SMARTBRIDGE
Let's Bridge the Gap
a Veranda Enterprise

Smart Internz

Final Project Report Template

SMS SPAM DETECTION

1.INTRODUCTION

1.1 PROJECT OVERVIEWS

The proliferation of spam messages in SMS communications poses a significant challenge to user security and convenience. Spam messages not only clutter inboxes but also serve as vectors for phishing attacks, fraudulent schemes, and privacy violations. As these unwanted messages evolve in complexity, traditional rule-based spam filters often fail to effectively identify and block them. To address this issue, this project leverages **Machine Learning (ML)** to build a more adaptive and accurate SMS spam detection system.

This project utilizes a dataset of SMS messages labeled as either **Spam** or **Not Spam**, applying advanced machine learning algorithms such as **Naive Bayes**. By analysing text-based features like message length, word frequency, punctuation usage, and specific keywords, the model learns to distinguish spam messages from legitimate ones. Unlike traditional methods, which rely on static rules, the ML model is dynamic and can adapt to new patterns, offering a more robust and scalable solution for spam detection.

The developed model is integrated into a **Flask-based web application**, allowing users to input an SMS message and receive real-time predictions. This not only enhances the user experience by providing an intuitive interface but also empowers users to combat spam more effectively. The application can be extended for use in personal devices, corporate communications, or mobile service providers, making it highly versatile and impactful.

1.2. OBJECTIVES

The main objective of this project is to create a machine learning model that accurately classifies SMS messages as Spam or Not Spam. This project goes beyond simple keyword matching and uses Natural Language Processing (NLP) techniques to understand the context and structure of SMS messages. By employing sophisticated ML algorithms, the project aims to achieve several key goals:

- Increase Detection Accuracy: Improve spam detection rates by using a Naive Bayes classifier and other algorithms to identify spam messages with high precision, reducing false positives and negatives.
- Enhance Adaptability: Build a system that adapts to evolving spam tactics, continuously learning from new datasets to maintain its effectiveness against emerging spam techniques.
- Real-Time Detection: Develop a real-time, web-based tool where users can input an SMS message and get instant feedback on whether the message is spam. This real-time capability ensures that users can manage spam messages efficiently and proactively.
- User Security: Enhance the security of SMS communications by minimizing the risk of phishing attacks and fraud schemes associated with spam messages. By automating the spam detection process, users can have more control over their SMS inbox, improving their overall digital security.
- User-Friendly Interface: Provide an intuitive web interface built using Flask to ensure ease of use. The tool is designed to cater to a broad range of users, from individuals managing personal SMS communications to businesses handling large volumes of messages.
- Scalability and Performance: Ensure the system can scale to handle large datasets and high traffic loads, making it deployable in real-world scenarios, such as integration with mobile service providers or enterprise-level communications.

- Demonstrate the Power of Machine Learning in NLP: Showcase how machine learning techniques, when applied to natural language processing, can solve real-world problems like spam detection. The project will contribute to the growing body of work in text classification and NLP.

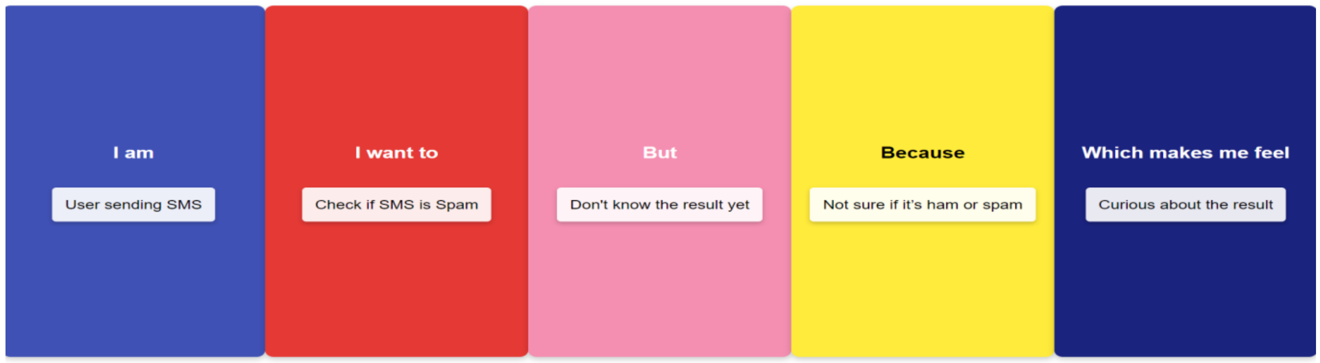
2.Project Initialization and Planning Phase

2.1 Define Problem Statements (Customer Problem Statement Template):

The increasing amount of SMS spam is a growing concern for users. Spam messages often contain fraudulent schemes, unwanted promotions, or phishing attempts, making it difficult for users to distinguish between legitimate and spam messages. This project aims to provide an efficient solution to automatically detect and classify SMS messages as either **Spam** or **Not Spam**, enhancing users' trust and experience.

By creating a tool to automatically detect SMS spam, we aim to alleviate the frustration of manually filtering unwanted messages. The solution will contribute to a cleaner inbox and a safer user experience.

EXAMPLE:



Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	A user regularly receiving SMS messages, both legitimate and spam.	Identify whether an SMS message is legitimate or spam without having to manually filter through my inbox.	Spam messages often look like legitimate messages, and filtering through them manually is time-consuming and frustrating.	Some spam messages contain fraudulent information or unwanted promotions, potentially leading to scams or security breaches.	Worried about security, and annoyed by the constant influx of spam messages.

2.2 Project Proposal (Proposed Solution)

This proposal AI-ML to develop a machine learning-based SMS spam detection system, enhancing the user experience by automatically classifying SMS messages as either spam or not spam. The project leverages a pre-trained model and web technologies to deliver real-time predictions in a user-friendly interface. It tackles the challenge of spam overload, promising to reduce unnecessary distractions and improve user security.

Project Overview	
Objective	The primary objective is to create a Flask web application capable of detecting spam SMS messages with high accuracy by utilizing machine learning techniques such as Naive Bayes and TF-IDF vectorization.

Scope	This project will allow users to input an SMS message via a web interface and receive instant feedback on whether the message is classified as spam or not spam. The application aims to improve user convenience by automating the spam detection process and offering an intuitive, easy-to-use web solution.
Problem Statement	
Description	Users often receive unwanted and potentially harmful SMS spam messages that affect their productivity and pose security risks. Manually filtering these messages is time-consuming and inefficient.
Impact	By addressing the issue of spam detection, the application can reduce the number of unsolicited messages users have to handle manually, thereby improving user experience, increasing productivity, and enhancing digital security.
Proposed Solution	
Approach	The proposed solution is to build a Flask-based web application that employs machine learning techniques, such as **Naive Bayes** and **TF-IDF vectorization**, to classify SMS messages as spam or not spam in real-time.
Key Features	1.Implementation of a machine learning-based model for **SMS spam detection**. 2.**Real-time predictions**: Users will receive instant feedback on the spam classification of the SMS. 3.User-friendly **web interface** for easy input of messages and result visualization. 4.Continuous updates and retraining of the model to adapt to new spam techniques.

Resource Requirements

Resource Type	Description	Specification/Allocation
Hardware		
Computing Resources	CPU/GPU specifications, number of cores	NVIDIA V100 GPUs
Memory	RAM specifications	4 GB / 8 GB / 16 GB
Storage	Disk space for data, models, and logs	512 GB SSD / 1 TB SSD
Software		
Frameworks	Python framework for web development	Flask
Libraries	Libraries for machine learning and data processing	scikit-learn, pandas, numpy, matplotlib, seaborn
Development Environment	IDEs for coding	Jupyter Notebook, Google Colab, Visual Studio Code
Data		
Data	Dataset used for training and testing	Kaggle dataset (SMS Spam Collection), 5,000+ rows, CSV format

2.3 Initial Project Planning

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)

Sprint-1	Data Collection and Preprocessing	SD-1	Understanding & loading dataset	High	Sajida Tharunam	26-09-2024	29-09-2024
Sprint-1	Data Collection and Preprocessing	SD-2	Data cleaning	High	Satya Pranith	27-09-2024	27-09-2024
Sprint-1	Data Collection and Preprocessing	SD-3	Exploratory Data Analysis (EDA)	Medium	Satya Pranith	28-09-2024	28-09-2024
Sprint-2	Model Development	SD-4	Training the Naive Bayes model	Medium	Sajida Tharunam	30-09-2024	1-10-2024
Sprint-2	Model Development	SD-5	Evaluating the model	Medium	R.L. Chaitanya	2-10-2024	4-10-2024
Sprint-2	Model Tuning and Testing	SD-6	Model tuning	High	R.L.Chaitanya	5-10-2024	7-10-2024

Sprint-2	Model Tuning and Testing	SD-7	Model tuning	Medium	Eshwar Reddy	7-10-2024	9-10-2024
Sprint-3	Web Integration and Deployment	SD-8	Building HTML templates	High	Eshwar Reddy	10-10-2024	13-10-2024
Sprint-3	Web Integration and Deployment	SD-9	Local deployment	Medium	Eshwar Reddy	14-10-2024	15-10-2024
Sprint-4	Project Report	SD-10	Writing the final project report	Medium	Sajida Tharunam	15-10-2024	18-10-2024

3.Data Collection and Preprocessing Phase

3.1 Data Collection Plan & Raw Data Sources Identification

This report outlines the data collection strategy and the identified raw data sources for the **SMS Spam Detection** project. It ensures a meticulous approach to data curation, integrity, and quality to support informed decision-making and accurate predictions.

