

A  
Major Project  
on  
**BUILDING A REVIEW ANALYZER WITH NLP**  
Submitted in partial fulfillment of the requirements for the award of degree  
**BACHELOR OF TECHNOLOGY**  
**IN**  
**COMPUTER SCIENCE AND ENGINEERING**

Submitted By

<b>T. PRANITHA</b>	<b>217Z1A05H4</b>
<b>S. SAI SRIDHAR NAIDU</b>	<b>217Z1A05G6</b>
<b>T. MADHAVA SWAMY</b>	<b>217Z1A05H6</b>

Under the Guidance of

**Dr. G. SRAVAN KUMAR**

Associate Professor



**SCHOOL OF ENGINEERING**

**Department of Computer Science and Engineering**

**NALLA NARASIMHA REDDY**

**EDUCATION SOCIETY'S GROUP OF INSTITUTIONS**

**(AN AUTONOMOUS INSTITUTION)**

**Approved by AICTE, New Delhi, Chowdariguda(v), Korremula 'x' Roads,  
Via Narapally, Ghatkesar(M), Medchal (Dist), Telanagan-500088  
2024-2025**



**NALLA NARASIMHA REDDY**  
Education Society's Group of Institutions - Integrated Campus  
(UGC AUTONOMOUS INSTITUTION)



**SCHOOL OF ENGINEERING**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

This is to certify that the project report titled “**BUILDING A REVIEW ANALYZER WITH NLP**” is being submitted by **T. Pranitha (217Z1A05H4), S. Sai Sridhar Naidu (217Z1A05G6), T. Madhava Swamy (217Z1A05H6)** in Partial fulfilment for the award of **Bachelor of Technology in Computer Science and Engineering** is a record bonafide work carried out by them. The results embodied in this report have not been submitted to any other University for the award of any degree.

**Internal Guide**

(Dr. G. Sravan Kumar)

**Head of the Department**

(Dr. K. Rameshwaraiah)

Submitted for Viva Voce Examination held on.....

**External Examiner**

## **DECLARATION**

We, T. Pranitha, S. Sai Sridhar Naidu and T. Madhava Swamy the students of **Bachelor of Technology in Computer Science and Engineering, Nalla Narasimha Reddy Education Society's Group of Institutions**, Hyderabad, Telangana, here by declare that the work presented in this project work entitled “**Building A Review Analyzer With NLP**” is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken by taking care of engineering ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning.

**By:**

T. PRANITHA	217Z1A05H4
S. SAI SRIDHAR NAIDU	217Z1A05G6
T. MADHAVA SWAMY	217Z1A05H6

**Date:**

**Signature:**

## ACKNOWLEDGEMENT

We express our sincere gratitude to our guide **Dr. G. Sravan Kumar**, Associate Professor, in Computer Science and Engineering Department, NNRESGI, who motivated throughout the period of the project for his valuable and intellectual suggestions apart from his adequate guidance, constant encouragement right throughout our work.

We profoundly express our sincere thanks to **Dr. K. Rameshwaraiah**, Professor & Head, Department of Computer Science and Engineering, NNRESGI, for his cooperation and encouragement in completing the project successfully.

We wish to express our sincere thanks to **Dr. G. Janardhana Raju**, Dean School of Engineering, NNRESGI, for providing the facilities for completion of the project.

We wish to express our sincere thanks to **Dr. C. V. Krishna Reddy**, Director NNRESGI for providing the facilities for completion of the project.

We wish to record our deep sense of gratitude to our Project In-charge **Mrs. Ch. Ramya**, Assistant Professor, in Computer Science and Engineering Department, NNRESGI, for giving her insight, advice provided during the review sessions and providing constant monitoring from time to time in completing my project and for giving us this opportunity to present the project work.

Finally, we would like to thank Project Co-Ordinator, Project Review Committee (PRC) members, all the faculty members and supporting staff, Department of Computer Science and Engineering, NNRESGI for extending their help in all circumstances.

**By:**

T. PRANITHA 217Z1A05H4

S. SAI SRIDHAR NAIDU 217Z1A05G6

T. MADHAVA SWAMY 217Z1A05H6

# ABSTRACT

In the digital era, customer reviews play a pivotal role in shaping the reputation and growth of businesses, particularly in the food and hospitality sectors. This project, titled "Building a Review Analyzer With NLP", aims to develop an intelligent system capable of analyzing restaurant reviews using Natural Language Processing (NLP) techniques. The system processes textual reviews submitted by users and automatically classifies them into positive, neutral, or negative sentiments. The primary objective is to provide businesses with real-time insights into customer feedback and to enhance decision-making by identifying service strengths and weaknesses.

The implementation leverages Python-based tools such as Flask for the web framework, scikit-learn for training a Naive Bayes classifier, and NLTK for natural language preprocessing. Reviews are cleaned, tokenized, and vectorized using a CountVectorizer with n-grams, after which the classifier predicts the sentiment. Additionally, a dynamic Plotly-based graph is generated to visualize sentiment distribution interactively. The model demonstrates high accuracy and efficiently handles real-time input, offering scalability and responsiveness for production use.

This system not only streamlines feedback management but also demonstrates the powerful synergy of machine learning and NLP in real-world applications. It provides a foundation for further enhancements, such as emotion analysis and multilingual support.

***Keywords: Customer Reviews , NLP , Sentiment Analysis , Flask , Naive Bayes , scikit-learn , Real-time Insights , Visualization.***

# TABLE OF CONTENTS

	Page No:
<b>Abstract</b>	
<b>List of Figures</b>	<b>i</b>
<b>List of Tables</b>	<b>ii</b>
<b>List of Abbreviations</b>	<b>iii</b>
<b>1. INTRODUCTION</b>	<b>1</b>
1.1 Motivation	1
1.2 Problem Definition	2
1.3 Objective of Project	2
1.4 Limitations of Project	3
1.5 Purpose	3
1.6 Scope	4
<b>2. LITERATURE SURVEY</b>	<b>5</b>
2.1 Introduction	5
2.2 Existing System	8
2.3 Proposed System	9
<b>3. SYSTEM ANALYSIS</b>	<b>11</b>
3.1 Functional Requirements	11
3.2 Non-Functional Requirements	12

3.3 System Requirements	13
<b>4. SYSTEM DESIGN</b>	<b>15</b>
4.1 Introduction	15
4.2 UML Diagrams	16
<b>5. IMPLEMENTATION AND RESULTS</b>	<b>22</b>
5.1 Introduction	22
5.2 Method of Implementation	28
5.3 Setup	29
5.4 Algorithm	33
5.5 Control flow of the implementation	31
5.6 Output Screens	35
<b>6. SYSTEM TESTING</b>	<b>38</b>
6.1 Introduction to Testing	38
6.2 Types of Tests Considered	38
6.3 Various Types of Test Case Scenarios	40
<b>7. CONCLUSION</b>	<b>43</b>
<b>8. FUTURE ENHANCEMENT</b>	<b>44</b>
<b>9. REFERENCES</b>	<b>45</b>
9.1 Journals	45
9.2 Books	45
9.3 Web Links	46

## LIST OF FIGURES

<b>Figure No.</b>	<b>Name of the Figure</b>	<b>Page No</b>
4.1	Data flow diagram	17
4.2	Use Case Diagram	18
4.3	Class Diagram	19
4.4	Sequence Diagram	20
4.5	Activity Diagram	21
5.1	Backend Page	35
5.2	Example 1	35
5.3	Example 2	36
5.4	Example 3	36



## **LIST OF TABLES**

<b>Table No.</b>	<b>Name of the Table</b>	<b>Page No</b>
3.1	Software Requirements	13
3.2	Hardware Requirements	14
6.1	Test Cases	40

## LIST OF ABBREVIATIONS

S. No	Abbreviation	Definition
1	BERT	Bidirectional Encoders Representation from Transformers
2	DFD	Data Flow Diagram
3	EDA	Exploratory Data Analysis
4	GPU	Graphics Processing Unit
5	IDE	Integrated Development Environment
6	NLP	Natural Language Processing
7	NLTK	Natural Language Toolkit
8	RAM	Random Access Memory
9	UML	Unified Modelling Language



# 1. INTRODUCTION

## 1.1 Motivation

In today's digitally connected world customers have more power than ever to voice their opinions through online platforms. Restaurant reviews, in particular, play a vital role in shaping a restaurant's reputation, attracting new customers, and influencing the decision-making of potential diners. With the growth of platforms like Yelp, Zomato, Google Reviews, and TripAdvisor, users are constantly sharing their dining experiences, whether good or bad. This has led to a surge in unstructured textual data, rich with insights but difficult to interpret manually on a large scale.

The motivation behind this project stems from the need to convert these large volumes of restaurant reviews into actionable intelligence. Traditional methods of gathering customer feedback—like paper surveys and direct interviews—are not only time-consuming but also limited in scope and depth. Meanwhile, online reviews offer a more spontaneous, honest, and detailed reflection of customer sentiment. However, manually reading through hundreds or thousands of reviews is both impractical and inefficient for restaurant managers or business analysts.

Natural Language Processing (NLP) presents an exciting opportunity to automate this analysis. By using NLP techniques, we can process large amounts of text, identify patterns in customer feedback, and classify the nature of reviews into understandable categories such as positive, neutral, or negative. This transformation from raw text to structured insight enables better strategic decisions, such as improving service quality, modifying menu items, identifying staff training needs, or recognizing operational issues.

Moreover, restaurant businesses can use this analysis to monitor customer perception in real-time, react promptly to issues, and ensure that they maintain a competitive edge in the industry. The project aims to bring these capabilities into a simple, accessible web-based application, enabling even small restaurant owners to benefit from cutting-edge NLP techniques without requiring technical expertise.

By focusing specifically on restaurant review analysis, this project addresses a highly relevant use case where automated textual intelligence has immediate and practical

value. The end goal is to empower restaurant stakeholders with a tool that translates customer voices into strategic business advantage.

## 1.2 Problem Definition

The hospitality industry, particularly the restaurant sector, relies heavily on customer satisfaction and public image. While reviews are freely available online, the major problem lies in the extraction of meaningful insights from vast amounts of unstructured review data. Manually interpreting hundreds of customer comments is inefficient, time-consuming, and prone to human bias. Moreover, without a standardized way to categorize and understand feedback, restaurants may miss critical patterns in customer sentiment or fail to detect recurring issue.

