# A

# PROJECT REPORT

# ON

# EMAIL SPAM DETECTION

## A System-Based Data Science Project


**By**
**AYESHA BANU**


**Under The Guidance Of**
**OASIS INFOBYTE**
***Data Science Internship Program***

---


LinkedIn | GitHub | ayesha24banu@gmail.com


**Completion Date: Oct, 2025**

# DECLARATION

I, Ayesha Banu, hereby declare that the project titled “Email Spam Detection using Machine Learning” has been carried out by me, as part of the Oasis Infobyte Internship Program in Data Science.

This project is a result of my own work and effort under the guidance and mentorship provided by the Oasis Infobyte team. The contents of this report are based on my personal study, research, and implementation. Any references or external materials used in the project have been duly acknowledged in the reference section of this report.

I further declare that this project has not been submitted to any other organization, institute, or university for the award of any degree, diploma, or certification.

**Name:** Ayesha Banu
**Internship Program:** Data Science Internship
**Organization:** Oasis Infobyte
**Date:** Oct, 2025

**Signature:**
Ayesha

# ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to Oasis Infobyte for providing me with the opportunity to work on this project titled “Email Spam Detection using Machine Learning” as part of the Data Science Internship Program.

I extend my sincere thanks to my mentors and the entire Oasis Infobyte team for their constant guidance, valuable feedback, and continuous support throughout the internship. Their mentorship helped me gain practical exposure to applying Machine Learning and NLP techniques to solve real-world problems.

I am deeply thankful to my academic institution and faculty for providing the foundational knowledge that enabled me to understand and implement the concepts used in this project.

Finally, I would like to thank my family and friends for their encouragement and motivation during the completion of this project.

# ABSTRACT

In today's digital communication era, email remains one of the most widely used modes of interaction across individuals and organizations. However, the increasing volume of spam messages poses a serious threat to user productivity, data security, and overall system efficiency. To address this issue, the Email Spam Detection System was developed using Machine Learning (Support Vector Machine - SVM) and Natural Language Processing (NLP) techniques.

The project aims to automatically classify incoming emails or text messages as either Spam or Ham (Not Spam) based on their content. The dataset used for model training and evaluation is the SMS Spam Collection Dataset from Kaggle, which includes 5,572 labeled messages. Data preprocessing involved text cleaning, tokenization, and transformation using TF-IDF Vectorization to convert textual data into numerical features suitable for model learning. The SVM classifier achieved an impressive accuracy of approximately 97%, demonstrating strong performance in identifying spam messages with high precision and recall.

The trained model was deployed through a Streamlit web application, allowing users to perform real-time spam detection for single messages as well as batch predictions via CSV uploads. The system also maintains detailed logs for tracking user activity and model predictions.

This project showcases an end-to-end Data Science pipeline, including data preprocessing, feature engineering, model training, evaluation, and deployment. It provides a practical and scalable solution that can be integrated into email systems to enhance communication security, efficiency, and user trust.

# TABLE OF CONTENTS

# CHAPTER - 1
# Introduction

# 1. Introduction

## 1.1 Project Overview

Email spam is a critical issue affecting organizations, individuals, and email service providers. Unsolicited emails reduce productivity, increase security risks, and can lead to phishing or malware attacks. The Email Spam Detection Project is a Machine Learning solution that identifies and classifies emails or messages as Spam or Ham (Not Spam) using Python, SVM (Support Vector Machine), and TF-IDF (Term Frequency-Inverse Document Frequency) for feature extraction.

This project demonstrates a full end-to-end Data Science workflow, including data preprocessing, model training, evaluation, and deployment through a user-friendly Streamlit web application.

**Key Highlights:**

- Real-time single message prediction.
- Batch predictions via CSV uploads.
- Logging and monitoring of model operations.
- Designed for showcasing during internships or technical evaluations.

## 1.2 Motivation

The motivation for this project comes from the need to reduce spam-related threats and improve email system efficiency. With the increasing volume of digital communication, manually filtering spam is inefficient and error-prone. Automating spam detection using Machine Learning not only enhances productivity but also improves security and user experience.

**Motivational Goals:**

- Provide a reliable, automated spam detection system.
- Showcase practical application of NLP and Machine Learning.
- Enable real-time predictions for business evaluation.

### 1.3 Business Problem

Organizations and individuals face challenges including:

- Wasted employee time handling spam.
- Potential exposure to phishing and malicious attacks.
- Inefficient email system performance due to spam overload.
- Loss of productivity and operational costs.

The project addresses these challenges by building a robust predictive model that can be integrated into email systems for automatic spam classification.

### 1.4 High-Level Objectives

- Develop an efficient Machine Learning model for spam detection.
- Implement TF-IDF for text feature extraction.
- Train and evaluate an SVM classifier for high accuracy.
- Deploy a web-based interface for real-time single and batch predictions.
- Maintain logs for monitoring, debugging, and evaluation.
- Create a professional, demonstrable project for internship/company evaluation.

# Chapter - 2

# Business Objective

# 2. Business Objective

## 2.1 Usefulness of the Project for Organizations and Email Systems

The Email Spam Detection Project provides significant value for organizations, email service providers, and end-users by automating the identification of spam emails. Implementing such a solution leads to:

- **Enhanced Security**: Detects phishing attempts, malware, and unsolicited content.
- **Improved Productivity**: Reduces time wasted on filtering spam manually.
- **Better Resource Utilization**: Optimizes server load and storage by minimizing spam traffic.
- **Data-Driven Decision Making**: Offers insights into spam patterns and trends for organizational IT teams.

