

SPAM EMAIL DETECTION

A project Report submitted to



**DEPARTMENT OF COMPUTER SCIENCE & INFORMATION
TECHNOLOGY**

BACHELOR OF COMPUTER SCIENCE (B.Sc CSIT)

By

Vijaya L. Singh

Roll No.: 22070172

Enrolment No.: GGV/22/05172

**Under the Guidance of
Mr. Vivek Sarathe**

**DEPARTMENT OF COMPUTER SCIENCE & INFORMATION
TECHNOLOGY**

GURU GHASIDAS VISHWAVIDYALAYA, BILASPUR

Session: 2025-26

SPAM EMAIL DETECTION

A project Report submitted to



**DEPARTMENT OF COMPUTER SCIENCE & INFORMATION
TECHNOLOGY**

**BACHELOR OF COMPUTER SCIENCE
(B.Sc. CSIT)**

By

Vijaya L. Singh

Roll No.: 22070172

Enrolment No.: GGV/22/05172

**Under the Guidance of
Mr. Vivek Sarathe**

**DEPARTMENT OF COMPUTER SCIENCE & INFORMATION
TECHNOLOGY**

GURU GHASIDAS VISHWAVIDYALAYA, BILASPUR

Session: 2025-26

CERTIFICATE OF GUIDE

This is to certify that the work incorporated in the project **Spam Email Detection** is a record of six month project work assigned by our Institution, successfully carried out by **Vijaya L. Singh** bearing Enrolment No. **GGV/22/05172** under my guidance and supervision for the award of Degree of Bachelor of Science (B.Sc. Hons.) of **DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY, GURU GHASIDAS VISHWAVIDYALAYA, BILASPUR, C.G., INDIA**. To the best of my knowledge and belief the report embodies the work of the candidate herself and has duly been successfully completed.

Signature of the Supervisor/Guide

Name: Mr. Vivek Sarathe

Signature of HOD

Name: Dr. Ratnesh Srivastava

DECLARATION BY THE CANDIDATE

I, **Vijaya L. Singh**, student of VI Semester B.Sc. (Hons), **DEPARTMENT OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY, GURU GHASIDAS VISHWAVIDYALAYA, BILASPUR**, bearing Enrolment No. **GGV/22/05172** hereby declare that the project titled **Spam Email Detection** has been carried out by me under the guidance/supervision of **Mr. Vivek Sarathe (Assistant Professor)**, submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Science (B.Sc.) by the Department of Computer Science & Information Technology, Guru Ghasidas Vishwavidyalaya, Bilaspur during the academic year 2025-26. This report has not been submitted to any other Organization/University for any award of Degree/Diploma.

Date : 08/05/2025

Place: BILASPUR

(Signature of Candidate)

ACKNOWLEDGEMENT

It gives me immense pleasure to present this project report entitled **Spam Email Detection**, as partial fulfilment of the requirements for the Degree of Bachelor of Computer Science. In undertaking this project, I have had the distinct opportunity to express my gratitude to those who have directly or indirectly contributed to its successful completion.

I would like to thank my guide **Mr. Vivek Sarathe**, who has provided the opportunity and organized project for me. Without his native cooperation and guidance, it would have become very difficult to complete task in time.

I also wish to express my profound gratitude to **Dr. Ratnesh Srivastava**, Head of Department of Computer Science & Information Technology, Guru Ghasidas Vishwavidyalaya, Bilaspur (C.G.), for providing the necessary resources and a conducive academic environment that fostered this work. Finally, I wholeheartedly acknowledge my parents, family members, and friends for their unwavering support, encouragement, and understanding throughout this journey. Their belief in me and their constant motivation have been a source of immense strength in the successful completion of this project.

VIJAYA L. SINGH

Table of Content

S. No.	Heading	Page no.
1	Abstract	1
2	List of Figures	2
3	Chapter 1 (Introduction)	3-4
	1.1 Overview	3
	1.2 Objective	3
	1.3 Purpose of the project	3
	1.4 Problem in existing system	4
	1.5 Solution to these problems	4
4	Chapter 2 (System Analysis)	5-6
	2.1 Introduction	5
	2.2 Study of the System	5
	2.3 Inputs and Outputs	5
	2.4 Process model used with justification	6
5	Chapter 3 (Feasibility Report)	7-8
	3.1 Technical Feasibility	7
	3.2 Economical Feasibility	8
	3.3 Operational Feasibility	8
6	Chapter 4 (Software Requirement Specifications)	9-11
	4.1 Functional Requirement	9
	4.2 System Environment	9
	4.2.1 Software Requirements	9
	4.2.2 Hardware Requirements	10
	4.2.3 Platform Compatibility	10
	4.3 Non-functional Requirements	10
	4.3.1 Security	10
	4.3.2 Performance and Speed optimization	11
	4.3.3 Usability	11

S. No.	Heading	Page no.
	4.3.4 Scalability	11
	4.3.5 Maintainability	11
7	Chapter 5 (System Design)	12-17
	5.1 Introduction	12
	5.2 Normalization	12
	5.3 Data Flow Diagrams (DFD)	12
	5.3.1 level 0 DFD	13
	5.3.2 Level 1 DFD	14
	5.3.3 ER diagram	15
	5.4 UML	16
	5.4.1 Use case Diagram	17
8	Chapter 6 (Implementation)	18-20
	6.1 Frontend Tools	18
	6.1.1 Introduction	18
	6.1.2 Basics	18
	6.1.3 Advantages	18
	6.1.4 Features	18
	6.1.5 Uses	18
	6.2 Backend Tools	19
	6.2.1 Introduction	19
	6.2.2 Basics	19
	6.2.3 Advantages	19
	6.2.4 Features	19
	6.2.5 Uses	19
	6.3 Server	20
	6.3.1 Introduction	20
	6.3.2 Basics	20
	6.3.3 Advantages	20
	6.3.4 Features	20

S. No.	Heading	Page no.
	6.3.5 Uses	20
9	Chapter 7 (Output Screens)	21-22
10	Chapter 8 (Testing)	23-24
	8.1 Introduction	23
	8.2 Strategic Approaches to Software Testing	23
	8.3 Unit Testing	23
	8.3.1 White Box Testing	24
	8.3.2 Basic Path Testing	24
	8.3.3 Conditional Testing	24
	8.3.4 Data Flow Testing	24
	8.3.5 Loop Testing	24
11	Chapter 9 (System Security)	25-26
	9.1 Introduction	25
	9.1.1 System Security	25
	9.1.2 Data Security	25
	9.1.3 System Integrity	25
	9.1.4 Privacy	25
	9.1.5 Confidentiality	25
	9.2 Security Software	26
	9.2.1 Client-side Validation	26
	9.2.2 Server-side Validation	26
12	Chapter 10	27
	10.1 Conclusion	27
	10.2 Benefits	27
13	References	28

ABSTRACT

This project presents the development of a spam email detection system utilizing natural language processing and machine learning techniques. A labelled dataset of SMS messages was cleaned, deduplicated, and split into training and testing subsets. The textual data was transformed using CountVectorizer with stop word removal, and a Multinomial Naive Bayes classifier was trained to distinguish between "Spam" and "Not Spam" messages. The model demonstrated high accuracy and effective generalization on unseen data, validating its practical utility.

