as Bag of Words (BoW) or Term Frequency-Inverse Document Frequency (TF-IDF), for further analysis. Proper pre-processing ensures that the data is in a form that can be efficiently used by machine learning models.

Requirement ID: FR-003
Description: Sentiment and Disorder Detection Model
Priority: High
Category: Functional
The system needs to implement machine learning algorithms such as k-Nearest Neighbors (k-NN) and Support Vector Machines (SVM) to classify social media posts based on their sentiment (positive, negative, neutral) and detect potential signs of mental health disorders. The system must be able to train these models on historical data and then apply them to predict the sentiment and detect mental health-related issues in new posts.

Requirement ID: FR-004
Description: Model Validation and Evaluation
Priority: High
Category: Functional
The system must evaluate the performance of the trained models by splitting the dataset into training and test sets. Various performance metrics like accuracy, precision, recall, and F1-score should be computed to assess the model's effectiveness in detecting mental health disorders. Based on the evaluation, the system should allow for model fine-tuning, such as adjusting hyperparameters, to improve the overall performance.

6.2.2 Non Functional Requirements

Requirement ID: NFR-001
Description: Documentation
Priority: Medium
Category: Non Functional

The addition of documentation and maintenance manuals as a non-functional requirement ensures that the system is not only usable in the short term but also maintainable and extensible over time. This guarantees that future updates and improvements to the system can be implemented without disrupting existing functionality or requiring a steep learning curve for new developers or users.

Requirement ID: NFR-002
Description: Scalability
Priority: Medium
Category: Non Functional
The system must be designed to efficiently process large volumes of data, considering the potential growth in the amount of social media posts that need to be analyzed. The data processing pipeline should be scalable to handle increasing data size without significant degradation in performance. This could involve implementing parallel processing techniques or leveraging cloud-based infrastructure to ensure that processing large datasets remains feasible even as the dataset scales.

7 Design

7.1 Technical Environment

The technical environment for the project ”Analyzing Social Media Posts for Mental Health Disorder Detection” comprises a combination of hardware, software, and tools that enable smooth data analysis, machine learning model training, and deployment. Below is a detailed overview of the minimum hardware configuration, software tools, and package details necessary to carry out this project effectively.

Minimum Hardware Configuration
Given the nature of the project, which involves processing textual data and training machine learning models, the hardware requirements are modest but significant enough to ensure optimal performance. The minimum configuration needed is:

- **Processor** : Intel Core i5 (or equivalent) with a base clock speed of at least 2.5 GHz. A multi-core processor is preferred as it helps in parallel processing, which is essential during model training and data preprocessing steps.
- **RAM** : 8 GB of RAM is recommended to handle the operations of data loading, cleaning, and transformation. Large datasets, like those used in this project, may require more

memory to prevent memory overflow errors and reduce delays during processing. For larger datasets, 16 GB of RAM would be ideal.

- **Storage** : At least 256 GB of SSD storage is recommended. Faster storage access significantly impacts loading time for datasets and dependencies. SSD is preferred over traditional HDD because of its faster read/write speeds, which benefit large datasets like the Reddit-based social media posts used in this project.

- **Graphics Processing Unit (GPU)** : For basic machine learning tasks like Logistic Regression or SVM, a dedicated GPU is not necessary. However, if deep learning models or more complex neural networks were introduced later, a GPU like NVIDIA GTX 1060 with 4 GB VRAM or higher would be advantageous.

- **Operating System** : Windows 10 (64-bit) or higher, macOS 10.13 (High Sierra) or higher, or any stable Linux distribution (e.g., Ubuntu 18.04 or higher). The operating system should support all necessary machine learning libraries and be compatible with the tools required for the project.

Software Tools and Packages

For the software stack, the project leverages a set of well-established tools, platforms, and programming libraries to ensure smooth execution from data preprocessing to model deployment:

- **Python** : The primary programming language used for data processing, model training, and evaluation. Python is chosen due to its rich ecosystem of libraries and frameworks tailored for machine learning and data science.

- **Google Colab** : A cloud-based platform used for writing, executing, and sharing Python code. Google Colab provides free access to GPU and TPU resources, which is beneficial for intensive model training tasks. It also offers seamless integration with libraries like TensorFlow and PyTorch.

- **Jupyter Notebooks** : An alternative environment for running Python code, offering an interactive interface to write and execute code in blocks. It allows for easy visualization of results and is highly suitable for collaborative projects.

Python Packages and Libraries

Package	Purpose
praw	Access Reddit posts for data collection.
pandas	Data manipulation and analysis.
textblob	Text processing and sentiment analysis.
time	Managing execution time.
re	Regular expressions for text cleaning.
TfidfVectorizer	Convert text to TF-IDF features.
stopwords	Remove stopwords from text.
tokenize	Tokenize text data.
nltk	Natural Language Toolkit for text processing.
CountVectorizer	Convert text to Bag-of-Words features.
split	Split the dataset into training and testing sets.
LogisticRegression	Build and train Logistic Regression models.
sklearn.metrics.accuracy_score	Calculate accuracy of the model.
report	Generate classification performance report.
RandomizedSearchCV	Hyperparameter tuning for models.
seaborn	Data visualization and plotting.
matplotlib.pyplot	Create and customize plots.
matrix	Generate confusion matrix.
roc_curve, auc, roc_auc_score	Evaluate model's ROC and AUC scores.
KNeighborsClassifier	Build k-Nearest Neighbors (k-NN) models.
svm.SVC	Build Support Vector Machines (SVM) models.
naive_bayes.MultinomialNB	Build Naive Bayes models.
scipy.stats.uniform	Statistical operations for tuning models.
RandomForestClassifier	Build Random Forest models.
sklearn.preprocessing.label_binarize	Convert labels for multi-class ROC analysis.

7.2 Hierarchy of Modules

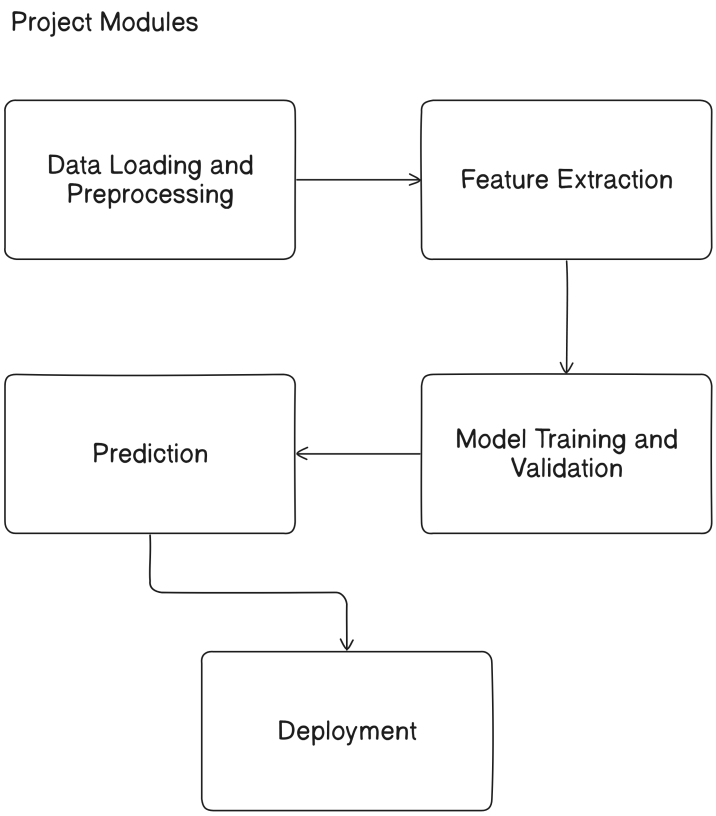


Figure 5: Project Modules

In this project, the system is organized into several key modules to efficiently classify mental health issues based on text input. The Data Loading and Preprocessing Module is responsible for loading the CSV data from preprocessed_mental_health_text.csv and performing essential text cleaning operations, such as tokenization and stop-word removal, to prepare the data for analysis. Following this, the Feature Extraction Module employs techniques like Bag of Words or TF-IDF to convert the cleaned text into numerical features suitable for classification. The Model Training and Validation Module takes these features, splitting the dataset into training and testing sets, and implements multiple machine learning models, including Logistic Regression, k-Nearest Neighbors (k-NN), Support Vector Machines (SVM), Naive Bayes, and Random Forest. Each model is evaluated using relevant metrics to assess accuracy and performance. The Prediction Module is designed to accept new input text and utilize the trained models to predict mental health issues, ensuring that various perspectives from the different algorithms are considered. Finally, the Deployment Module offers a free solution for serving the models, potentially utilizing platforms like Google Colab or Hugging Face, allowing for real-time predictions in practical applications.

7.3 Detailed Design

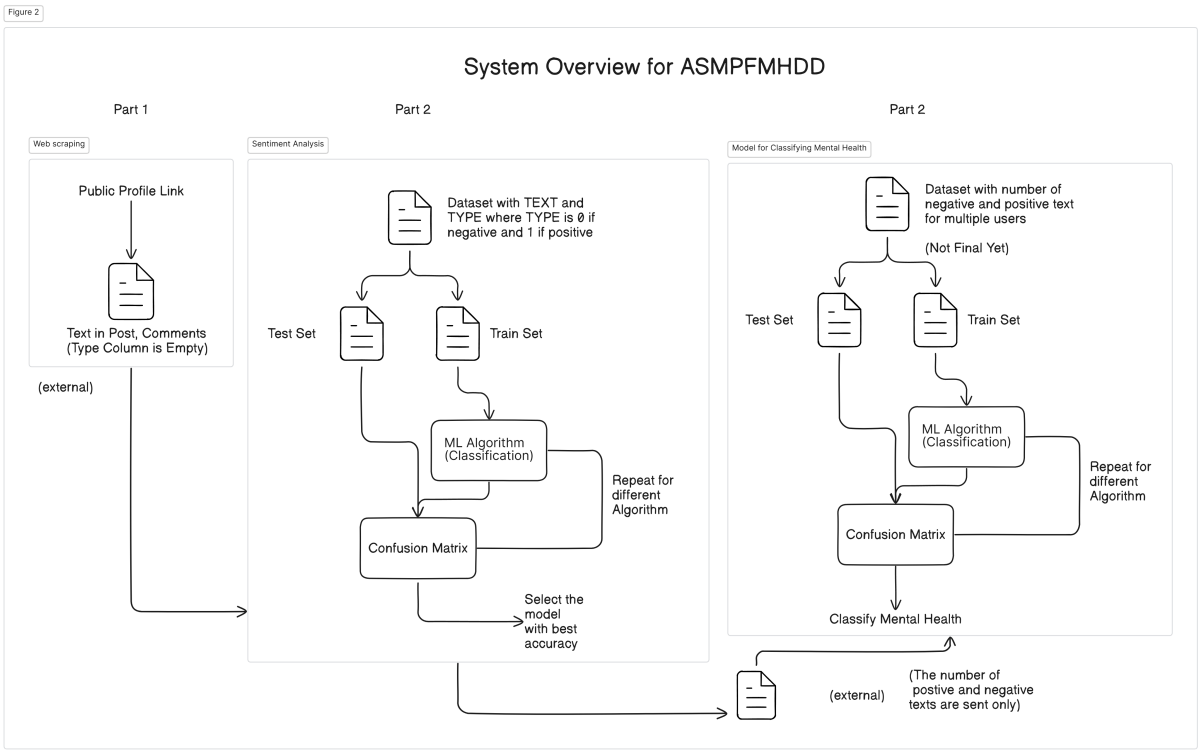


Figure 6: System Overview

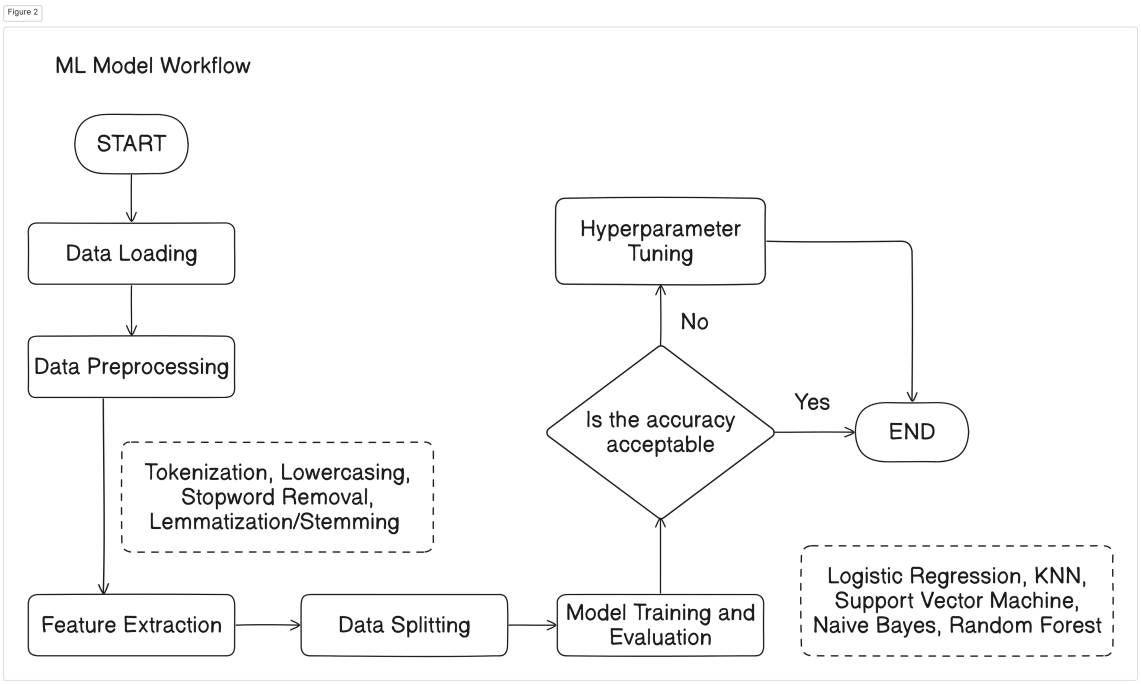


Figure 7: Model Workflow

7.3.1 Data Loading and Preprocessing

The Data Loading and Preprocessing Module is the foundation of the system, responsible for ingesting and preparing the text data for analysis. This module begins by loading the dataset from the preprocessed_mental_health_text.csv file, which contains various mental health-related text entries. Once the data is loaded, a series of preprocessing steps are conducted to ensure the text is clean and ready for feature extraction. This includes tokenization, where the text is split into individual words or tokens, and lowercasing to maintain uniformity across the dataset. Additionally, stop-word removal is performed to eliminate common words that do not contribute to the meaning, such as ”and,” ”the,” and ”is.” Finally, lemmatization or stemming is applied to reduce words to their base or root forms. These preprocessing techniques are crucial as they help improve the quality of the input data, ultimately leading to better model performance.

7.3.2 Feature Extraction

In the Feature Extraction Module, the preprocessed text data is transformed into a numerical format that machine learning algorithms can process. This module allows for the selection between two primary feature extraction methods: Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF). The Bag of Words model creates a representation of the text based on the frequency of words, disregarding the order in which they appear, which simplifies the input for classification algorithms. Alternatively, the TF-IDF approach evaluates the importance of words in the dataset by considering their frequency in individual documents relative to their overall occurrence across all documents. This helps in highlighting the most informative words. By converting text into numerical features, this module prepares the data for the subsequent training and validation stages, ensuring that the classification models can effectively interpret the input.

7.3.3 Model Training and Validation

The Model Training and Validation Module is critical to developing a robust classification system. In this module, the dataset is split into training and testing sets to evaluate the performance of the models accurately. Various classification algorithms are employed, including Logistic Regression, k-Nearest Neighbors (k-NN), Support Vector Machines (SVM), Naive Bayes, and Random Forest. Each model is trained on the training set, which involves adjusting the model parameters based on the input features and their corresponding labels. Following training, the models undergo validation to assess their performance using various metrics such as accuracy, precision, recall, and F1-score. A decision point is included to determine if the achieved accuracy meets the project requirements. If the accuracy is deemed acceptable, the model proceeds to the deployment stage.