Data Collection Plan

Section	Description

Project Overview	The machine learning project aims to classify SMS messages as **Spam** or **Not Spam** using a dataset of labeled SMS messages. The objective is to build a model that can accurately detect spam messages, enhancing the user experience and security. Features of the dataset include the raw SMS text and a label (spam or ham).
Data Collection Plan	Search for datasets related to **SMS messages** and **spam classification**. - Prioritize datasets with large sample sizes to cover a variety of spam message patterns. - Ensure datasets include labeled messages to facilitate supervised learning.

Raw Data Sources T

Source Name	Description	Location/URL	Format	Size	Access Permissions
Kaggle Dataset	The dataset contains labeled SMS messages, each classified as either **Spam** or **Not Spam (ham)**.	spam_ham_dataset.csv - Google Drive	CSV	5 MB	Public

UCI Dataset	A dataset of SMS messages labeled for spam detection, providing a good balance of spam and ham messages.	https://archive.ics.uci.edu/dataset/228/sms+spam+collection	CSV	4.7 MB	Public

3.2 Data Quality Report Template

The Data Quality Report outlines data quality issues from the selected dataset, including their severity levels and the proposed resolution plans. This report helps systematically identify and address any discrepancies or issues with the dataset to ensure high-quality input for the machine learning model.

Data Source	**Data Quality Issue**	**Severity**	**Resolution Plan**
Kaggle SMS Spam Collection Dataset	Presence of special characters, URLs, and numbers in text data	Moderate	Clean the text by removing special characters, links, and numbers using regular expressions.
Kaggle SMS Spam Collection Dataset	Imbalanced classes between spam and non-spam messages	High	Use class weighting or oversampling techniques like SMOTE to handle imbalance.

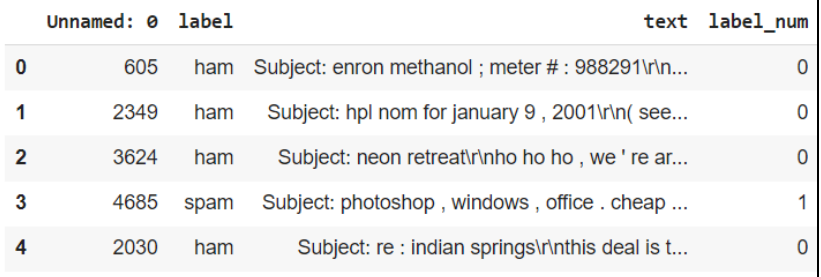
Kaggle SMS Spam Collection Dataset	Text data needs to be converted into numerical format	Moderat e	Use Tfidf Vectorizer to transform text into numerical features for model input.
Kaggle SMS Spam Collection Dataset	Presence of duplicate messages in the dataset	Low	Remove duplicate rows to avoid model bias.

3.3 Data Preprocessing

Dataset variables will be statistically analyzed to identify patterns, trends, and potential anomalies in the text data. Python will be employed for preprocessing tasks such as text cleaning, tokenization, and vectorization to prepare the data for modeling. **Tfidf Vectorization** will be used to convert the text data into numerical features that can be processed by the machine learning model. Data cleaning will address noise such as special characters, links, and numbers, ensuring high-quality input for subsequent analysis and prediction, thus forming a solid foundation for accurate spam detection.

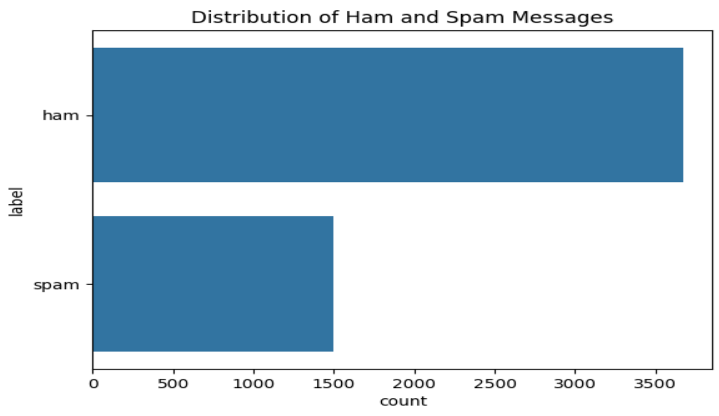
Section	**Description**
Data Overview	Dimension: 5171 Rows x 5 Columns

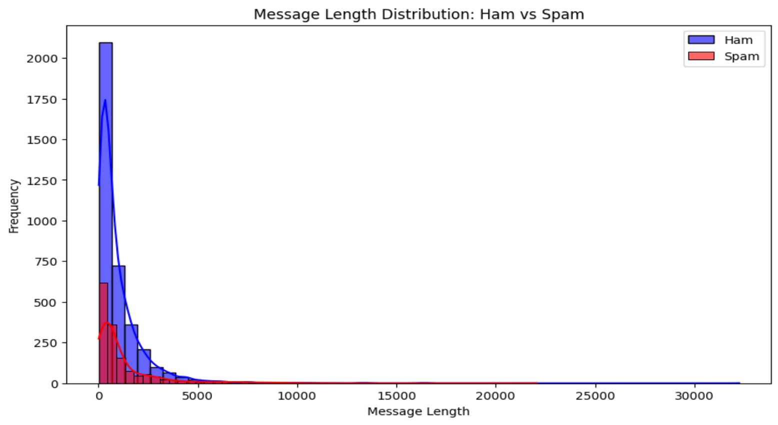
Descriptive statistics:

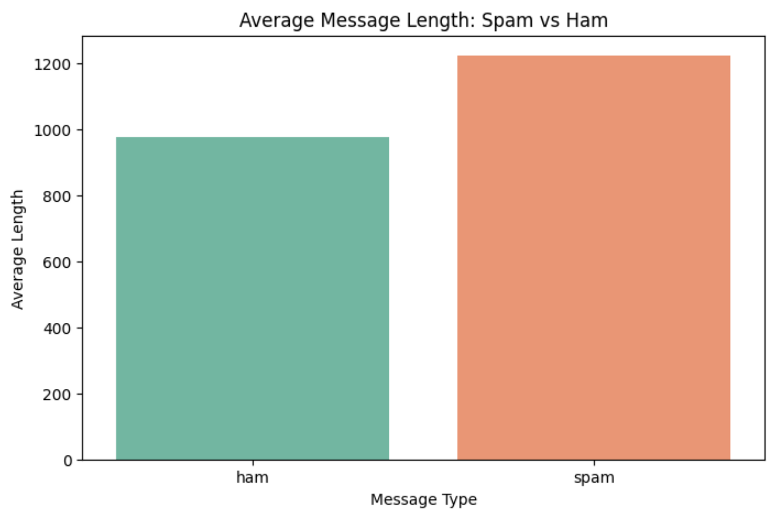
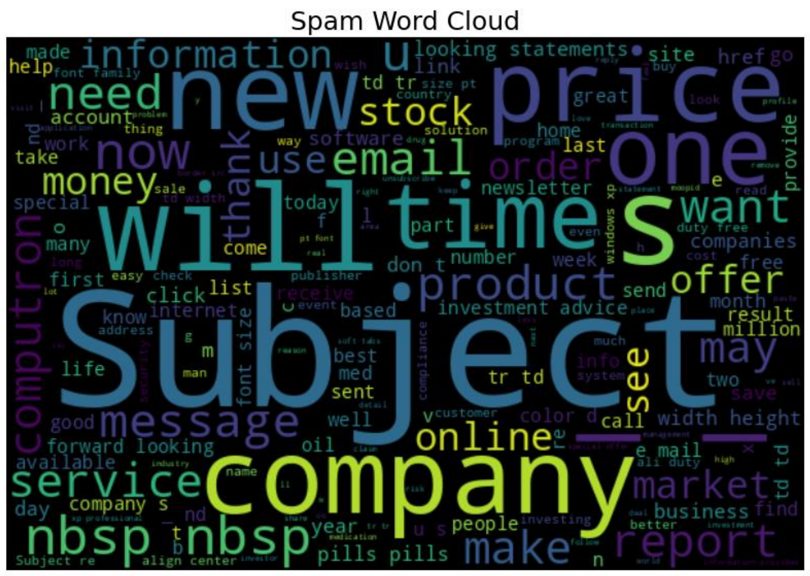