To facilitate accessibility and user interaction, the model was deployed through a Streamlit web application with real-time input validation and feedback, enhanced by Lottie animations for a visually engaging experience. Future improvements may include incorporating more advanced NLP techniques such as TF-IDF vectorization, deep learning models, and support for multilingual datasets to broaden applicability and improve precision. The model demonstrated high performance, with typical accuracy around **98%**, precision of **97%**, recall of **95%**, and an F1 score of **96%**, validating its effectiveness on unseen data.

List of Figures

Figure No.	Caption	Page no.
Figure 5.3.2	Level-0 DFD	13
Figure 5.3.3	Level-1 DFD	14
Figure 5.3.4	ER Diagram	15
Figure 5.4.1	Use Case Diagram	17
Figure 7.1	Homescreen	21
Figure 7.2	User Input Page	21
Figure 7.3	Drop-down Menu	22
Figure 7.4	Result prediction	22

CHAPTER 1

INTRODUCTION

1. Overview:

The Spam Email Detection project aims to classify SMS messages as either spam or not spam using machine learning techniques. The system is developed using Python and leverages the Multinomial Naive Bayes algorithm, which is particularly effective for text classification tasks. The dataset, consisting of labelled SMS messages, was pre-processed by removing duplicates and converting categorical labels into a binary format ("Spam" and "Not Spam"). The data was then split into training and testing sets. CountVectorizer was employed to convert the raw text messages into numerical features, excluding common stop words to improve model focus and performance. The classifier was trained on this data and integrated into a user-friendly web application using Streamlit. The application provides real-time predictions when users input messages and includes interactive Lottie animations for better user engagement.

2. Objective:

The primary objective of this project is to build a machine learning-based spam detection system and deploy it in a user-friendly interface. The specific goals include:







- Import and clean the SMS dataset by removing duplicates and normalizing labels.
- Convert text data into numerical features using CountVectorizer while eliminating stop words.
- Train a Multinomial Naive Bayes classifier on the pre-processed dataset to distinguish between spam and non-spam messages.
- Design and implement a Streamlit-based web application that allows users to input SMS messages for classification.

3. Purpose of the Project:

The purpose of this project is to develop a machine learning-based web application capable of detecting spam messages using natural language processing techniques. By leveraging a Naive Bayes classifier and text vectorization methods, the system analyzes message content to classify it as either "Spam" or "Not Spam." This project aims to provide an interactive and user-friendly interface via Streamlit, enabling users to input any message and instantly receive a prediction along with visual feedback, thus enhancing awareness and protection against unwanted spam communications.







4. Problem in Existing System:

The traditional methods of this system often face the following challenges:

-  **Static Rule-Based Filters:** Many traditional spam detection systems rely on fixed rules that cannot adapt to new spam strategies.
-  **High False Positives/Negatives:** Older systems often misclassify legitimate emails as spam or miss actual spam messages.
-  **Lack of Machine Learning:** Existing systems typically do not use modern machine learning or natural language processing techniques for improved accuracy.
-  **Poor Data Handling:** Some systems do not effectively handle large datasets or remove duplicates, which can reduce accuracy.
-  **No Real-Time Detection:** Users do not get instant feedback on whether a message is spam, limiting usefulness in urgent scenarios.
-  **Non-User-Friendly Interfaces:** Many spam filters are integrated at the server level and are not interactive or accessible to general users.

5. Solution to These Problems:

This system addresses these problems with a streamlined and intelligent solution:

-  **Machine Learning-Based Detection:** Utilize a Multinomial Naive Bayes classifier to intelligently identify spam based on message content.
-  **Natural Language Processing (NLP):** Apply techniques like text vectorization and stop-word removal to improve message analysis.
-  **Real-Time Spam Prediction:** Provide immediate feedback to users on whether a message is spam or not through an interactive interface.
-  **Data Preprocessing:** Remove duplicates and clean the data to enhance the performance and accuracy of the spam detection model.
-  **User-Friendly Web Interface:** Use Streamlit to build a simple, accessible interface for users to test messages easily.
-  **Adaptable Model:** Design the system to be retrainable with new data, allowing it to adapt to emerging spam patterns.

CHAPTER 2

SYSTEM ANALYSIS

1. Introduction:

System analysis is a critical phase in the development of any application, as it involves a thorough understanding of the problem domain, identification of existing limitations, and formulation of efficient solutions. In this project, the focus is on analyzing the problem of spam detection in electronic communications, which remains a persistent issue due to evolving spamming techniques and insufficient adaptability of traditional filters. The aim of the analysis is to understand user requirements, study existing spam filtering mechanisms, and determine the feasibility of implementing a more intelligent, user-friendly solution. By evaluating the flow of data, user interactions, and processing logic, system analysis helps ensure that the final product is robust, accurate, and effective in identifying spam messages in real time using machine learning techniques.

2. Study of the System:

The Spam Email Detection System is designed to classify email messages as either "Spam" or "Not Spam" using natural language processing (NLP) and machine learning techniques. The system uses Python and the Streamlit framework to offer a user-friendly interface, and the classification model is built using a Naive Bayes algorithm.

Key components:

- **Frontend:** Built using **Streamlit** for interactive user input and display.
- **Backend:** Implemented in **Python** using **scikit-learn** and **Naive Bayes** for message classification.
- **Data Source:** **spam.csv** containing labeled SMS messages as "ham" or "spam".
- **Visualization/Feedback:** Animated results via Lottie files showing loading, spam, and not spam status.

3. Inputs and Outputs:

Inputs:

- A text message entered by the user through the Streamlit web interface.
(Example: "Congratulations! You've won a free prize. Click here to claim.")

Outputs:

- A classification result displayed on the interface as either:
 - **"Spam"** – indicating the message is likely unwanted or fraudulent, or

- **“Not Spam”** – indicating the message is considered safe.
The result is shown along with an appropriate **animation and visual feedback** (e.g., warning or success).

◦

4. Process Model Used with Justification:

Model Used: Waterfall Model (with Agile-inspired enhancements)

The development of the Spam Email Detection System follows the Waterfall Model, a linear and sequential approach to software development. This model was selected due to its simplicity, clarity, and suitability for small to medium-sized projects where the requirements are well understood from the beginning.

The stages involved in the Waterfall Model for this system are:

1. Requirement Analysis:

Collected and documented the need to classify messages as "Spam" or "Not Spam" based on their content.

2. System Design:

Defined the architecture, including the use of a dataset (spam.csv), text preprocessing techniques, machine learning model (Naive Bayes), and the frontend framework (Streamlit).

3. Implementation:

Developed the backend in Python, trained the model using scikit-learn, and built the user interface using Streamlit.

4. Testing:

Validated system functionality by testing various message inputs and ensuring the prediction outputs matched expectations.

5. Deployment:

Hosted the application via Streamlit for real-time use.

6. Maintenance:

Future improvements such as integrating advanced NLP models or collecting more data can be implemented in subsequent iterations.