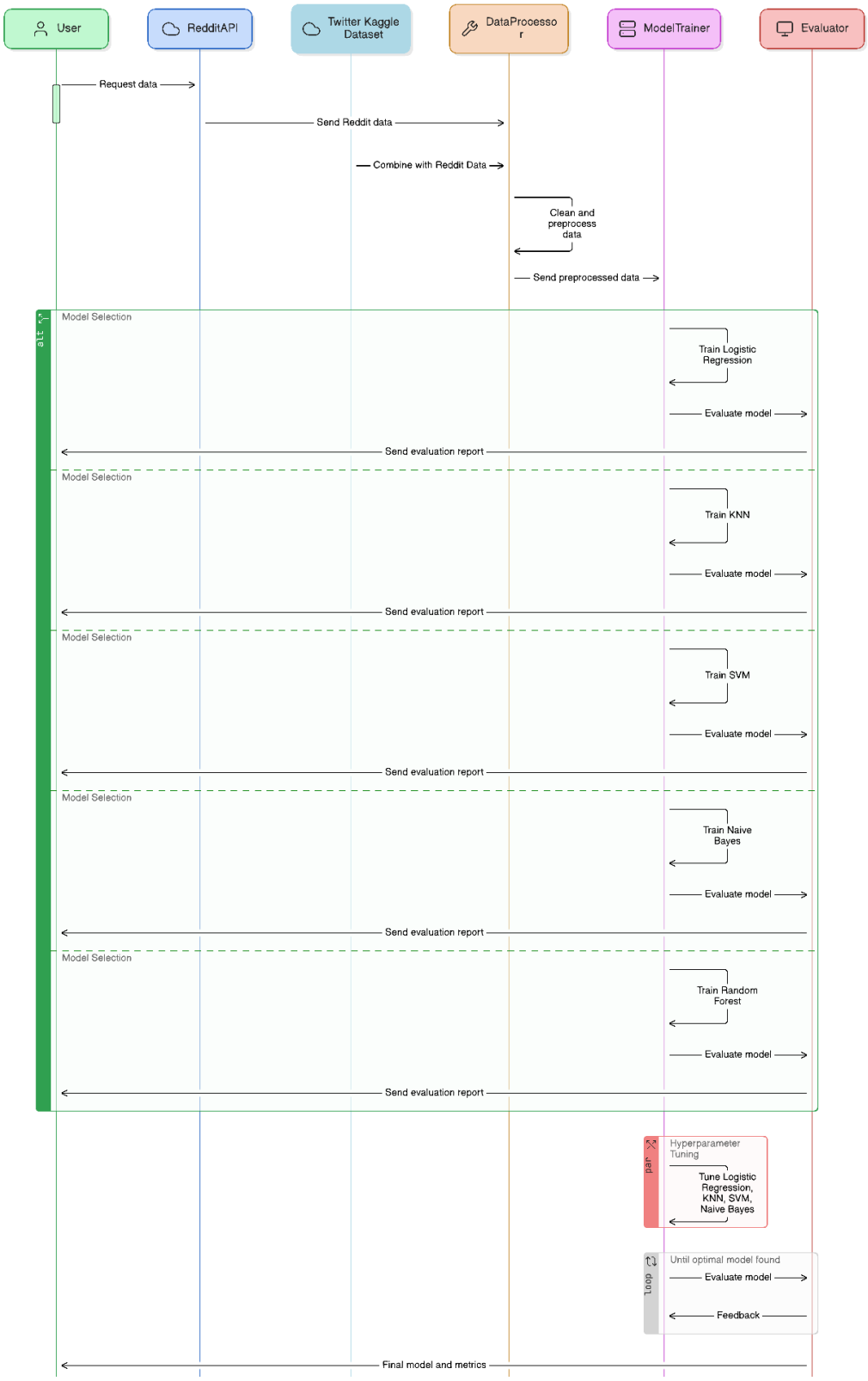


Figure 8: Project Sequence

7.3.4 Prediction

The Prediction Module is designed to provide real-time classification of new input text related to mental health issues. Upon receiving user input, this module initiates a preprocessing workflow that mirrors the steps applied during the training phase, including tokenization, lowercasing, stop-word removal, and lemmatization or stemming. Once the input text is preprocessed, it is fed into the trained classification models to generate predictions. Each model may provide a classification result, allowing for a comprehensive analysis of the input. This module not only delivers the predicted mental health issue but also ensures that users receive an informative output that reflects the confidence level of each prediction, enabling them to understand the model's reasoning. The seamless integration of this module into the overall system enhances the user experience by providing instant and relevant feedback.

7.3.5 Deployment

The Deployment Module focuses on making the trained models accessible for real-time predictions. Once the models have been validated and selected based on their performance, this module prepares them for deployment on suitable platforms, such as Google Colab or Hugging Face. This involves packaging the models and creating a user interface where users can input text and receive predictions. The deployment process also includes considerations for scaling, ensuring that the system can handle multiple requests simultaneously while maintaining responsiveness. By providing a free and efficient deployment solution, this module enables users to access the mental health classification service easily. The deployment of the models ensures that the insights generated from the analysis can be utilized effectively in real-world applications, contributing to better mental health awareness and support.

8 Implementation

8.1 Features From RM

For the initial prototype development, a subset of the requirements from the Requirement Matrix (RM) was carefully selected to focus on implementing core functionalities and demonstrating proof of concept. The selection was based on a combination of high-priority functional requirements that form the backbone of the system, ensuring that critical features are built and validated before expanding the scope.

The requirements chosen for the prototype primarily involve data preprocessing, model training, and evaluation processes. These requirements were selected because they are fundamental to the project's success, ensuring that the data pipeline and model implementation

work seamlessly together. This subset of features lays the groundwork for later integration with additional components and more complex functionality.

The filtered part of the RM focuses on the following high-priority requirements: data cleaning and feature extraction, training of machine learning models, and evaluation of model performance using standard metrics. These requirements were identified as crucial because they directly impact the system's ability to handle data, learn patterns, and provide meaningful outputs. Without successfully implementing these core features, the overall effectiveness of the solution would be significantly reduced.

Furthermore, these selected features align with the project's goals and provide a clear pathway for incremental development. By narrowing down the requirements to these foundational aspects, the development team can ensure that the prototype is not only functional but also extensible, providing a robust framework for future enhancements.

	A	B	C	D
1	**Requirement ID**	**Requirement Description**	**Priority**	**Category**
2	FR-001	Collect and preprocess social media data from Kaggle and Reddit.	High	Functional
3	FR-002	Implement data cleaning and feature extraction for NLP.	High	Functional
4	FR-003	Train machine learning and deep learning models (k-NN, SVM) for sentiment analysis.	High	Functional
5	FR-004	Evaluate models using performance metrics (accuracy, recall, F1).	High	Functional

Figure 9: Features from Requirement Matrix

8.2 Steps of Compilation, Execution and Setup

1. **Setup the Development Environment**

   - Ensure Python 3.x is installed on your system.
   - Install a virtual environment package if not already available:

     ```
     pip install virtualenv
     ```

   - Create a virtual environment for the project:

     ```
     python -m venv project_env
     ```

- Activate the virtual environment:
  - For Windows:

```
.\project_env\Scripts\activate
```

  - For Linux/Mac:

```
source project_env/bin/activate
```

2. **Install Required Libraries**

- Install the necessary Python libraries using the following command:

```
pip install pandas numpy scikit-learn matplotlib
```

- If your project has additional dependencies, include them as well:

```
pip install tensorflow keras seaborn
```

3. **Set Up the Project Directory**

- Organize your project directory as follows:

```
project_folder/
 data/
    sample_data.csv
 notebooks/
    project_prototype.ipynb
 src/
    main.py
 requirements.txt
```

- Place your input data files in the `data/` folder.
- Store Jupyter notebooks in the `notebooks/` folder.
- Add the main code files in the `src/` folder.

4. **Configure the Jupyter Notebook Environment**

- Navigate to the `notebooks/` folder using the command line:

```
cd notebooks
```

- Launch Jupyter Notebook:

```
jupyter notebook
```

- Open the `project_prototype.ipynb` file from the Jupyter interface and execute each cell sequentially to run the prototype.

5. **Compilation and Execution of the Main Code (If Using Scripts)**

   - If you are running the project using a script (e.g., `main.py`), navigate to the `src/` folder:

```
cd ../src
```

   - Run the main script:

```
python main.py
```

   - Ensure that the paths to the data files and other dependencies are correctly set in your code.

6. **Setup Test Data for Evaluation**

   - Place the test dataset in the `data/` folder, ensuring it follows the format specified in the documentation.
   - If using the command line, you can test with different data files by updating the file path in the code or using command-line arguments.

7. **View and Analyze the Output**

   - For Jupyter Notebooks, observe the outputs directly in the notebook cells.
   - If using scripts, the results (e.g., model performance, accuracy metrics, or visualizations) will be printed to the console or saved as files in the `output/` folder, depending on the implementation.

8. **Document and Verify the Execution**

   - Document any issues or errors encountered during the compilation and execution.
   - Perform a verification of results against the expected outputs to ensure correctness.

8.3 Code Details and Output

8.3.1 Data Collection: Scraping Reddit Posts

The code snippet below demonstrates how Reddit posts related to mental health issues are collected and preprocessed for further analysis. This process involves using the ‘praw‘ library to access the Reddit API, collecting posts from specified subreddits, extracting text data, calculating sentiment scores using ‘TextBlob‘, and saving the processed data into a CSV file.

- **Library Installation and Importation:**

```
!pip install praw
import praw
import pandas as pd
from textblob import TextBlob
import time
```

The code starts by installing and importing necessary libraries: `praw` (Python Reddit API Wrapper) for accessing Reddit data, `pandas` for handling tabular data, `TextBlob` for sentiment analysis, and `time` for managing pauses during the scraping process to avoid rate limits.

- **Reddit API Authentication:**

```
reddit = praw.Reddit(client_id=<Reddit Client ID>,
                     client_secret=<Reddit Client Secret>,
                     user_agent='Mental Health')
```

Here, the Reddit API credentials (‘client_id‘, ‘client_secret‘, and ‘user_agent‘) are specified to create an authorized ‘praw.Reddit‘ object, which will be used to interact with Reddit.

- **Defining Subreddits to Scrape:**

```
subreddits = ['normal','depression','anxiety','bipolar','ptsd']
```

The subreddits related to mental health are stored in a list named ‘subreddits‘. These communities are targeted for data collection.

- **Setting Up Post Collection:**

```
post_types = ['hot', 'new', 'top']
posts_per_type = 1500
```

Three categories of posts are selected: ‘hot‘, ‘new‘, and ‘top‘. The number of posts to be collected from each category is set to 1500.

- **Iterating through Subreddits and Post Types:**

```
for subreddit in subreddits:
    for post_type in post_types:
        if post_type == 'hot':
            subreddit_posts = reddit.subreddit(subreddit).
            hot(limit=posts_per_type)
        elif post_type == 'new':
            subreddit_posts = reddit.subreddit(subreddit).
            new(limit=posts_per_type)
        elif post_type == 'top':
            subreddit_posts = reddit.subreddit(subreddit).
            top(limit=posts_per_type)
```

Nested loops are used to iterate through each subreddit and post type, creating a collection of posts from each combination.

- **Collecting and Labeling Data:**

```
for post in subreddit_posts:
    post_content = post.title + " " + post.selftext
    sentiment = TextBlob(post_content).sentiment.polarity
```

Each post’s title and content are concatenated into a single string. Sentiment analysis is then performed using ‘TextBlob‘ to generate a polarity score, which ranges from -1 (negative) to 1 (positive).

- **Assigning Sentiment Labels:**

```
if sentiment > 0:
    sentiment_label = 1.0
elif sentiment < 0:
    sentiment_label = -1.0
else:
    sentiment_label = 0.0
```

Based on the polarity score, a label is assigned: 1.0 for positive, -1.0 for negative, and 0.0 for neutral sentiment.

- **Storing Data with Labels:**

```
issue = subreddit
data.append([post_content, sentiment_label, issue])
```

Each post is associated with its respective subreddit label (mental health issue type) and stored in the ‘data‘ list.

- **Handling Rate Limits:**

```
time.sleep(2)
```

A two-second delay is introduced after processing each type of post to avoid triggering Reddit’s rate limits.

- **Converting Data to DataFrame and Saving to CSV:**

```
df = pd.DataFrame(data, columns=['text', 'sentiment',
mental_health_issue'])
df.to_csv('mental_health.csv', index=False)
```

Finally, the collected data is converted into a ‘pandas‘ DataFrame with columns for text, sentiment, and mental health issue, and saved to a CSV file named ‘mental_health.csv‘.

8.3.2 Generating the Final Dataset

The following code snippet combines two separate datasets—‘mental_health.csv‘ and ‘twitter_mh.csv‘—into a single CSV file named ‘mental_health_text.csv‘. This step is crucial for consolidating different sources of mental health-related data into a unified dataset for further analysis.

```
import pandas as pd
```

The `pandas` library is imported as `pd`. It is a powerful library for data manipulation and analysis, allowing easy handling of structured data formats such as CSV files.

```
# Load the two CSV files
csv1 = pd.read_csv('mental_health.csv')
csv2 = pd.read_csv('twitter_mh.csv')
```

Here, the two CSV files, `mental_health.csv` and `twitter_mh.csv`, are read into separate `pandas` DataFrames: `csv1` and `csv2`. This operation loads each file into memory for further manipulation.

```
# Combine the two dataframes vertically
combined_csv = pd.concat([csv1, csv2], ignore_index=True)
```

The `pd.concat()` function is used to concatenate the two DataFrames vertically. By setting `ignore_index=True`, the function reindexes the rows in the resulting DataFrame, ensuring a continuous index across both files. This approach is used to merge datasets containing similar structures (i.e., same columns).

```
# Save the result to a new CSV file
combined_csv.to_csv('mental_health_text.csv', index=False)
```

The combined DataFrame, stored in `combined_csv`, is saved as a new CSV file named `mental_health_text.csv`. The parameter `index=False` ensures that the index column is not included in the output file, resulting in a cleaner dataset.

```
print("CSV files combined successfully!")
```

The final line prints a success message: `"CSV files combined successfully!"`, indicating that the operation completed without errors and the files were merged as intended.