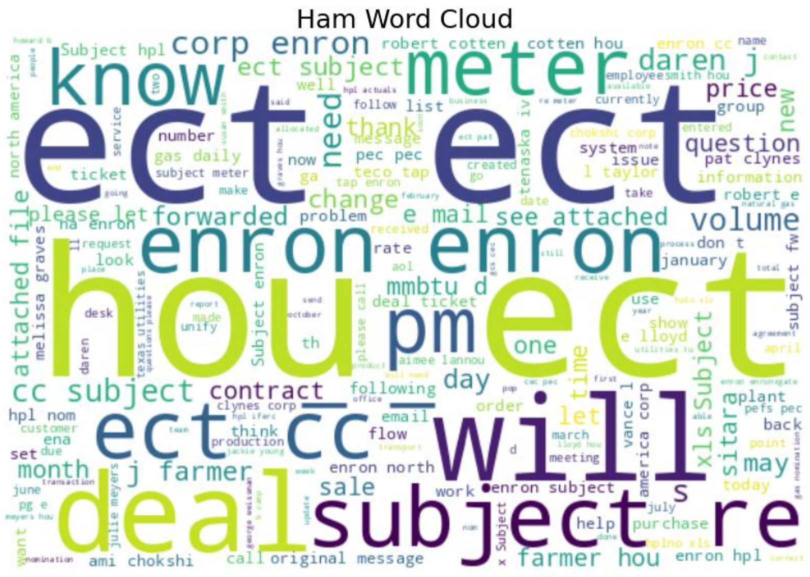
	Unnamed: 0	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	3624	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	4685	spam	Subject: photoshop , windows , office . cheap ...	1
4	2030	ham	Subject: re : indian springs\r\nthis deal is t...	0

Univariate Analysis



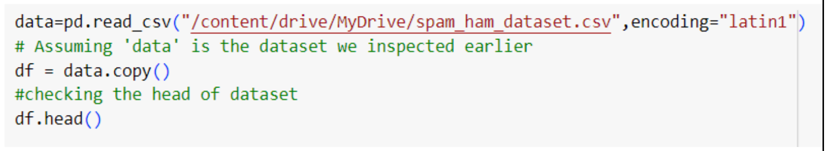


Bivariate Analysis	
Multivariate Analysis	

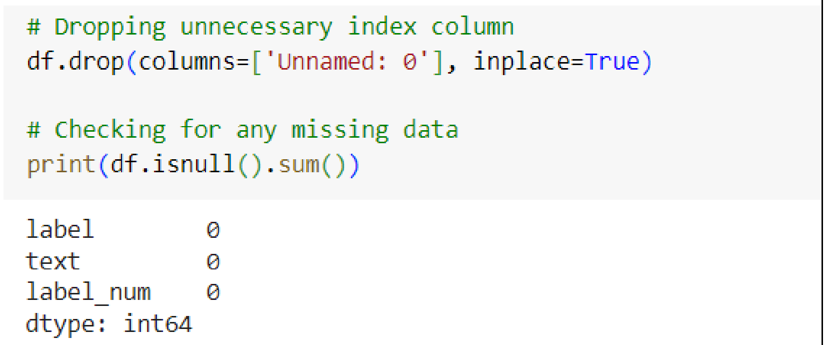
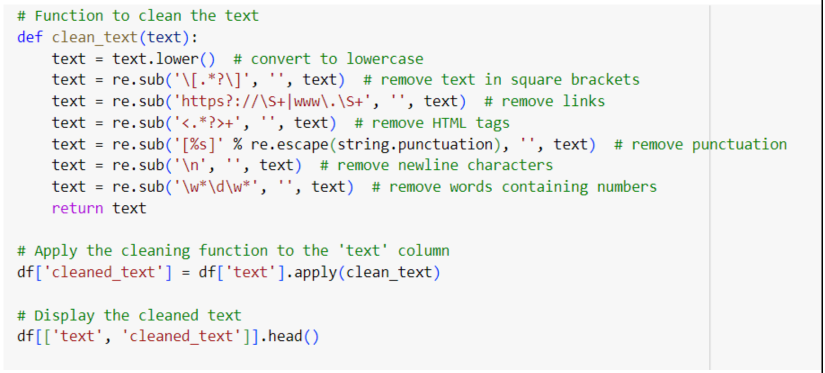
	
Outliers and Anomalies	Addressed noise like irrelevant special characters, numbers, and excessive spaces.

Data Preprocessing Code Screenshots

Loading Data



```
data=pd.read_csv("/content/drive/MyDrive/spam_ham_dataset.csv",encoding="latin1")
# Assuming 'data' is the dataset we inspected earlier
df = data.copy()
#checking the head of dataset
df.head()
```

Handling Missing Data	
Data Transformation	
Feature Engineering	Attached the code in the final submission.
Save Processed Data	-

```
# Dropping unnecessary index column
df.drop(columns=['Unnamed: 0'], inplace=True)

# Checking for any missing data
print(df.isnull().sum())
```

```
label        0
text         0
label_num    0
dtype: int64
```

```
# Function to clean the text
def clean_text(text):
    text = text.lower()  # convert to lowercase
    text = re.sub('\[.*?\]', '', text)  # remove text in square brackets
    text = re.sub('https?://\S+|www\.\S+', '', text)  # remove links
    text = re.sub('<.*?>+', '', text)  # remove HTML tags
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)  # remove punctuation
    text = re.sub('\n', '', text)  # remove newline characters
    text = re.sub('\w*\d\w*', '', text)  # remove words containing numbers
    return text

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

# Display the cleaned text
df[['text', 'cleaned_text']].head()
```

4. Model Development Phase

4.1 Model Selection Report

In this project, multiple machine learning models are evaluated to identify the best approach for classifying SMS messages as **Spam** or **Not Spam**. The models are compared based on performance, accuracy, complexity, and computational requirements. The goal is to select the most suitable model for real-time spam detection in a web-based application.

Model	**Description**
Model 1: Naive Bayes	A probabilistic classifier that uses Bayes' theorem with strong independence assumptions. This model is commonly used for text classification and works well with small datasets. Low computational cost and good performance for spam detection tasks.
Model 2: Decision Tree	A non-parametric supervised learning model used for classification tasks. It works by splitting data into subsets based on feature values. It's easy to interpret but can lead to overfitting.
Model 3: Logistic Regression	A linear model used for binary classification. It predicts probabilities and outputs binary outcomes. Efficient and interpretable but may struggle with complex data relationships.
Model 4: Support Vector Machine (SVM)	A powerful classifier that finds a hyperplane separating spam from non-spam messages. SVM is highly accurate but computationally expensive and may require more time to train.

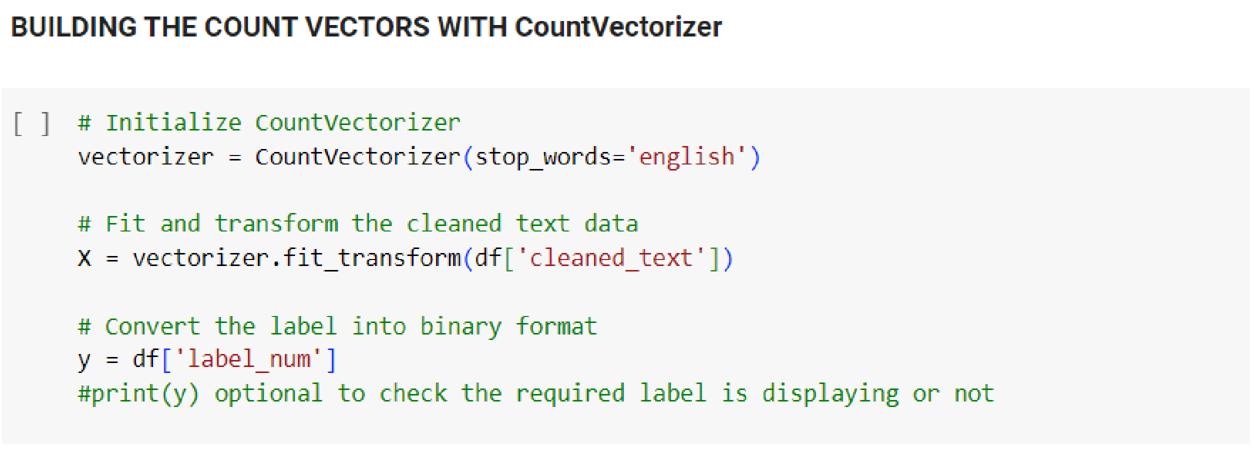
Conclusion

After evaluating these models, the **Naive Bayes classifier** was selected due to its simplicity, low computational cost, and strong performance on text data. The model offers an optimal balance between accuracy and efficiency, making it ideal for real-time SMS spam detection in a web-based application.