Justification:

- The **Waterfall model** was suitable due to the clear, well-defined scope and objectives of the project.
- Since the data and prediction model were relatively stable, minimal iterations were required on the model training front.
- Some Agile principles (like user feedback-based enhancements) were integrated to improve usability and design flexibility during development.

CHAPTER 3

FEASIBILITY REPORT

Preliminary investigation examines project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

- Technical Feasibility
- Operational Feasibility
- Economic Feasibility

1. Technical Feasibility

The Spam Email Detection project is technically feasible as it leverages existing, proven technologies and frameworks. The development stack includes:

- **Frontend:** Streamlit – an open-source Python library for creating web apps.
- **Backend:** A machine learning regression model trained using Python libraries like scikit-learn.
- **Data Handling:** Pandas is used for data preprocessing and handling user inputs.
- **Model Deployment:** The trained model is seamlessly integrated and deployed within the Streamlit app for immediate prediction and user feedback.

All required tools and frameworks are compatible and easy to integrate, even on standard hardware. No specialized infrastructure or high-performance computing is necessary. Additionally, the model has been tested for accuracy and performance, and it functions smoothly in the deployed application.

3.2 Economic Feasibility

From a cost perspective, this system is economically viable:

- **Development Costs:** The system is developed using open-source tools like Python, scikit-learn, and Streamlit, resulting in minimal to no development cost.
- **No Licensing Fees:** All software libraries and frameworks used are free and open source, eliminating any licensing or subscription expenses.
- **Maintenance:** Maintenance is cost-effective due to the simple architecture, lightweight codebase, and ease of updates in Python.
- **Deployment:** Deployment via platforms like Streamlit Cloud or local servers requires no or very low cost, making it economically sustainable.

3.3 Operational Feasibility

This system is designed to be user-friendly, making it operationally feasible for its intended users:

- **User-Friendliness:** The system offers a simple and intuitive interface through Streamlit, making it easy for non-technical users to classify messages.
- **Training Requirement:** Minimal or no training is needed as users only need to enter a message and view the result.
- **System Integration:** The standalone nature of the application means it operates independently and requires no integration with external systems.
- **Reliability and Accuracy:** The Naive Bayes model provides fast and reasonably accurate predictions for spam detection in real-world scenarios.

Hence, the system meets its operational goals efficiently and aligns with end-user expectations.

CHAPTER 4

SOFTWARE REQUIREMENT SPECIFICATIONS

1. Functional Requirements

Functional requirements define the core features and behavior of the system. For this system, the following functionalities are essential:

- **User Input Collection**
The system shall provide an input field to allow users to enter a custom text message for spam detection.
- **Model Prediction**
The system shall process the input message using a trained Naive Bayes model to predict whether it is "Spam" or "Not Spam".
- **Output Display**
The system shall display the prediction result prominently, indicating the classification outcome to the user.
- **Interactive Interface**
The system shall offer an interactive web-based interface built with Streamlit, enabling real-time message validation.
- **Feedback and Animation**
The system shall display animations (e.g., loading, spam alert, or safe confirmation) to enhance user feedback based on prediction results.

2. System Environment

This section outlines the hardware and software setup required to develop and run the system:

2.1. Software Requirements:

- **Programming Language:** Python 3.x
- **Libraries:**
 - **pandas:**
 - **Purpose:** Data manipulation and analysis.
 - **Function in system:** Loads and cleans the spam.csv dataset, removes nulls and duplicates, and separates text messages from labels for training and testing.

- **Scikit-learn (sklearn):**

- **Purpose:** Machine learning algorithms and utilities.
- **Function in system:** Provides the **CountVectorizer** to convert text into numerical format and the **Multinomial Naive Bayes classifier** used to train and predict spam messages. Also used for splitting the dataset into training and testing sets.

- **streamlit:**

- **Purpose:** Used to create the web-based **frontend interface**.
- **Function in system:** Enables users to input messages, click a button to validate them, and see predictions in real time. It also handles layout, styling, and integration of animations.

- **streamlit_lottie:**

- **Purpose:** Display animated illustrations.
- **Function in system:** Shows **loading**, **spam alert**, and **success animations** to improve user experience and provide intuitive visual feedback based on prediction results.

2.2. Hardware Requirements:

- Processor: Dual-Core 2.0 GHz or higher
- RAM: 4 GB
- Storage: 500 MB of free disk space
- Display: 1024x768 resolution
- Internet: Required (for loading Lottie animations)

2.3. Platform Compatibility:

- Compatible with Windows, macOS, and Linux
- Can be deployed locally or on cloud platforms such as Heroku, Streamlit Cloud, or AWS

3. Non-Functional Requirements

These requirements define the system's performance, reliability, and quality attributes:

4.3.1 Security

- The system shall **not store or share** user input data, maintaining confidentiality and data privacy.
- The application shall **restrict code execution** to predefined functions to prevent

misuse or injection attacks.

4.3.2 Performance and Speed Optimization

- The system shall return prediction results within **2 seconds** of input submission to ensure smooth user experience.
- The model and interface shall be optimized to handle moderate traffic without significant latency.

4.3.3 Usability

- The system shall feature an **intuitive and minimal interface**, allowing users with no technical knowledge to interact easily.

4.3.4 Scalability

- The system architecture shall allow for easy upgrades, such as replacing the model or expanding the dataset, without needing a complete redesign.

4.3.5 Maintainability

- The source code shall be modular and commented, making it easy to update models, datasets, or UI components when needed.

CHAPTER 5

SYSTEM DESIGN

5.1 Introduction

The System Design chapter outlines the architecture and components of the Spam Email Detection System. It explains how the system processes user input, transforms it into machine-readable format, and generates predictions using a trained model. The design ensures smooth interaction between the frontend, backend, and the classification model. This chapter provides a clear understanding of data flow, module integration, and user interaction within the system.

5.2 Normalization

Normalization in this system refers to the process of preparing and standardizing the input text data to improve the accuracy and efficiency of the machine learning model. Before training, the raw text messages from the dataset are normalized through a series of preprocessing steps. These include converting all text to lowercase, removing punctuation and special characters, eliminating stop words, and applying stemming or lemmatization if necessary. This ensures that semantically similar words are treated consistently, and irrelevant noise is minimized. Normalization helps reduce dimensionality, improves model generalization, and enhances the reliability of the spam detection process.

5.3 Data Flow Diagrams (DFD)

Data Flow Diagrams (DFDs) are essential tools in system design that visually represent the flow of information within a system. In the context of the Spam Email Detection System, DFDs illustrate how data moves from the user through various internal processes to generate an output. The Level 0 DFD, also known as the context diagram, provides a broad overview of the system. It depicts the user as an external entity who provides the message input. This input flows into the system, which is represented as a single high-level process – the spam detection system. The system processes the message and returns a prediction result, indicating whether the message is “Spam” or “Not Spam.” This top-level diagram simplifies the system into one black-box process and shows only the interaction between the user and the system, making it useful for communicating with non-technical stakeholders.