8.3.3 Text Preprocessing and Feature Extraction

The following code snippet demonstrates the process of text preprocessing and feature extraction using the Term Frequency-Inverse Document Frequency (TF-IDF) method. This is an essential step in preparing textual data for machine learning models, particularly in natural language processing tasks.

```
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
```

The necessary libraries are imported: - `pandas` is imported as `pd` for data manipulation. - `re` is imported for regular expression operations, useful for text cleaning. - `TfidfVectorizer` from `sklearn` is imported for feature extraction. - `nltk.corpus.stopwords` and - `nltk.tokenize.word_tokenize` are imported from the Natural Language Toolkit (NLTK) for text processing. - The `nltk` library is imported to access various functionalities.

```
# Download stopwords (if you haven't already)
nltk.download('stopwords')
nltk.download('punkt')
```

NLTK's stopwords and punkt tokenizer resources are downloaded if not previously available. Stopwords are common words (e.g., "and", "the") that are usually removed during text preprocessing, while the punkt tokenizer is necessary for breaking text into words.

```
# Load the dataset
df = pd.read_csv('mental_health_text.csv')
```

The dataset, `mental_health_text.csv`, is loaded into a DataFrame named `df` for further processing.

```
# 1. Handling Missing Values
# Remove rows with missing text
df.dropna(subset=['text'], inplace=True)
```

The first preprocessing step handles missing values by removing any rows that contain null values in the `text` column. The `dropna()` function is called with `subset` set to `text`, and `inplace=True` ensures that changes are made directly to the original DataFrame.

```
# 2. Removing duplicates (if any)
df.drop_duplicates(subset=['text'], inplace=True)
```

Next, any duplicate rows based on the `text` column are removed using the `drop_duplicates()` method. This step ensures that each entry in the dataset is unique.

```
# 3. Text Preprocessing
# Define a function to clean the text
def clean_text(text):
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove mentions (@username)
    text = re.sub(r'@\w+', '', text)
    # Remove special characters, numbers, and punctuations
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Convert text to lowercase
    text = text.lower()
    # Tokenize the text
    tokens = word_tokenize(text)
    # Remove stopwords
    tokens = [word for word in tokens if word not in
    stopwords.words('english')]
    # Join the tokens back into a single string
    clean_text = ' '.join(tokens)
    return clean_text
```

A function named `clean_text` is defined to preprocess the text data. The following operations are performed within the function: - URLs are removed using a regular expression. - Mentions (usernames starting with @) are stripped out. - Special characters, numbers, and punctuation are eliminated to retain only alphabetical characters and spaces. - The text is converted to lowercase to ensure uniformity. - The `word_tokenize` function is applied to split the cleaned text into individual words (tokens). - Stopwords are removed from the token list. - Finally, the tokens are rejoined into a single string and returned.

```
# Apply the cleaning function to the 'text' column
df['cleaned_text'] = df['text'].apply(clean_text)
```

The `clean_text` function is applied to the `text` column of the DataFrame, and the cleaned text is stored in a new column named `cleaned_text`. The `apply()` method applies the function to each row in the specified column.

```
# 4. Feature Extraction using TF-IDF Vectorization
# Initialize the TF-IDF vectorizer
vectorizer = TfidfVectorizer(max_features=5000)
# You can adjust the max_features
```

The TF-IDF vectorizer is initialized with a maximum feature limit of 5000. This setting ensures that only the top 5000 most important words are considered, which helps in managing the dimensionality of the dataset.

```
# Fit and transform the cleaned text data
X = vectorizer.fit_transform(df['cleaned_text'])
```

The `fit_transform()` method is called on the cleaned text data, transforming the text into a matrix of TF-IDF features. The result is stored in the variable `X`.

```
# Convert the result to a DataFrame for easier understanding (optional)
X_df = pd.DataFrame(X.toarray(), columns=vectorizer.
get_feature_names_out())
```

The resulting TF-IDF matrix `X` is converted into a DataFrame, `X_df`, with columns named according to the feature names extracted by the vectorizer. This step enhances the readability and usability of the data.

```
# You now have a cleaned and vectorized dataset.
print(X_df.head())
```

The first few rows of the cleaned and vectorized dataset are printed to the console, providing a quick overview of the transformed data.

```
# Save the preprocessed dataset (optional)
df.to_csv('preprocessed_mental_health_text.csv', index=False)
```

Lastly, the preprocessed DataFrame, which now includes the cleaned text, is saved to a new CSV file named `preprocessed_mental_health_text.csv`. The parameter `index=False` ensures that the index column is not included in the output file.

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
    aa  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  abandon  abandoned  abandonment  \
0  0.0                                      0.0      0.0        0.0          0.0
1  0.0                                      0.0      0.0        0.0          0.0
2  0.0                                      0.0      0.0        0.0          0.0
3  0.0                                      0.0      0.0        0.0          0.0
4  0.0                                      0.0      0.0        0.0          0.0

   abdominal  abilify  ability  able  abnormal  ...  youth  youtube  youve  \
0        0.0      0.0      0.0   0.0   0.22298  ...    0.0      0.0    0.0
1        0.0      0.0      0.0   0.0   0.00000  ...    0.0      0.0    0.0
2        0.0      0.0      0.0   0.0   0.00000  ...    0.0      0.0    0.0
3        0.0      0.0      0.0   0.0   0.00000  ...    0.0      0.0    0.0
4        0.0      0.0      0.0   0.0   0.00000  ...    0.0      0.0    0.0

    yr  yrs  zero  zoloft  zombie  zone  zoning
0  0.0  0.0   0.0     0.0     0.0   0.0     0.0
1  0.0  0.0   0.0     0.0     0.0   0.0     0.0
2  0.0  0.0   0.0     0.0     0.0   0.0     0.0
3  0.0  0.0   0.0     0.0     0.0   0.0     0.0
4  0.0  0.0   0.0     0.0     0.0   0.0     0.0

[5 rows x 5000 columns]
```

Figure 10: Data Preprocessing

8.3.4 Implementation of Bag Of Words

The following code snippet implements the Bag of Words (BoW) model for the preprocessed dataset related to mental health. This process converts text data into a numerical format that can be used for machine learning tasks.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
# Load the preprocessed dataset
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
# Check if 'cleaned_text' column exists
if 'cleaned_text' not in dataset.columns:
    raise ValueError("The dataset must have a 'cleaned_text' column.
    Ensure text preprocessing has been done.")
# Remove rows with missing values in 'cleaned_text' column
dataset.dropna(subset=['cleaned_text'], inplace=True)
# Initialize the CountVectorizer
vectorizer = CountVectorizer()
# Fit and transform the cleaned text data
X = vectorizer.fit_transform(dataset['cleaned_text'])
```

```
# Convert the result to a DataFrame for better visualization (optional)
X_df = pd.DataFrame(X.toarray(), columns=vectorizer.
get_feature_names_out())
# Print the shape of the resulting matrix
print(f'Shape of Bag of Words matrix: {X_df.shape}')
# Print the first few rows of the Bag of Words DataFrame (optional)
print(X_df.head())
```

The code begins by importing the necessary libraries: `pandas` for data manipulation and `CountVectorizer` from `sklearn` for creating the Bag of Words model. It then loads a preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv`. A check is performed to ensure that the dataset contains a column labeled `cleaned_text`. If this column is absent, a `ValueError` is raised, prompting the user to ensure text preprocessing is completed. Following this, any rows with missing values in the `cleaned_text` column are removed to maintain data integrity. Next, an instance of `CountVectorizer` is initialized, which will convert the cleaned text into a matrix of token counts. The `fit_transform()` method is called on the cleaned text data, generating a sparse matrix X that represents the Bag of Words model. This matrix is then converted into a DataFrame `X_df` for easier visualization, with columns corresponding to the unique words identified in the text. Finally, the shape of the resulting Bag of Words matrix is printed to the console, along with the first few rows of the DataFrame to provide a preview of the transformed data.

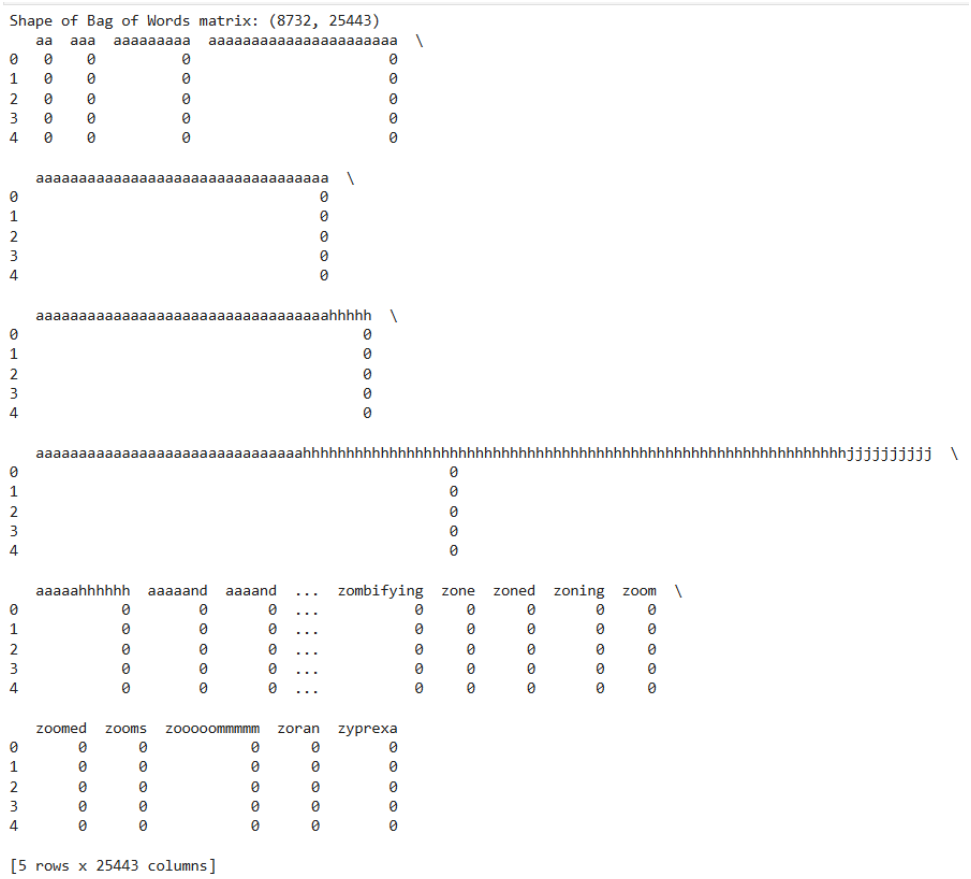
8.3.5 Splitting Preprocessed Dataset

The following code snippet demonstrates how to split the preprocessed dataset into training and test sets, which is a crucial step in preparing data for machine learning models.

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the preprocessed dataset
dataset = pd.read_csv('preprocessed_mental_health_text.csv')

# Check if 'cleaned_text' and 'mental_health_issue' columns exist
if 'cleaned_text' not in dataset.columns or 'mental_health_issue'
not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
```



```
Shape of Bag of Words matrix: (8732, 25443)
   aa  aaa  aaaaaaaaa  aaaaaaaaaaaaaaaaaaaa  \
0   0    0          0                     0
1   0    0          0                     0
2   0    0          0                     0
3   0    0          0                     0
4   0    0          0                     0

   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  \
0                                     0
1                                     0
2                                     0
3                                     0
4                                     0

   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaahhhhh  \
0                                          0
1                                          0
2                                          0
3                                          0
4                                          0

   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaahhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhjjjjjjjjjj  \
0                                                                                                                                0
1                                                                                                                                0
2                                                                                                                                0
3                                                                                                                                0
4                                                                                                                                0

   aaaaahhhhhh  aaaaand  aaaand  ...  zombifying  zone  zoned  zoning  zoom  \
0            0       0       0  ...           0     0      0       0     0
1            0       0       0  ...           0     0      0       0     0
2            0       0       0  ...           0     0      0       0     0
3            0       0       0  ...           0     0      0       0     0
4            0       0       0  ...           0     0      0       0     0

   zoomed  zooms  zooooommmmm  zoran  zyprexa
0       0      0            0      0        0
1       0      0            0      0        0
2       0      0            0      0        0
3       0      0            0      0        0
4       0      0            0      0        0

[5 rows x 25443 columns]
```

Figure 11: Implementation of Bag Of Words

```
# Remove rows with missing values in 'cleaned_text' column
dataset.dropna(subset=['cleaned_text'], inplace=True)
#This line has been added

# Initialize the CountVectorizer and fit/transform the cleaned text
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['cleaned_text'])

# Prepare the target variable
y = dataset['mental_health_issue']

# Split the dataset into Training and Test Sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
# Print the shapes of the resulting datasets
print(f'Shape of X_train: {X_train.shape}')
print(f'Shape of X_test: {X_test.shape}')
print(f'Shape of y_train: {y_train.shape}')
print(f'Shape of y_test: {y_test.shape}')
```

The code begins by importing the necessary libraries, specifically `pandas` for data manipulation and `train_test_split` from `sklearn.model_selection` for splitting the dataset. It then loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv`. A validation check is performed to ensure that both the `cleaned_text` and `mental_health_issue` columns exist in the dataset; if not, a `ValueError` is raised to alert the user. Next, rows with missing values in the `cleaned_text` column are removed to maintain data integrity. After this, the `CountVectorizer` is initialized to transform the cleaned text into a numerical format suitable for machine learning. The transformed text data is stored in the variable X, while the target variable representing mental health issues is stored in y. The dataset is then split into training and test sets, with 80% of the data used for training and 20% for testing, controlled by the `random_state` parameter to ensure reproducibility. Finally, the shapes of the resulting training and test datasets are printed to the console, providing insights into the number of samples allocated for training and testing, which is essential for understanding the data distribution.

```
Shape of X_train: (6985, 25443)
Shape of X_test: (1747, 25443)
Shape of y_train: (6985,)
Shape of y_test: (1747,)
```

Figure 12: Splitting Dataset

8.3.6 Logistic Regression

The following code snippet demonstrates the training and evaluation of a Logistic Regression model using a preprocessed dataset related to mental health.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

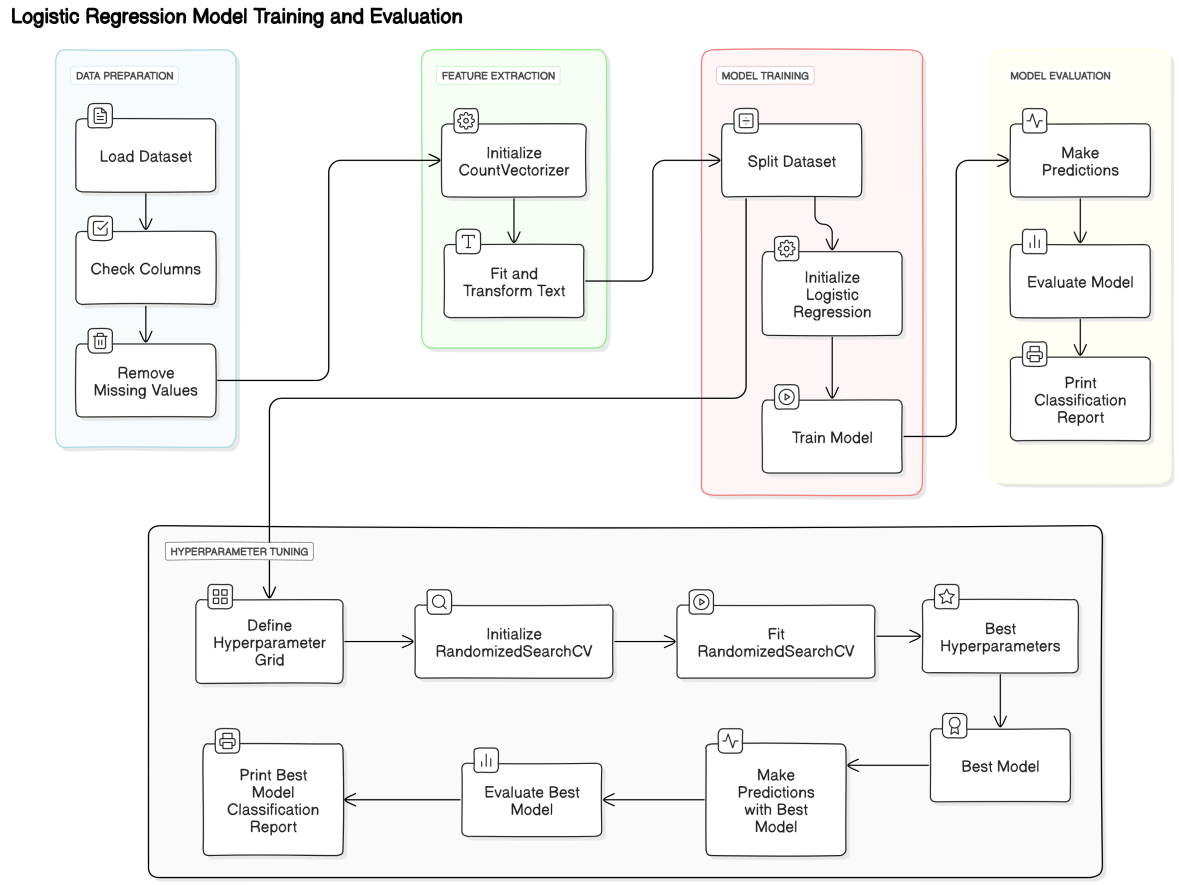


Figure 13: Logistic Regression

```
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
if 'cleaned_text' not in dataset.columns or 'mental_health_issue'
not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
dataset.dropna(subset=['cleaned_text'], inplace=True)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['cleaned_text'])
y = dataset['mental_health_issue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=2000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f'Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n", classification_report(y_test,
y_pred))
```

The code begins by importing necessary libraries, including `pandas` for data manipulation and various functions from `sklearn` for model training and evaluation. It then loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv`. A validation check ensures that the dataset contains the required columns, raising an error if either `cleaned_text` or `mental_health_issue` is missing. Rows with missing values in the `cleaned_text` column are removed to maintain data quality. The `CountVectorizer` is initialized and used to convert the cleaned text data into a matrix of token counts, stored in X. The target variable, representing mental health issues, is assigned to y. The dataset is then split into training and testing sets, with 80% used for training and 20% for testing. A Logistic Regression model is initialized and trained on the training data. Predictions are made on the test set, and the model's accuracy is calculated and printed. Finally, a classification report is generated, providing detailed metrics on the model's performance across different categories.