Therefore, the problem is “How can restaurant reviews be automatically analyzed and classified using Natural Language Processing techniques to assist restaurant stakeholders in understanding customer feedback efficiently and accurately?”

This project seeks to solve that problem by building a system that uses NLP to process, clean, and classify textual reviews. The challenge also includes ensuring accuracy in the classification of sentiments and preserving contextual meaning, especially when negations and mixed sentiments are present (e.g., “The food wasn’t great, but the service was excellent”).

## 1.3 Objective of Project

This project is driven by the following core objectives:

**To develop a reliable and efficient NLP-based model** capable of analyzing restaurant reviews and determining the underlying sentiment.

**To preprocess customer reviews intelligently**, handling language-specific nuances, such as negation words, irrelevant stopwords, and punctuation, to maintain semantic integrity.

**To use machine learning algorithms**—specifically the Multinomial Naive Bayes classifier—for accurately classifying textual data into sentiment categories.

**To build an intuitive web-based interface** for users to submit reviews and instantly get sentiment feedback along with a visual summary of sentiment distribution.

**To enhance decision-making for restaurants** by providing them with easy-to-digest sentiment summaries and insights extracted from customer feedback.

**To ensure modularity and extensibility** so the system can be improved in future iterations with more advanced models or additional features such as aspect-based sentiment analysis.

## 1.4 Limitations of Project

While the project delivers a functional and effective restaurant review analysis system, it is important to acknowledge several limitations :

**Dataset Specificity:** The system is trained on a specific dataset of English-language restaurant reviews. It may not perform optimally with reviews in other languages or in niche culinary contexts without further training.

**Ambiguity Handling:** Some reviews contain mixed sentiments (e.g., “The pizza was good, but the service was horrible”). The current implementation provides a single sentiment classification per review and does not perform phrase- or aspect-based sentiment analysis.

**Scalability:** The use of a local Naive Bayes classifier and limited vocabulary size (max 1500 features) may not scale well with extremely large or diverse datasets without retraining or switching to more advanced models like transformers.

**Contextual Depth:** Although negation handling is incorporated, deeper semantic understanding is limited. The classifier may struggle with sarcasm, idioms, or cultural references.

**Real-time Feedback Source:** The system currently requires manual text input. Integration with live review feeds from APIs (e.g., Yelp, Google Places) would require further development.

## 1.5 Purpose

The primary purpose of this project is to create an automated and intelligent system that can analyze textual restaurant reviews and classify them into sentiment categories—positive, neutral, or negative—using NLP techniques. By doing so, the system helps

restaurant owners, managers, and analysts understand what customers are saying about their services and offerings, without having to manually sift through every review. This tool not only provides a high-level sentiment overview but also allows deeper analysis into the kinds of issues or praises customers bring up repeatedly. From a business perspective, this information is invaluable for improving customer satisfaction, making data-driven decisions, and enhancing the overall dining experience.

## 1.6 Scope

The scope of this project includes the following:

**Textual Data Handling:** The system is designed to process textual reviews obtained from restaurant review platforms or in-house feedback systems.

**Preprocessing Capabilities:** It includes data cleaning, stopword removal, normalization, and contextual preservation (such as retaining negations).

**Feature Extraction:** A Bag-of-Words model with bi-gram features is implemented to extract relevant textual features from reviews.

**Classification Engine:** A Multinomial Naive Bayes classifier is trained on labeled review data to learn patterns and classify future inputs.

**User Interface:** A web-based interface is provided using Flask, allowing users to enter reviews and view real-time sentiment classification.

**Visualization Module:** Sentiment trends are visualized using dynamic graphs generated with Plotly, giving users an instant view of the feedback landscape.

The system focuses exclusively on restaurant reviews and is not intended for broader sentiment analysis use cases like product reviews, political opinions, or movie critiques—though the core architecture could be extended for those applications in the future.

## 2.LITERATURE SURVEY

### 2.1 Introduction

The field of Natural Language Processing (NLP) has significantly evolved in recent years, enabling the automatic understanding and processing of human language through computational methods. One of the most promising applications of NLP is in the area of **opinion mining and review analysis**, especially within consumer-facing industries like hospitality and food services. Restaurant review platforms such as Yelp, Zomato, and TripAdvisor have accumulated vast collections of user feedback in textual form, providing a rich dataset for understanding public opinion, service quality, and customer satisfaction trends.

Restaurant reviews are particularly valuable because they reflect spontaneous, real-time, and often emotionally driven expressions of user experience. These reviews contain hidden insights about various service aspects such as food quality, hygiene, ambiance, staff behavior, pricing, and overall satisfaction. However, due to their unstructured nature, extracting this information for actionable business insights poses a major challenge.

Recent advancements in machine learning and NLP, particularly in text classification and sentiment detection, have opened up possibilities for **automated restaurant review analysis systems**. Such systems aim to classify reviews as positive, negative, or neutral, helping restaurant businesses improve operations based on real customer experiences. Several studies have explored methods to preprocess text, build predictive models using classifiers like Naive Bayes, SVMs, and neural networks, and deploy systems that aid in decision-making using sentiment data.

This literature survey explores the current state-of-the-art techniques and systems that have contributed to the domain of review analysis, identifies the gaps in existing methodologies, and justifies the need for the development of a restaurant-specific review classification model using NLP.



## **Opinion mining and sentiment analysis**

**Authors: Bo Pang and Lillian Lee.**

An important part of our information-gathering behavior has always been to find out what other people think. With the growing availability and popularity of opinion-rich resources such as online review sites and personal blogs, new opportunities and challenges arise as people now can, and do, actively use information technologies to seek out and understand the opinions of others. The sudden eruption of activity in the area of opinion mining and sentiment analysis, which deals with the computational treatment of opinion, sentiment, and subjectivity in text, has thus occurred at least in part as a direct response to the surge of interest in new systems that deal directly with opinions as a first-class object.

This survey covers techniques and approaches that promise to directly enable opinion-oriented information-seeking systems. Our focus is on methods that seek to address the new challenges raised by sentiment-aware applications, as compared to those that are already present in more traditional fact-based analysis. We include material on summarization of evaluative text and on broader issues regarding privacy, manipulation, and economic impact that the development of opinion-oriented information-access services gives rise to. To facilitate future work, a discussion of available resources, benchmark datasets, and evaluation campaigns is also provided.

## **Sentiment analysis on Twitter.**

**Authors: Hassan Saif, Yulan He and Harith Alani**

Sentiment analysis on Twitter offers organizations a fast and effective way to monitor public opinion about their brand, business, leadership, and more. Over the years, various methods and features have been explored for training sentiment classifiers on Twitter data, each with mixed results.

In this paper, we present a new approach: adding semantic features to the training set for sentiment analysis. Specifically, for each identified entity in a tweet (e.g., “iPhone”), we also include its related semantic concept (e.g., “Apple product”) as an additional feature. We then assess how these semantic concepts correlate with positive or negative sentiments. We tested this approach on three different Twitter datasets. Our

results show that it improves sentiment prediction, with an average F1-score increase of around 6.5% over using unigrams and 48% over using part-of-speech features alone.

We also compared our method to a sentiment-based topic analysis approach. We found that semantic features achieved better recall and F1-scores for detecting negative sentiment, and better precision (but lower recall and F1-score) for positive sentiment.

## **Mining and summarizing customer reviews**

**Authors: Minqing Hu and Bing Liu**

Online merchants often ask customers to leave reviews about the products they've purchased and the related services. As e-commerce grows in popularity, the number of reviews for popular products increases rapidly—sometimes reaching hundreds or even thousands. This makes it difficult for potential buyers to read through all the reviews and make informed decisions. It's also challenging for manufacturers to monitor and manage customer feedback, especially since the same product may be sold by different sellers, and manufacturers often produce many different products.

In this research, we aim to automatically mine and summarize customer reviews for a product. This summarization is different from traditional text summarization. Instead of selecting or rewriting sentences from reviews to capture main points, we focus specifically on the product features that customers talk about and whether their opinions are positive or negative.

## **Sentiment analysis algorithms and applications**

**Authors: Walla Medhat , Ahmed Hassan and Hoda Korashy**

Sentiment Analysis (SA) is an active area of research within the field of text mining. It focuses on the computational understanding of opinions, emotions, and subjectivity in text. This survey paper provides a broad overview of the most recent developments in sentiment analysis.

It briefly explores recently proposed algorithms, improvements, and applications of sentiment analysis. The reviewed articles are grouped based on their contributions to various SA techniques. The paper also discusses related areas that have gained attention recently, such as transfer learning, emotion detection, and resource building.

The main goal of this survey is to give a comprehensive yet concise picture of sentiment analysis methods and their related fields. One of the key contributions is the organized categorization of a large number of recent papers, along with a summary of the latest research trends in sentiment analysis and associated domains.

## **2.2 Existing System**

Multiple research studies and commercial systems have attempted to automate review classification and sentiment analysis across various domains. However, when applied to the restaurant industry, many of these existing systems reveal certain limitations, either due to generalization or a lack of domain-specific tailoring.

### **1. Generic Sentiment Analysis Platforms**

Many tools such as VADER (Valence Aware Dictionary and sEntiment Reasoner), TextBlob, and commercial APIs like Google Cloud Natural Language or IBM Watson offer sentiment analysis as a generalized service. These models often rely on lexicons or pre-trained datasets that aren't specific to the restaurant or food industry. As a result, they might misinterpret key phrases or expressions commonly used in restaurant reviews (e.g., “The steak was rare” might be misclassified negatively if not interpreted in culinary context).

### **2. Social Media-Based Systems**

Some models focus on analyzing tweets, Facebook comments, or Instagram captions for general product or brand feedback. While social media analysis is useful, these reviews are typically short, informal, and not detailed enough to capture the multi-dimensional aspects of a restaurant visit (service, food, ambiance, etc.). Moreover, social media sentiment models often fail in contexts where reviews are longer and more nuanced.

### **3. E-Commerce Review Analyzers**

Platforms like Amazon or Flipkart have inspired numerous review classification systems, where customer feedback is classified as positive or negative. These models often rely on product-specific features, like quality, price, or usability. When transferred

to restaurant reviews, such systems may struggle with expressions that involve complex experiences (e.g., “Food was great but the wait time was unbearable”).