4.2 Initial Model Training Code, Model Validation and Evaluation Report

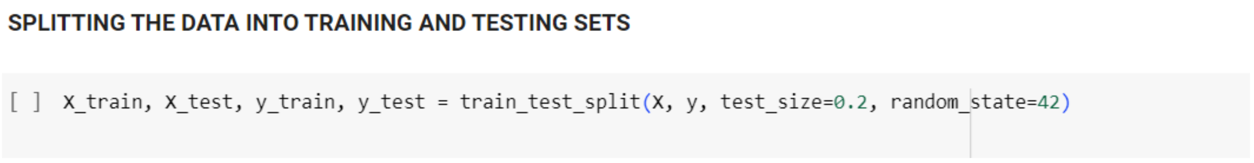
The initial model training code will be included in the final submission through screenshots. The **model validation and evaluation report** will summarize the performance of the spam detection model using metrics such as **accuracy**, **precision**, **recall**, **F1-score**, and **confusion matrices**. Multiple models, including Naive Bayes, Decision Tree, and Logistic Regression, will be evaluated to select the best-performing model.

Initial Model Training Code



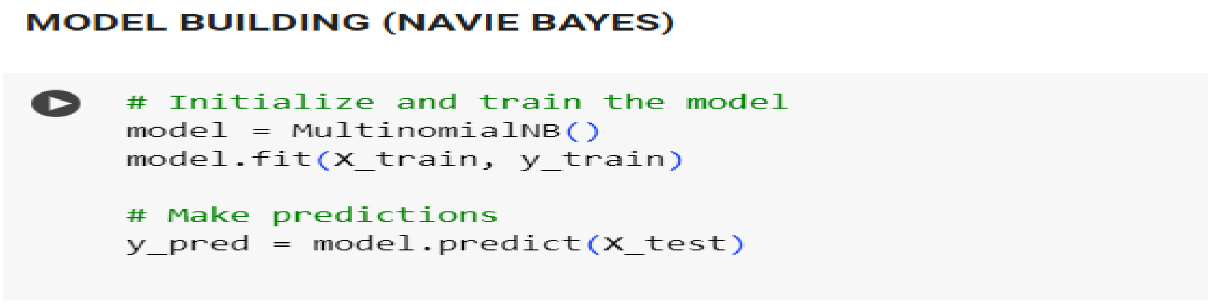
BUILDING THE COUNT VECTORS WITH CountVectorizer

```
[ ]  # Initialize CountVectorizer
     vectorizer = CountVectorizer(stop_words='english')

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

     # Convert the label into binary format
     y = df['label_num']
     #print(y) optional to check the required label is displaying or not
```



SPLITTING THE DATA INTO TRAINING AND TESTING SETS

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

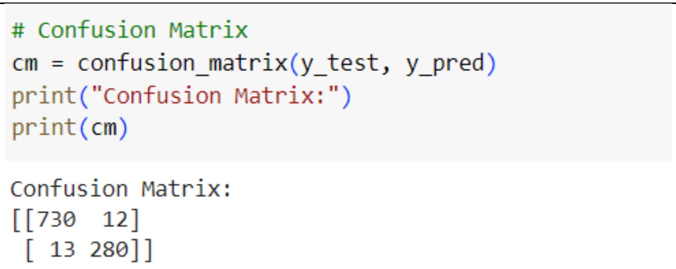


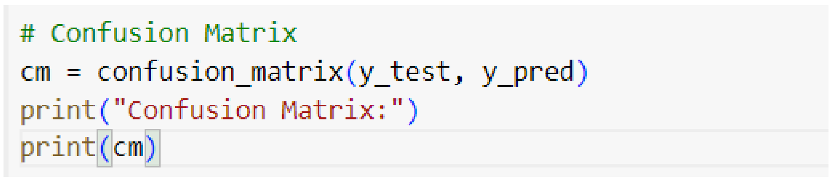
MODEL BUILDING (NAVIE BAYES)

```
# Initialize and train the model
model = MultinomialNB()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

Model Validation and Evaluation Report:

Model	F1 Scor e	Confusion Matrix

Naive Bayes	97.58%	
		`# Confusion Matrix`
		`cm = confusion_matrix(y_test, y_pred)`
		`print("Confusion Matrix:")`
		`print(cm)`
		
		`Confusion Matrix:`
		`[[730  12]`
		`[ 13 280]]`

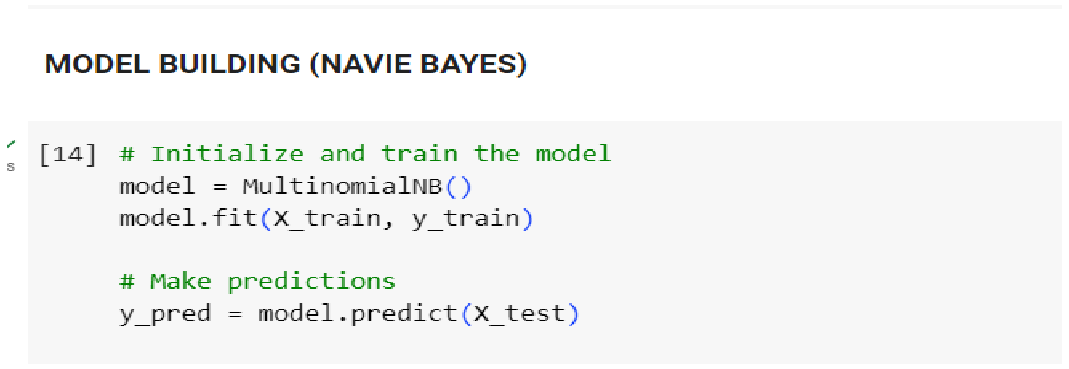
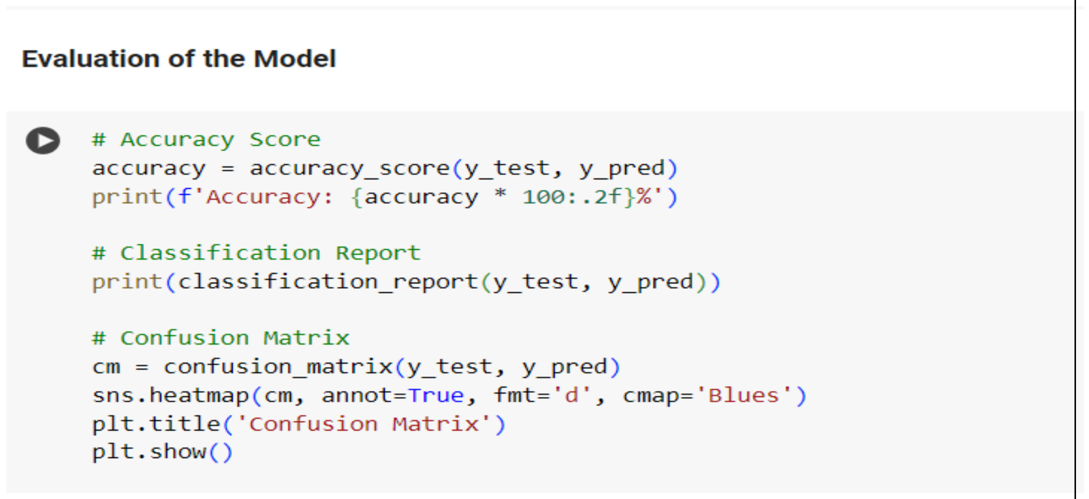
Decision Tree	94.63%	
		`# Confusion Matrix`
		`cm = confusion_matrix(y_test, y_pred)`
		`print("Confusion Matrix:")`
		`print(cm)`
		
		Confusion Matrix:
		[[730 12]
		[18 274]]

5. Model Optimization and Tuning Phase

5.1Tuning Phase

In this phase, the Naive Bayes model was optimized for the SMS Spam Detection task. Hyper parameters were fine-tuned to achieve optimal performance. Although various models were considered, the primary focus was on Naive Bayes, as it is well-suited for text classification tasks due to its simplicity and effectiveness.

Hyperparameter Tuning Documentation:

Model	**Tuned Hyperparameters**
Model 1: Naive Bayes	- **alpha**: Smoothing parameter, tested with values of 0.1, 0.5, and 1.0 to handle zero probabilities in sparse data. - **fit_prior**: Set to True to learn class priors from the training data.  

MODEL BUILDING (NAVIE BAYES)

```
[14] # Initialize and train the model
     model = MultinomialNB()
     model.fit(X_train, y_train)