```
Accuracy: 73.78%
Classification Report:
              precision    recall  f1-score   support

     anxiety       0.78      0.73      0.75       416
     bipolar       0.66      0.84      0.74       412
  depression       0.74      0.72      0.73       443
     neutral       0.08      0.06      0.07        17
      normal       0.78      0.22      0.34        32
        ptsd       0.83      0.74      0.78       427

    accuracy                           0.74      1747
   macro avg       0.64      0.55      0.57      1747
weighted avg       0.75      0.74      0.74      1747
```

Figure 14: Logistic Regression Result

8.3.7 Hyperparameter Tuning on Logistic Regression using Random Search

The following code snippet demonstrates how to perform hyperparameter tuning on a Logistic Regression model using Random Search.

```
import pandas as pd
from sklearn.model_selection import train_test_split,
```

```
RandomizedSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
if 'cleaned_text' not in dataset.columns or 'mental_health_issue'
not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
dataset.dropna(subset=['cleaned_text'], inplace=True)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['cleaned_text'])
y = dataset['mental_health_issue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=200)
param_distributions = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'solver': ['liblinear', 'saga']
}
random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_distributions,
n_iter=10, scoring='accuracy', cv=5, n_jobs=-1, random_state=42)
random_search.fit(X_train, y_train)
print("Best Hyperparameters:", random_search.best_params_)
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

The code begins by importing the necessary libraries, including `pandas` for data manipulation and various functions from `sklearn` for model training and evaluation. It loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv` and checks for the required columns `cleaned_text` and `mental_health_issue`. If these columns are not present, a `ValueError` is raised.

The code then removes any rows with missing values in the `cleaned_text` column. The `CountVectorizer` is initialized to convert the cleaned text into a numerical format, which is stored in X, while the target variable representing mental health issues is stored in y. The dataset is split into training and testing sets with 80% for training and 20% for testing. A Logistic Regression model is created with a maximum of 200 iterations. The hyperparameter grid for tuning is defined, specifying values for the inverse of regularization strength (`C`), types of regularization (`penalty`), and solvers. A `RandomizedSearchCV` object is initialized to perform the random search over the hyperparameter grid, and the model is fitted using the training data. The best hyperparameters are printed, and the best model is used to make predictions on the test set. Finally, the model's accuracy is calculated and displayed, along with a classification report that provides detailed metrics on the model's performance across different categories.

```
Best Hyperparameters: {'solver': 'liblinear', 'penalty': 'l1', 'C': 1}
Accuracy: 75.27%
Classification Report:
              precision    recall  f1-score   support

     anxiety       0.80      0.75      0.77       416
     bipolar       0.66      0.85      0.74       412
  depression       0.75      0.74      0.75       443
     neutral       0.14      0.12      0.13        17
      normal       0.78      0.22      0.34        32
        ptsd       0.86      0.75      0.80       427

    accuracy                           0.75      1747
   macro avg       0.67      0.57      0.59      1747
weighted avg       0.76      0.75      0.75      1747
```

Figure 15: Result of Hyperparameter Tuning on Logistic Regression

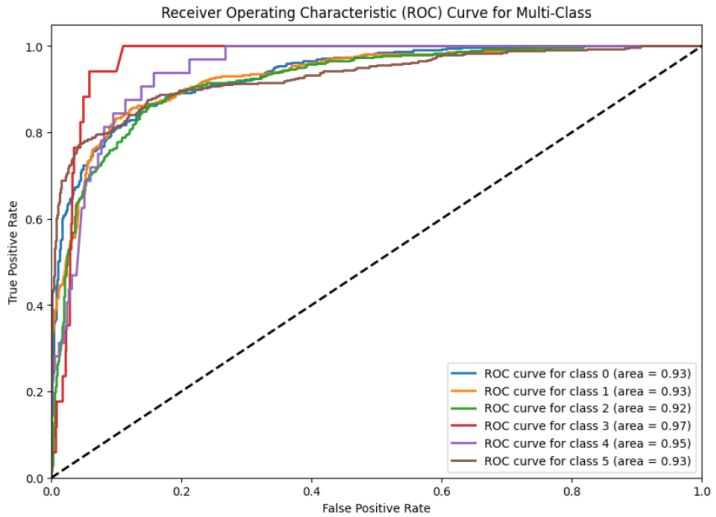


Figure 16: ROC Curve on Logistic Regression

8.3.8 K Nearest Neighbours

The following code snippet demonstrates the implementation of the k-Nearest Neighbors (k-NN) algorithm for classifying mental health issues based on preprocessed text data.

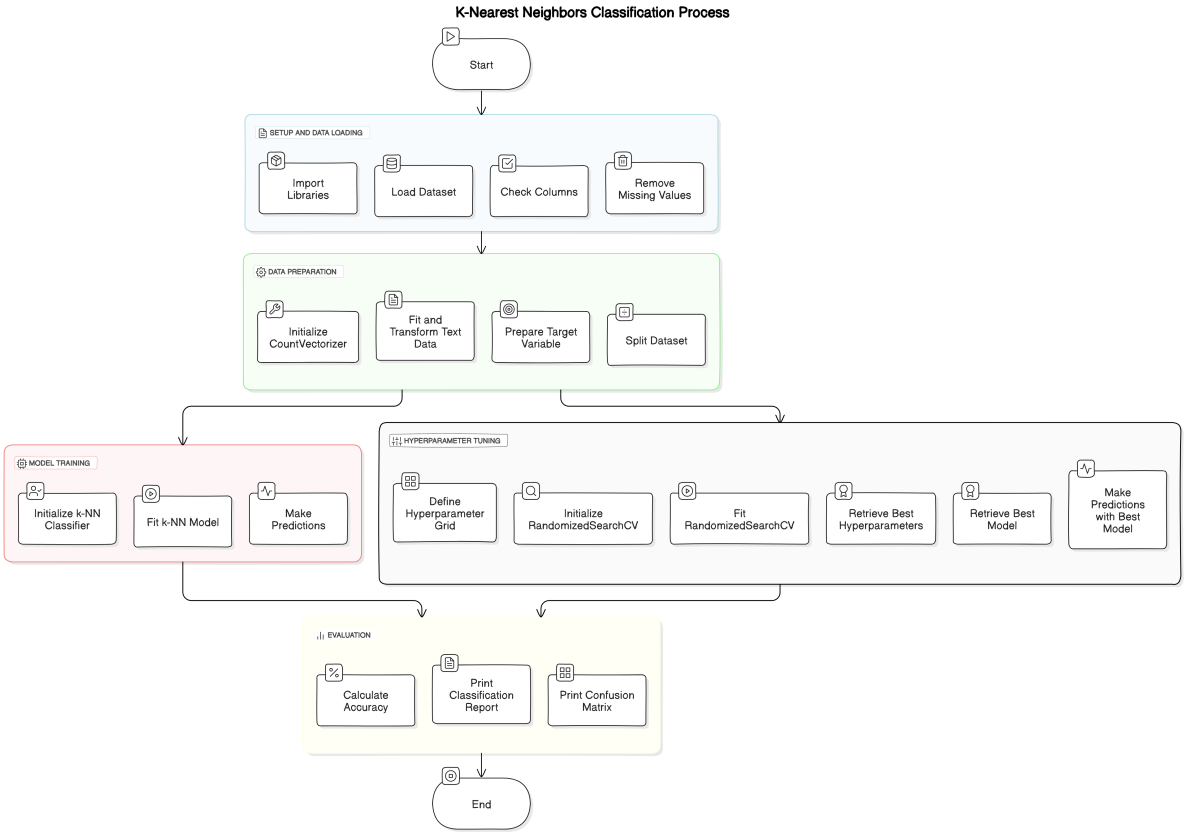


Figure 17: K Nearest Neighbours Workflow

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
if 'cleaned_text' not in dataset.columns or 'mental_health_issue'
not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
dataset.dropna(subset=['cleaned_text'], inplace=True)
vectorizer = CountVectorizer()
```

```
X = vectorizer.fit_transform(dataset['cleaned_text'])
y = dataset['mental_health_issue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n",
classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test,y_pred))
```

The code begins by importing the necessary libraries, including `pandas` for data manipulation, `train_test_split` for splitting the dataset, `CountVectorizer` for transforming text data, and `KNeighborsClassifier` along with various metrics from `sklearn` for model evaluation. It loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv` and checks that both the `cleaned_text` and `mental_health_issue` columns are present; if not, a `ValueError` is raised. Rows with missing values in the `cleaned_text` column are removed to maintain data integrity. The `CountVectorizer` is initialized to convert the cleaned text into a numerical format, stored in X, while the target variable representing mental health issues is assigned to y. The dataset is split into training and testing sets, with 80% used for training and 20% for testing. A k-NN classifier is initialized with 5 neighbors, and the model is fitted on the training data. Predictions are made on the test set, and the model's accuracy is calculated and displayed. Finally, a classification report and confusion matrix are printed, providing detailed metrics on the model's performance and insight into its classification results.

8.3.9 Hyperparameter Tuning on KNN Using Random Search

The following code snippet demonstrates how to perform hyperparameter tuning on a k-Nearest Neighbors (k-NN) classifier using Random Search to optimize its performance.

```
import pandas as pd
from sklearn.model_selection import train_test_split,
RandomizedSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
```

Accuracy: 39.61%
Classification Report:

	precision	recall	f1-score	support
anxiety	0.60	0.36	0.45	416
bipolar	0.31	0.83	0.45	412
depression	0.46	0.35	0.40	443
neutral	0.00	0.00	0.00	17
normal	0.29	0.06	0.10	32
ptsd	0.81	0.11	0.20	427
accuracy			0.40	1747
macro avg	0.41	0.28	0.27	1747
weighted avg	0.54	0.40	0.36	1747

Figure 18: K Nearest Neighbours Result

```
confusion_matrix
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
if 'cleaned_text'
not in dataset.columns or 'mental_health_issue' not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
dataset.dropna(subset=['cleaned_text'], inplace=True)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['cleaned_text'])
y = dataset['mental_health_issue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
knn = KNeighborsClassifier()
param_distributions = {
    'n_neighbors': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
    14, 15, 16, 17, 18, 19, 20],
    'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski'],
    'weights': ['uniform', 'distance']
}
random_search = RandomizedSearchCV(estimator=knn,
param_distributions=param_distributions,
n_iter=200, scoring='accuracy', cv=5, n_jobs=-1, random_state=42)
random_search.fit(X_train, y_train)
print("Best Hyperparameters:", random_search.best_params_)
best_knn = random_search.best_estimator_
y_pred = best_knn.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n",
classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

The code starts by importing the required libraries, including `pandas` for data manipulation, and various functions from `sklearn` for model training, evaluation, and hyperparameter tuning. It loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv` and checks for the presence of the necessary columns, raising a `ValueError` if they are missing. Rows with missing values in the `cleaned_text` column are removed to ensure data quality. The `CountVectorizer` is initialized to convert the cleaned text into a numerical format, stored in X, while the target variable representing mental health issues is assigned to y. The dataset is split into training and testing sets, with 80% used for training and 20% for testing. A k-NN classifier is initialized, and a hyperparameter grid is defined, specifying different values for the number of neighbors, distance metrics, and weighting options for the neighbors. The `RandomizedSearchCV` object is created to perform random search over the hyperparameter grid, and the model is fitted using the training data. After fitting, the best hyperparameters are printed, along with the best k-NN model derived from the search. Predictions are made on the test set, and the accuracy of the model is calculated and displayed. Finally, a classification report and confusion matrix are printed to evaluate the model's performance and provide insights into its classification results.

```
Accuracy: 39.61%
Classification Report:
              precision    recall  f1-score   support

     anxiety       0.60      0.36      0.45       416
     bipolar       0.31      0.83      0.45       412
  depression       0.46      0.35      0.40       443
     neutral       0.00      0.00      0.00        17
      normal       0.29      0.06      0.10        32
        ptsd       0.81      0.11      0.20       427

    accuracy                           0.40      1747
   macro avg       0.41      0.28      0.27      1747
weighted avg       0.54      0.40      0.36      1747
```

Figure 19: Result of Hyperparameter Tuning on KNN

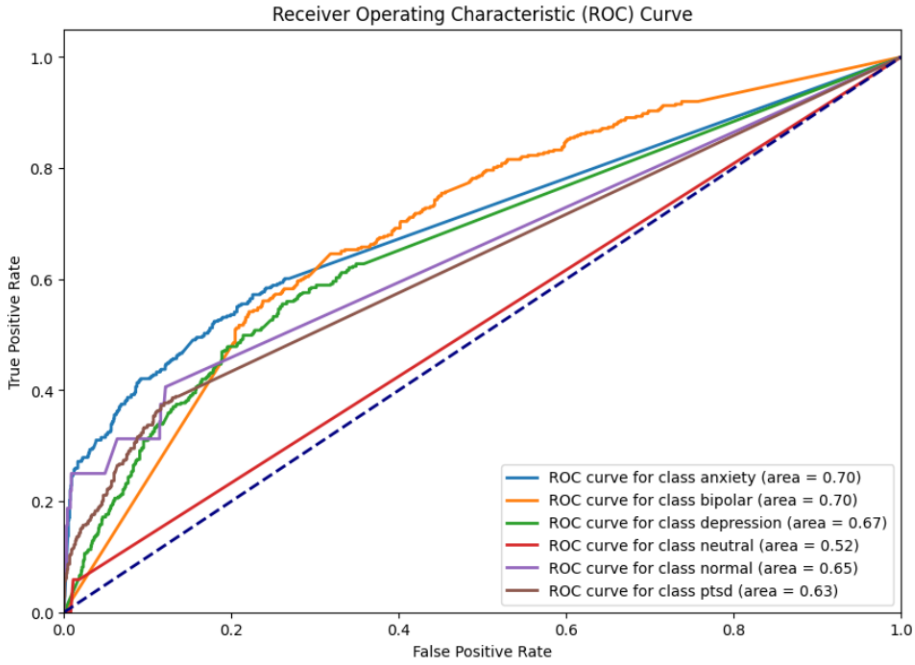


Figure 20: ROC curve on KNN

8.3.10 Support Vector Machine

The following code snippet demonstrates the implementation of a Support Vector Machine (SVM) classifier for classifying mental health issues based on preprocessed text data.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
if 'cleaned_text' not in dataset.columns or 'mental_health_issue'
not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
dataset.dropna(subset=['cleaned_text'], inplace=True)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['cleaned_text'])
y = dataset['mental_health_issue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

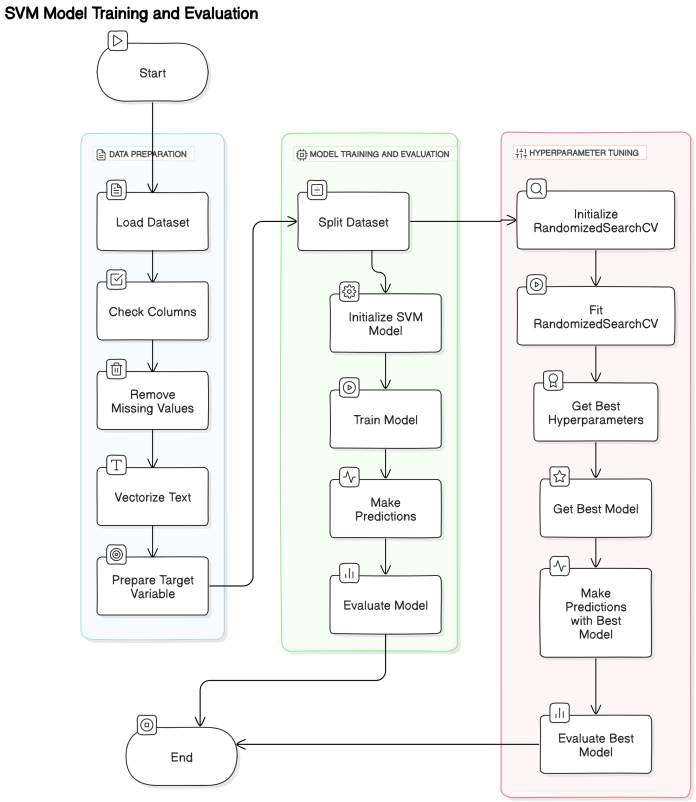


Figure 21: Support Vector Machine Workflow

```
svm_model = SVC(kernel='linear', C=1, random_state=42)
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n",
classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

The code begins by importing the necessary libraries, including `pandas` for data manipulation and various functions from `sklearn` for model training and evaluation. It loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv` and verifies that both the `cleaned_text` and `mental_health_issue` columns exist, raising a `ValueError` if they are missing. Rows with missing values in the `cleaned_text` column are removed to maintain data quality. The `CountVectorizer` is initialized to convert the cleaned text into a numerical format, stored in X, while the target variable representing mental health issues is assigned to y. The

dataset is then split into training and testing sets, with 80% used for training and 20% for testing. An SVM model is initialized with a linear kernel and a regularization parameter C set to 1, which can be adjusted based on the specific needs of the analysis. The model is trained using the training data, and predictions are made on the test set. The accuracy of the model is calculated and displayed, along with a classification report and confusion matrix, which provide insights into the model's performance across different categories and the distribution of classification results.

Accuracy: 70.75%
Classification Report:

	precision	recall	f1-score	support
anxiety	0.70	0.70	0.70	416
bipolar	0.65	0.81	0.72	412
depression	0.72	0.67	0.69	443
neutral	0.19	0.18	0.18	17
normal	0.62	0.25	0.36	32
ptsd	0.81	0.71	0.76	427
accuracy			0.71	1747
macro avg	0.61	0.55	0.57	1747
weighted avg	0.71	0.71	0.71	1747

Figure 22: Support Vector Machine (Linear Kernel) Result

8.3.11 Hyperparameter Tuning on SVM using Random Search

The following code snippet demonstrates how to perform hyperparameter tuning on a Support Vector Machine (SVM) classifier using Random Search to optimize its performance.