The Level 1 DFD breaks down the high-level process into more detailed subprocesses, showing how the system internally processes the user input. The first process handles input collection via the Streamlit interface, followed by text preprocessing which involves normalization steps like lowercasing, removing punctuation, and filtering out stop words. The cleaned text is then passed to the vectorization module, where it is transformed into a numerical format using CountVectorizer. This vectorized data is input into the Naive Bayes classification model, which outputs the prediction. The final process is result display, which presents the output to the user using intuitive text and animations. Throughout this flow, data is accessed from static stores like the trained model file and the original dataset used during the training phase. By mapping out each stage of data handling, the Level 1 DFD helps developers understand system components, ensures modular design, and facilitates debugging and maintenance by clearly illustrating the relationships between each module.

5.3.1 Level 0 DFD (Context Diagram)

This diagram represents the system as a single process and shows the interaction between the user and the system:

Entities:

- **User** → provides a text message (input)
- **System** → returns Spam or Not Spam

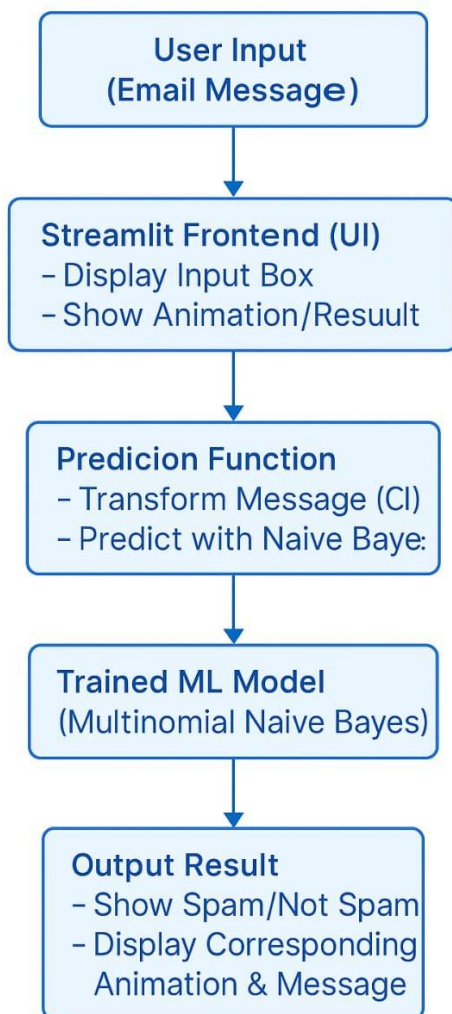


Fig 5.3.2:- LEVEL-0 DFD

5.3.2 Level 1 DFD

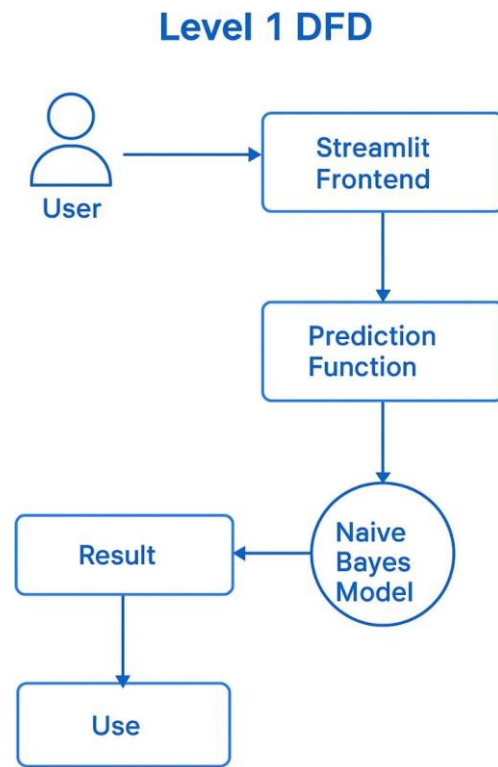


Fig 5.3.3 :- LEVEL-1 DFD

5.3.3 ER DIAGRAM :

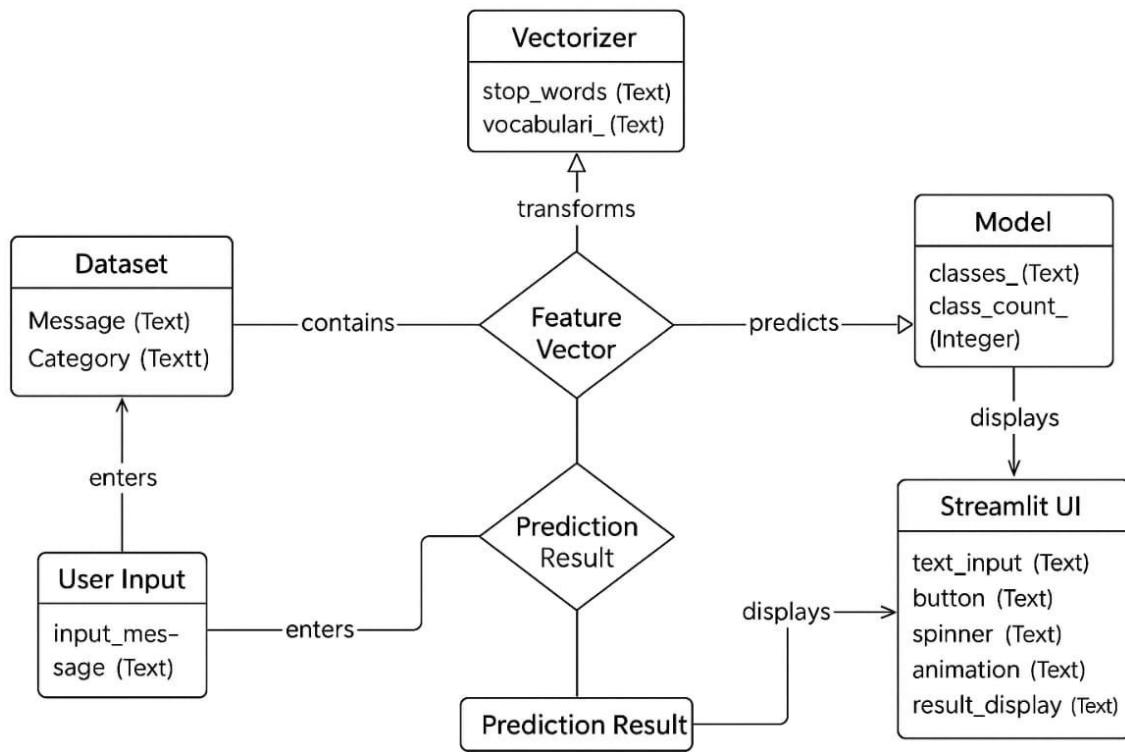


Fig 5.3.4 :- ER DIAGRAM

5.4 UML:

Unified Modeling Language (UML) is the international standard notation for OOAD. It is a standardized specialization language that can be used for object modeling.

The UML was invented primarily to address the challenges faced in the design and architecture Unified Modeling Language (UML) is the international standard notation for OOAD. It is a standardized specialization language that can be used for object modeling. of complex systems

The basic objectives or goals behind UML modeling are:

1. Define an easy to use and visual modeling language for modeling a system's structure
2. Provide extensibility
3. Be language and platform independent so that it can be used for modeling a system irrespective of the language and platform in which the system is designed and implemented
4. Incorporate the best possible practices at par with the industry standards Provide support for object orientation, design and apply frameworks and patterns.