     # Make predictions
     y_pred = model.predict(X_test)
```

Evaluation of the Model

```
# Accuracy Score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Classification Report
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```

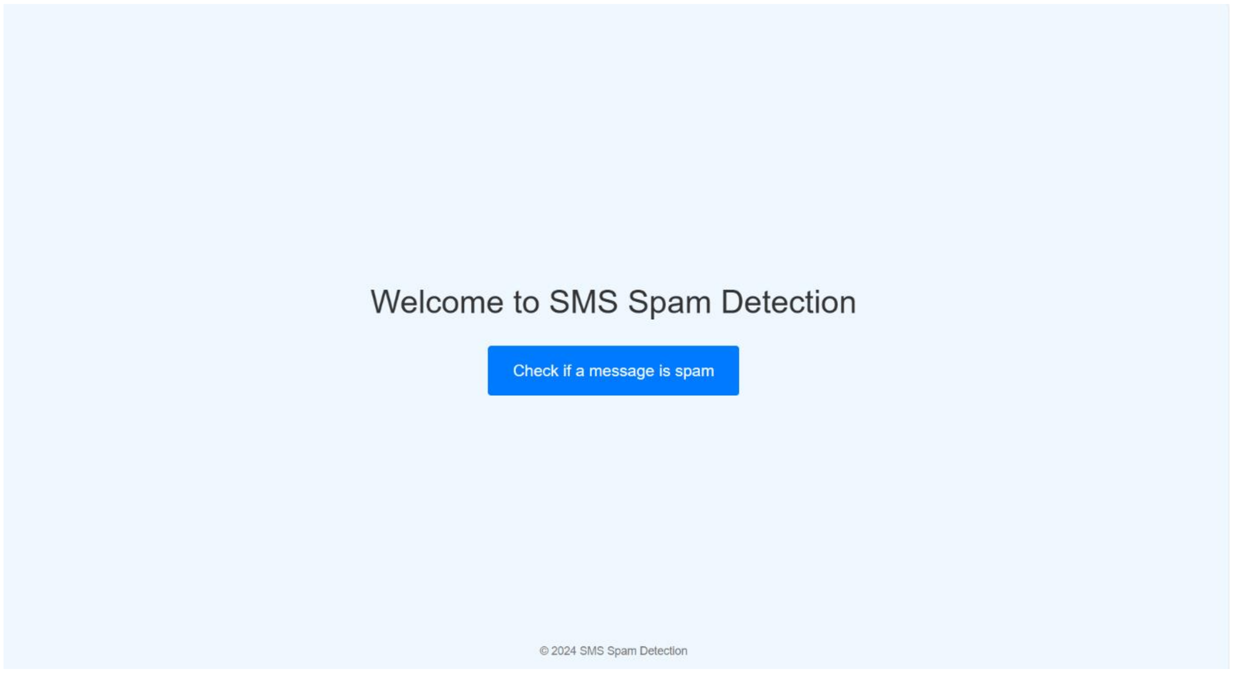
Note: Due to the specific requirements of this project and the dataset's structure, the other models were not implemented in this phase, as Naive Bayes demonstrated superior performance for text classification.

5.2 Final Model Selection Justification:

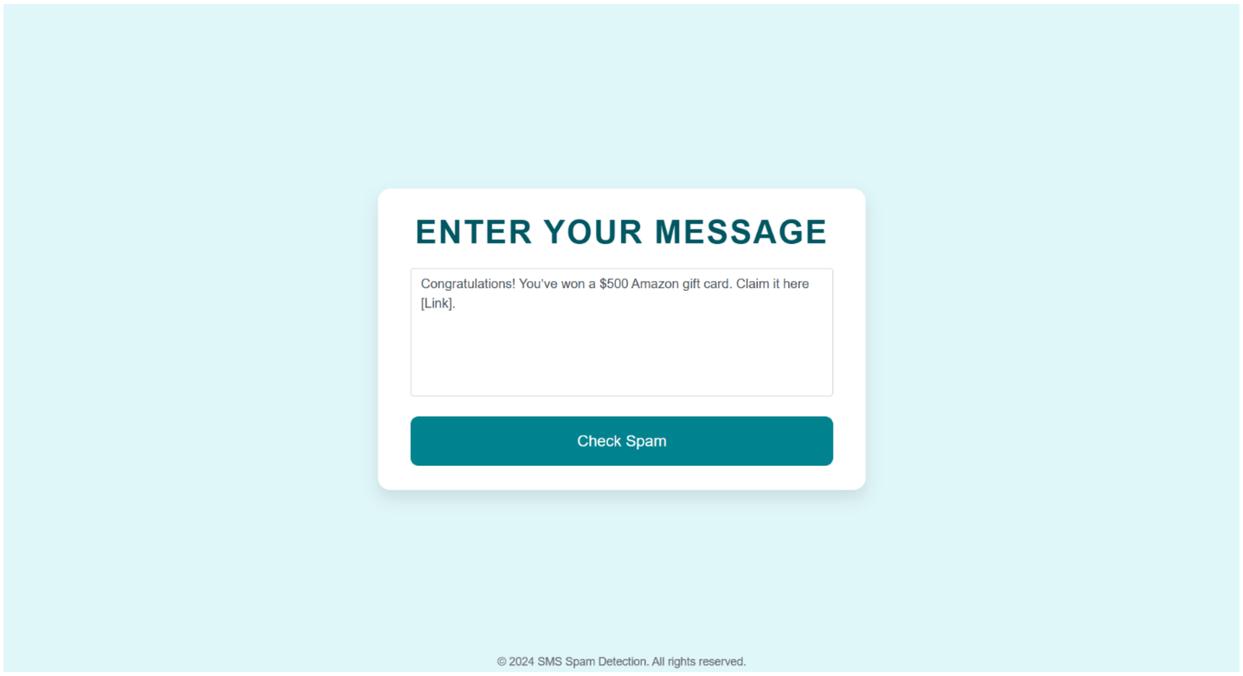
Final Model	**Reasoning**
Model 1: Naive Bayes	Naive Bayes was selected because it consistently achieved the highest accuracy and F1-score for the given text-based dataset. Its ability to handle sparse data and its low computational cost make it an excellent choice for real-time spam detection.