```
import pandas as pd
from sklearn.model_selection import train_test_split,
RandomizedSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
if 'cleaned_text' not in dataset.columns or 'mental_health_issue'
not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
dataset.dropna(subset=['cleaned_text'], inplace=True)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['cleaned_text'])
```

```
y = dataset['mental_health_issue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = SVC()
param_distributions = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto', 0.1, 1],
}
random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_distributions,
n_iter=200, scoring='accuracy',
cv=5, n_jobs=-1, random_state=42)
random_search.fit(X_train, y_train)
print("Best Hyperparameters:", random_search.best_params_)
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

The code begins by importing the required libraries, including `pandas` for data manipulation and various functions from `sklearn` for model training and evaluation. It loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv` and checks for the necessary columns, raising a `ValueError` if either the `cleaned_text` or `mental_health_issue` columns are missing. Rows with missing values in the `cleaned_text` column are then removed to ensure data quality. The `CountVectorizer` is initialized to convert the cleaned text into a numerical format, stored in X, while the target variable representing mental health issues is assigned to y. The dataset is split into training and testing sets, with 80% used for training and 20% for testing. An SVM model is initialized, and a hyperparameter grid is defined that specifies different values for the regularization parameter C, various kernel types, and kernel coefficients. The `RandomizedSearchCV` object is created to perform random search over the hyperparameter grid, and the model is fitted using the training data. After fitting, the best hyperparameters are printed along with the best SVM model derived from the search. Predictions are made on the test set, and the accuracy of the model is calculated and displayed, along with a classification report that provides detailed metrics on the model's performance.

```
Best Hyperparameters: {'kernel': 'linear', 'gamma': 'scale', 'C': 0.1}
Accuracy: 72.07%
Classification Report:
              precision    recall  f1-score   support

     anxiety       0.78      0.71      0.74       416
     bipolar       0.61      0.83      0.70       412
  depression       0.75      0.70      0.72       443
     neutral       0.13      0.12      0.12        17
      normal       0.57      0.12      0.21        32
        ptsd       0.84      0.71      0.77       427

    accuracy                           0.72      1747
   macro avg       0.61      0.53      0.54      1747
weighted avg       0.73      0.72      0.72      1747
```

Figure 23: Result of Hyperparameter Tuning on SVM

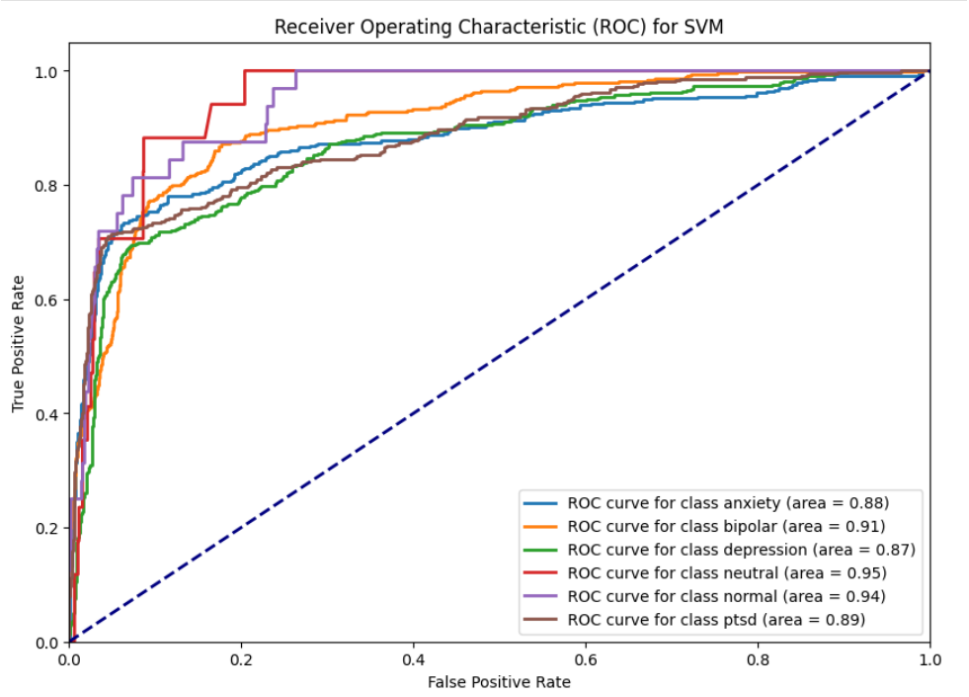


Figure 24: ROC Curve on SVM

8.3.12 Naive Bayes Result

The following code snippet demonstrates the implementation of a Naive Bayes classifier for classifying mental health issues based on preprocessed text data.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
```

Naive Bayes Model Training and Evaluation

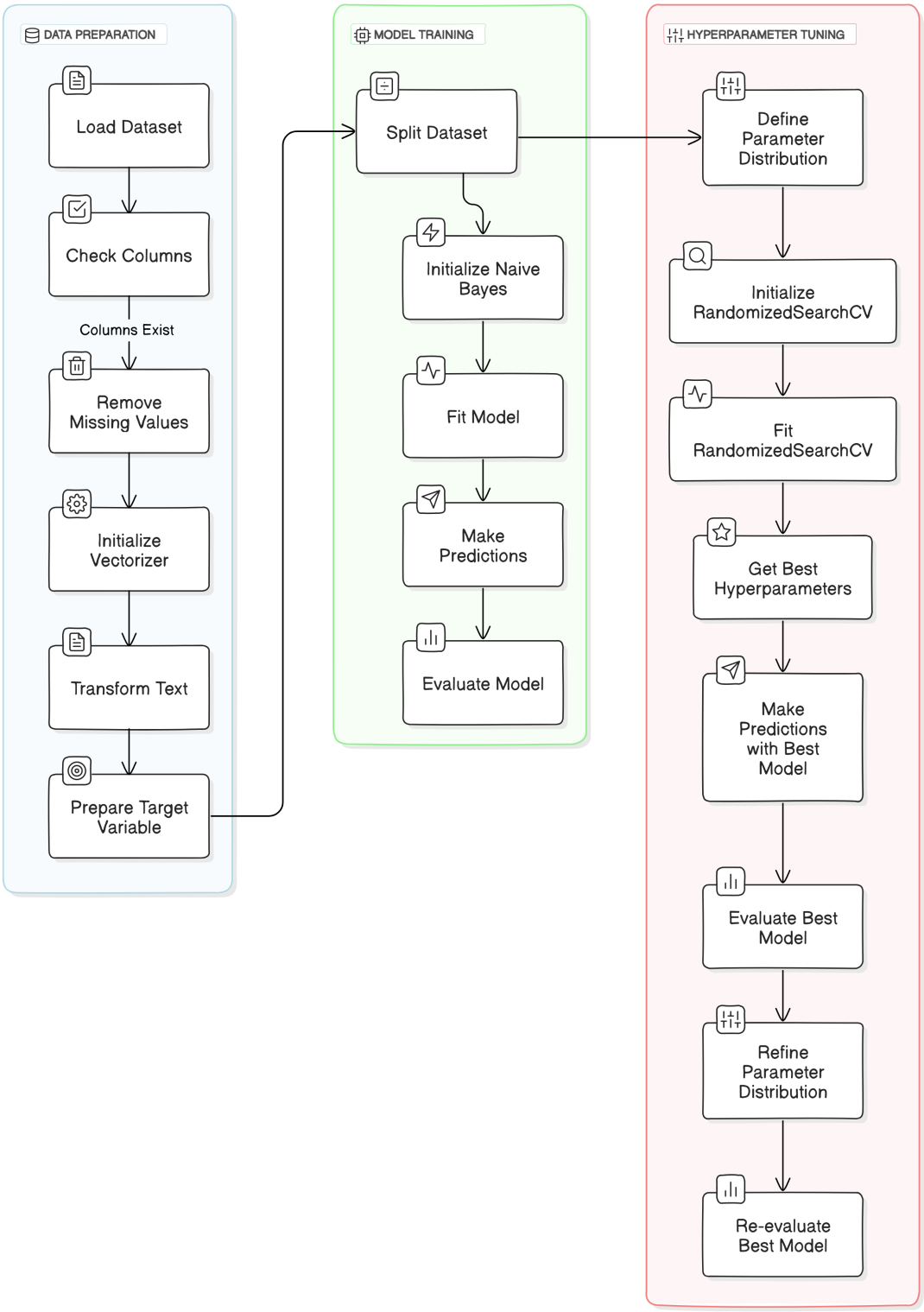


Figure 25: Naive Bayes Workflow

```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
if 'cleaned_text' not in dataset.columns or 'mental_health_issue'
not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
dataset.dropna(subset=['cleaned_text'], inplace=True)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['cleaned_text'])
y = dataset['mental_health_issue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
naive_bayes_model = MultinomialNB()
naive_bayes_model.fit(X_train, y_train)
y_pred = naive_bayes_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n",
classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

The code begins by importing the necessary libraries, including `pandas` for data manipulation and various functions from `sklearn` for model training and evaluation. It loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv` and checks for the required columns, raising a `ValueError` if either the `cleaned_text` or `mental_health_issue` columns are missing. Rows with missing values in the `cleaned_text` column are then removed to ensure data quality. The `CountVectorizer` is initialized to convert the cleaned text into a numerical format, which is stored in X, while the target variable representing mental health issues is assigned to y. The dataset is split into training and testing sets, with 80% used for training and 20% for testing. A Naive Bayes classifier, specifically `MultinomialNB`, is initialized and fitted using the training data. Predictions are made on the test set, and the model's accuracy is calculated and displayed. Finally, a classification report and confusion matrix are printed, providing detailed metrics on the model's performance across different categories and insights into its classification results.

Accuracy: 68.92%
Classification Report:

	precision	recall	f1-score	support
anxiety	0.69	0.72	0.70	416
bipolar	0.79	0.56	0.65	412
depression	0.65	0.82	0.72	443
neutral	0.17	0.29	0.21	17
normal	0.14	0.03	0.05	32
ptsd	0.73	0.72	0.72	427
accuracy			0.69	1747
macro avg	0.53	0.52	0.51	1747
weighted avg	0.70	0.69	0.68	1747

Figure 26: Naive Bayes

8.3.13 Hyperparameter Tuning on Naive Bayes using Random Search

The following code snippet demonstrates how to perform hyperparameter tuning on a Naive Bayes classifier using Random Search to optimize its performance.

```
import pandas as pd
from sklearn.model_selection import train_test_split,
RandomizedSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from scipy.stats import uniform
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
if 'cleaned_text' not in dataset.columns or 'mental_health_issue'
not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
dataset.dropna(subset=['cleaned_text'], inplace=True)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['cleaned_text'])
y = dataset['mental_health_issue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
naive_bayes_model = MultinomialNB()
param_distributions = {
    'alpha': uniform(0.001, 5.0)
}
random_search = RandomizedSearchCV(estimator=naive_bayes_model,
param_distributions=param_distributions, n_iter=2000,
scoring='accuracy', cv=5, n_jobs=-1, random_state=42)
random_search.fit(X_train, y_train)
print("Best Hyperparameters:", random_search.best_params_)
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n",
classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

The code begins by importing the necessary libraries, including `pandas` for data manipulation, various functions from `sklearn` for model training and evaluation, and `uniform` from `scipy.stats` for sampling. It loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv` and checks for the presence of the required columns, raising a `ValueError` if either the `cleaned_text` or `mental_health_issue` columns are missing. Rows with missing values in the `cleaned_text` column are removed to maintain data quality. The `CountVectorizer` is initialized to convert the cleaned text into a numerical format, which is stored in X, while the target variable representing mental health issues is assigned to y. The dataset is split into training and testing sets, with 80% used for training and 20% for testing. A Naive Bayes classifier, specifically `MultinomialNB`, is initialized, and a parameter distribution is defined for hyperparameter tuning using Random Search. The `RandomizedSearchCV` object is created to perform random search over the hyperparameter grid, and the model is fitted using the training data. After fitting, the best hyperparameters are printed along with the best Naive Bayes model derived from the search. Predictions are made on the test set, and the accuracy of the model is calculated and displayed, along with a classification report and confusion matrix that provide detailed metrics on the model's performance across different categories.

Random Forest The following code snippet demonstrates the implementation of a Random Forest classifier for classifying mental health issues based on preprocessed text data.