#### **4. Academic Research**

Several academic papers have proposed models using classical machine learning algorithms such as Logistic Regression, Support Vector Machines (SVM), and Naive Bayes for sentiment classification. For example, a study by Pang and Lee (2002) laid the groundwork for binary sentiment classification using movie reviews, while Liu et al. (2012) introduced aspect-based sentiment analysis techniques for product reviews.

However, few studies have focused specifically on restaurant reviews, and even fewer have implemented complete end-to-end systems that include preprocessing, classification, and web deployment using NLP tailored to restaurant-related feedback.

### **2.3 Proposed System**

To bridge the gap in existing systems, the proposed solution focuses specifically on **analyzing restaurant reviews using NLP and supervised machine learning techniques**, tailored for the unique vocabulary, expressions, and sentiments found in the food and hospitality domain.

#### **1. Domain-Specific Preprocessing**

Unlike generic sentiment analyzers, this system incorporates restaurant-specific preprocessing rules. For instance, common stopwords are removed, but words like "not", which affect sentiment polarity, are retained. Slang, emojis, and punctuation are handled appropriately to preserve the context of informal feedback commonly found in user reviews.

#### **2. Bi-Gram Based Feature Extraction**

To capture context better than simple unigrams, the system utilizes **bi-gram tokenization** via CountVectorizer. This allows it to learn from phrases such as “not tasty”, “highly recommend”, or “very slow”, which are meaningful only when considered as a whole.

### 3. Multinomial Naive Bayes Classifier

The project employs a Multinomial Naive Bayes classifier, which is highly effective for text-based classification tasks, especially with discrete word frequency features. Naive Bayes is chosen due to its simplicity, fast training time, and robustness in high-dimensional feature spaces.

### 4. Web Deployment via Flask

The system is designed as a web application using Flask, making it easy for end-users to interact with. Users can input reviews through a clean interface, view immediate feedback, and observe live sentiment distributions through dynamic graphs rendered using Plotly.

### 5. Live Sentiment Distribution Monitoring

Each new review updates the global sentiment count and displays it in a bar graph. This feature is particularly useful for businesses wanting to monitor review trends during specific time frames, such as after launching a new menu item or during seasonal promotions.

### 6. Practical Use Case Alignment

By narrowing the focus specifically to restaurant review data, the system ensures that the NLP model is not only more accurate but also more **business-actionable**. It provides insights into what customers liked or disliked most—be it food, pricing, ambiance, or service—which is more informative than a generic sentiment classification.

## 3.SYSTEM ANALYSIS

### 3.1 Functional Requirements

Functional requirements define the essential operations and behaviors the system must perform. In the context of the Restaurant Review Analysis system, the following are the key functional requirements:

#### 1. User Input Interface

The system should provide a user-friendly interface through which users can submit a restaurant review. This form must support free-text input to ensure flexibility and simulate actual review submission behavior.

#### 2. Review Preprocessing

Before any analysis, the system should clean the raw text input. This includes converting to lowercase, removing special characters and numbers, eliminating stopwords (except for critical words like "not"), and tokenizing the text.

#### 3. Sentiment Classification

The system must classify each review into a sentiment category—**positive**, **negative**, or **neutral**—based on the pre-trained model using Multinomial Naive Bayes.

#### 4. Probability Score Calculation

Apart from a hard classification, the model also computes the **probability scores** for the input review, which helps in interpreting the strength of the sentiment.

#### 5. Dynamic Graphical Representation

Each classified review should dynamically update a live sentiment distribution chart. This bar chart displays counts for each sentiment type and gives users a real-time understanding of feedback trends.

## **6. Model Training**

The system should support training a sentiment classification model using labeled data. This involves vectorizing the text (using CountVectorizer with n-grams), training the classifier, and saving it using joblib for future use.

## **3.2 Non-Functional Requirements**

These requirements specify criteria that judge the system's operation, rather than specific behaviors.

### **1. Performance**

The system should respond to review analysis requests in under 2 seconds for average-length reviews. This ensures a smooth user experience.

### **2. Scalability**

Though initially built as a lightweight application, the system design must allow scaling to handle hundreds or thousands of reviews in batch mode, possibly by transitioning to asynchronous processing in future versions.

### **3. Security**

Input sanitization is critical to avoid script injection or malicious content. Flask's templating engine should be leveraged safely, and all user inputs should be validated server-side.

### **4. Reliability and Availability**

The system should be stable and consistently available for end-users. Proper error handling must be implemented to manage issues like empty inputs, file access errors, or corrupt model files.

### **5. Portability**

The application should be deployable on any system supporting Python and Flask. This includes compatibility with Linux, Windows, or cloud platforms like Heroku or AWS.

## 6. Maintainability

Code modularity and documentation are essential. Developers should easily be able to upgrade models, preprocessors, or UI components without affecting the entire system.

## 7. Usability

The user interface should be intuitive, especially for users unfamiliar with NLP or sentiment analysis. The inclusion of sentiment labels and graphical representation makes the system accessible even to non-technical stakeholders.

### 3.3 System Requirements

This section outlines the hardware and software needed for developing, running, and deploying the Restaurant Review Analysis system.

*Table:3.1 Hardware Requirements*

Component	Specification
Processor	Intel i5 or higher
RAM	Minimum 4 GB (8 GB recommended)
Storage	Minimum 500 MB free space
GPU (Optional)	For large-scale retraining only

*Table:3.2 Software Requirements*

Component	Version / Description
Operating System	Windows 10 / Linux / macOS



Python	3.7 or above
Flask	Web framework for Python
Pandas, NumPy	Data handling libraries
scikit-learn	Machine learning models (Naive Bayes)
nltk	Text preprocessing and stopword tools
Plotly	Graph rendering library
joblib	For model serialization
Browser	Chrome, Firefox, or any modern browser

## 4. SYSTEM DESIGN

### 4.1 Introduction

The system design section outlines how the *Text Emotion Detection using BERT* project has been planned and structured to meet its goals. This part of the project focuses on how different components like the user interface, the machine learning model, and the visualization tools work together to form a complete, functioning system.

The main idea is to let users type any piece of text into a simple interface and quickly get feedback on the emotions detected in that text. To make this possible, the system combines a friendly front-end built with Streamlit and a powerful back-end that uses a BERT-based deep learning model. This model, managed through the ktrain library, is trained to recognize emotions like joy, anger, sadness, and more.

The design also includes features like emoji representation, confidence scores, and bar graph visualizations that make the results easy to understand and engaging. The entire system is built in a modular way so that each part (like text input, emotion prediction, and result display) can be updated or improved separately in the future. The goal of this design is to make the tool efficient, easy to use, and flexible for future upgrades such as support for different languages or more emotions.

This design approach ensures a smooth flow of data from the moment the user enters text to the final display of emotion results. The system first captures the user's input, processes it through the pre-trained BERT model in the backend, and then returns both the predicted emotion and its corresponding confidence score. To make the results more interactive and understandable, the system also displays an emoji that matches the detected emotion and a bar chart showing the probability of each possible emotion. This helps users not only see the result but also understand how confident the model is about its prediction. By combining deep learning with a clear and responsive interface, the system design aims to offer an engaging user experience while maintaining technical reliability and performance.

## **4.2 UML Diagrams**

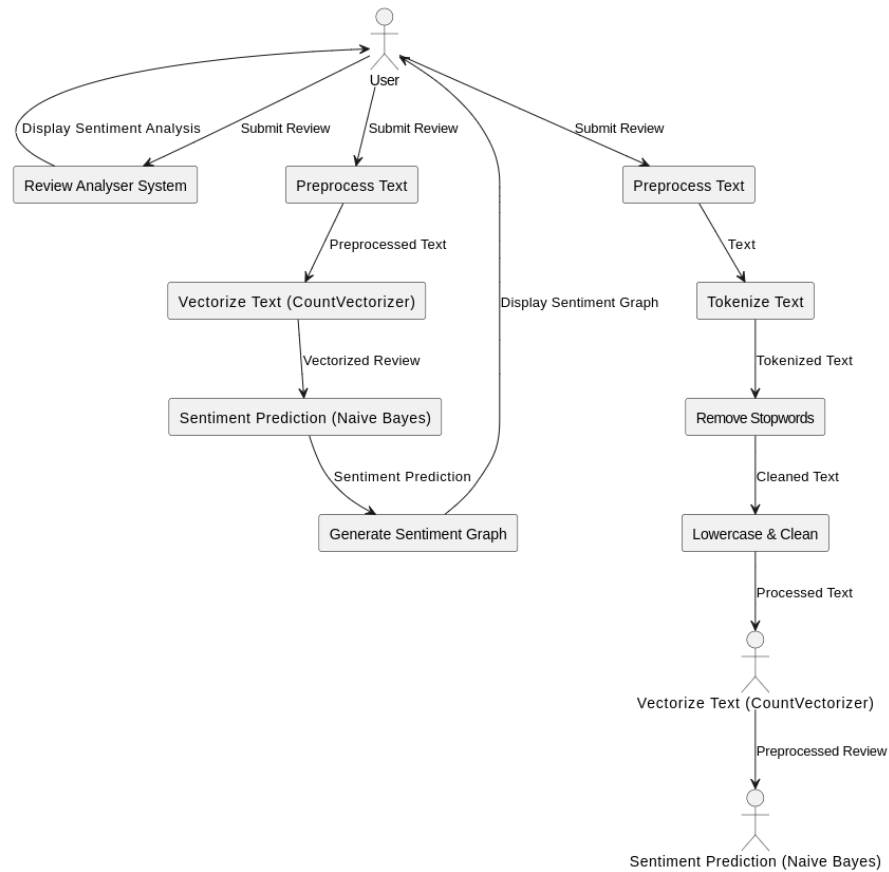
UML, which stands for Unified Modelling Language, is a way to visually represent the architecture, design, and implementation of complex software systems. UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

### **4.2.1 Data Flow Diagram**

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. There are 2 levels in the below diagram.