6.Results

6.1 Output Screen shots

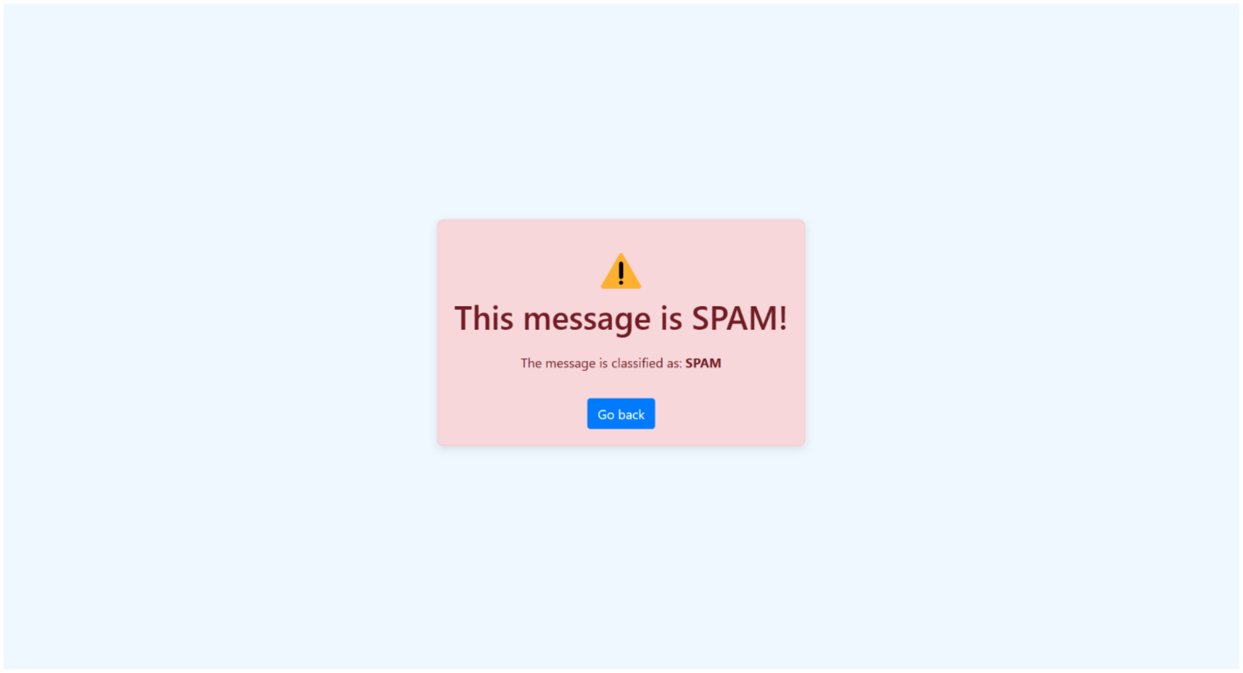

Welcome to SMS Spam Detection

© 2024 SMS Spam Detection



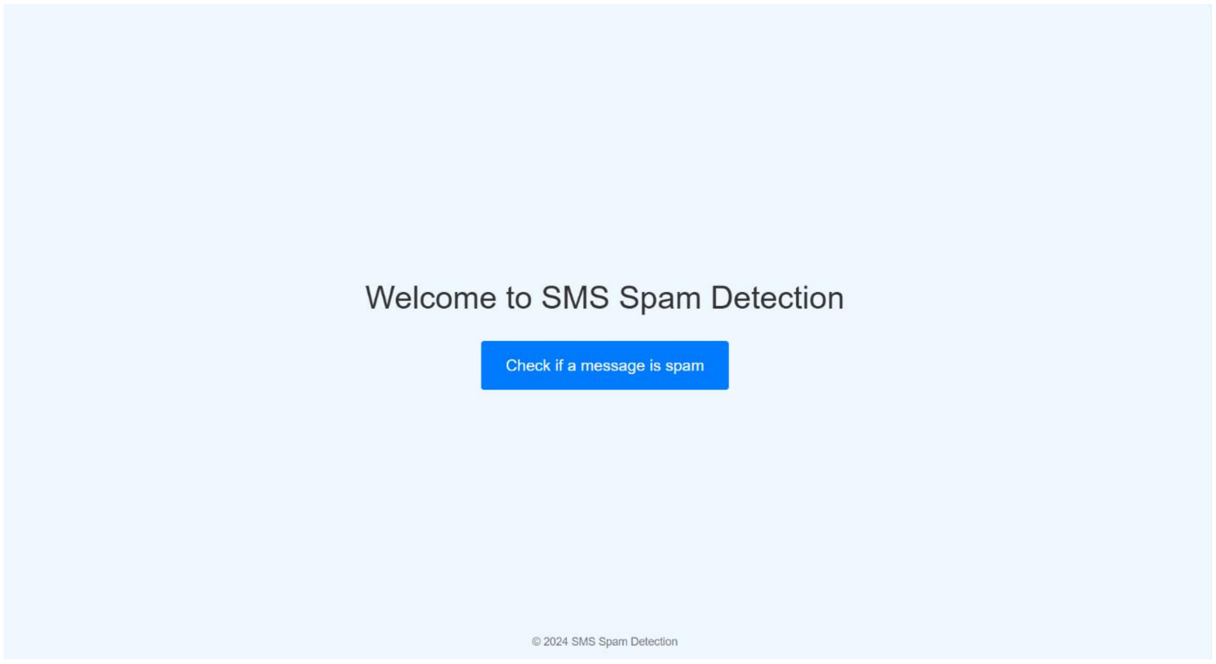
Congratulations! You've won a $500 Amazon gift card. Claim it here [Link].

© 2024 SMS Spam Detection. All rights reserved.

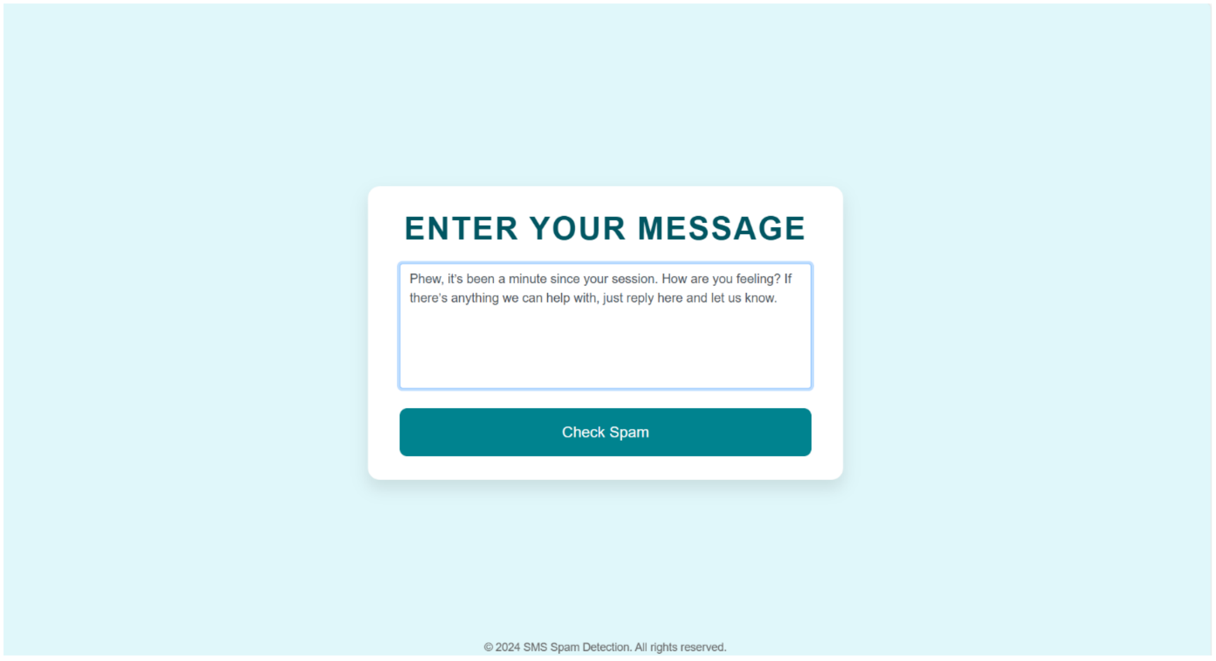


This message is SPAM!

The message is classified as: **SPAM**

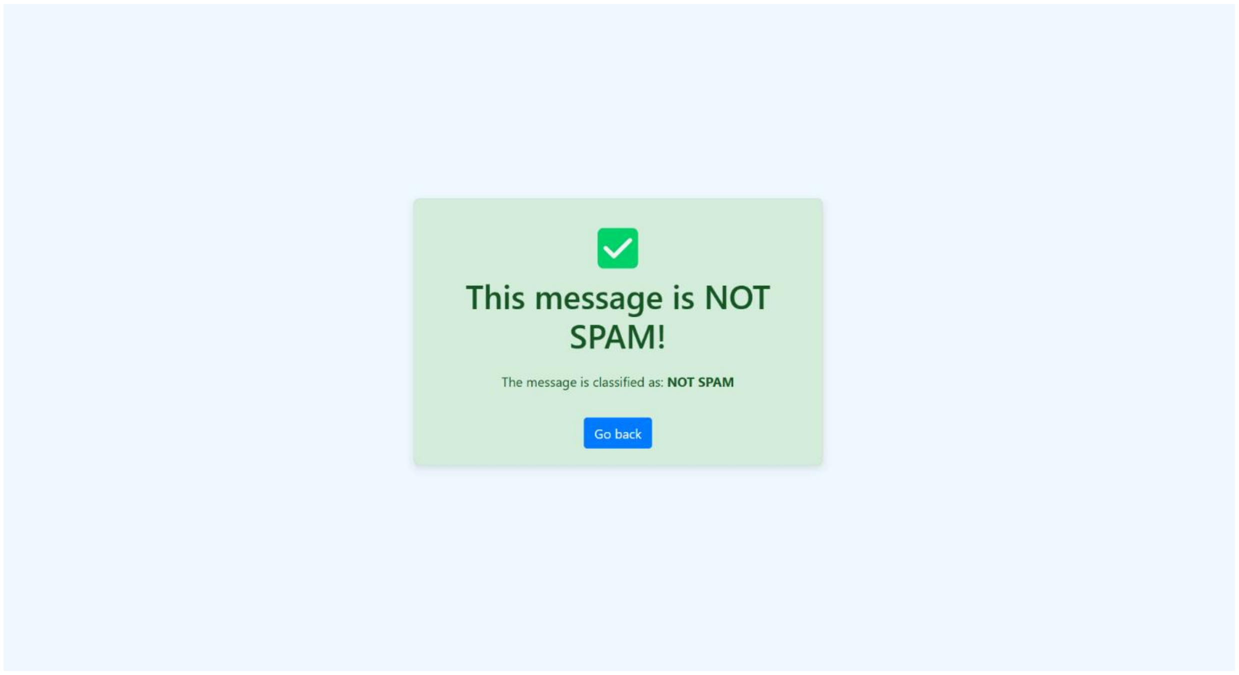

Welcome to SMS Spam Detection

© 2024 SMS Spam Detection


ENTER YOUR MESSAGE

Phew, it's been a minute since your session. How are you feeling? If there's anything we can help with, just reply here and let us know.

© 2024 SMS Spam Detection. All rights reserved.


This message is NOT SPAM!