```
import pandas as pd
```

```
Best Hyperparameters: {'fit_prior': True, 'alpha': 0.5}
Accuracy: 70.16%
Classification Report:
              precision    recall  f1-score   support

     anxiety       0.68      0.74      0.71       277
     bipolar       0.77      0.63      0.69       289
  depression       0.69      0.81      0.74       316
     neutral       0.20      0.43      0.27        14
      normal       0.17      0.04      0.07        23
        ptsd       0.75      0.69      0.72       304

    accuracy                           0.70      1223
   macro avg       0.54      0.56      0.53      1223
weighted avg       0.71      0.70      0.70      1223
```

Figure 27: Result of Hyperparameter Tuning on Naive Bayes

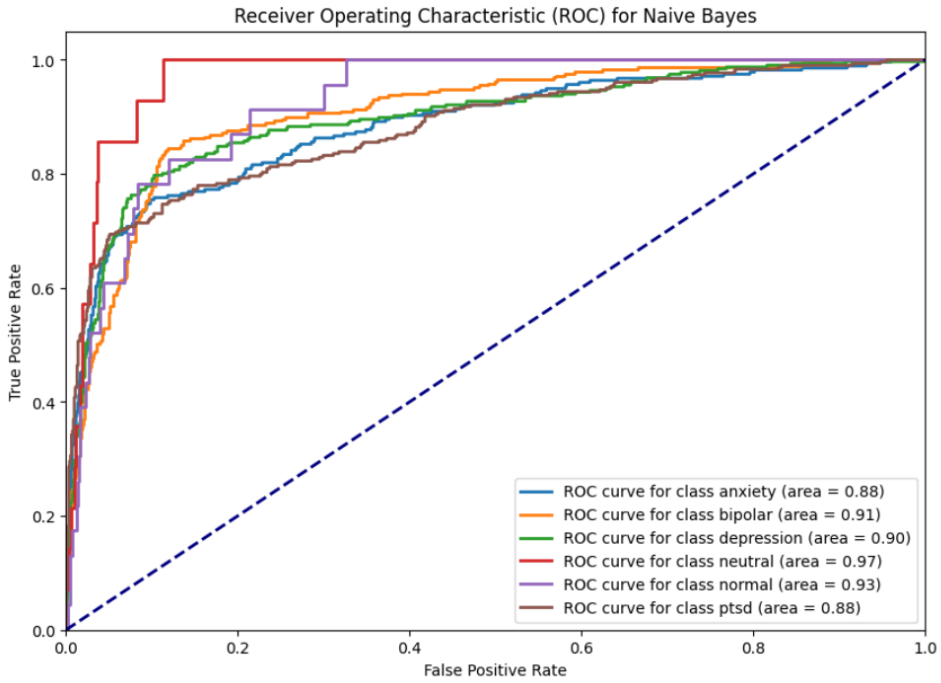


Figure 28: ROC Curve on Naive bayes

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

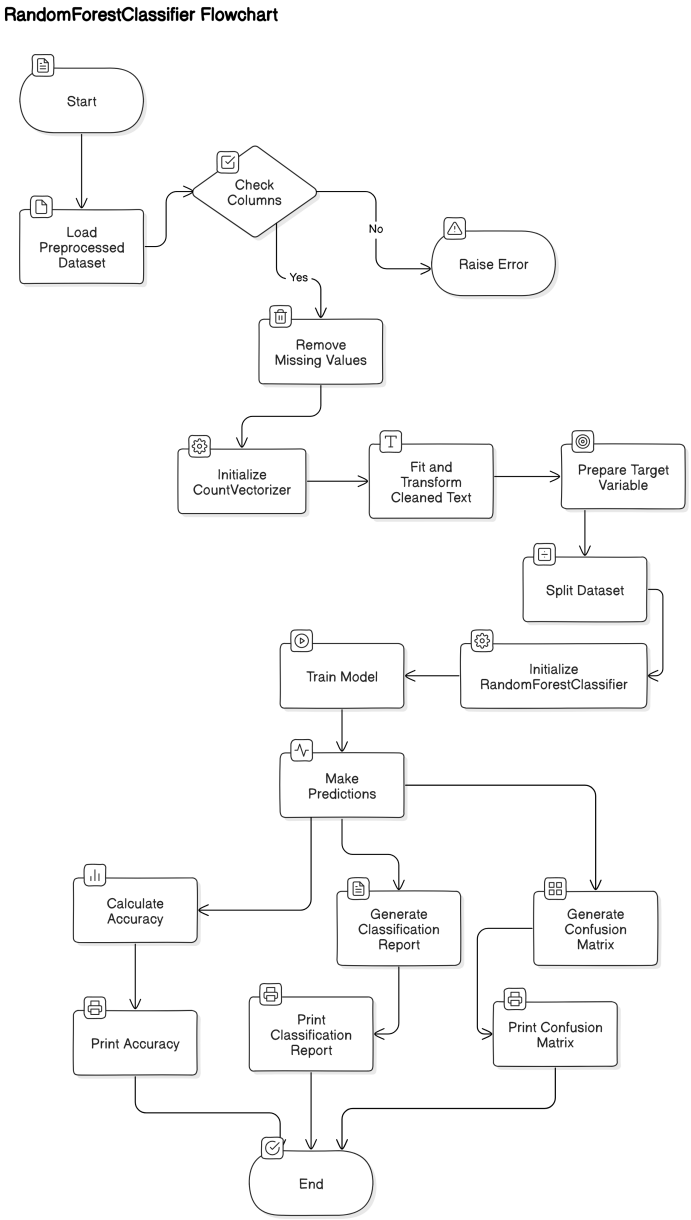


Figure 29: Random Forest Workflow

```
dataset = pd.read_csv('preprocessed_mental_health_text.csv')
if 'cleaned_text' not in dataset.columns or 'mental_health_issue'
not in dataset.columns:
    raise ValueError("The dataset must have 'cleaned_text' and
    'mental_health_issue' columns.")
dataset.dropna(subset=['cleaned_text'], inplace=True)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(dataset['cleaned_text'])
```

```
y = dataset['mental_health_issue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
rf_model = RandomForestClassifier(
    n_estimators=3000,
    max_depth=None,
    min_samples_split=20,
    min_samples_leaf=1,
    max_features='sqrt',
    bootstrap=False,
    random_state=42
)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n",
classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

The code begins by importing the necessary libraries, including `pandas` for data manipulation and various functions from `sklearn` for model training and evaluation. It loads the preprocessed dataset from a CSV file named `preprocessed_mental_health_text.csv` and checks for the presence of the required columns, raising a `ValueError` if either the `cleaned_text` or `mental_health_issue` columns are missing. Rows with missing values in the `cleaned_text` column are then removed to ensure data quality.

The `CountVectorizer` is initialized to convert the cleaned text into a numerical format, which is stored in X, while the target variable representing mental health issues is assigned to y. The dataset is split into training and testing sets, with 80% used for training and 20% for testing. A Random Forest classifier is initialized with specific parameters, including the number of trees (`n_estimators`), the minimum number of samples required to split a node (`min_samples_split`), the minimum number of samples in a leaf node (`min_samples_leaf`), and the number of features to consider when looking for the best split (`max_features`). The model is then trained using the training data, and predictions are made on the test set. The accuracy of the model is calculated and displayed, along with a classification report and confusion matrix that provide detailed metrics on the model's performance across different categories.

Accuracy: 74.87%
Classification Report:

	precision	recall	f1-score	support
anxiety	0.76	0.77	0.76	416
bipolar	0.70	0.77	0.74	412
depression	0.69	0.80	0.74	443
neutral	0.20	0.06	0.09	17
normal	0.83	0.16	0.26	32
ptsd	0.88	0.73	0.80	427
accuracy			0.75	1747
macro avg	0.68	0.55	0.57	1747
weighted avg	0.75	0.75	0.74	1747

Figure 30: Random Forest Result

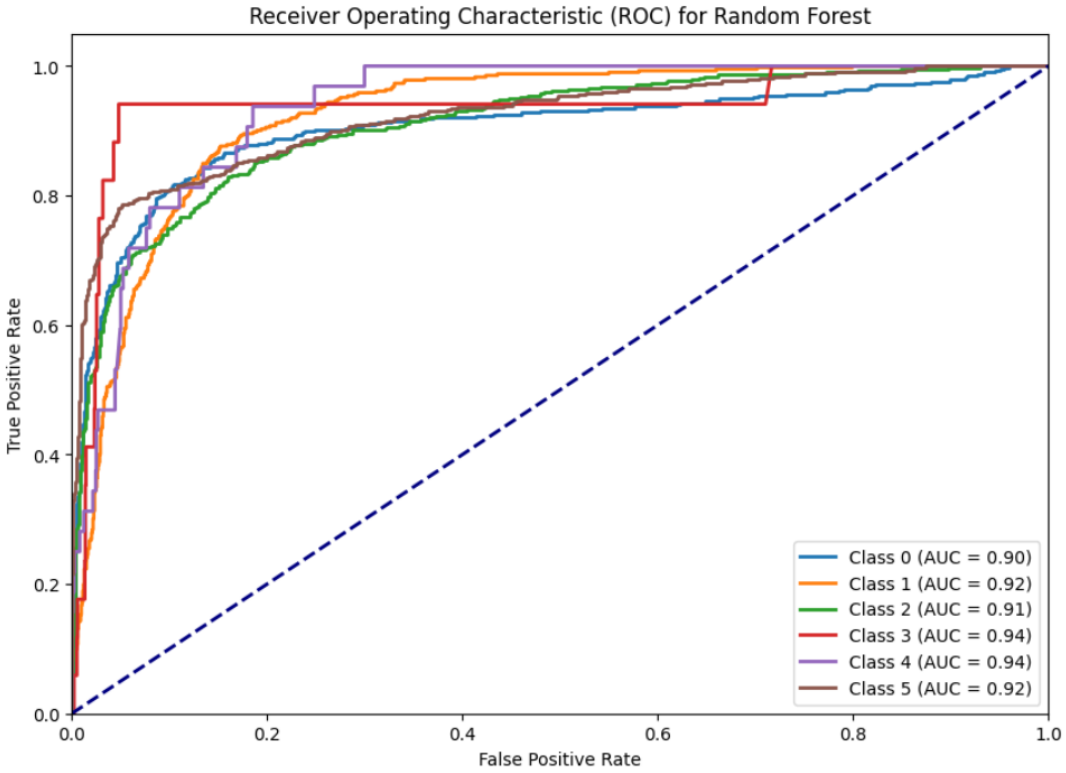


Figure 31: ROC Curve on Random Forest

9 Results and Analysis

The metrics used for evaluating the performance of the classification models include Precision, Recall, F1-Score, and Support, along with the Confusion Matrix. These metrics are crucial for assessing how well the models are able to differentiate between various classes, providing insight into their accuracy, ability to capture relevant instances, and the overall balance between precision and recall. By analyzing these metrics, a comprehensive understanding of the model's performance can be obtained, enabling informed decisions for further optimization and tuning.

9.1 Classification Metrics and Confusion Matrix

- **Precision:** Precision is the ratio of correctly predicted positive observations to the total predicted positives. It can be calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

  where TP represents True Positives, and FP represents False Positives.

- **Recall:** Recall is the ratio of correctly predicted positive observations to all observations in the actual class:

$$\text{Recall} = \frac{TP}{TP + FN}$$

  where TP is True Positives, and FN is False Negatives.

- **F1-Score:** F1-Score is the harmonic mean of Precision and Recall, calculated as:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Support:** Support refers to the number of actual occurrences of each class in the dataset:

$$\text{Support} = \text{Number of samples in the true class}$$

- **Confusion Matrix:** A confusion matrix is used to evaluate the performance of a classification model. It is structured as follows:

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

  where:

  - TP = True Positives
  - FP = False Positives
  - FN = False Negatives
  - TN = True Negatives

9.2 Results of Logistic Regression and Hyperparameter Tuning

Logistic Regression Classification Report (Before Hyperparameter Tuning)

Class	Precision	Recall	F1-Score	Support
Anxiety	0.78	0.73	0.75	416
Bipolar	0.66	0.84	0.74	412
Depression	0.74	0.72	0.73	443
Neutral	0.08	0.06	0.07	17
Normal	0.78	0.22	0.34	32
PTSD	0.83	0.74	0.78	427
Accuracy	73.78%			
Macro Avg	0.64	0.55	0.57	1747
Weighted Avg	0.75	0.74	0.74	1747

Logistic Regression Classification Report (After Hyperparameter Tuning)

Class	Precision	Recall	F1-Score	Support
Anxiety	0.80	0.75	0.77	416
Bipolar	0.66	0.85	0.74	412
Depression	0.75	0.74	0.75	443
Neutral	0.14	0.12	0.13	17
Normal	0.78	0.22	0.34	32
PTSD	0.86	0.75	0.80	427
Accuracy	75.27%			
Macro Avg	0.67	0.57	0.59	1747
Weighted Avg	0.76	0.75	0.75	1747

Confusion Matrix for Logistic Regression (After Hyperparameter Tuning)

True Class	Anxiety	Bipolar	Depression	Neutral	Normal	PTSD
Anxiety	310	45	37	2	1	21
Bipolar	12	349	35	5	1	10
Depression	34	62	327	2	0	18
Neutral	1	8	5	2	0	1
Normal	1	23	1	0	7	0
PTSD	31	44	29	3	0	320

9.3 Explanation of the Results (Logistic Regression)

Overall Performance:

The Logistic Regression model's initial accuracy was **73.78%**, which improved to **75.27%** after hyperparameter tuning using Randomized Search.

Improvement After Hyperparameter Tuning:

- The tuned model showed a slight increase in accuracy, reflecting an improved ability to distinguish between the different mental health classes.

- The precision, recall, and F1-scores for most of the individual classes also showed minor improvements.

Class-wise Analysis:

- **Anxiety:** The precision and recall improved slightly, indicating better classification of anxiety-related texts.
- **Bipolar:** A notable increase in recall suggests the tuned model became more sensitive to identifying bipolar instances correctly.
- **Depression:** The performance metrics remained relatively stable, showing the model was consistently able to handle depression cases.
- **Neutral & Normal:** Low precision and recall values indicate that the model struggles to correctly classify these rare classes, likely due to insufficient support (sample size).

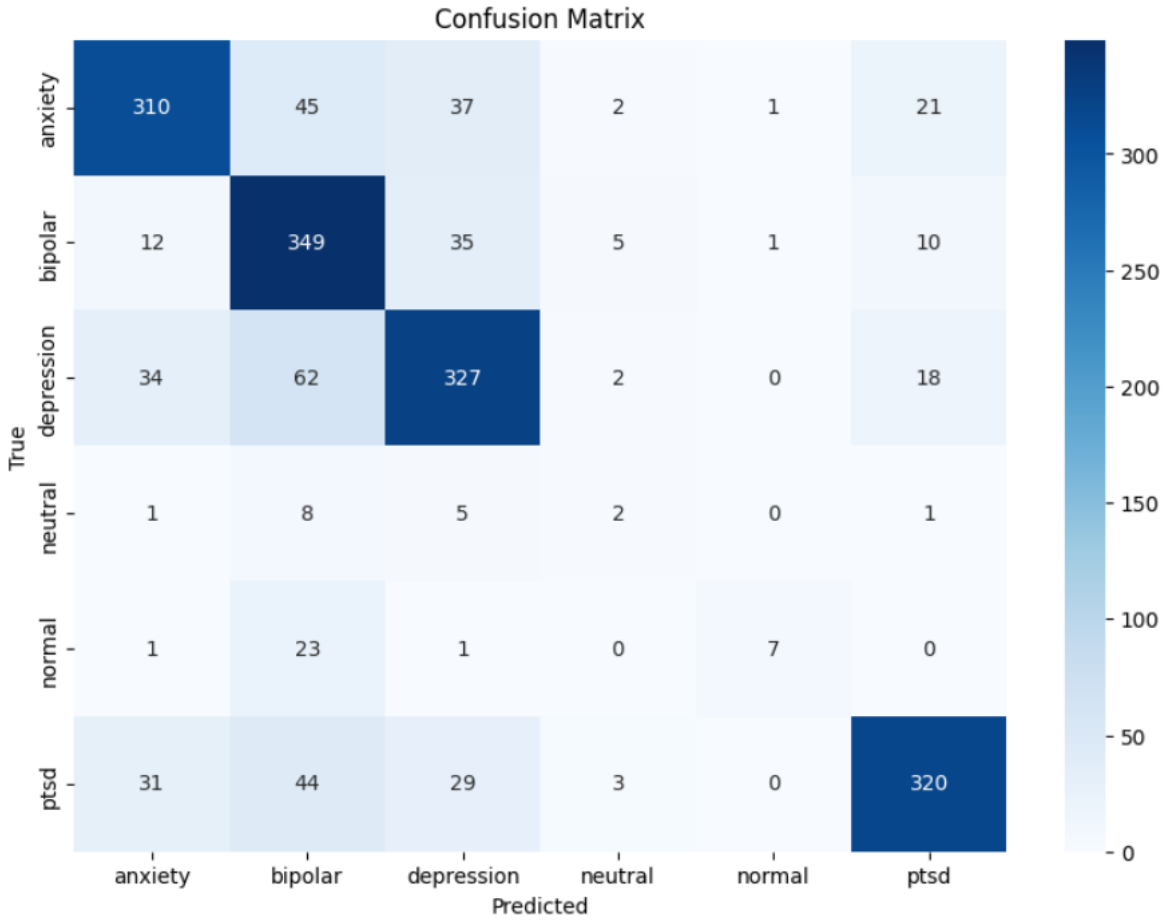


Figure 32: Confusion Matrix for Logistic Regression

Confusion Matrix Analysis:

- The confusion matrix reveals where the model misclassifies different categories. For example:
  - **Anxiety** is often confused with **Bipolar** and **Depression**, indicating overlapping features between these classes.
  - **PTSD** is more distinct, as shown by a higher number of true positive predictions (320), suggesting that the model can better differentiate this category.
- Overall, the confusion matrix helps pinpoint specific classes where the model's predictions are less reliable, providing direction for further improvements.