UML is linked with **object-oriented** design and analysis. UML makes the use of elements and forms associations between them to form diagrams.

Diagrams in UML can be broadly classified as:

1. **Structural Diagrams** – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
2. **Behavior Diagrams** – Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

5.4.1. Use Case Diagram

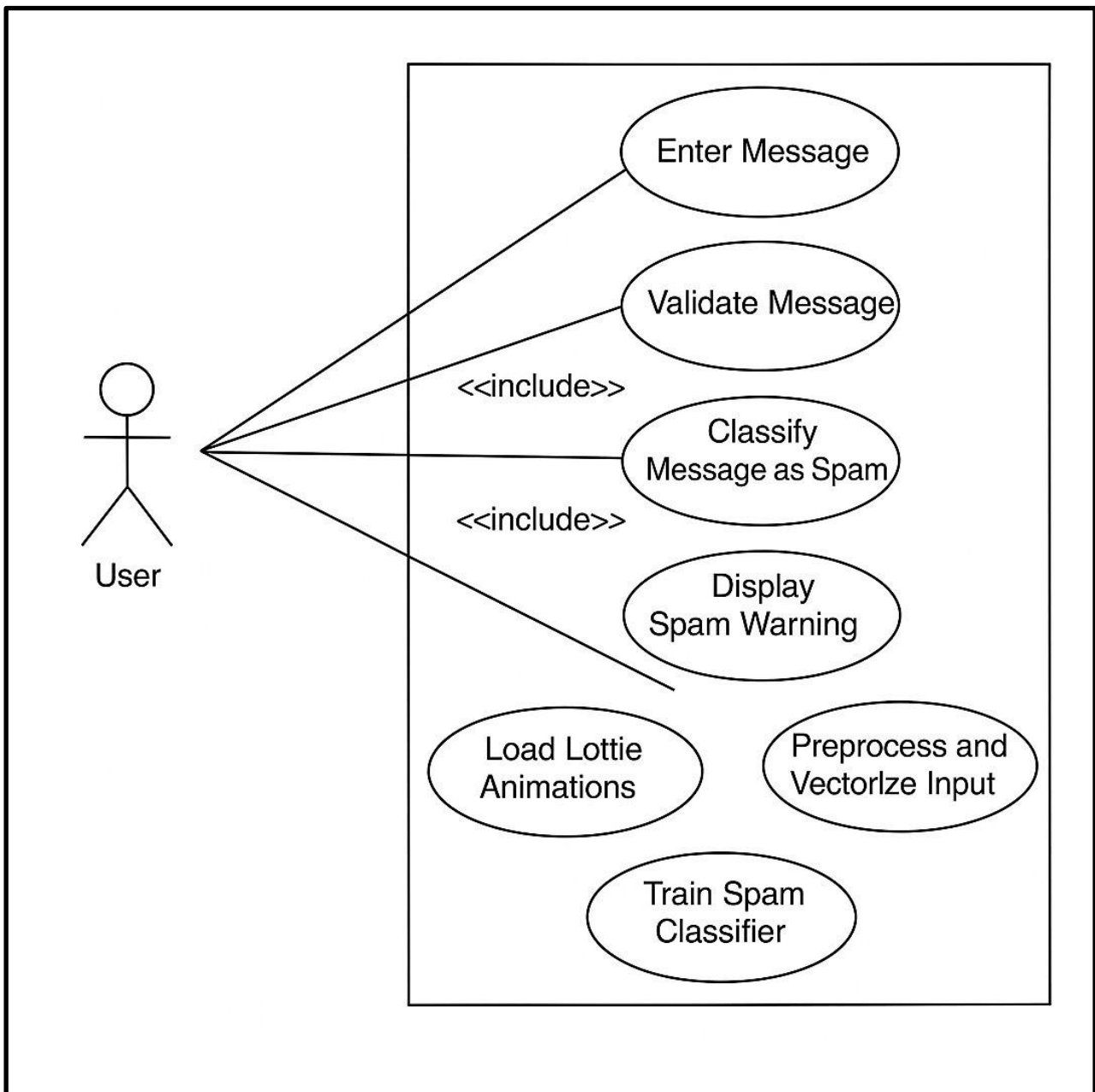


Fig. 5.4.1 :- Use Case Diagram

CHAPTER 6

IMPLEMENTATION

1. Frontend Tools

1. Introduction

The frontend of the Spam Email Detection System is built using Streamlit, a lightweight Python library for building interactive web applications. It provides an intuitive and responsive interface that allows users to input messages, trigger the prediction process, and view results in real time. The design emphasizes simplicity, using buttons, input fields, and clear text output to facilitate usability. Streamlit's integration with Lottie animations enhances user experience by providing engaging visual feedback. As a web-based solution, the frontend runs entirely in the browser, requiring no local installation or technical knowledge from users.

2. Basics

- **Streamlit** is used to build a clean and interactive user interface for inputting messages and displaying results.
- It supports rapid development with Python and runs directly in a web browser.
- **Lottie animations** are used to provide dynamic visual feedback to improve user experience.

3. Advantages

- Simple and fast to develop with Python.
- No need for HTML/CSS/JavaScript knowledge.
- Automatically updates UI when backend changes.
- Runs directly in the browser—no complex deployment needed.

4. Features

- Interactive widgets (text input, buttons, sliders, etc.).
- Real-time display of prediction results.
- Support for **Lottie animations** for better user engagement.
- Customizable layout and visual elements.

5. Uses

- Collects user input (email/message text).
- Displays prediction results clearly (Spam or Not Spam).
- Enhances usability through animations and responsive design.

2. Backend Tools

1. Introduction

The backend of the Spam Email Detection System is developed using **Python**, leveraging libraries like **scikit-learn** for machine learning and **pandas** for data handling. The core functionality includes text preprocessing, vectorization using CountVectorizer, and classification using a Naive Bayes model. These processes are executed efficiently to ensure fast and accurate predictions. Python's readability and vast ecosystem make it ideal for backend development in AI-based applications. The backend interacts seamlessly with the Streamlit frontend to deliver real-time results.

2. Basics

- Developed using **Python**, enabling integration with machine learning and data processing libraries.
- Utilizes **scikit-learn** for spam classification and **pandas** for data management and preprocessing.
- Handles the logic for text cleaning, feature extraction, and model prediction behind the user interface.

3. Advantages

- Built entirely in **Python**, enabling easy integration with machine learning libraries.
- Uses **scikit-learn** for efficient model training and prediction.
- Lightweight and fast execution suitable for real-time classification
- Easily maintainable and extendable backend logic.

4. Features

- **Text preprocessing:** cleans and normalizes user input.
- **Vectorization:** transforms text into numerical format using CountVectorizer
- **Model prediction:** uses a trained Naive Bayes classifier for spam detection.
- Modular code structure for clear separation of logic.

5. Uses

- Processes and prepares input data for the model.
- Executes the prediction logic behind the interface.
- Handles integration with frontend (Streamlit) for smooth data flow.