The message is classified as: **NOT SPAM**

7. Advantages & Disadvantages

Advantages

1. Accuracy: Machine learning models, like Naive Bayes, can achieve high accuracy in detecting spam messages by learning from vast datasets of labeled SMS messages.
2. Automation: Once trained, the ML model can automatically classify SMS messages as spam or not spam in real-time, eliminating the need for manual filtering.
3. Scalability: The model can handle large datasets of SMS messages and adapt to new spam patterns as more data becomes available, ensuring continuous improvement.
4. Pattern Recognition: ML models excel at identifying patterns in text data that traditional keyword-based spam filters may miss, such as the use of certain phrases or structures typical of spam.
5. User-Friendly: Integrated into a web-based Flask application, the system provides a simple, intuitive interface for users to check if a message is spam.

6. Real-Time Detection: The system can classify incoming messages instantly, helping users manage their SMS inbox efficiently and reduce the risk of fraud or phishing.

Disadvantages

1. Data Dependency: The performance of the model heavily depends on the quality and size of the dataset used for training. Insufficient or biased data can reduce the model's effectiveness.
2. Complexity: Developing and fine-tuning machine learning models requires expertise in both machine learning and natural language processing, which can be a challenge for non-experts.
3. Resource Intensive: Although the Naive Bayes model is relatively light, more complex models like Random Forest or SVM can be computationally expensive to train, especially with large datasets.
4. Model Maintenance: Spam tactics evolve over time, requiring regular updates to the model and the dataset to maintain high detection accuracy.
5. Interpretability: While the Naive Bayes model is generally interpretable, more complex models like deep learning algorithms can behave like a "black box," making it harder to understand how predictions are made.
6. Bias: If the training data is biased, the model may produce biased results, affecting its ability to detect certain types of spam messages correctly.

8. Conclusion

The application of machine learning in the **SMS Spam Detection Flask Application** has proven to be an effective solution for detecting and filtering spam messages in real time. By leveraging the **Naive Bayes** algorithm, this project demonstrates how machine learning can improve upon traditional rule-based spam filters, offering a more adaptive and accurate spam detection system.

In this project, we focused on preprocessing the SMS dataset to ensure high-quality inputs for the model. The **Naive Bayes classifier** was chosen for its efficiency in handling text-based data and its ability to identify patterns in SMS messages that indicate spam. The project successfully integrates the machine learning model into a web-based Flask application, allowing users to input an SMS message and receive instant classification feedback.

Our evaluation metrics, including **accuracy**, **precision**, and **F1-score**, highlight the effectiveness of the Naive Bayes model in distinguishing spam messages from legitimate ones. This model is particularly well-suited for this task, given its low computational cost and ability to handle the sparse nature of text data. The application provides users with an easy-to-use tool for managing spam in their inbox, reducing the risk of phishing and fraudulent messages.

While the project has been successful in achieving its goals, it is important to note the challenges that come with model maintenance and potential bias in the dataset. Regular updates and tuning are required to ensure that the model remains accurate as spam tactics evolve.

In conclusion, the **SMS Spam Detection Flask Application** is a valuable tool for improving user security and reducing the frustration associated with unwanted spam messages. The project demonstrates the practical applications of machine learning in natural language processing, with potential for further improvements and expansions in the future.

9. Future Scope

1. Model Enhancement: Continuously refine the model by incorporating new spam datasets, improving the model's robustness and ability to detect emerging spam trends.
2. Real-Time Data Integration: Integrate real-time data from SMS gateways or mobile networks to provide immediate spam detection for users.
3. Mobile and Web Application: Develop a full-fledged mobile or web application that allows users to seamlessly input messages and receive spam classifications with an intuitive interface.

4. Geographical and Language Expansion: Adapt the model to support different geographic regions and languages, allowing for global applicability of the spam detection system.
5. Handling Complex Spam Patterns: Enhance the model to detect more sophisticated spam patterns that may include obfuscation techniques or spam that mimics legitimate communication.
6. Collaboration with Telecom Providers: Work with mobile network operators to deploy the model as part of a broader anti-spam strategy, enabling spam detection at the network level for enhanced security.

10.Appendex

10.1 Source Code

```
#IMPORTING THE LIBRARIES
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import re
import string
#READING THE DATA SET
data=pd.read_csv("/content/drive/MyDrive/spam_ham_dataset.csv",enc
oding="latin1")
```

```
# Assuming 'data' is the dataset we inspected earlier
df = data.copy()
#checking the head of dataset
df.head()
df.shape
df.size
df.describe()
df.info()
# Dropping unnecessary index column
df.drop(columns=['Unnamed: 0'], inplace=True)
# Checking for any missing data
print(df.isnull().sum())
#EDA ON DATSET
# Visualize the distribution of spam vs ham
sns.countplot(df['label'])
plt.title('Distribution of Ham and Spam Messages')
plt.show()
#checking the length of each message including spam and ham
df['message_length'] = df['text'].apply(len)
# Plotting the distribution of ham and spam message lengths
plt.figure(figsize=(10,6))
# Ham messages
sns.histplot(df[df['label']=='ham']['message_length'],
             bins=50, color='blue', kde=True, label='Ham', alpha=0.6)
# Spam messages
```

```
sns.histplot(df[df['label']=='spam']['message_length'],
             bins=50, color='red', kde=True, label='Spam', alpha=0.6)

plt.title('Message Length Distribution: Ham vs Spam')
plt.xlabel('Message Length')
plt.ylabel('Frequency')
plt.legend()
plt.show()
#VISULIZATIONS
# Grouping by label and calculating mean message length
avg_length = df.groupby('label')['message_length'].mean().reset_index()
# Plotting
plt.figure(figsize=(8,5))
sns.barplot(x='label', y='message_length', data=avg_length,
palette="Set2")
plt.title('Average Message Length: Spam vs Ham')
plt.xlabel('Message Type')
plt.ylabel('Average Length')
plt.show()
from wordcloud import WordCloud

# Generating word clouds for spam and ham messages
spam_words = ' '.join(df[df['label']=='spam']['text']) # Use 'text' column
instead of 'cleaned_text'
ham_words = ' '.join(df[df['label']=='ham']['text']) # Use 'text' column
instead of 'cleaned_text'
```

```
# Spam word cloud

spam_wordcloud = WordCloud(width=600, height=400,
background_color='black').generate(spam_words)


# Ham word cloud

ham_wordcloud = WordCloud(width=600, height=400,
background_color='white').generate(ham_words)


# Plotting the Spam word cloud

plt.figure(figsize=(10,8))

plt.imshow(spam_wordcloud, interpolation='bilinear')

plt.title('Spam Word Cloud', fontsize=18)

plt.axis('off')

plt.show()


# Plotting the Ham word cloud

plt.figure(figsize=(10,8))

plt.imshow(ham_wordcloud, interpolation='bilinear')

plt.title('Ham Word Cloud', fontsize=18)

plt.axis('off')

plt.show()

# Generating word clouds for spam and ham messages

spam_words = ' '.join(df[df['label']=='spam']['text']) # Use 'text' column
instead of 'cleaned_text'

ham_words = ' '.join(df[df['label']=='ham']['text']) # Use 'text' column
instead of 'cleaned_text'
```

```
# Spam word cloud
spam_wordcloud = WordCloud(width=600, height=400,
background_color='black').generate(spam_words)
# Ham word cloud
ham_wordcloud = WordCloud(width=600, height=400,
background_color='white').generate(ham_words)
# Plotting the Spam word cloud
plt.figure(figsize=(10,8))
plt.imshow(spam_wordcloud, interpolation='bilinear')
plt.title('Spam Word Cloud', fontsize=18)
plt.axis('off')
plt.show()

# Plotting the Ham word cloud
plt.figure(figsize=(10,8))
plt.imshow(ham_wordcloud, interpolation='bilinear')
plt.title('Ham Word Cloud', fontsize=18)
plt.axis('off')
plt.show()

#DATA PREPROCESSING (CLEANING THE TEXT)

# Function to clean the text
def clean_text(text):
    text = text.lower()  # convert to lowercase
    text = re.sub('\[.*?\]', '', text)  # remove text in square brackets
    text = re.sub('https?://\S+|www\.\S+', '', text)  # remove links
    text = re.sub('<.*?>+', '', text)  # remove HTML tags
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)  #
remove punctuation
    text = re.sub('\n', '', text)  # remove newline characters
    text = re.sub('\w*\d\w*', '', text)  # remove words containing numbers
    return text
# Apply the cleaning function to the 'text' column
df['cleaned_text'] = df['text'].apply(clean_text)
# Display the cleaned text
df[['text', 'cleaned_text']].head()

#BUILDING THE COUNT VECTORS WITH CountVectorizer
# Initialize CountVectorizer
vectorizer = CountVectorizer(stop_words='english')
# Fit and transform the cleaned text data
X = vectorizer.fit_transform(df['cleaned_text'])
# Convert the label into binary format
y = df['label_num']
#print(y) optional to check the required label is displaying or not


#SPLITTING THE DATA INTO TRAINING AND TESTING SETS
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
#MODEL BUILDING (NAVIE BAYES)
# Initialize and train the model
model = MultinomialNB()
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)

#Evaluation of the Model


# Accuracy Score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
# Classification Report
print(classification_report(y_test, y_pred))
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
#TESTING WITH NEW DATA
# Assuming the model is already trained
# Define a new SMS message to test
random_sms = ["Congratulations! You've won a free ticket to Bahamas.
Call now to claim your prize!"]
# or else we can give input on runtime
# Vectorize the new SMS message using the same vectorizer used for
training data
random_sms_counts = vectorizer.transform(random_sms) # Use the
trained vectorizer to transform the new SMS
# Predict whether it's spam (1) or ham (0)
prediction = model.predict(random_sms_counts) # Use the transformed
data for prediction
# Output the result
if prediction[0] == 1:
    print("The message is SPAM.")
else:
    print("The message is NOT SPAM.")

#SAVING THE MODEL USING PICKLE

import pickle
# Save the model using pickle
with open('sms_spam_model.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)
# Save the vectorizer
with open('vectorizer.pkl', 'wb') as vectorizer_file:
    pickle.dump(vectorizer, vectorizer_file)
with open('sms_spam_model.pkl', 'rb') as model_file:
    model = pickle.load(model_file)
with open('vectorizer.pkl', 'rb') as vectorizer_file:
    vectorizer = pickle.load(vectorizer_file)
```