9.4 Results of K-Nearest Neighbors (KNN) and Hyperparameter Tuning

KNN Classification Report (Before Hyperparameter Tuning)

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Anxiety	0.60	0.36	0.45	416
Bipolar	0.31	0.83	0.45	412
Depression	0.46	0.35	0.40	443
Neutral	0.00	0.00	0.00	17
Normal	0.29	0.06	0.10	32
PTSD	0.81	0.11	0.20	427
Accuracy	39.61%			
Macro Avg	0.41	0.28	0.27	1747
Weighted Avg	0.54	0.40	0.36	1747

Confusion Matrix for KNN (Before Hyperparameter Tuning)

True Class	**Anxiety**	**Bipolar**	**Depression**	**Neutral**	**Normal**	**PTSD**
Anxiety	148	221	46	0	0	1
Bipolar	24	340	42	1	3	2
Depression	36	246	154	0	0	7
Neutral	2	14	0	0	0	1
Normal	0	30	0	0	2	0
PTSD	38	248	91	0	2	48

KNN Classification Report (After Hyperparameter Tuning)

Class	Precision	Recall	F1-Score	Support
Anxiety	0.65	0.34	0.45	416
Bipolar	0.32	0.82	0.46	412
Depression	0.47	0.39	0.43	443
Neutral	0.00	0.00	0.00	17
Normal	0.26	0.19	0.22	32
PTSD	0.73	0.15	0.24	427
Accuracy	41.27%			
Macro Avg	0.40	0.31	0.30	1747
Weighted Avg	0.53	0.41	0.39	1747

Confusion Matrix for KNN (After Hyperparameter Tuning)

True Class	Anxiety	Bipolar	Depression	Neutral	Normal	PTSD
Anxiety	143	214	52	0	2	5
Bipolar	19	337	45	1	7	3
Depression	31	223	173	1	1	14
Neutral	0	15	1	0	0	1
Normal	0	26	0	0	6	0
PTSD	28	233	97	0	7	62

9.5 Explanations for KNN Results

- **Initial KNN Performance:**
  - The initial accuracy of the KNN model is low (**39.61%**), indicating that the model struggles to classify the mental health categories effectively.
  - The recall for the **Bipolar** class is high (0.83), suggesting that the model is sensitive to detecting bipolar cases but tends to misclassify other categories as bipolar.
  - Other classes like **Neutral** and **Normal** have very low precision and recall, highlighting the model's inability to distinguish these less frequent categories.
- **After Hyperparameter Tuning:**
  - The accuracy improved slightly to **41.27%** after tuning with the best hyperparameters {'weights': 'distance', 'n_neighbors': 4, 'metric': 'euclidean'}, indicating minor gains in the classification.
  - The recall for the **Anxiety** class decreased, while the precision increased, showing a trade-off in performance.
  - For the **Depression** class, both precision and recall increased slightly, indicating that the tuning helped the model distinguish depression cases better.

- **Confusion Matrix Analysis:**
  - Many instances of **Anxiety** and **Depression** are misclassified as **Bipolar**, showing overlapping features between these classes.
  - The class **PTSD** has a low recall, indicating that most of the PTSD samples are misclassified into other categories.
  - The class distribution imbalance and similar feature patterns might be contributing to the misclassifications, causing the model to favor more common classes like Bipolar.

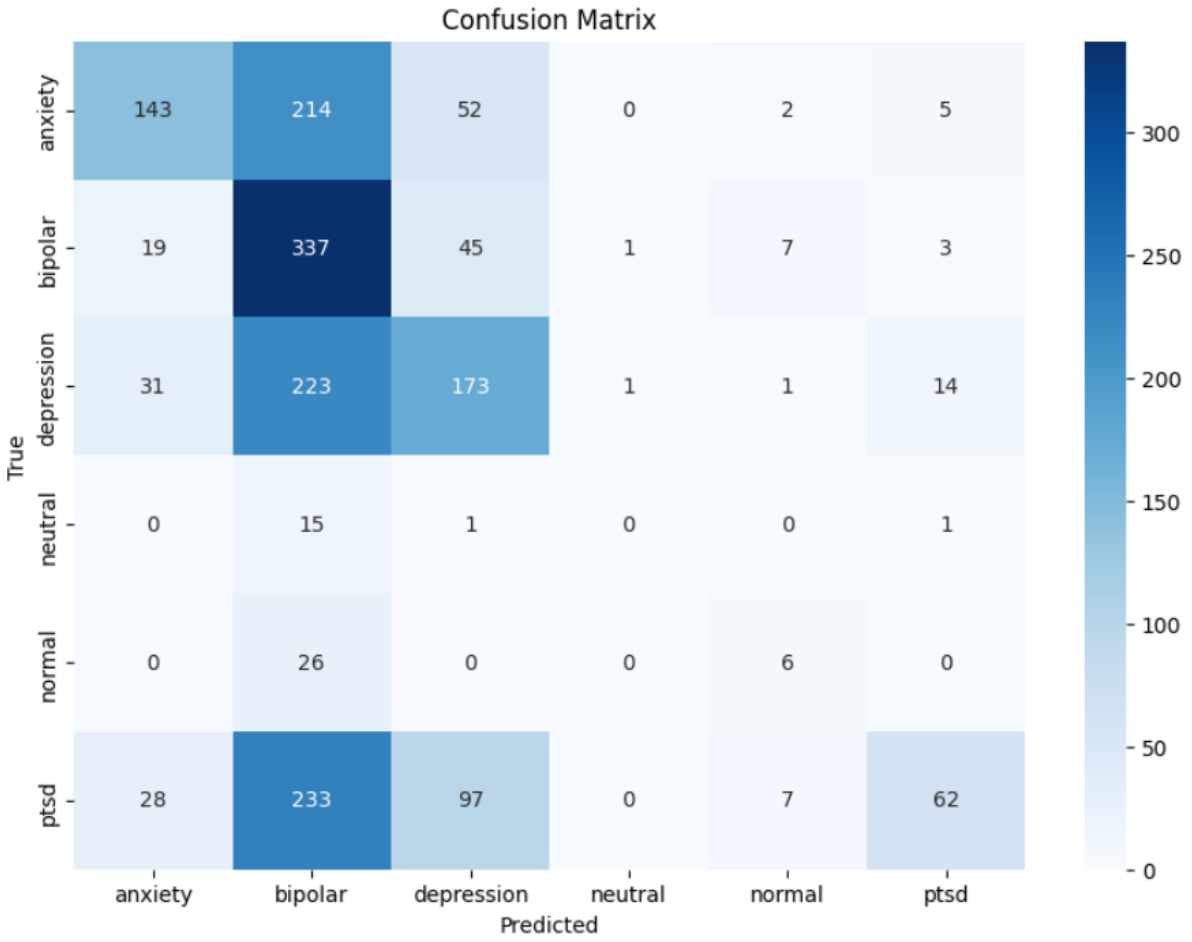


Figure 33: Confusion Matrix for KNN

9.6 Results of Support Vector Machine (SVM) and Hyperparameter Tuning

SVM Classification Report (Before Hyperparameter Tuning)

Class	Precision	Recall	F1-Score	Support
Anxiety	0.70	0.70	0.70	416
Bipolar	0.65	0.81	0.72	412
Depression	0.72	0.67	0.69	443
Neutral	0.19	0.18	0.18	17
Normal	0.62	0.25	0.36	32
PTSD	0.81	0.71	0.76	427
Accuracy	70.75%			
Macro Avg	0.61	0.55	0.57	1747
Weighted Avg	0.71	0.71	0.71	1747

Confusion Matrix for SVM (Before Hyperparameter Tuning)

True Class	Anxiety	Bipolar	Depression	Neutral	Normal	PTSD
Anxiety	292	45	44	2	1	32
Bipolar	23	333	37	5	3	11
Depression	59	58	296	2	0	28
Neutral	1	7	5	3	0	1
Normal	0	24	0	0	8	0
PTSD	41	47	30	4	1	304

SVM Classification Report (After Hyperparameter Tuning)

Class	Precision	Recall	F1-Score	Support
Anxiety	0.78	0.71	0.74	416
Bipolar	0.61	0.83	0.70	412
Depression	0.75	0.70	0.72	443
Neutral	0.13	0.12	0.12	17
Normal	0.57	0.12	0.21	32
PTSD	0.84	0.71	0.77	427
Accuracy	72.07%			
Macro Avg	0.61	0.53	0.54	1747
Weighted Avg	0.73	0.72	0.72	1747

Confusion Matrix for SVM (After Hyperparameter Tuning)

True Class	Anxiety	Bipolar	Depression	Neutral	Normal	PTSD
Anxiety	292	45	44	2	1	32
Bipolar	23	333	37	5	3	11
Depression	59	58	296	2	0	28
Neutral	1	7	5	3	0	1
Normal	0	24	0	0	8	0
PTSD	41	47	30	4	1	304

9.7 Explanations for SVM Results

- **Initial SVM Performance:**
  - The initial accuracy of the SVM model is **70.75%**, indicating a moderate performance in classifying mental health categories.
  - The **Bipolar** class has a high recall of 0.81, indicating the model's ability to identify most bipolar cases correctly. However, the precision is lower, showing misclassification with other classes.
  - The **Neutral** and **Normal** classes have low precision and recall values, indicating that the model struggles to identify these minority classes.
- **After Hyperparameter Tuning:**
  - The accuracy improved slightly to **72.07%** with the best hyperparameters {'kernel': 'linear', 'gamma': 'scale', 'C': 0.1}.
  - The recall for the **Anxiety** and **Depression** classes increased, indicating better sensitivity to these categories.
  - The precision for **PTSD** also improved, suggesting the model has a clearer distinction for this class after tuning.

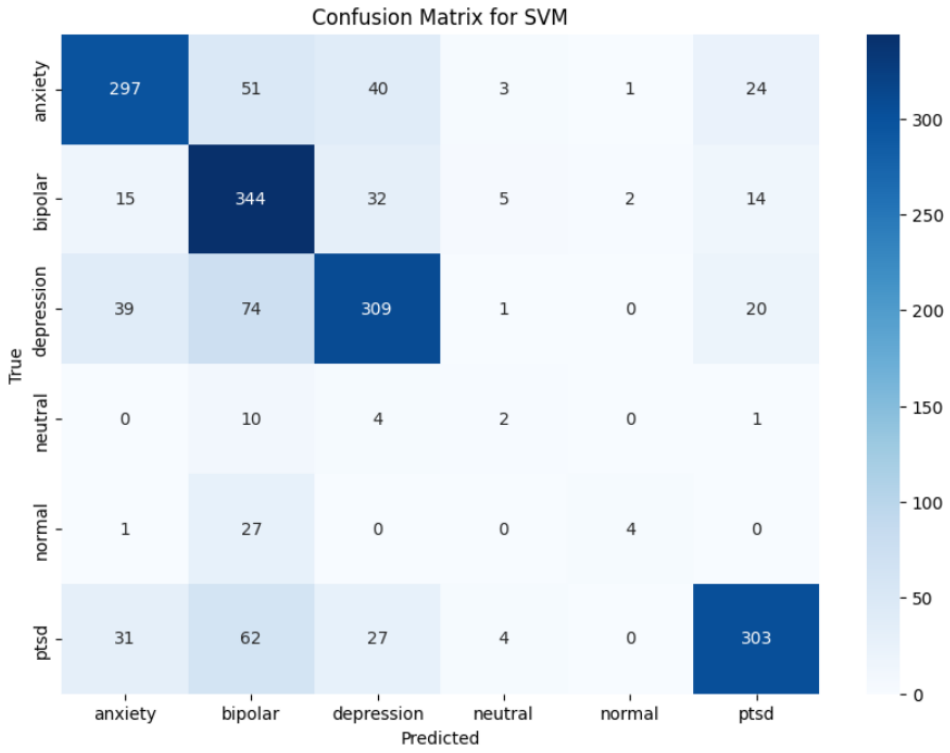


Figure 34: Confusion Matrix for SVM

- **Confusion Matrix Analysis:**
  - Many instances of **Anxiety** and **Depression** are misclassified as **Bipolar**, showing overlapping features between these classes.
  - The **Neutral** and **Normal** classes have very few true positive predictions, indicating that the SVM model still struggles to capture these less frequent categories.
  - Overall, the slight improvement in performance after hyperparameter tuning suggests that tuning helped the SVM classifier better balance the trade-offs between precision and recall for most major classes.

9.8 Results of Naive Bayes and Hyperparameter Tuning

Naive Bayes Classification Report (Before Hyperparameter Tuning)

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Anxiety	0.69	0.72	0.70	416
Bipolar	0.79	0.56	0.65	412
Depression	0.65	0.82	0.72	443
Neutral	0.17	0.29	0.21	17
Normal	0.14	0.03	0.05	32
PTSD	0.73	0.72	0.72	427
Accuracy	68.92%			
Macro Avg	0.53	0.52	0.51	1747
Weighted Avg	0.70	0.69	0.68	1747

Confusion Matrix for Naive Bayes (Before Hyperparameter Tuning)

True Class	**Anxiety**	**Bipolar**	**Depression**	**Neutral**	**Normal**	**PTSD**
Anxiety	300	14	61	8	3	30
Bipolar	51	229	78	5	0	49
Depression	27	20	362	5	1	28
Neutral	1	6	4	5	1	0
Normal	12	2	9	1	1	7
PTSD	46	20	47	6	1	307

Naive Bayes Classification Report (After Hyperparameter Tuning)

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Anxiety	0.68	0.74	0.71	277
Bipolar	0.77	0.63	0.69	289
Depression	0.69	0.81	0.74	316
Neutral	0.20	0.43	0.27	14
Normal	0.17	0.04	0.07	23
PTSD	0.75	0.69	0.72	304
Accuracy	70.16%			
Macro Avg	0.54	0.56	0.53	1223
Weighted Avg	0.71	0.70	0.70	1223

Confusion Matrix for Naive Bayes (After Hyperparameter Tuning)

True Class	Anxiety	Bipolar	Depression	Neutral	Normal	PTSD
Anxiety	300	14	61	8	3	30
Bipolar	51	229	78	5	0	49
Depression	27	20	362	5	1	28
Neutral	1	6	4	5	1	0
Normal	12	2	9	1	1	7
PTSD	46	20	47	6	1	307

9.9 Explanations for Naive Bayes Results

- **Initial Naive Bayes Performance:**
  - The initial accuracy of the Naive Bayes model is **68.92%**, indicating moderate classification performance.
  - The **Anxiety** and **Depression** classes have relatively higher recall values, suggesting that the model is sensitive to identifying these categories correctly.
  - The **Neutral** and **Normal** classes have very low precision and recall, indicating that the model struggles to classify these minority classes.
- **After Hyperparameter Tuning:**
  - The accuracy increased slightly to **70.16%** with the best hyperparameters {'fit_prior': True, 'alpha': 0.5}, showing a minor performance improvement.
  - The recall for the **Anxiety** and **Depression** classes increased, indicating better sensitivity and improved classification for these classes.
  - However, the precision for the **Bipolar** class decreased slightly, suggesting a trade-off in correctly identifying true positive cases for this class.
- **Confusion Matrix Analysis:**
  - Many instances of **Anxiety** and **Depression** are misclassified as **Bipolar**, indicating overlapping feature characteristics between these categories.
  - The **Neutral** and **Normal** classes continue to have very few true positive predictions, highlighting the model's struggle to capture these less frequent categories.
  - The increase in recall for major classes like **Anxiety** and **Depression** after tuning suggests that the model learned better class-specific distributions.