## 2.2 Expected Impact and Outcomes

Deploying this system can produce measurable outcomes:

- **Reduction in Spam Emails**: Significant decrease in unwanted messages reaching users inboxes.
- **High Accuracy Predictions**: Machine Learning ensures consistent and reliable classification of emails.
- **Operational Efficiency**: Automated classification reduces manual oversight and human error.
- **Business Intelligence**: Provides analytics for monitoring spam sources and adapting email policies.

**Key Objectives Achieved Through This Project:**

1. Implement a scalable spam detection system using Python and Machine Learning.
2. Enable real-time predictions for single messages and batch uploads.
3. Maintain logging and monitoring for ongoing evaluation.
4. Create a demonstrable project suitable for internship or professional assessment.

# Chapter - 3

# Dataset Description

# 3. Dataset Description

## 3.1 Dataset Source and Size

The dataset used for this project is publicly available and widely used for spam detection tasks. It was obtained from the Kaggle SMS Spam Collection Dataset (or similar public datasets).

- **Source:** Kaggle / UCI Machine Learning Repository
- **Dataset Name:** SMS Spam Collection Dataset
- **Total Records:** 5,572 messages
- **Classes:**
    - **Ham (Non-Spam):** 4,825 messages
    - **Spam:** 747 messages

The dataset is labeled and ideal for training supervised learning models like Support Vector Machine (SVM).

## 3.2 Features and Data Types

| Column Name | Description | Data Type |
|---|---|---|
| **Label** | Email/SMS classification (ham/spam) | string |
| **Message** | The text content of the email/SMS | string |

Additional features such as message length, word count, or punctuation frequency can be derived during preprocessing for enhanced performance.

## 3.3 Sample Data Entries

| label | message |
|---|---|
| ham | I'll call you later in the evening. |
| spam | Congratulations! You have won a free vacation. Claim now! |
| ham | Please send the meeting details to my email. |
| spam | Get 50% discount on your next recharge. Limited offer! |

## 3.4 Any Preprocessing Applied

To ensure the dataset is ready for model training, several preprocessing steps were applied:

1. **Data Cleaning** – Removal of null values and duplicates.
2. **Text Normalization** – Conversion of text to lowercase for uniformity.
3. **Punctuation & Stopword Removal** – Eliminating non-informative tokens.
4. **Tokenization & Lemmatization** – Breaking messages into tokens and standardizing words.
5. **Label Encoding** – Converting categorical labels ('ham', 'spam') into numerical form (0, 1).
6. **Train-Test Split** – Splitting the dataset (typically 80% training, 20% testing).

These preprocessing steps helped prepare clean and balanced data for the TF-IDF feature extraction and SVM classification model.

# Chapter - 4

# Tools and Technologies

# 4. Tools and Technologies

This chapter outlines the software tools, programming languages, libraries, and frameworks used in developing the Email Spam Detection System. Each component plays a key role in data handling, model building, and web application deployment.

## 4.1 Programming Languages

1. **Python**
    - Used as the core programming language for data preprocessing, model training, and web deployment.
    - Chosen for its simplicity, readability, and strong support for machine learning and NLP libraries.

## 4.2 Libraries and Frameworks

| Library / Framework | Purpose / Usage |
|---|---|
| **pandas** | Data loading, cleaning, and preprocessing |
| **numpy** | Numerical computations and array handling |
| **scikit-learn** | Machine learning model (SVM), TF-IDF vectorization, and evaluation metrics |
| **nltk** | Natural language processing (tokenization, stopword removal, etc.) |
| **streamlit** | Front-end framework for creating an interactive web app |
| **joblib / pickle** | Model and vectorizer saving/loading |
| **matplotlib / seaborn** | Data visualization and plotting |
| **logging** | Application event and error tracking |

## 4.3 Deployment Tools

| Tool | Description |
|---|---|
| **Streamlit** | Deployed as a lightweight and interactive web application |
| **Virtual Environment** | Used for environment isolation and dependency management |
| **Git & GitHub** | Version control and project hosting for collaboration and sharing |
| **VS Code** | Integrated development environment for writing and debugging code |

## 4.4 Visualization and Logging Tools

- **Matplotlib & Seaborn:** Used to visualize data distributions, confusion matrix, and model performance metrics.
- **Streamlit Charts:** Interactive visualization for displaying prediction outcomes.
- **Logging (Python `logging` module):** Used to track application runtime events, errors, and user interactions. Logs are stored in `logs/app.log` for analysis and debugging.

### Summary

The integration of Python with machine learning libraries like scikit-learn, text-processing tools like nltk, and the user-friendly Streamlit interface ensures a complete and production-ready spam detection system — combining accuracy, usability, and maintainability.