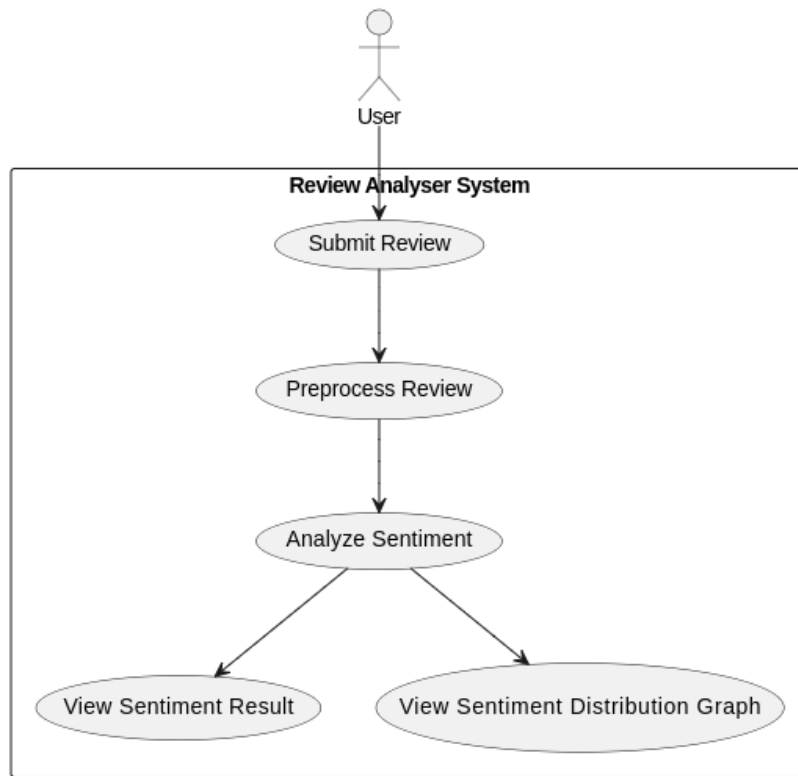


**Fig 4.1: Data flow diagram**

## 4.2.2 Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

Use Case Diagram - Building a Review Analyser Using NLP

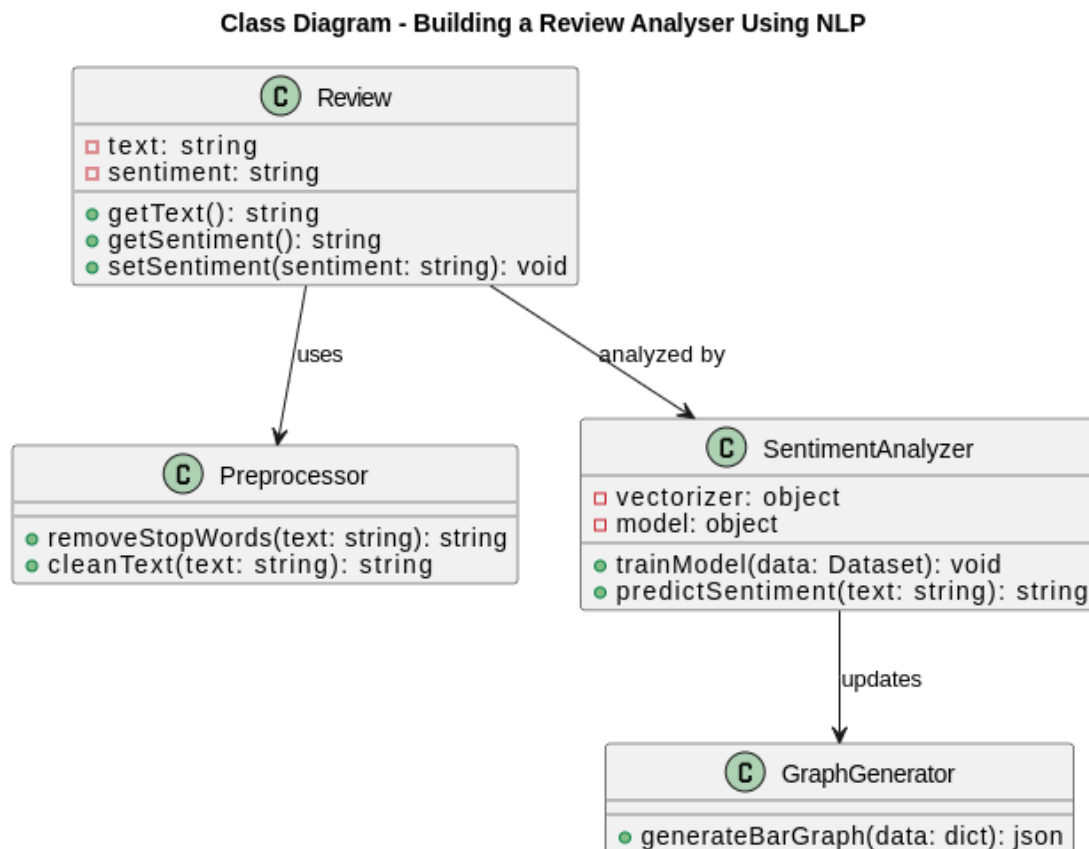


*Fig 4.2: Use case diagram*

The use case diagram provides a high-level view of the system from the perspective of end users (actors). For this project, the primary actor is the **User**, who interacts with the system to analyze restaurant reviews. The key use cases include: "Submit Review," "View Sentiment Result," and "View Sentiment Graph." Optional actors such as an **Admin** could be added to manage feedback or perform system monitoring in advanced versions. The use case diagram clarifies the functionalities the system offers and the boundaries of user interaction. It also helps in identifying dependencies between different user actions and how the system should respond to them. For instance, the "Submit Review" use case would trigger the NLP pipeline, and the "View Sentiment Result" would show whether the input was positive, neutral, or negative. This diagram is crucial for requirement gathering and communicating with non-technical stakeholders since it abstracts implementation details and focuses on what the system should do.

### 4.2.3 Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

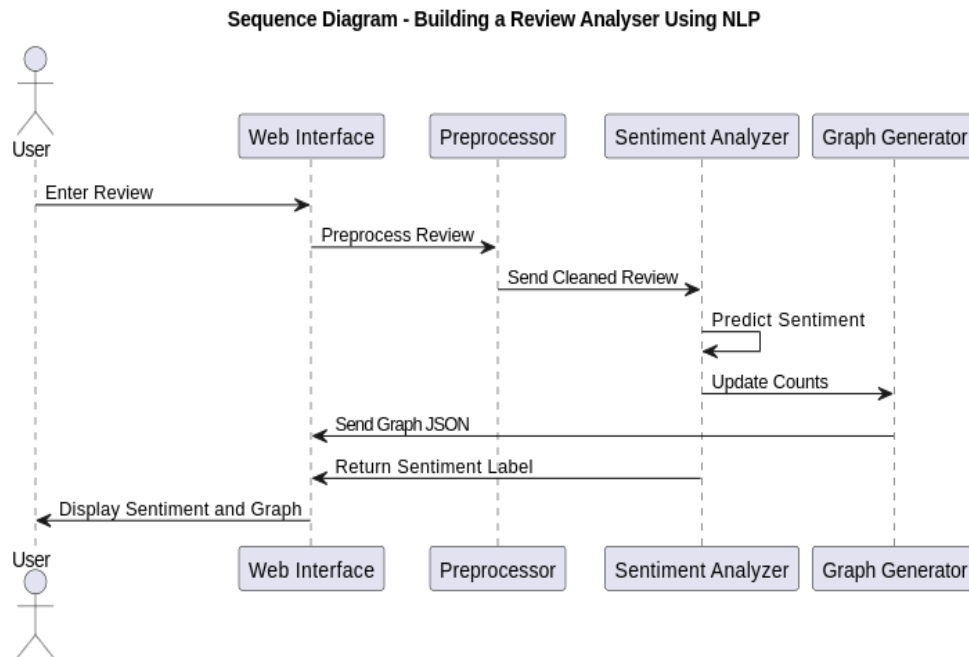


*Fig 4.3: Class diagram*

The class diagram represents the static structure of the system, highlighting classes, their attributes, methods, and relationships. In the review analyser system, key classes might include **ReviewAnalyzer**, **Preprocessor**, **SentimentModel**, and **GraphGenerator**. The **ReviewAnalyzer** class orchestrates the flow between preprocessing, classification, and result generation. The **Preprocessor** class handles tasks like cleaning the text, removing stopwords, and tokenizing. The **SentimentModel** class wraps the trained Naïve Bayes model and provides prediction methods. The **GraphGenerator** class manages the creation and encoding of sentiment graphs using Plotly. The class diagram also illustrates relationships like composition (e.g., **ReviewAnalyzer** uses **Preprocessor**)

and associations (e.g., ReviewAnalyzer accesses GraphGenerator). This diagram is vital for developers to understand code organization, promote reuse, and ensure maintainability and scalability of the system.