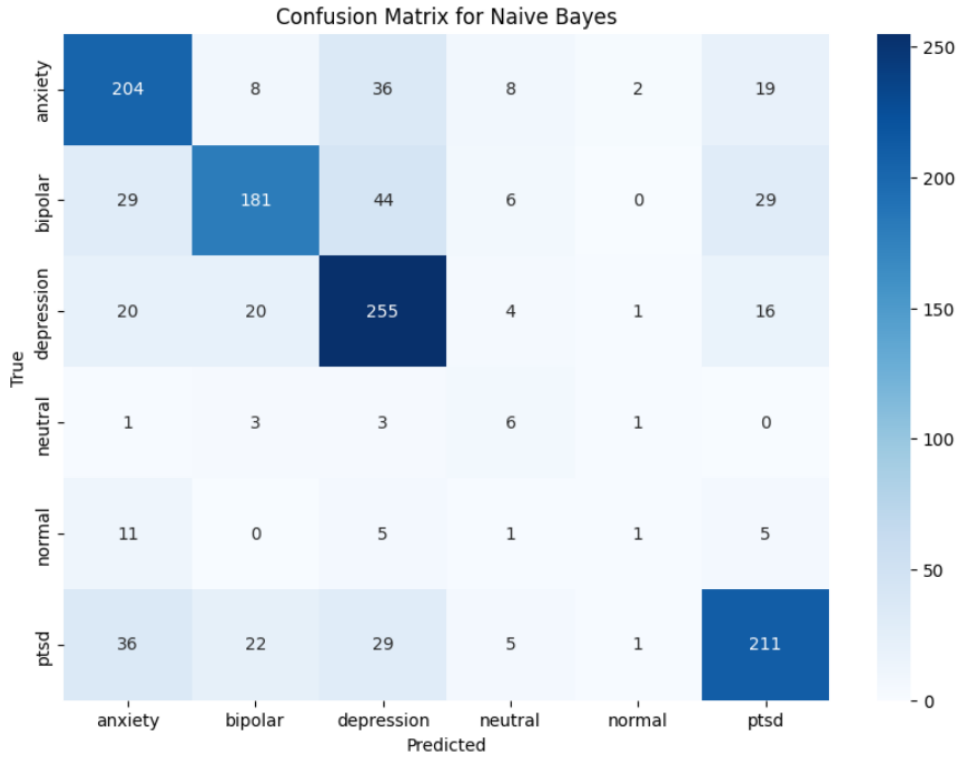


Figure 35: Confusion Matrix for Naive Bayes

9.10 Results of Random Forest and Hyperparameter Tuning

Random Forest Classification Report

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Anxiety	0.76	0.77	0.76	416
Bipolar	0.70	0.77	0.74	412
Depression	0.69	0.80	0.74	443
Neutral	0.20	0.06	0.09	17
Normal	0.83	0.16	0.26	32
PTSD	0.88	0.73	0.80	427
Accuracy	74.87%			
Macro Avg	0.68	0.55	0.57	1747
Weighted Avg	0.75	0.75	0.74	1747

Confusion Matrix for Random Forest

True Class	**Anxiety**	**Bipolar**	**Depression**	**Neutral**	**Normal**	**PTSD**
Anxiety	319	38	50	0	0	9
Bipolar	23	319	53	2	1	14
Depression	31	41	353	0	0	18
Neutral	1	9	5	1	0	1
Normal	3	24	0	0	5	0
PTSD	43	22	49	2	0	311

9.11 Explanations for Random Forest Results

- **Overall Performance:**
  - The accuracy of the Random Forest model is **74.87%**, indicating a good level of performance in classifying the different mental health categories.
  - The **PTSD** class shows the highest precision (0.88) and recall (0.73), suggesting that the model is effective at identifying PTSD cases correctly.
- **Class-wise Analysis:**
  - The precision and recall for the **Anxiety** and **Depression** classes are reasonably balanced, indicating that the model is adept at classifying these categories.
  - The **Bipolar** class has a slightly lower precision but high recall, suggesting that while the model can identify bipolar instances effectively, it also misclassifies some non-bipolar cases as bipolar.
  - The performance for the **Neutral** and **Normal** classes is concerning, with very low recall and F1-scores, indicating that these classes are often misclassified.

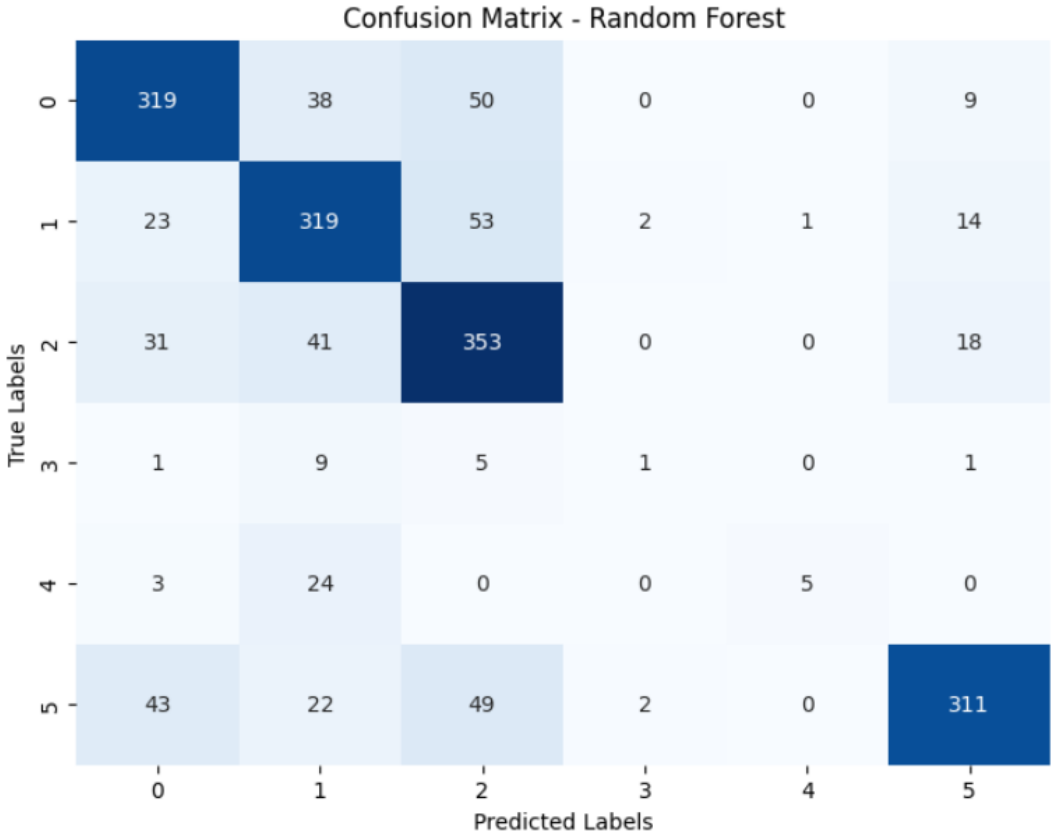


Figure 36: Confusion Matrix for Random Forest

- **Confusion Matrix Analysis:**
  - The confusion matrix reveals that many instances of **Anxiety** and **Depression** are correctly classified, with a significant number of **Neutral** instances being misclassified.
  - The low count for **Normal** indicates that the model has trouble distinguishing this category, as evidenced by the high number of false positives.
  - The high number of true positive predictions for **PTSD** (311) suggests that the Random Forest model is effective in distinguishing PTSD from other classes, likely due to its ability to handle complex feature interactions well.

9.12 Comparison of Different Models

Datasets Overview

Dataset	**Capacity**	**Modality**
Dataset - Reddit	12000	Single
Dataset - Twitter	900	Single
DS - Reddit + Twitter	12900	Single

Comparison of Classification Metrics for Anxiety

Model	**Precision**	**Recall**	**F1-Score**	**Support**
Logistic Regression	0.78	0.73	0.75	416
KNN	0.60	0.36	0.45	416
SVM	0.70	0.70	0.70	416
Naive Bayes	0.69	0.72	0.70	416
Random Forest	0.76	0.77	0.76	416

Comparison of Classification Metrics for Bipolar

Model	**Precision**	**Recall**	**F1-Score**	**Support**
Logistic Regression	0.66	0.84	0.74	412
KNN	0.31	0.83	0.45	412
SVM	0.65	0.81	0.72	412
Naive Bayes	0.79	0.56	0.65	412
Random Forest	0.70	0.77	0.74	412

Comparison of Classification Metrics for Depression

Model	**Precision**	**Recall**	**F1-Score**	**Support**
Logistic Regression	0.74	0.72	0.73	443
KNN	0.46	0.35	0.40	443
SVM	0.72	0.67	0.69	443
Naive Bayes	0.65	0.82	0.72	443
Random Forest	0.69	0.80	0.74	443

Comparison of Classification Metrics for Neutral

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.08	0.06	0.07	17
KNN	0.00	0.00	0.00	17
SVM	0.19	0.18	0.18	17
Naive Bayes	0.17	0.29	0.21	17
Random Forest	0.20	0.06	0.09	17

Comparison of Classification Metrics for Normal

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.78	0.22	0.34	32
KNN	0.29	0.06	0.10	32
SVM	0.62	0.25	0.36	32
Naive Bayes	0.14	0.03	0.05	32
Random Forest	0.83	0.16	0.26	32

Comparison of Classification Metrics for PTSD

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.83	0.74	0.78	427
KNN	0.81	0.11	0.20	427
SVM	0.81	0.71	0.76	427
Naive Bayes	0.73	0.72	0.72	427
Random Forest	0.88	0.73	0.80	427

Comparison of Classification Metrics for Anxiety (After Hyperparameter Tuning)

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.80	0.75	0.77	416
KNN	0.65	0.34	0.45	416
SVM	0.78	0.71	0.74	416
Naive Bayes	0.68	0.74	0.71	277

Comparison of Classification Metrics for Bipolar (After Hyperparameter Tuning)

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.66	0.85	0.74	412
KNN	0.32	0.82	0.46	412
SVM	0.61	0.83	0.70	412
Naive Bayes	0.77	0.63	0.69	289

Comparison of Classification Metrics for Depression (After Hyperparameter Tuning)

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.75	0.74	0.75	443
KNN	0.47	0.39	0.43	443
SVM	0.75	0.70	0.72	443
Naive Bayes	0.69	0.81	0.74	316

Comparison of Classification Metrics for Neutral (After Hyperparameter Tuning)

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.14	0.12	0.13	17
KNN	0.00	0.00	0.00	17
SVM	0.13	0.12	0.12	17
Naive Bayes	0.20	0.43	0.27	14

Comparison of Classification Metrics for Normal (After Hyperparameter Tuning)

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.78	0.22	0.34	32
KNN	0.26	0.19	0.22	32
SVM	0.57	0.12	0.21	32
Naive Bayes	0.17	0.04	0.07	23

Comparison of Classification Metrics for PTSD (After Hyperparameter Tuning)

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.86	0.75	0.80	427
KNN	0.73	0.15	0.24	427
SVM	0.84	0.71	0.77	427
Naive Bayes	0.75	0.69	0.72	304

10 Conclusion

10.1 Project Benefits

The project on detecting mental health disorders through social media analysis offers a wide array of significant benefits, both immediate and long-term, across multiple dimensions. First and foremost, it addresses a critical issue in mental health care—early detection and intervention. Social media has become a ubiquitous platform where people express their thoughts, feelings, and emotional states, often unconsciously. By leveraging the vast amounts of data available on social media platforms, our project seeks to tap into this resource to identify early signs of mental health disorders such as anxiety, depression, and more severe conditions like bipolar disorder or schizophrenia. The ability to detect mental health issues through real-time social media data is a game-changer for public health systems, mental health practitioners, and even individuals who may not realize they are at risk. Early detection enables timely intervention, reducing the overall burden of mental health disorders on society by preventing escalation into more severe conditions that often lead to hospitalization, self-harm, or even suicide. In this sense, the project aligns with global health initiatives that emphasize early diagnosis and preventive care.

Moreover, this project holds significant potential for improving the accuracy and efficiency of mental health diagnostics. Traditional diagnostic methods are often time-consuming, sub-

jective, and reliant on self-reporting, which can lead to underdiagnosis or misdiagnosis. By utilizing machine learning algorithms and natural language processing techniques, our project automates the process of sentiment and behavioral analysis on social media platforms, offering a more objective and data-driven approach. This automated system can process large volumes of data much faster than human professionals, providing insights that would be impossible to glean from manual analysis. The algorithms developed as part of this project can be easily scaled to analyze millions of social media posts, enabling a broader reach in monitoring public mental health trends. Additionally, the project offers practical benefits for mental health professionals, allowing them to focus on treatment and intervention rather than diagnosis. It provides a tool that can be integrated into telehealth systems, offering mental health screening at scale, which is particularly valuable in underserved or rural areas where access to mental health professionals is limited.

From a technological standpoint, the project offers a host of reusable components and methodologies. The machine learning models developed, the sentiment analysis tools, and the overall data pipeline are designed to be scalable and modular. These components can be adapted and extended to other domains beyond mental health, such as market sentiment analysis, public opinion monitoring, or even detecting harmful behavior like cyberbullying and harassment online. By advancing the state of the art in social media analytics, this project contributes to the growing field of AI-driven health care solutions. Furthermore, it provides a blueprint for future interdisciplinary work that integrates data science, psychology, and public health.

10.2 Future Scope for Improvements

While this project offers numerous immediate benefits, there is substantial room for future enhancements that can broaden its applicability, accuracy, and effectiveness. One of the key areas for improvement lies in the expansion of data sources. Currently, the project focuses on analyzing Twitter sentiment data from Kaggle, which is limited to a specific social media platform and dataset. In the future, incorporating data from other platforms like Facebook, Instagram, Reddit, and even niche forums could provide a more comprehensive understanding of an individual's mental health status. Different platforms cater to different demographics and social behaviors, and expanding the dataset will allow for a more holistic analysis of mental health indicators across various user bases. Additionally, expanding the dataset to include multilingual posts or integrating language translation capabilities could make the system applicable to a global audience, helping to identify mental health issues in non-English speaking populations.

Another area ripe for future development is the improvement of the machine learning models used for mental health prediction. Current models, while effective, could benefit from more advanced techniques such as deep learning architectures, which are particularly powerful in handling unstructured data like text and images. For instance, implementing models like transformers or neural networks that can better understand the context and nuance in human language could lead to more accurate predictions. Additionally, incorporating multi-modal data, such as analyzing images and videos along with textual data, could provide richer insights into a user's mental health state. Emotions and mental health issues are often expressed visually as well, and combining these different data types could significantly improve the system's accuracy.

Moreover, future improvements could focus on integrating real-time data analysis capabilities. Currently, our project is based on batch processing of historical data. However, in future iterations, the system could be developed to perform real-time monitoring, offering immediate feedback and potentially alerting health professionals or loved ones when someone shows signs of mental distress. This real-time capability would be invaluable in emergency situations, allowing for immediate intervention. Developing a mobile application or a web-based interface where users can voluntarily connect their social media accounts to monitor their mental health status could also increase user engagement and provide individuals with direct feedback on their well-being.

Another significant future enhancement could involve incorporating ethical considerations and improving user privacy. As mental health is a sensitive subject, ensuring that the system is designed with robust privacy protections is critical. Future work could focus on using differential privacy or other anonymization techniques to ensure that user data remains confidential while still allowing for effective analysis. Moreover, collaborating with psychologists, ethicists, and legal experts could help refine the system to ensure it adheres to ethical guidelines and avoids potential harm, such as misdiagnosis or privacy violations.

Lastly, the future scope of this project could include expanding its use in clinical settings. While the current system is primarily designed as a research tool, future iterations could be developed in collaboration with mental health professionals to ensure that it meets clinical standards. By integrating this system with electronic health records or telehealth platforms, it could become a critical tool in mental health care, helping practitioners monitor their patients' mental health between sessions. This would allow for a more proactive approach to mental health care, potentially reducing the need for emergency interventions and improving overall treatment outcomes.

11 References

[1] Hatoon S AlSagri and Mourad Ykhlef. Machine learning-based approach for depression detection in twitter using content and activity features. *IEICE Transactions on Information and Systems*, 103(8):1825–1832, 2020.

[2] Munmun De Choudhury, Michael Gamon, Scott Counts, and Eric Horvitz. Predicting depression via social media. *Proceedings of the International AAAI Conference on Web and Social Media*, 2013.

[3] Sharath Chandra Guntuku, David Bryce Yaden, Margaret L. Kern, Lyle H. Ungar, and Johannes C. Eichstaedt. Detecting depression and mental illness on social media: an integrative review. *Current Opinion in Behavioral Sciences*, 18:43–49, 2017.

[4] Priya Mathur, Amit Kumar Gupta, and Abhishek Dadhich. Mental health classification on social-media: Systematic review. *Proceedings of the 4th International Conference on Information Management & Machine Intelligence*, 2022.

[5] Moin Nadeem. Identifying depression on twitter. *arXiv preprint arXiv:1607.07384*, 2016.

[6] Ramin Safa, S. A. Edalatpanah, and Ali Sorourkhah. Predicting mental health using social media: A roadmap for future development, 2023.

[7] Konda Vaishnavi, U Nikhitha Kamath, B Ashwath Rao, and N V Subba Reddy. Predicting mental health illness using machine learning algorithms. *Journal of Physics: Conference Series*, 2161(1):012021, jan 2022.