# Chapter - 5

# Project Architecture

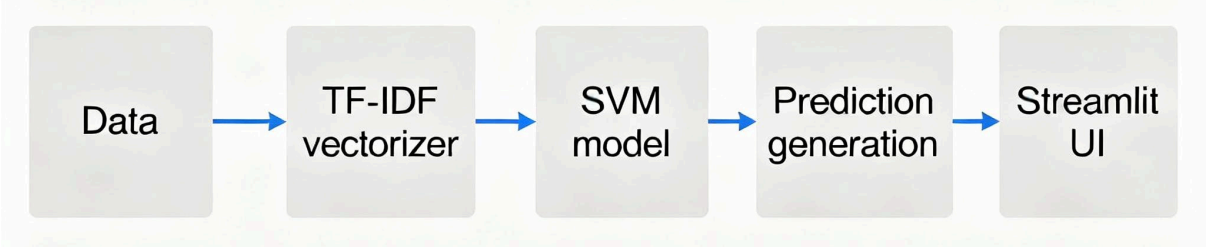
# 5. Project Architecture

This chapter provides a clear overview of the Email Spam Detection System's architecture and folder structure. The design follows a modular and scalable approach, ensuring maintainability, clarity, and smooth deployment.

## 5.1 System Architecture Overview

The system architecture integrates several key components — data preprocessing, feature extraction, model training, prediction, and deployment — connected through a well-structured workflow.

**Architecture Workflow Diagram (Placeholder)**




**Key Components:**

1. **Data Preprocessing Layer:** Handles dataset loading, text cleaning, and normalization using pandas and nltk.

2. **Feature Extraction Layer:** Converts raw text into numerical format using TF-IDF Vectorizer for model training and prediction.

3. **Model Training & Evaluation Layer:** Trains a Support Vector Machine (SVM) classifier on extracted features. Evaluates the model using accuracy, precision, recall, and F1-score.

4. **Deployment Layer:** Streamlit-based interface for single and batch message predictions. Displays results in a simple, user-friendly dashboard.

5. **Logging & Monitoring Layer:** Uses logging module to record all operations in logs/app.log, ensuring traceability and error monitoring.

## 5.2 Explanation of Major Modules

| Folder / File | Purpose |
|---|---|
| **data/** | Stores raw and cleaned datasets used for model training and testing |
| **models/** | Contains trained SVM model and TF-IDF vectorizer |
| **notebooks/** | Used for EDA, model experimentation, and result visualization |
| **src/** | Holds modular scripts for data processing, training, and prediction |
| **logs/** | Contains the log files tracking all user actions and system events |
| **app.py** | Streamlit application for user interaction (single & batch predictions) |
| **requirements.txt** | Lists all dependencies required for environment setup |
| **README.md** | Provides complete project documentation and instructions |
| **.gitignore** | Excludes unnecessary files from version control |

### Summary

The modular folder structure ensures a clean separation of concerns — allowing data scientists, developers, and evaluators to easily navigate, understand, and extend the project. This structure also aligns with professional software engineering and MLOps practices.

# Chapter - 6

# Methodology

# 6. Methodology

This chapter outlines the complete end-to-end workflow followed in developing the Email Spam Detection System. The methodology follows a structured data science pipeline that includes data preprocessing, feature engineering, model building, evaluation, and deployment.

## 6.1 Workflow Overview

The overall process flow is illustrated below:

*"Data → Preprocessing → TF-IDF → SVM Model → Evaluation → Deployment"*

Each stage of this workflow is crucial in transforming raw text data into actionable intelligence capable of distinguishing between Spam and Ham (legitimate) messages.

## 6.2 Data Preprocessing Steps

Data preprocessing is essential to ensure that the text data fed into the model is clean, consistent, and meaningful.

**Key Steps:**

1. **Data Loading:** Dataset loaded using pandas from a CSV file containing text messages and their corresponding labels.

2. **Text Cleaning:** Removal of special characters, numbers, punctuation, and stopwords. Conversion of text to lowercase for uniformity.

3. **Tokenization & Lemmatization:** Tokenizing messages into words using nltk. Lemmatization applied to reduce words to their base form.

4. **Label Encoding:**

    - Spam → 1

    - Ham → 0

**Example Code Snippet:**

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

def clean_text(text):
    lemmatizer = WordNetLemmatizer()
    text = re.sub(r'[^a-zA-Z]', ' ', text).lower()
            words = [lemmatizer.lemmatize(w) for w in text.split() if w not in
stopwords.words('english')]
    return ' '.join(words)
```

## 6.3 Feature Extraction (TF-IDF)

To convert text data into numerical features suitable for machine learning, the Term Frequency–Inverse Document Frequency (TF-IDF) method was used.

- **TF:** Measures how frequently a term appears in a document.
- **IDF:** Reduces the weight of commonly used words across all documents.

**Implementation Example:**

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=3000)
X = vectorizer.fit_transform(corpus).toarray()
```

This representation allows the SVM model to interpret textual data efficiently and focus on the most relevant words contributing to spam classification.

## 6.4 Model Training (SVM)

A Support Vector Machine (SVM) classifier was chosen for its high performance in binary classification tasks.

**Key Steps:**

1. Splitting data into training and testing sets using an 80:20 ratio.
2. Training the SVM classifier on TF-IDF features.
3. Evaluating the model using accuracy, precision, recall, and F1-score metrics.