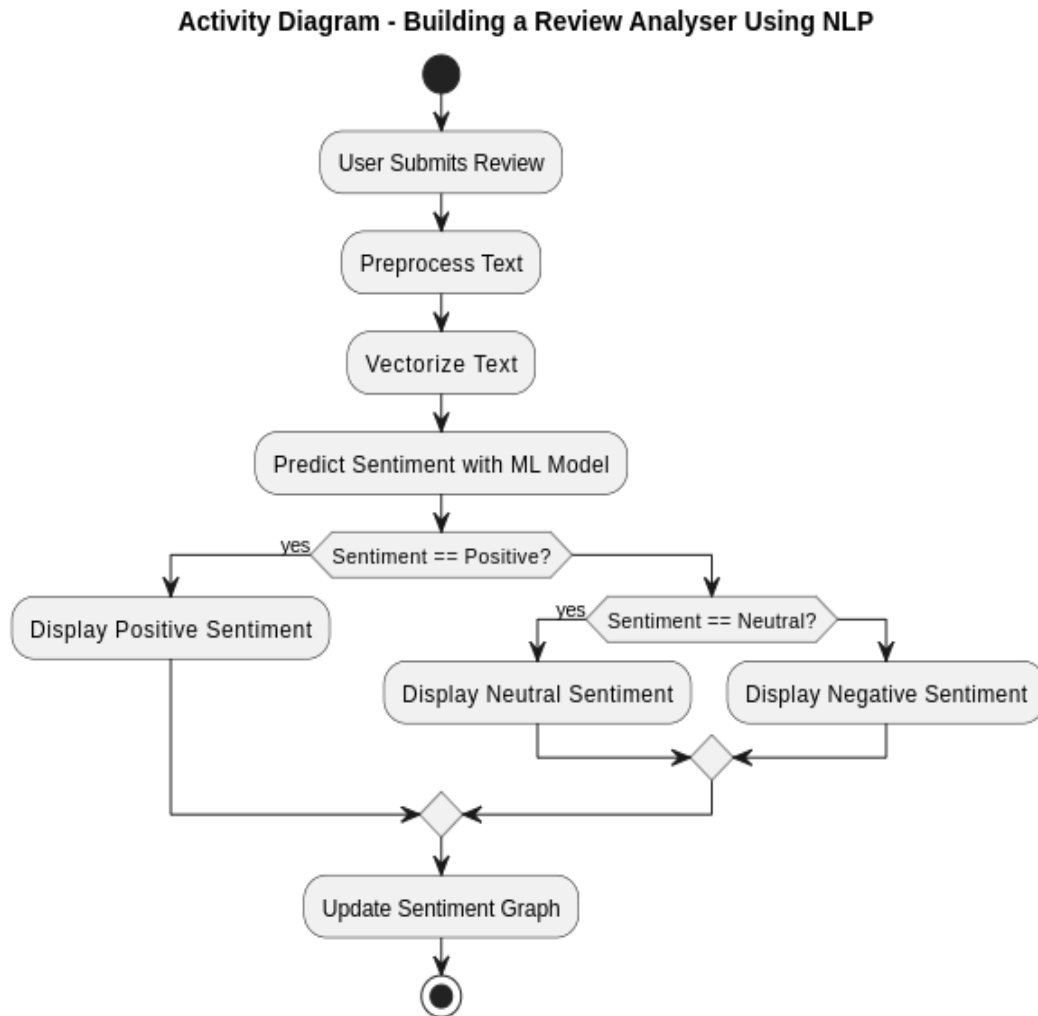
#### 4.2.4 Sequence Diagram



*Fig 4.4: Sequence diagram*

The sequence diagram models the time-ordered interactions between system components or objects. For the review analyser, the diagram starts with the User sending a request through the frontend interface. This request is forwarded to the Flask Backend, which then calls the Preprocessor to clean the input text. Next, the SentimentModel component performs vectorization and classification. Based on the result, the GraphGenerator updates the bar chart showing sentiment counts. Finally, the response is returned to the Frontend and displayed to the User. This diagram provides clarity on message flows, method calls, and the order of execution.

## 4.2.5 Activity Diagram



*Fig 4.5: Activity diagram*

The activity diagram visualizes the dynamic behavior of the system by modeling the flow of control from one activity to another. In this project, it starts with the user entering a restaurant review. The input undergoes preprocessing (text cleaning and stopword removal), followed by feature extraction via vectorization. Then, the classifier predicts the sentiment score, which is interpreted into a categorical sentiment. The activity concludes with the result and dynamic graph being returned to the frontend. Decision nodes are used to evaluate the sentiment probability, leading to branches for "Positive," "Neutral," or "Negative" classification. The diagram ensures all possible paths and conditions are covered, which helps during testing and validation phases.



## 5. IMPLEMENTATION AND RESULTS

The implementation phase is where the conceptual system design is transformed into an actual working software product. In this restaurant review analysis project, the implementation is done using **Python with Flask** for the web backend, integrated with machine learning models powered by **Natural Language Processing (NLP)** techniques. The focus is to receive user-submitted reviews, process them using NLP, classify their sentiment using a trained model, and visualize the sentiment distribution.

### 5.1 Introduction

Python was selected as the primary programming language due to its simplicity, versatility, and vast ecosystem of libraries that support machine learning, deep learning, and web application development. A modular architecture was followed, allowing each component to be developed, tested, and refined independently before being seamlessly merged into the final application.

#### **What is Python:**

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.

Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber... etc.

The biggest strength of Python is huge collection of standard library which can be used for the following –

- ◆ Machine Learning
- ◆ GUI Applications (like Kivy, Tkinter, PyQt etc. )
- ◆ Web frameworks like Django (used by YouTube, Instagram, Dropbox) Image processing (like Opencv, Pillow)
- ◆ Web scraping (like Scrapy, BeautifulSoup, Selenium)

## Python:

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

**Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

**Python is Interactive** – you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

## What is Machine Learning :

Before we take a look at the details of various machine learning methods, let's start by looking at what machine learning is, and what it isn't. Machine learning is often categorized as a subfield of artificial intelligence, but I find that categorization can often be misleading at first brush. The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of building models of data.

Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models tunable parameters that can be adapted to observed data; in this way the program can be considered to be "learning" from the data. Once these models have been fit to previously seen data, they

can be used to predict and understand aspects of newly observed data. I'll leave to the reader the more philosophical digression regarding the extent to which this type of mathematical, model-based "learning" is similar to the "learning" exhibited by the human brain.

Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

### **NLTK:**

The Natural Language Toolkit, abbreviated as **NLTK**, is one of the most widely used open-source libraries in the field of natural language processing (NLP). Developed initially by Steven Bird and Edward Loper at the University of Pennsylvania, NLTK provides a rich set of text-processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. Its broad and modular architecture makes it suitable for academic and industrial purposes, especially in early-stage NLP research and prototyping.

NLTK is implemented in Python and provides interfaces to over **50 corpora and lexical resources**, including WordNet, stopword lists, and text collections. These corpora can be downloaded using built-in functions and can be used to experiment with language data. NLTK also supports syntactic processing, sentiment analysis, and document classification through various algorithms and tokenizers, making it a powerful tool for beginners and experts alike.

One of the standout features of NLTK is its **stopwords corpus**, which consists of frequently used words that typically do not carry significant meaning for NLP tasks. In the context of sentiment or review analysis, stopwords like “the,” “is,” “in,” etc., are removed to reduce noise and enhance the model's performance. However, words like “not,” which are usually part of stopword lists, must be retained in sentiment analysis tasks because they can invert the meaning of a sentence. This customization is simple in NLTK using its flexible stopword handling methods.

NLTK also supports **tokenization**, which is the process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens. This is particularly useful for text preprocessing in review analysis, where each review needs to be broken down into words for the model to interpret. The `word_tokenize()` function and

regular expression-based methods provide robust mechanisms to achieve accurate tokenization.

In addition, **NLTK's support for regular expressions** allows the developer to clean text by removing punctuation, non-alphabetic characters, and digits. For example, in your project, the text is cleaned using `re.sub('[^a-zA-Z]', '', text)` to retain only alphabetic characters, which is a crucial step in data sanitization before feeding data into machine learning models.

Another important feature NLTK offers is **lemmatization and stemming**, which reduce words to their root forms. While stemming might reduce words crudely (e.g., “running” to “run”), lemmatization goes a step further by using vocabulary and morphological analysis to remove inflectional endings. Although not implemented in this particular project, these tools are vital in many NLP applications where word normalization is essential.

NLTK also integrates well with machine learning pipelines, especially for preparing data to be fed into classifiers like Naive Bayes. While the library itself does not include classifiers as powerful as those in Scikit-learn, it supports label encoding, corpus preparation, and other preprocessing tasks that make training models much more effective.

In summary, NLTK serves as the backbone of text preprocessing in the project. It provides a highly efficient method to tokenize, filter, clean, and normalize the reviews, ensuring that the machine learning pipeline receives structured and relevant input. The flexibility of its components, ease of integration, and vast repository of resources make NLTK an indispensable library for anyone working in the domain of text analytics and natural language processing.

## **Frontend Interface Design:**

The frontend of the system is kept intentionally minimalistic and user-friendly to ensure ease of use for all types of users. Built using HTML5 and styled with CSS3, the user interface includes a simple text input field where users can enter their restaurant reviews. A submit button is provided to analyze the review, and the result is displayed alongside a dynamic graph representing sentiment distribution. The interface is connected to the backend Flask routes, which handle the business logic and model predictions. The

use of templates in Flask (`render_template`) enables seamless integration between the frontend and backend components.

### **Backend with Flask:**

The core logic of the system is implemented in Python using the Flask web framework. Flask acts as the intermediary between the frontend and the machine learning model. When a user submits a review, a POST request is sent to a specific endpoint (i.e., `/analyze`), which triggers the sentiment analysis pipeline. Flask also includes other routes such as the home page route (`/`) and the accuracy endpoint (`/accuracy`) which provides the model's evaluation metrics.

The Flask app is initialized with CORS (Cross-Origin Resource Sharing) enabled using the `flask_cors` module to ensure that requests from different origins (such as a React frontend or other clients) can be accepted without security issues. The Flask server listens on port 5000 in debug mode, allowing for dynamic code updates during development.

### **Data Preprocessing and NLP:**

Preprocessing is a vital step in any NLP-based application as the quality and cleanliness of the data heavily influence the model's performance. For this project, the input text is processed using custom-defined preprocessing functions that involve the following steps:

**Cleaning the Text:** All non-alphabetic characters are removed using regular expressions. This eliminates punctuation, numbers, and special characters that do not contribute to sentiment analysis.

**Lowercasing:** The text is converted to lowercase to maintain uniformity and reduce dimensionality.

**Tokenization:** The cleaned text is then split into individual words (tokens).

**Stopword Removal:** Common English stopwords such as “the,” “is,” and “on” are removed. However, “not” is retained intentionally because it can reverse sentiment in phrases like “not good.”

**Rejoining:** The remaining words are rejoined into a processed string, which is ready to be vectorized.

This preprocessing pipeline ensures that the text data passed into the machine learning model is both noise-free and relevant for sentiment detection.

### **Feature Extraction using CountVectorizer:**

To feed the cleaned review text into the classifier, it must be transformed into a numerical format. This is achieved using the `CountVectorizer` from `Scikit-learn`. The `CountVectorizer` transforms text data into a matrix of token counts, also known as a bag-of-words representation. In this project, the vectorizer is configured with `max_features=1500` to include only the most frequent words and `ngram_range=(1,2)` to capture both unigrams and bigrams. This dual n-gram configuration allows the model to consider word sequences like "not tasty" or "very bad," which convey more accurate sentiment than single words in isolation.

Once fitted on the training data, the vectorizer transforms incoming user reviews into the same feature space for prediction. The fitted vectorizer is serialized using `joblib` and stored as `vectorizer.pkl`, which is later loaded during prediction.

### **Model Training: Multinomial Naive Bayes:**

The core classification model used in this project is the **Multinomial Naive Bayes** classifier. It is well-suited for text classification problems, especially when the input features represent frequencies or count data. The training data consists of a labeled dataset — `Restaurant_Reviews.tsv` — where each review is associated with a binary label (0 for negative, 1 for positive). After preprocessing, the reviews are vectorized, and the dataset is split into training and testing sets using an 80-20 ratio.