#download the pickle files

For deploying the flask application we need to create the directory with the project name (optional)

```
sms spam detection/        (main directory)
├── app.py                 (python file consists of flask application)
└── templates/                      (directory that contains html files)
        ├── index.html
        ├── spam.html
        └── result.html
```

Building the flask Application(app.py)

```
from flask import Flask, render_template, request
import pickle
# Load the trained model and vectorizer
with open("C:\\Users\\eshwa\\Downloads\\sms_spam_model.pkl", 'rb')
as model_file:
    model = pickle.load(model_file)
with open("C:\\Users\\eshwa\\Downloads\\vectorizer.pkl", 'rb') as
vectorizer_file:
    vectorizer = pickle.load(vectorizer_file)
app = Flask(__name__)
# Route for the home page
@app.route('/')
def home():
    return render_template('index.html')
# Route for the spam input page
@app.route('/spam')
def spam():
    return render_template('spam.html')
# Route to handle prediction
@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        # Get the message from the form
        message = request.form['message']
        # Vectorize the input message using the trained vectorizer
        message_counts = vectorizer.transform([message])
        # Make a prediction
        prediction = model.predict(message_counts)
        # Determine if it's spam or not
        result = "SPAM" if prediction[0] == 1 else "NOT SPAM"
        # Render the result page
        return render_template('result.html', prediction=result)

if __name__ == "__main__":
    app.run(debug=True, port=5001)
```

Templates

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <title>SMS Spam Detection</title>
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css">
   <style>
      body {
         background-color: #f0f8ff; /* Light blue background */
         display: flex;
         flex-direction: column;
         justify-content: center;
         align-items: center;
         height: 100vh; /* Full height of the viewport */
         margin: 0;
         font-family: Arial, sans-serif;
      }
      h1 {
         color: #333; /* Dark gray color for the heading */
         text-align: center;
         margin-bottom: 30px;
```

```
        }
        .btn-primary {
            background-color: #007bff; /* Button color */
            border-color: #007bff; /* Button border color */
            padding: 15px 30px; /* Padding for the button */
            font-size: 20px; /* Larger font size */
            transition: background-color 0.3s ease; /* Smooth transition for
hover */
        }
        .btn-primary:hover {
            background-color: #0056b3; /* Darker shade on hover */
        }

        footer {
            position: absolute; /* Footer at the bottom */
            bottom: 20px;
            font-size: 14px;
            color: #777; /* Light gray color for footer text */
        }
    </style>
</head>
<body>
    <div class="container text-center">
        <h1>Welcome to SMS Spam Detection</h1>
        <a href="/spam" class="btn btn-primary">Check if a message is
spam</a>
```

```
    </div>
    <footer>
        © 2024 SMS Spam Detection
    </footer>
</body>
</html>
```

Spam.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>SMS Spam Detection</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css">
    <style>
        body {
            background-color: #e0f7fa; /* Soft cyan background */
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            font-family: 'Poppins', sans-serif;
        }

        h1 {
            color: #005662; /* Dark teal for heading */
            font-size: 2.5rem;
            font-weight: 600;
            margin-bottom: 20px;
            text-transform: uppercase;
            letter-spacing: 2px;
            text-align: center;
            animation: fadeInDown 1s ease; /* Animation for the heading */
        }

        @keyframes fadeInDown {
            0% {
                opacity: 0;
                transform: translateY(-50px);
            }
            100% {
                opacity: 1;
                transform: translateY(0);
```

```
            }
        }

        .container {
            background-color: #ffffff; /* White background for the form
container */
            padding: 30px 40px;
            border-radius: 15px; /* Rounded container corners */
            box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1); /* Soft shadow */
            max-width: 600px;
            width: 100%;
            text-align: center;
        }

        .form-group {
            margin-bottom: 25px; /* Larger space between elements */
        }

        textarea {
            resize: none;
            border: 2px solid #00838f; /* Teal border */
            border-radius: 10px; /* Rounded corners for textarea */
            padding: 15px;
            font-size: 16px;
            width: 100%;
            transition: border-color 0.3s ease; /* Smooth border transition */
        }

        textarea:focus {
            border-color: #005662; /* Darker border on focus */
            outline: none;
            box-shadow: 0 0 5px rgba(0, 131, 143, 0.5); /* Shadow effect on
focus */
        }

        .btn-primary {
            background-color: #00838f; /* Teal color */
            border-color: #00838f;
            color: #fff;
            padding: 15px 30px;
            font-size: 1.2rem; /* Larger font size */
            border-radius: 10px; /* Rounded corners */
            cursor: pointer;
            transition: background-color 0.3s ease, transform 0.3s ease; /*
Smooth hover effect */
            display: inline-block;
            width: 100%; /* Make the button fill the available space */
            text-align: center;
        }

        .btn-primary:hover {
            background-color: #005662; /* Darker teal on hover */
```

```
    transform: scale(1.05); /* Slight zoom effect on hover */
}

footer {
    margin-top: 20px;
    font-size: 14px;
    color: #666;
    position: fixed;
    bottom: 10px;
    left: 0;
    right: 0;
    text-align: center;
}

/* Responsive Design */
@media (max-width: 768px) {
    h1 {
        font-size: 2rem;
    }
    .container {
        padding: 20px;
    }
    .btn-primary {
        font-size: 1rem;
    }
```

```
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Enter your message</h1>
        <form action="/predict" method="POST">
            <div class="form-group">
                <textarea class="form-control" name="message" rows="6"
required placeholder="Type your message here..."></textarea>
            </div>
            <button type="submit" class="btn btn-primary">Check
Spam</button>
        </form>
    </div>
    <footer>
        © 2024 SMS Spam Detection. All rights reserved.
    </footer>
</body>
</html>
```

Result.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Prediction Result</title>
    <!-- Include Bootstrap for better styling -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css">
    <style>
        body {
            background-color: #f0f8ff; /* Light blue background */
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
        }
        .alert-box {
            max-width: 500px;
            padding: 20px;
            border-radius: 8px;
            text-align: center;
            box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.1);
        }
        .spam-icon {
            font-size: 50px;
            color: red;
        }
        .not-spam-icon {
            font-size: 50px;
            color: green;
        }
        h1 {
            margin-bottom: 20px;
        }
    </style>
</head>
<body>

    <!-- Conditionally show alert box based on the prediction result -->
    <div class="alert-box alert
        {% if prediction == 'SPAM' %}
            alert-danger
        {% else %}
            alert-success
        {% endif %}
        ">
        <!-- Display an alert icon depending on spam or not spam -->
        {% if prediction == 'SPAM' %}
            <div class="spam-icon">⚠</div>
            <h1>This message is SPAM!</h1>
        {% else %}
            <div class="not-spam-icon">✅</div>
            <h1>This message is NOT SPAM!</h1>
        {% endif %}
        <p>The message is classified as: <strong>{{ prediction
}}</strong></p>
        <a href="/" class="btn btn-primary mt-3">Go back</a>
    </div>
    <!-- Include Bootstrap JS (optional) -->
    <script src="https://code.jquery.com/jquery-
3.5.1.slim.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.2/dist/umd/popp
er.min.js"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.mi
n.js"></script>
</body>
</html>
```

11.GitHub & Project Demo Link