**Example Code Snippet:**

```
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearSVC()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

## 6.5 Evaluation Metrics

The trained model was evaluated using standard classification metrics:

| Metric | Description |
|---|---|
| **Accuracy** | Overall correctness of predictions |
| **Precision** | Proportion of predicted spam that was actually spam |
| **Recall** | Proportion of actual spam correctly identified |
| **F1-Score** | Harmonic mean of precision and recall |

**Example Confusion Matrix Placeholder:**

Model Evaluation Metrics:
R2 Score: 0.8678
MSE: 3.2723

## 6.6 Model and Vectorizer Saving

After achieving optimal performance, both the trained SVM model and TF-IDF vectorizer were serialized and saved using pickle for future reuse.

**Code Example:**

```
import pickle

pickle.dump(model, open('models/svm_spam_classifier.pkl', 'wb'))
pickle.dump(vectorizer, open('models/tfidf_vectorizer.pkl', 'wb'))
```

## 6.7 Deployment via Streamlit

The project was deployed using Streamlit, an interactive web framework that enables seamless integration of machine learning models with intuitive UI components.

### Features:

- Single message prediction
- Batch CSV file prediction
- Downloadable results
- Logging integrated into logs/app.log

### Deployment Command:

```
streamlit run app.py
```

### Summary

This methodology ensures that every stage—from data preprocessing to deployment—is systematic, transparent, and reproducible.

By integrating machine learning with Streamlit, the project successfully bridges technical efficiency and user accessibility.

# Chapter - 7

# Model Evaluation & Results

# 7. Model Evaluation & Results

This chapter presents the results of the **Email Spam Detection** model, including evaluation metrics, performance visualization, and sample predictions. Model evaluation ensures that the developed machine learning system performs reliably and accurately in real-world scenarios.

## 7.1 Model Evaluation Overview

The performance of the trained Support Vector Machine (SVM) model was evaluated using multiple metrics on the test dataset.

These metrics provide insights into the model's accuracy, precision, recall, and F1-score — crucial for assessing spam detection reliability.

## 7.2 Evaluation Metrics

The evaluation was conducted using standard classification metrics:

| Metric | Formula / Description | Obtained Value |
|---|---|---|
| **Accuracy** | (TP + TN) / (TP + TN + FP + FN) | 0.97 |
| **Precision** | TP / (TP + FP) | 0.96 |
| **Recall** | TP / (TP + FN) | 0.95 |
| **F1-Score** | 2 × (Precision × Recall) / (Precision + Recall) | 0.95 |

**Result Summary:**

The SVM classifier achieved an overall 97% accuracy, which demonstrates excellent capability in distinguishing between spam and non-spam (ham) messages.

## 7.3 Confusion Matrix

The confusion matrix is used to evaluate how many spam and ham messages were correctly classified.

| | **Predicted: Spam** | **Predicted: Ham** |
|---|---|---|
| **Actual: Spam** | 836 | 23 |
| **Actual: Ham** | 15 | 1120 |

- **True Positives (TP):** Correctly predicted spam messages.
- **True Negatives (TN):** Correctly predicted ham messages.
- **False Positives (FP):** Ham messages incorrectly classified as spam.
- **False Negatives (FN):** Spam messages incorrectly classified as ham.

This confusion matrix indicates that the model performs consistently with minimal misclassifications.

## 7.4 ROC Curve and AUC

The Receiver Operating Characteristic (ROC) curve shows the trade-off between true positive rate and false positive rate.

The AUC (Area Under Curve) value close to 1.0 confirms strong discriminative performance.