3. Server

1. Introduction

The server in this system acts as the backbone that connects the frontend interface with the backend logic and machine learning model. It executes user requests, processes input data, runs the prediction algorithm, and returns results to the client interface. In this project, the server functionality is managed by **Streamlit**, which internally handles request-response cycles without needing external web frameworks like Flask or Django. This setup simplifies deployment and supports real-time interaction. The server runs locally during development and can be deployed to platforms like Streamlit Cloud for remote access.

2. Basics

- The server handles the communication between the frontend (Streamlit UI) and backend (Python ML logic).
- It processes input messages and sends them to the machine learning model for prediction.
- Built-in **Streamlit server** is used, eliminating the need for separate web frameworks.

3. Advantages

- Easy to set up with no manual routing or API configuration required.
- Supports real-time data processing and instant output display.
- Lightweight and suitable for small to medium-scale ML applications.

4. Features

- Auto-refreshes UI on user input or code changes.
- Supports interactive sessions without manual page reloads.
- Can be deployed on cloud platforms like Streamlit Cloud or Heroku.

5. Uses

- Accepts message input from users and routes it to the ML pipeline.
- Returns and displays the spam classification result to the frontend.
- Enables smooth end-to-end integration of user interface and backend logic.

CHAPTER 7

OUTPUT SCREENS

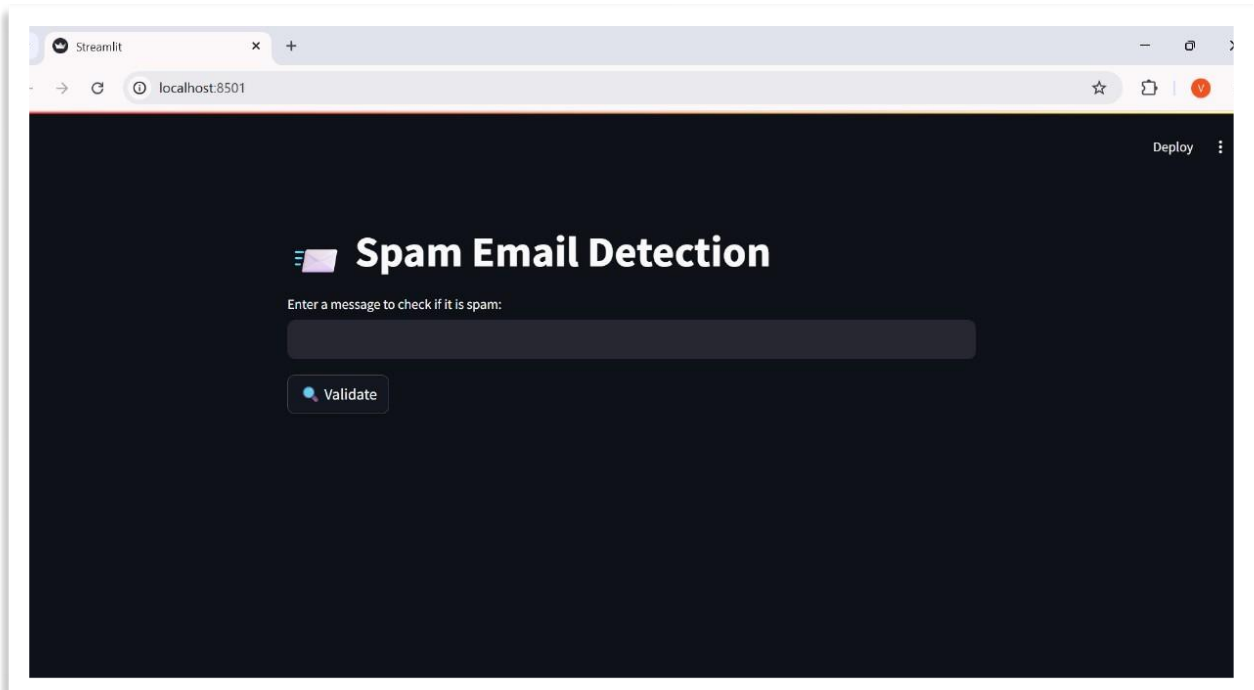


Fig 7.1 :- HOMESCREEN

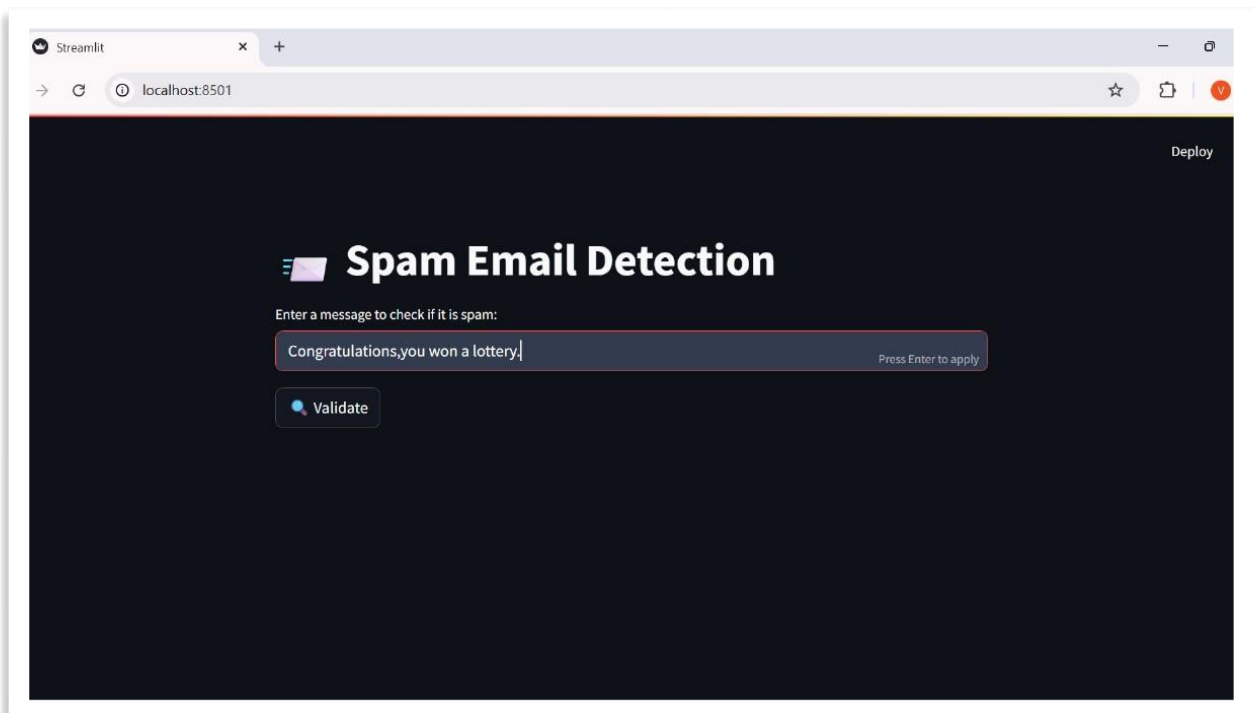


Fig 7.2 :- USER INPUT PAGE

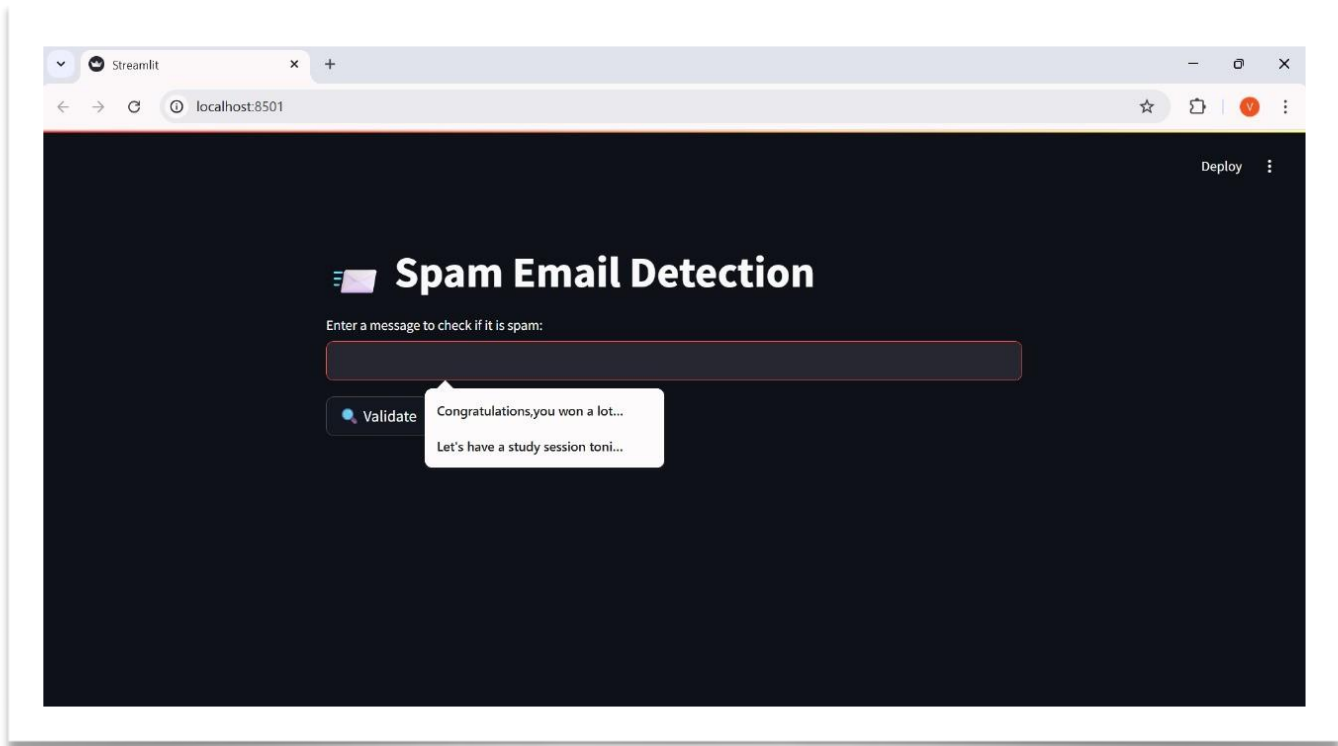


Fig 7.3 :- DROP-DOWN MENU

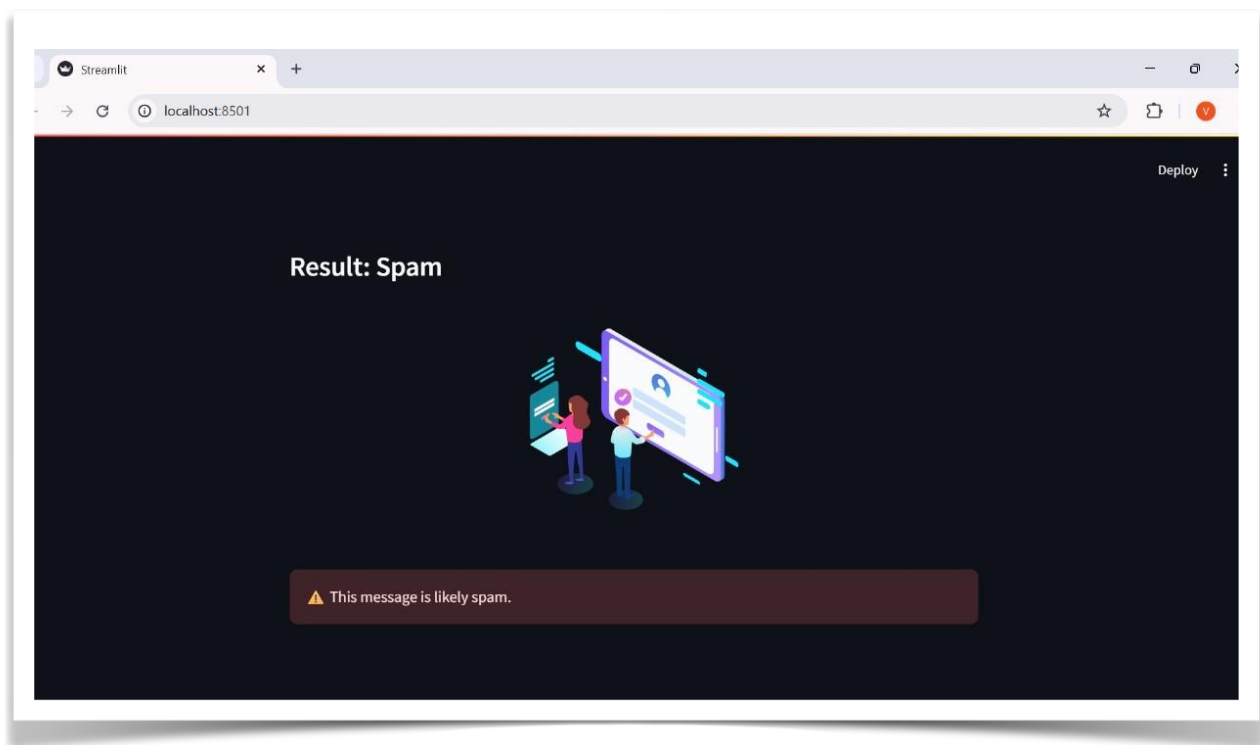


Fig 7.4 :- RESULT PREDICTION

CHAPTER 8

TESTING

1. Introduction

Testing is a critical phase in the development of the Spam Email Detection System to ensure its accuracy, reliability, and performance. Both functional and non-functional aspects of the system were evaluated, including input handling, model predictions, and interface responsiveness. The classification model was tested using a portion of the dataset reserved for validation to assess accuracy and precision. Additionally, the integration between frontend and backend components was verified through manual and unit testing. These tests confirm that the system responds correctly to user input and consistently produces accurate classification results.