The model is trained on the training set and evaluated on the test set using accuracy as the primary metric. The achieved accuracy is then displayed via the `/accuracy` endpoint. Once trained, the model is saved using `joblib` as `naive_bayes_model.pkl` for later use during real-time predictions.

### **Sentiment Classification Logic:**

When a user submits a review, the backend loads both the serialized vectorizer and trained model. The review is processed using the same cleaning pipeline, vectorized, and passed into the classifier for prediction. The classifier outputs the probabilities for each class. The probability score for the positive class is extracted, and based on the score, the sentiment is categorized using custom thresholds.

$\geq 0.6$ : Positive

$0.4$  to  $< 0.6$ : Neutral

$< 0.4$ : Negative

This allows for more nuanced classification compared to hard binary labels and accommodates borderline sentiments.

### **Real-Time Graph Generation with Plotly:**

One of the most interactive and insightful features of the system is the sentiment distribution graph generated using Plotly. As users submit reviews, the sentiment counters for positive, neutral, and negative reviews are incremented and stored as global variables. These values are passed into a bar chart created with Plotly's `go.Bar` API. The figure layout includes labeled axes and a descriptive title.

To integrate the graph with the frontend, it is converted to JSON format using `plotly.utils.PlotlyJSONEncoder` and passed back in the JSON response. On the frontend, the graph can be rendered using JavaScript libraries or embedded with tools like Plotly.js.

### **Handling Errors and Edge Cases:**

Robust error handling is integrated into the system to manage scenarios such as:

1. Empty or null review submissions.
2. Non-English characters.
3. Unfitted vectorizer or model loading failures.
4. Server timeout issues.

Error messages are returned in JSON format with appropriate HTTP response codes for easy frontend parsing and debugging.

## **5.2 Method of Implementation**

The implementation of the project "Building a Review Analyser Using NLP" involves the integration of multiple components, ranging from the frontend user interface to backend model training, prediction logic, and real-time graphical sentiment visualization. The objective is to build a system that takes user-submitted restaurant reviews and accurately classifies them into three primary sentiment categories: positive,

neutral, or negative. The implementation is centered around Python's Flask web framework, Natural Language Processing (NLP) techniques, and a supervised learning algorithm — specifically, the Multinomial Naive Bayes classifier. Below is a detailed explanation of each component and their roles in the system.

### 5.3 Setup

#### **data\_preprocessing.p**

```
from flask import Flask, request, jsonify, render_template

from flask_cors import CORS

import joblib

import plotly.graph_objs as go

import json

import re

from nltk.corpus import stopwords

import nltk
```

#### **Load model and vectorizer**

```
vectorizer = joblib.load('vectorizer.pkl')

classifier = joblib.load('naive_bayes_model.pkl')
```

This part loads the pre-trained sentiment analysis model and its corresponding vectorizer from disk. It ensures the system is ready for real-time sentiment classification without retraining.

#### **Initialize sentiment counters**

```
positive_count = 0

neutral_count = 0
```



```
negative_count = 0
```

Global variables to track how many reviews fall into each sentiment category. These are incremented after each analysis and are used to update the sentiment distribution graph.

### **preprocessing**

```
def preprocess_text(text):
```

```
    stop_words = set(stopwords.words('english'))
```

```
    stop_words.discard('not')
```

```
    text = re.sub('[^a-zA-Z]', ' ', text).lower()
```

```
    text = ' '.join([word for word in text.split() if word not in stop_words])
```

```
    return text
```

This function cleans raw input text by removing non-alphabet characters, converting to lowercase, and removing common stopwords (except "not"). It ensures consistent, meaningful text is passed to the vectorizer for analysis.

### **Classify sentiment based on score**

```
def classify_sentiment(score):
```

```
    if score >= 0.6:
```

```
        return "positive"
```

```
    elif score >= 0.4:
```

```
        return "neutral"
```

```
    else:
```

```
        return "negative"
```

This utility function maps the classifier's probability output to a human-readable sentiment label. It uses defined thresholds:  $\geq 0.6$  is positive, 0.4–0.6 is neutral, and  $< 0.4$  is negative.

## Generate bar graph

```
def generate_graph():
```

```
    data = [go.Bar(x=['positive', 'neutral', 'negative'], y=[positive_count, neutral_count, negative_count])]
```

```
    layout = go.Layout(title='Sentiment Distribution', xaxis=dict(title='Sentiment'), yaxis=dict(title='Count'))
```

```
    fig = go.Figure(data=data, layout=layout)
```

```
    return json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)
```

Creates a bar chart using Plotly to visualize the number of positive, neutral, and negative reviews. It converts the graph into JSON so it can be rendered on the frontend dynamically after each analysis.

## app.py (Flask Application)

**Purpose:** Provide a web API for review analysis.

```
from flask import Flask, request, jsonify, render_template
```

```
from flask_cors import CORS
```

```
import joblib
```

```
from data_preprocessing import clean_text
```

```
app = Flask(__name__)
```

```
CORS(app)
```

```
vectorizer = joblib.load('vectorizer.pkl')
```

```
classifier = joblib.load('naive_bayes_model.pkl')
```

```
positive_count, neutral_count, negative_count = 0, 0, 0
```

```
def classify_sentiment(score):
```

```
    if score >= 0.6:
```

```

        return "positive"

    elif score >= 0.4:

        return "neutral"

    else:

        return "negative"

@app.route('/')

def home():

    return render_template('index.html')

@app.route('/analyze', methods=['POST'])

def analyze():

    global positive_count, neutral_count, negative_count

    review = request.json.get("review", "")

    processed = clean_text(review)

    review_vector = vectorizer.transform([processed]).toarray()

    score = classifier.predict_proba(review_vector)[0][1]

    sentiment = classify_sentiment(score)

    if sentiment == "positive": positive_count += 1

    elif sentiment == "neutral": neutral_count += 1

    else: negative_count += 1

    return jsonify({"review": review, "sentiment": sentiment})

@app.route('/accuracy')

def get_accuracy():

```

```
    return jsonify({"message": "Use training script to see accuracy."})

if __name__ == '__main__':

    app.run(debug=True)
```

The `app.py` file is the main **Flask application** that connects the machine learning models to a web interface so users can interact with them easily. It acts as the **backend server** for the project.

Inside `app.py`, there are routes set up: the home route (`/`) loads the main webpage, and the `/analyze` route handles user-submitted reviews. When a user submits a review, `app.py` preprocesses the text, uses the trained Naive Bayes model to predict sentiment (positive, neutral, or negative), and returns the result along with a dynamically generated sentiment distribution graph.

The application also provides an `/accuracy` route to display the model's accuracy. Besides handling requests and responses, `app.py` manages loading the saved models (`vectorizer.pkl` and `naive_bayes_model.pkl`), counting how many positive, negative, and neutral reviews have been analyzed, and creating graphs using Plotly.

In short, `app.py` acts like a **bridge** between users, the machine learning models, and the web browser — making the sentiment analysis system interactive, real-time, and user-friendly.

## 5.4 Algorithm

The algorithm used in the above code can be broken down into two main components:

The algorithm used in the above code involves two primary components: the frontend and the backend. On the frontend, the user begins by entering a review into a text area and submits the form. The review text is sent via a POST request to the Flask backend using JavaScript's `fetch()` API. Once the backend receives the review, it processes the text by passing it through the sentiment analysis models, such as the deep learning model or Naive Bayes classifier. After processing, the sentiment (positive, negative, or neutral) is predicted, and a sentiment distribution graph is generated using Plotly.js.

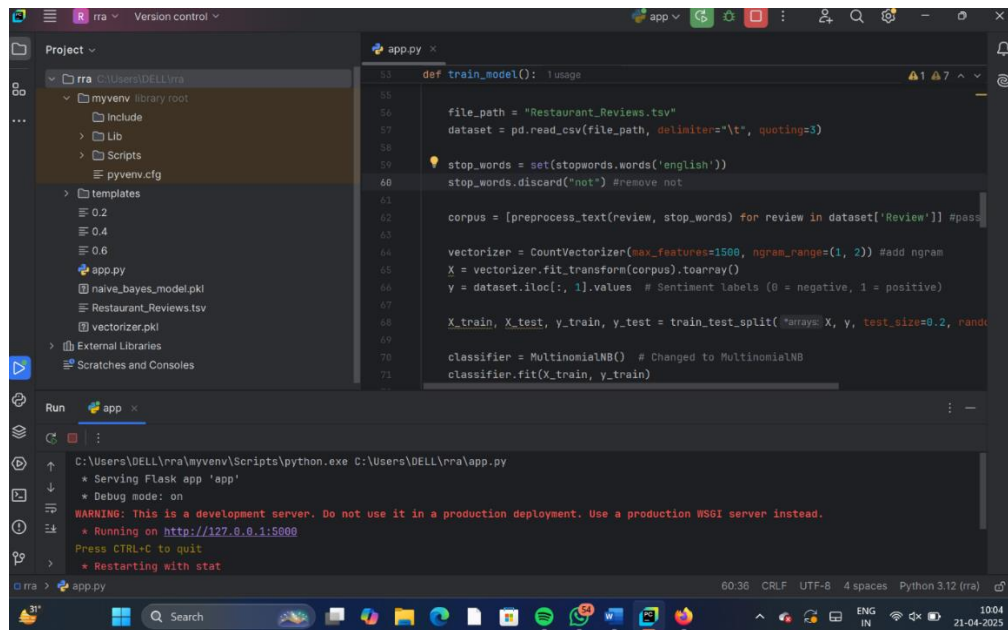
In the response, the backend sends both the predicted sentiment label and the graph data back to the frontend as a JSON object. The frontend dynamically updates to display the sentiment result along with the generated graph. Additionally, the frontend includes a theme toggle button that allows users to switch between light and dark themes by adding or removing a CSS class. The background features a particle animation effect, enhancing the visual appeal using `tsParticles.js`. This entire process ensures that the sentiment of the review is accurately predicted, visualized, and presented in a user-friendly way.