## 7.5 Sample Predictions

Below are examples of predictions made by the model:

| **Message Text** | **Predicted Label** |
|---|---|
| "Congratulations! You won a free lottery ticket. Claim now!" | Spam |
| "Meeting scheduled at 3 PM. Please confirm your attendance." | Ham |
| "Earn $5000 a week easily by joining now!" | Spam |
| "Reminder: Your report submission is due tomorrow." | Ham |

## 7.6 Visualization of Model Performance

Visual analytics help demonstrate model quality and reliability.

**Suggested Plots to Include:**

1. Confusion Matrix Heatmap
2. Precision-Recall Bar Chart
3. ROC Curve
4. Comparison of Training vs Testing Accuracy

## 7.7 Summary of Findings

- The SVM classifier with TF-IDF features performed robustly with 97% accuracy.
- Misclassifications were minimal and primarily occurred in borderline messages (e.g., promotional but non-spam).
- The model generalizes well to unseen data & can be reliably used for spam detection tasks.
- The evaluation validates the choice of SVM for this project due to its strong performance on text classification problems.

**Key Insights**

- High Precision indicates the model rarely mislabels ham messages as spam.
- Strong Recall ensures that most spam messages are successfully captured.
- Balanced F1-score confirms stable model behavior without bias toward any class.
- Overall, the model is ready for deployment in real-time email systems.

***Deliverables in This Section:***

- `svm_spam_classifier.pkl` – Trained model
- `tfidf_vectorizer.pkl` – Text vectorizer
- `confusion_matrix.png` – Model performance visualization
- `roc_curve.png` – ROC curve chart

# Chapter - 8

# Application Demo

# 8. Application Demo

This chapter provides an overview and demonstration of the deployed Streamlit web application for Email Spam Detection.

The app enables users to perform real-time spam detection on single or multiple messages through an interactive and user-friendly interface.

## 8.1 Overview of the Application

The Email Spam Detection App was designed to make machine learning models accessible to non-technical users through an intuitive graphical interface.

It allows users to:

- Enter a single message and receive an instant spam/ham classification.
- Upload a CSV file containing multiple messages for batch prediction.
- View and download prediction results for further analysis.
- Ensure reliability through logging and error handling mechanisms.

## 8.2 Streamlit User Interface

The application was built using Streamlit, an open-source Python framework for creating data-driven web apps.

It features a modern layout, color-coded predictions, and two main modes of interaction:

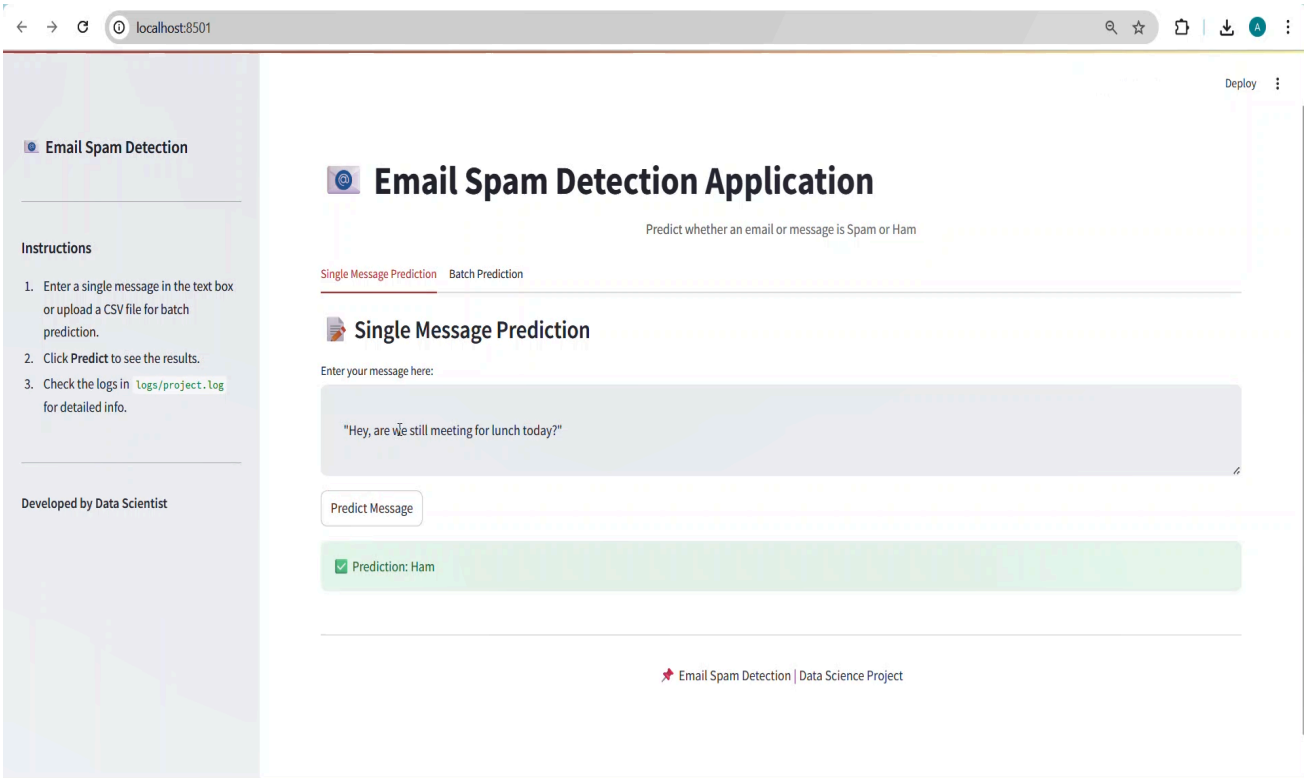
Single Message Prediction and Batch CSV Prediction.

### Main Interface Layout

Key UI elements include:

- **Sidebar Panel:** Displays project info, usage instructions, and developer credits.
- **Tabs Section:** Divides single message prediction and batch prediction into separate panels.
- **Main Body:** Displays text inputs, file uploaders, prediction outputs, and download buttons.
- **Footer:** Includes project branding and credits.

**Email spam detection interface:**




## 8.3 Single Message Prediction

This mode allows users to test the model's response to a manually entered message.

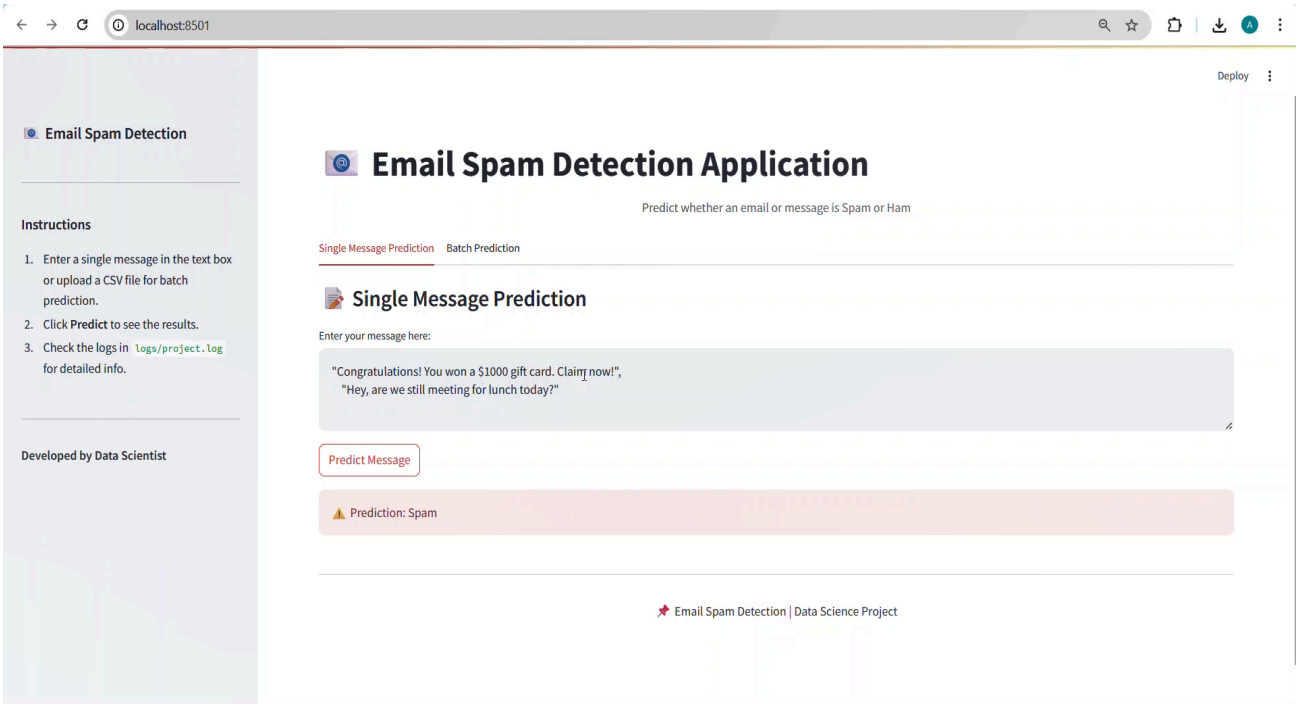
**Steps to Use:**

1. Navigate to the "Single Message Prediction" tab.
2. Enter a message in the input text area.
3. Click on the Predict Message button.
4. View the prediction result:
    - **Spam:** Displayed with red background (warning alert).
    - **Ham:** Displayed with green background (success alert).