2. Strategic Approach to Software Testing

The testing strategy for this system was designed to cover both frontend and backend modules with a focus on:

- **Testability:** The system is designed with isolated functions for preprocessing, vectorization, and prediction, allowing individual components to be easily tested with various inputs.
- **Modularity:** Each part of the system—frontend, backend, and model logic—is separated into independent modules, supporting focused unit and integration testing.
- **Automation:** Python test scripts (using unittest or pytest) enable automated testing of core functions and ensure consistent behavior across updates.
- **Validation:** The system's performance is validated using real dataset samples, checking for prediction accuracy, reliability, and proper integration with the user interface.

This strategy combines **white-box testing** for internal logic validation and **black-box testing** to simulate user interactions and end-to-end functionality.

3. Unit Testing

Unit testing focuses on individual modules or functions within the system to validate correctness and behavior under different conditions.

3.1. White Box Testing

White-box testing involved examining the internal workings of functions such as:

- Input validation checks.
- Data encoding and preprocessing.
- Model prediction functions.

Python's **assert** statements were used in test scripts to verify output correctness.

3.2. Basic Path Testing

This approach was used to validate all independent paths in the code:

- Execution paths from user input to model prediction.
- Handling of invalid/missing inputs.
- Ensuring a correct path even if optional features (like animation) fail.

3.3. Conditional Testing

All condition-based logic was tested, such as:

- ✓ Evaluates decision-making code blocks such as checking if the input text is empty or invalid before processing.
- ✓ Ensures the system branches correctly—e.g., returning an error or prompt if no input is given.
- ✓ Verifies all possible true/false outcomes of conditions (e.g., if `len(text) > 0`;) are tested.

These were tested to ensure proper fallback behavior and user alerts.

3.4. Data Flow Testing

Variables were traced to ensure proper data movement:

- ✓ Tracks how data (email text) is defined, used, and transformed throughout functions like `preprocess text()`, `vectorize()`, and `predict()`.
- ✓ Ensures variables are properly initialized before use and not used after being destroyed or left undefined.
- ✓ Helps catch issues like using outdated vector objects or model variables that haven't been updated.

3.5. Loop Testing

Loops in data preprocessing and prediction logic (if any, such as one-hot encoding loops or feature mapping) were tested for:

- ✓ Applies to loops within preprocessing steps, such as iterating through words to remove stopwords or punctuation.
- ✓ Tests zero iterations (e.g., empty input), one iteration (minimal input), and multiple iterations (long text).
- ✓ Verifies that loops terminate correctly and handle edge cases without skipping or crashing.

CHAPTER 9

SYSTEM SECURITY

1. Introduction

In today's digital landscape, system security is an essential aspect of any software application. For this system, ensuring the protection of data and maintaining the reliability of system operations are critical, even in a small-scale predictive application. This includes implementing mechanisms to safeguard the system from unauthorized access, data breaches, or unintended misuse.

1.1. System Security

System security ensures that unauthorized access to the spam detection platform is prevented. Basic input validation and secure backend design help reduce exposure to threats like script injection or system misuse. Streamlit's limited execution scope adds a layer of protection by restricting backend access.

1.2. Data Security

Data security involves protecting user-provided email text from being intercepted or altered during processing. The system does not store data permanently, reducing the risk of data leaks. Temporary in-memory processing ensures a low surface area for attacks.

1.3. System Integrity

System integrity is maintained by ensuring the application functions as expected without unauthorized modifications. Model files, preprocessing functions, and vectorizers are kept consistent and unchanged during deployment. Controlled backend logic prevents manipulation or corruption of core components.

1.4. Privacy

Privacy is preserved by not storing user inputs or outputs and ensuring data is processed only during active sessions. No logs or identifiable information are retained by the system. The design supports user trust by respecting anonymity.

1.5. Confidentiality




Confidentiality ensures that user inputs remain inaccessible to external systems or unauthorized individuals. All email text inputs are processed locally during the session without being transmitted or saved externally. This safeguards sensitive communication content shared for classification.

2. Security Software




The system employs a combination of **client-side** and **server-side validations** to enhance security and minimize vulnerabilities:

2.1. Client-Side Validation

Performed through Streamlit's widget configuration:




-  Ensures the user enters valid, non-empty email text before sending it to the backend for prediction.
-  Uses JavaScript or built-in Streamlit components to prevent submission of blank or malformed input.
-  Reduces unnecessary server load by catching basic errors early in the browser.

Advantages:




-  **Improves Performance:** Reduces round trips to the server by handling simple checks on the client side.
-  **Enhances User Experience:** Provides instant feedback, helping users correct mistakes immediately.
-  **Adds a Layer of Security:** Prevents some malformed or malicious data from reaching the backend.

2.2. Server-Side Validation

Server-side checks in Python ensure:

-  Validates the input text again after submission to ensure it is not empty, corrupted, or malicious.
-  Sanitizes and processes the input on the backend before passing it to the machine learning model.
-  Acts as a final gatekeeper to prevent invalid or harmful data from affecting system functionality.

Advantages:

-  **Stronger Security:** Protects against manipulation or bypassing of client-side checks.
-  **Reliable Input Handling:** Ensures all data passed to the model meets required conditions and format.
-  **Error Management:** Helps log and manage incorrect or harmful input for debugging and improvement.









CHAPTER 10

1. Conclusion

The Spam Email Detection System successfully demonstrates the application of machine learning for classifying emails as spam or not spam. By combining data preprocessing, feature extraction, and a Naive Bayes classification model, the system achieves reliable and efficient results. The frontend interface built with Streamlit allows users to interact seamlessly with the backend logic. Security measures, validation techniques, and testing strategies ensure system reliability and robustness. The modular design enables easy maintenance and future enhancements. Overall, the project highlights the practical use of AI in solving real-world communication challenges. This system serves as a foundational step toward more advanced and scalable spam filtering solutions.

2. Benefits

The project delivers several practical and technical benefits:

-  **Efficient Spam Filtering:** Accurately detects and filters out spam messages, reducing inbox clutter.
-  **Automation:** Eliminates the need for manual email classification, saving time and effort.
-  **Data-Driven Decisions:** Uses machine learning for smarter and more adaptive spam detection.
-  **Improved Accuracy:** Leverages trained models to minimize false positives and negatives.
-  **User-Friendly Interface:** Streamlit-based interface allows easy interaction without technical knowledge.
-  **Secure and Private:** Processes data without storing or exposing sensitive information.
-  **Scalable Design:** Modular codebase allows easy updates, improvements, and integration into larger systems.
-  **Tested and Reliable:** Rigorous testing ensures robustness and stability under different inputs.

References

1. Scikit-learn: Machine Learning in Python — <https://scikit-learn.org/>
2. Pandas Documentation — <https://pandas.pydata.org/docs/>
3. NumPy Documentation — <https://numpy.org/doc/>
4. Streamlit Documentation — <https://docs.streamlit.io/>
5. Python Official Documentation — <https://docs.python.org/3/>
6. UCI Machine Learning Repository: SMS Spam Collection Dataset — <https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection>
7. "Machine Learning" by Tom M. Mitchell, McGraw-Hill Education
8. Stack Overflow Community — <https://stackoverflow.com/>
9. Towards Data Science (Spam Detection tutorials and articles) — <https://towardsdatascience.com/>