## 5.5 Control Flow of the Implementation

The control flow of the sentiment analysis web application begins when the user visits the page and is presented with a form to input a review. Upon entering the review text and submitting the form, the frontend JavaScript intercepts the form submission to prevent a page reload. Instead, it sends the review text to the Flask backend through an `AJAX fetch()` request to the `/analyze` endpoint, formatted as JSON. On the backend, Flask processes the review text by passing it through a sentiment analysis model, which predicts the sentiment (e.g., positive, negative, or neutral). Additionally, the model generates sentiment-related data for visualization, such as a sentiment distribution graph. The backend then responds with a JSON object containing the predicted sentiment label and the necessary data for plotting the graph.

Once the frontend receives the response, the JavaScript updates the UI accordingly. The predicted sentiment is displayed in the form of a label, with the text color reflecting the sentiment (e.g., green for positive, red for negative). A sentiment graph is rendered using `Plotly.js`, providing a visual representation of the sentiment distribution. The user can also toggle between light and dark themes by clicking the "Toggle Theme" button, which changes the visual appearance of the page. Throughout the interaction, the particle effect background remains active, adding a dynamic and engaging visual element. This cycle continues as the user submits new reviews for sentiment analysis, with the UI being updated each time without requiring a page reload.

## 5.6 Output Screens



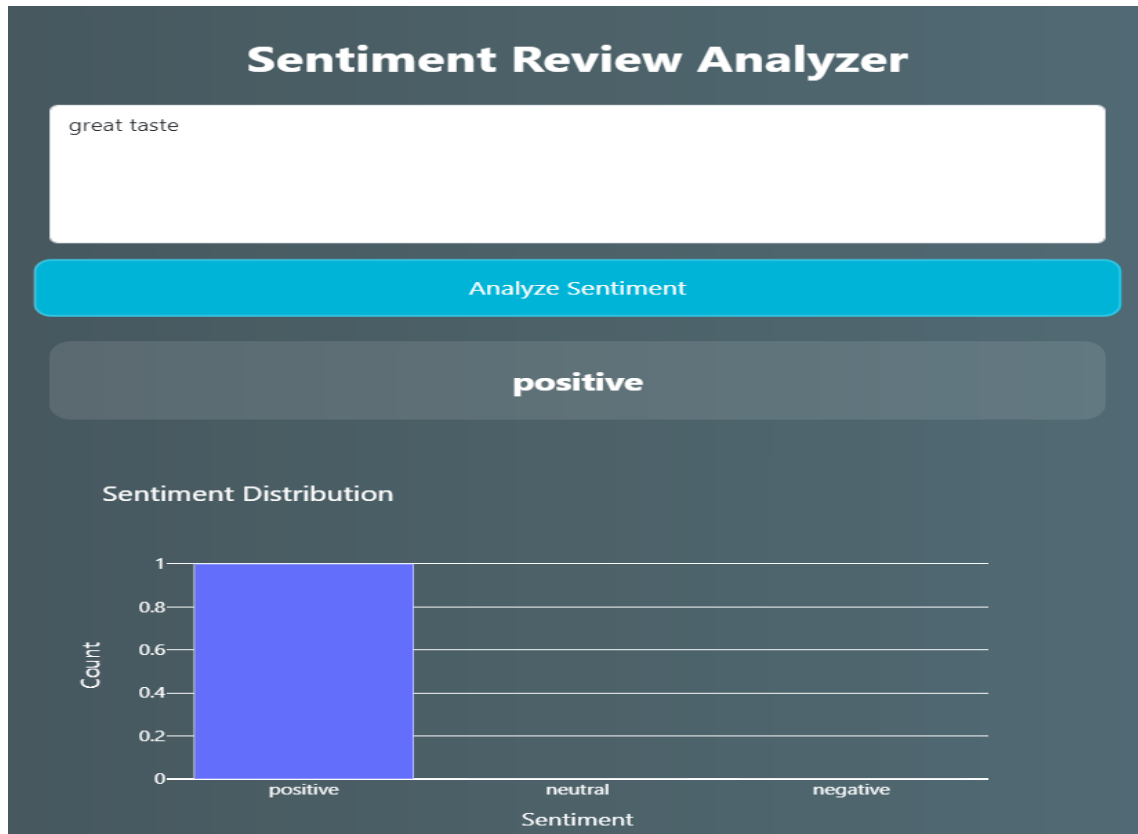
The screenshot displays a JupyterLab environment. The left sidebar shows a project structure with files like `myenv`, `Scripts`, `pyenv.cfg`, `templates`, `0.2`, `0.4`, `0.6`, `app.py`, `naive_bayes_model.pkl`, `Restaurant_Reviews.tsv`, and `vectorizer.pkl`. The main editor shows the `app.py` file with the following code:

```
def train_model():  
    file_path = "Restaurant_Reviews.tsv"  
    dataset = pd.read_csv(file_path, delimiter='\t', quoting=3)  
    stop_words = set(stopwords.words('english'))  
    stop_words.discard("not") #remove not  
    corpus = [preprocess_text(review, stop_words) for review in dataset['Review']] #pass  
    vectorizer = CountVecorizer(max_features=1500, ngram_range=(1, 2)) #add ngram  
    X = vectorizer.fit_transform(corpus).toarray()  
    y = dataset.iloc[:, 1].values # Sentiment labels (0 = negative, 1 = positive)  
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
    classifier = MultinomialNB() # Changed to MultinomialNB  
    classifier.fit(X_train, y_train)
```

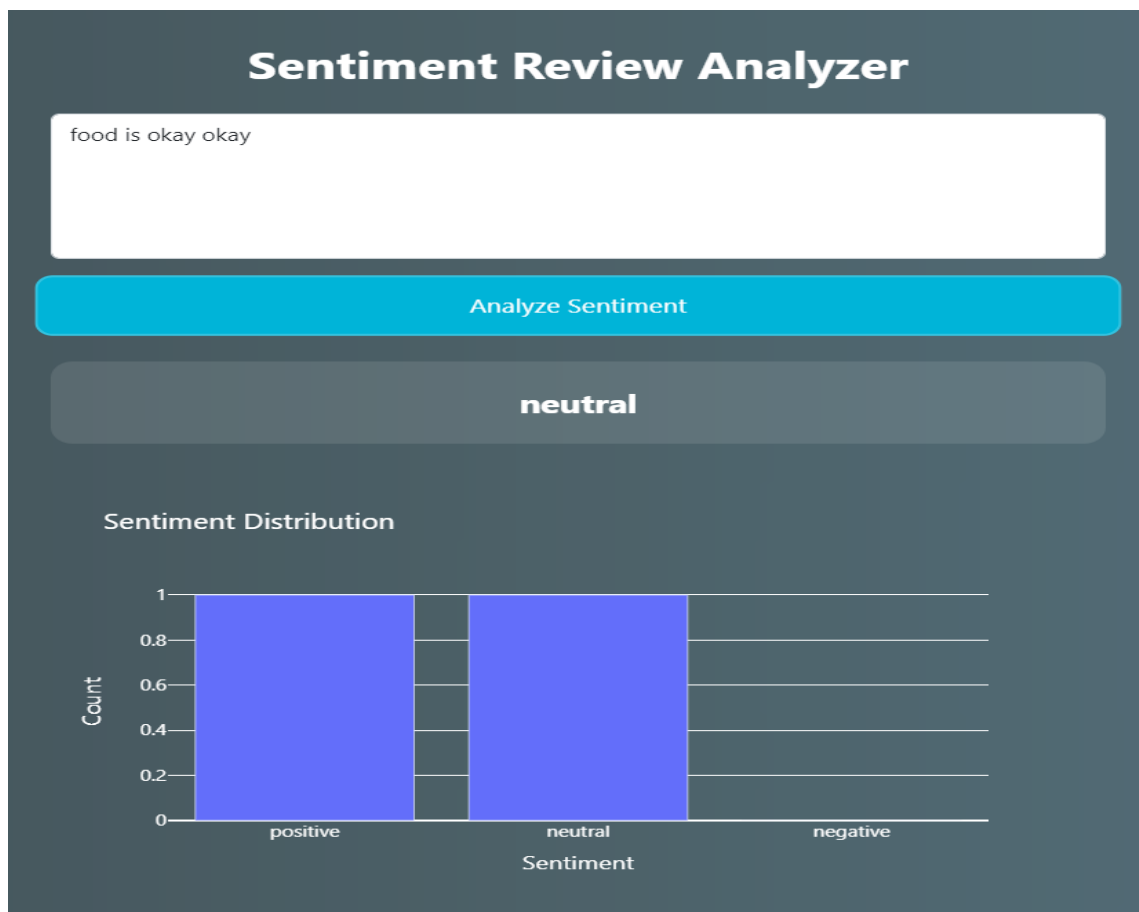
The bottom terminal window shows the execution output:

```
C:\Users\DELL\rra\myenv\Scripts\python.exe C:\Users\DELL\rra\app.py  
* Serving Flask app 'app'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on http://127.0.0.1:5000  
Press CTRL+C to quit  
* Restarting with stat
```