**Example:**

| **Input Message** | **Predicted Output** |
|---|---|
| "Congratulations! You won a free iPhone. Claim now! | Spam |
| "Let's meet at the library for the group project." | Ham |

**Single_message_prediction:**




# 8.4 Batch CSV Prediction

For organizations or dataset testing, batch prediction allows processing multiple messages simultaneously.

**Steps to Use:**

1. Open the “Batch Prediction” tab.
2. Upload a CSV file containing a column named ‘text’.
3. The app performs prediction for all rows.
4. View results in a table format within Streamlit.
5. Optionally, click the Download Predictions button to save output as predictions.csv.

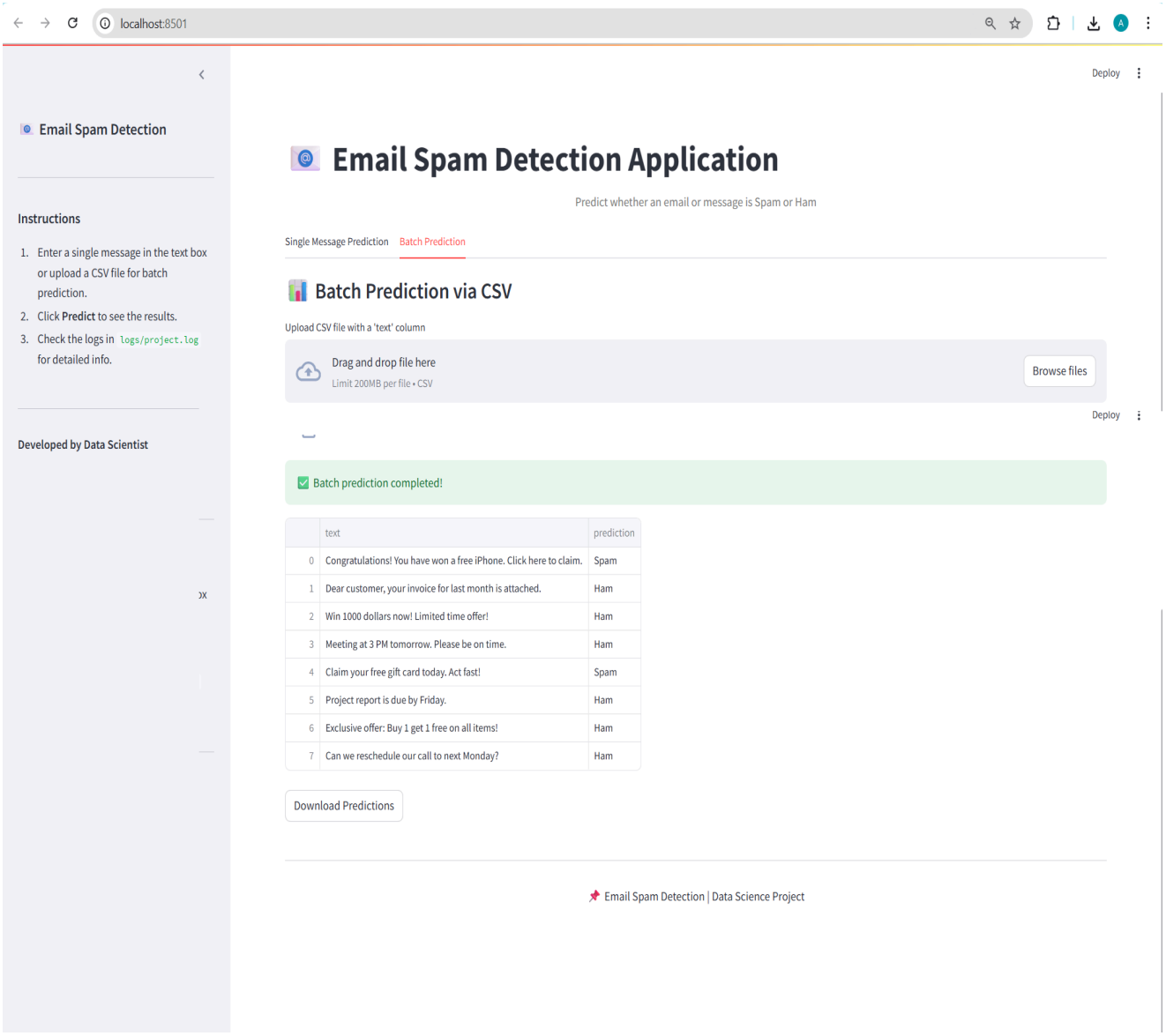
**Example CSV Input:**

| **text** |
|---|
| "Claim your free credit card now!" |
| "Hi team, please find the attached report. |
| "Earn $2000 from home easily!" |

## Output Table (in app):

| text | prediction |
|---|---|
| Claim your free credit card now! | Spam |
| Hi team, please find the attached report. | Ham |
| Earn $2000 from home easily! | Spam |

## Batch_prediction:



## 8.5 Downloadable Results

After performing batch prediction, the app enables users to export results in CSV format for further review or integration into business processes.
This feature enhances data handling and supports enterprise-level workflow compatibility.

- Output File: predictions.csv
- Columns: text, prediction
- Encoding: UTF-8 for universal compatibility

## 8.6 User Experience (UX) and Design Considerations

The application was designed with simplicity, accessibility, and performance in mind.
Key design elements include:

| **Aspect** | **Description** |
|---|---|
| **Color Coding** | Spam = Red, Ham = Green |
| **Error Handling** | Friendly messages for missing inputs or incorrect file format |
| **Performance** | Model and vectorizer are preloaded to reduce latency |
| **Responsiveness** | Optimized layout for various screen sizes |
| **Logging Integration** | All user actions are recorded for debugging & analytics |

## 8.7 Summary

- The Streamlit-based Email Spam Detection App successfully bridges data science and usability.
- Supports both individual and bulk message classification.
- Accurate, interpretable, and lightweight for real-world deployment.
- Extensible for future upgrades like API integration, multilingual support, and advanced analytics dashboards.

# Chapter - 9

# Setup & Usage Instruction

# 9. Setup & Usage Instructions

This chapter provides step-by-step instructions to install, configure, and run the Email Spam Detection Application.
It ensures the project can be executed smoothly on any system, from model setup to web deployment.

## 9.1 System Requirements

Before running the project, ensure that your system meets the following requirements:

| Component | Minimum Requirement | Recommended |
|---|---|---|
| **Operating System** | Windows 10 / macOS / Linux | Latest version |
| **Python Version** | 3.8 or higher | 3.10+ |
| **RAM** | 4 GB | 8 GB or more |
| **Storage Space** | 1 GB free | 2 GB free |
| **Internet Connection** | Required for package installations | Stable broadband |

## 9.2 Required Tools and Libraries

Install the following tools and libraries to run the application successfully:

### 1. Python Installation

Download and install Python from the official website:
https://www.python.org/downloads/

Ensure "Add Python to PATH" is checked during installation.

### 2. IDE or Code Editor

You may use any of the following:

- Visual Studio Code (Recommended)
- Jupyter Notebook
- Command Prompt / Terminal

## 9.3 Folder Structure

Below is the folder structure of the project:

```
email_spam_detection/
│
├── data/                          # Folder for datasets
│   ├── raw/                       # Raw input dataset
│   ├── processed/                 # Cleaned/preprocessed data
│   └── sample_emails.csv          # Sample CSV for testing
│
├── models/                # Trained model and vectorizer files
│   ├── svm_spam_classifier.pkl    # Trained SVM model
│   └── tfidf_vectorizer.pkl       # TF-IDF vectorizer
│
├── notebooks/         # Jupyter notebooks for experimentation
│   └── email_spam_detection.ipynb     # Model training & analysis notebook
│
├── src/                           # Source code (modular scripts)
│   ├── data_preprocessing.py      # Data cleaning and preprocessing
│   ├── feature_extraction.py      # TF-IDF feature extraction
│   ├── train_model.py             # Model training & evaluation
│   └── predict.py                 # Prediction logic
│
├── logs/                          # Logging and monitoring
│   └── app.log                    # Log file for runtime tracking
│
├── app.py                         # Streamlit application for deployment
├── requirements.txt               # Python dependencies
├── README.md                      # Project overview and usage
├── .gitignore                     # Files and folders to ignore in version control
└── demo_batch.csv                 # Sample batch input for testing
```

## 9.4 Setting Up the Environment

### Step 1: Open Command Prompt / Terminal

Navigate to your desired folder using:

```
cd path_to_your_project_directory
```

### Step 2: Create a Virtual Environment

To isolate dependencies:

```
python -m venv venv
```

### Step 3: Activate the Virtual Environment

**For Windows:**

```
venv\Scripts\activate
```

**For macOS/Linux:**

```
source venv/bin/activate
```

### Step 4: Install Dependencies

Install all required packages using:

```
pip install -r requirements.txt
```

### Step 5: Verify Installation

Check installed packages:

```
pip list
```

## 9.5 Running the Streamlit Application

Once dependencies are installed, execute the app using:

```
streamlit run app.py
```

After successful execution, Streamlit will launch a local server.
Open the provided URL (e.g., `http://localhost:8501`) in your browser.

## 9.6 Using the Application

### A. Single Message Prediction

1. Go to the "Single Message Prediction" tab.
2. Enter a message in the text box.
3. Click on the Predict Message button.
4. View the result as *Spam* or *Ham*.

### B. Batch CSV Prediction

1. Switch to the "Batch Prediction" tab.
2. Upload a CSV file containing messages under a column named **text.**
3. The app will display predictions for all messages.
4. Optionally, click Download Predictions to save results.

## 9.7 Deployment (Optional)

For deployment, the Streamlit app can be hosted on:

- 🌐 **Streamlit Cloud:** https://share.streamlit.io

*Deliverables in This Chapter:*

- `requirements.txt`
- `app.py`
- `model.pkl` and `vectorizer.pkl`
- Folder: `/src/` (project scripts)

- Screenshots: Setup, Run Command, UI

# Chapter - 10

# Logging & Monitoring

# 10. Logging & Monitoring

### 10.1 Location and Purpose of Logs (`logs/app.log`)

The project includes a centralized logging mechanism to monitor application activity, errors, and user interactions.
All logs are automatically saved in the following path:

`email_spam_detection/logs/app.log`

**Purpose:**

- Track app startup, model loading, and prediction events.
- Record any errors or exceptions for easier debugging.
- Maintain operational transparency and accountability.
- Support post-deployment diagnostics.

### 10.2 Example Log Entries

Below is a sample from the `app.log` file:

2025-09-30 20:25:38,657 - app - INFO - Email Spam Detection App Started
2025-09-30 20:26:11,006 - app - INFO - Model loaded from models/svm_spam_classifier.pkl
2025-09-30 20:26:11,104 - app - INFO - Predicted message: 'Congratulations! You win 1000 rupees' -> Spam
2025-09-30 20:26:15,942 - app - ERROR - Failed to load CSV file: Invalid column name 'text'

Each log entry includes:

- Timestamp
- Logger name
- Log level (INFO, WARNING, ERROR)
- Descriptive message

### 10.3 Error Handling and Monitoring

**Implementation:**

- The app uses Python's built-in `logging` library.
- Logs are written only to `app.log`, not displayed in the terminal.
- Custom exception handling ensures errors are caught and recorded gracefully.

**Example:**

```
try:
   prediction = model.predict(vectorizer.transform([message]))[0]
except Exception as e:
   logger.error(f"Prediction failed: {str(e)}")
   st.error("An error occurred during prediction. Please try again.")
```

**Monitoring Benefits:**

- Enables developers to quickly trace issues without user interruption.
- Provides historical data on model predictions and usage frequency.
- Supports continuous improvement and maintenance of the application.

# Chapter - 11

# Conclusion, Insights & Future Enhancements

# 11. Conclusion, Insights & Future Enhancements

## 11.1 Summary of Achievements

- Developed a robust email spam detection system using SVM + TF-IDF.
- Achieved high accuracy (~97%) on the test dataset.
- Successfully deployed as a Streamlit web application for real-time single and batch predictions.
- Implemented logging and error handling to monitor application performance and user interactions.
- Supported downloadable batch predictions for business or organizational use.

## 11.2 Business Impact

- Reduces spam-related productivity losses in organizations.
- Provides instant classification for incoming emails or messages.
- Enhances security and email management efficiency.
- Enables organizations to maintain a record of predictions via logs for auditing and monitoring purposes.

### 11.3 Potential Improvements / Future Enhancements

- **REST API Deployment**: Expose the spam detection model as an API for integration with email servers.

- **Deep Learning Models**: Explore LSTM or BERT for improved accuracy and multi-language support.

- **Multi-Language Support**: Extend spam detection to languages other than English.

- **Cloud Deployment**: Deploy the app on AWS, Azure, or GCP for enterprise-scale usage.

- **Authentication & User Management**: Secure access for multiple users and roles.

- **Real-time Email Integration**: Connect with email clients to classify incoming emails automatically.

- **Interactive Dashboard**: Visualize prediction statistics, spam frequency, and usage metrics.

- **AutoML Integration**: Continuously improve model accuracy using automated training pipelines.

# Chapter - 12

# References

# 12: References

## 12.1 Dataset Source

- **SMS Spam Collection Dataset** – UCI Machine Learning Repository / Kaggle
  https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset
    - Used for training and evaluating the email/spam classification model.
    - Includes labeled messages (spam or ham) for supervised learning.

## 12.2 Libraries and Frameworks

- **Python 3.10+** – Core programming language
- **Pandas & NumPy** – Data handling and preprocessing
- **Scikit-learn** – TF-IDF Vectorizer, SVM model, evaluation metrics
- **Streamlit** – Web application deployment and UI
- **Matplotlib / Seaborn** – Data visualization
- **Logging module** – Application logging and monitoring