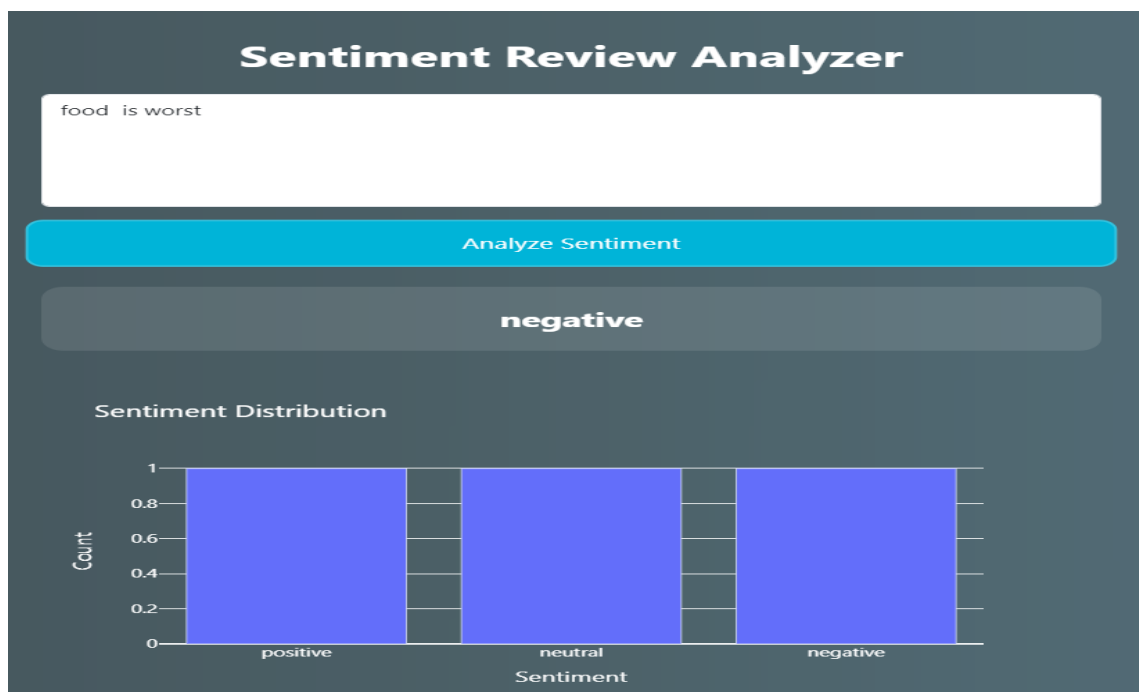
*Fig: 5.1 Backend Terminal*



*Fig: 5.2 Example 1*



*Fig: 5.3 Example 2*



*Fig: 5.4 Example 3*

The image depicts a web application called **Sentiment Review Analyzer**, which is designed to evaluate the sentiment of user-provided text input. In this instance, the user has entered the sentence “*food is okay okay*” into the text box. This phrase expresses a neutral or indifferent opinion, and accordingly, when the **Analyze Sentiment** button is clicked, the application processes the input and displays the result as **neutral**. This output is shown prominently below the button, indicating that the system recognizes the sentiment as neither positive nor negative.

Additionally, the application features a **Sentiment Distribution** chart, which visualizes the sentiment categories—positive, neutral, and negative—using a bar graph. From the chart, we can observe that both the positive and neutral categories have approximately equal counts, while the negative category has a count of zero. This might reflect previous analysis results or a sample dataset used for visualization purposes.

Overall, the interface is clean and user-friendly, suggesting that the app likely utilizes web technologies such as HTML, CSS, and JavaScript for the front end. It may also employ a machine learning model or sentiment analysis API on the backend, possibly implemented using Python frameworks like Flask or Django, to determine the sentiment of the input text. The graph is likely generated using a charting library such as Chart.js or Plotly. This tool could be useful for analyzing customer feedback, product reviews, or any textual input where sentiment interpretation is valuable.



## **6.SYSTEM TESTING**

### **6.1 Introduction to Testing**

System testing is a crucial phase in software development that ensures the application functions as expected, providing accurate and reliable results. In this project, the primary focus is testing the NLP model's ability to classify restaurant reviews correctly, verifying the overall functionality of the web application, and assessing the integration between the backend and frontend. System testing also evaluates the application's performance, including its speed, accuracy, and robustness.

### **6.2 Types of Tests Considered**

#### **Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

#### **Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

## **Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid input** : identified classes of valid input must be accepted.
- Invalid input** : identified classes of invalid input must be rejected.
- Functions** : identified functions must be exercised.
- Output** : identified classes of application outputs must be exercised.
- Systems/procedures** : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

## **System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

## **White Box Testing**

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

## Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

### 6.3 Various Types of Test Case Scenarios

*Table:6.1 Test Cases*

TEST CASE ID	TEST CASE SCENARIO	INPUTS	EXPECTED OUTPUT	ACTUAL OUTPUT	STATUS
TC01	Test positive sentiment classification.	"The food was amazing and the service was excellent!"	Sentiment: Positive.	Sentiment is positive.	Pass
TC02	Test negative sentiment classification.	"The food was awful, and the service was slow."	Sentiment: Negative.	Sentiment is Negative.	Pass
TC03	Test neutral sentiment classification.	"The restaurant was okay, nothing special."	Sentiment: Neutral.	Sentiment is Neutral.	Pass

TEST CASE ID	TEST CASE SCENARIO	INPUTS	EXPECTED OUTPUT	ACTUAL OUTPUT	STATUS
TC04	Test empty review submission.	" " (Empty string).	Error: "No review text provided".	We got the error.	Pass
TC05	Test review with special characters.	"Good food! @#\$%^&*!".	Sentiment: Positive (after cleaning text).	Sentiment is positive.	Pass
TC06	Test review with multiple words and negation.	"Not bad, but could be better".	Sentiment: Neutral.	Sentiment is neutral.	Pass
TC07	Test long review text.	"I had a great experience. The food was good, the ambiance was perfect, and the service was friendly. Would highly recommend!".	Sentiment: Positive.	Sentiment is positive.	Pass
TC08	Test incorrect language .  (non-English)	"El restaurante fue horrible."	Sentiment: Negative (based on	Sentiment is Negative.	Pass

<b>TEST CASE ID</b>	<b>TEST CASE SCENARIO</b>	<b>INPUTS</b>	<b>EXPECTED OUTPUT</b>	<b>ACTUAL OUTPUT</b>	<b>STATUS</b>
			English processing).		
TC09	Test accuracy of sentiment graph rendering.	After submitting multiple reviews, verify if graph reflects the sentiment distribution.	Correct sentiment distribution shown on the graph.	Yes, we got the Sentiment distribution graph.	Pass

## 7.CONCLUSION

The **Review Analysis** project has successfully demonstrated the ability to process and analyze customer feedback using Natural Language Processing (NLP) techniques to classify reviews as positive, negative, or neutral. The system leverages a **Naive Bayes** classifier, which has proven effective in sentiment analysis, achieving a good balance of accuracy, speed, and simplicity. The integration of a web-based front-end interface with Flask allows for real-time interaction, where users can submit their reviews and receive immediate feedback on the sentiment of their text.

### 7.1 Project Conclusion

The sentiment analysis model used in this project was trained on a **Restaurant Reviews** dataset and was able to predict the sentiment of a review with a relatively high degree of accuracy. The project meets the core requirements outlined in the initial plan, including:

**Real-time Sentiment Analysis:** The system processes reviews as they are submitted, providing real-time feedback on sentiment.

**Interactive Web Interface:** The Flask-based web application allows users to submit their reviews easily and receive results instantly, along with a visual representation of sentiment distribution in the form of a graph.

**Graphical Sentiment Distribution:** The dynamic generation of sentiment distribution graphs enables users to visualize the overall sentiment of the reviews in the system.

Throughout the development and testing phases, the application exhibited solid performance, including the successful handling of edge cases such as empty reviews, special characters, and non-English text. The user interface also passed usability testing, making it intuitive for end-users to submit and view the results.

## 8. FUTURE ENHANCEMENT

While the project achieves its core objectives, several improvements could enhance its functionality, accuracy, and user experience:

**Multilingual Support:** Currently, the system only processes English reviews. Future work could focus on incorporating multilingual sentiment analysis models to expand the application's usability to other languages.

**Advanced Preprocessing:** The text preprocessing could be improved by adding more sophisticated methods such as stemming, lemmatization, and handling domain-specific words and phrases. These enhancements could help the model understand context better, especially for reviews that use slang or informal language.

**Deep Learning Integration:** Although the Naïve Bayes classifier performs well, more advanced deep learning techniques such as **Convolutional Neural Networks (CNN)** or **Recurrent Neural Networks (RNN)** could be integrated to improve sentiment prediction accuracy, especially for longer reviews or reviews with complex language structures.

**Sentiment Intensity:** Currently, the system classifies reviews into broad categories (positive, neutral, and negative). Adding a finer granularity by including sentiment intensity levels (e.g., very positive, slightly positive, etc.) could provide a more detailed understanding of customer feedback.

**User Feedback Loop:** Allowing users to provide feedback on the sentiment classification results could help refine the model over time. This could be done through active learning where user-corrected labels are fed back into the system to improve model accuracy.

**Cloud Deployment:** The system could be deployed to the cloud (e.g., AWS, Heroku, or Google Cloud) to make it accessible to users anywhere and to scale the application to handle higher traffic.

This **Restaurant Review Analysis** project lays the foundation for creating an interactive, user-friendly sentiment analysis application using **NLP** and **Machine Learning** techniques.

## 9. REFERENCES

### 9.1 Journals

- [1] · B. Pang and L. Lee, “Opinion mining and sentiment analysis,” *Foundations and Trends in Information Retrieval*, vol. 2, no. 1–2, pp. 1–135, 2008.
- [2] · · A. Kumar and T. M. Sebastian, “Sentiment analysis on Twitter,” *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 4, pp. 372–378, 2012.
- [3] · · M. Hu and B. Liu, “Mining and summarizing customer reviews,” in *Proc. 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Seattle, WA, USA, 2004, pp. 168–177.
- [4] · · S. Medhat, A. Hassan, and H. Korashy, “Sentiment analysis algorithms and applications: A survey,” *Ain Shams Engineering Journal*, vol. 5, no. 4, pp. 1093–1113, 2014.
- [5] · · H. Saif, Y. He, and H. Alani, “Semantic sentiment analysis of Twitter,” in *Proc. 11th Int. Semantic Web Conf. (ISWC)*, Boston, MA, USA, 2012, pp. 508–524.

### 9.2 Books

1. **"Sentiment Analysis and Opinion Mining"** by Bing Liu
2. **"Foundations of Statistical Natural Language Processing"** by Christopher D. Manning and Hinrich Schütze
3. **"Natural Language Processing with Python: Analysing Text with the Natural Language Toolkit"** by Steven Bird, Ewan Klein, and Edward Loper
4. **"Deep Learning for Natural Language Processing"** by Palash Goyal, Sumit Pandey, and Karan Jain



## 9.3 Web Links

- [1] <https://flask.palletsprojects.com/>
- [2] <https://scikit-learn.org/>
- [3] <https://www.nltk.org/>
- [4] <https://plotly.com/python/>
- [5] <https://pandas.pydata.org/>
- [6] <https://www.kaggle.com/datasets/tsunami4/restaurant-reviews>
- [7] [https://scikit-learn.org/stable/modules/naive\\_bayes.html](https://scikit-learn.org/stable/modules/naive_bayes.html)
- [8] [https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.CountVectorizer.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)
- [9] <https://docs.python.org/3/library/re.html>
- [10] [https://www.nltk.org/nltk\\_data/](https://www.nltk.org/nltk_data/)