# System Requirements Specification Index

For

# NLP Usecase

## SPAM FILTER L1

1.0

## IIHT Pvt. Ltd.

[fullstack@iiht.com](mailto:fullstack@iiht.com)

Problem Statement : **SPAM Filter NLP -L1**

Description : Use relevant methods operations to perform specified activities which are given in the instructions.

#### NLP -L1

**Building a Spam Filter for a New Messaging App**

Background: Imagine you are working as a data scientist at a new messaging app called "ChatLink." The app allows users to send text messages, but with its growing popularity, the number of unsolicited and unwanted messages (also known as "spam") has increased significantly. These spam messages include promotional content, phishing attempts, and other forms of digital noise that disrupt the user experience.

To address this issue, the product team has tasked you with developing an automated spam detection system that will help users easily filter out spam messages from their inboxes. The goal is to implement a machine learning-based spam filter that can accurately classify incoming messages as either "spam" or "ham" (legitimate, non-spam messages).

The Process:

1. Problem Definition: The main problem you need to solve is to automatically classify incoming messages as either "spam" or "ham." Spam messages are unwanted, while ham messages are regular, legitimate messages that the user wants to read.
2. Data Collection: To build the spam detection system, you begin by gathering a dataset of text messages labeled as either "spam" or "ham." These messages represent a variety of scenarios:
   * Spam Messages: "Win a free iPhone!", "Get rich quick with this simple investment scheme!", "Click here to claim your free prize."
   * Ham Messages: "Hey, can we meet for lunch tomorrow?", "Just wanted to check if you received the report I sent?", "Hope you're doing well, let's catch up soon!"

This labeled dataset is crucial for training a machine learning model that will distinguish between spam and ham messages.

1. Preprocessing the Data: Once the dataset is ready, the next step is preprocessing. You clean the text data to make it suitable for machine learning:
   * Removing Special Characters: You eliminate punctuation marks, numbers, and other non-alphabetic characters from the messages since they do not contribute to the classification task.
   * Lowercasing: All text is converted to lowercase to ensure that the model treats words like "Free" and "free" as the same word.
2. Feature Extraction: After preprocessing the data, you need to convert the text messages into a numerical format that a machine learning model can understand. For this, you use a CountVectorizer, which converts each message into a vector representing the frequency of each word in the message. This step is essential because the model needs to learn patterns in word frequencies to distinguish between spam and ham.
3. Model Training: Now that the data is preprocessed and transformed into numerical features, you can split the dataset into two parts: one for training the model and the other for testing it. You decide to use a Multinomial Naive Bayes classifier, which is a popular algorithm for text classification tasks, especially when the data consists of word frequencies.

The Naive Bayes model is trained using the training dataset. It learns the probability of a message being spam based on the words in the message. For example, the word "free" might be highly indicative of spam, while words like "dinner" or "report" are more likely to be found in ham messages.

1. Model Evaluation: After training the model, you evaluate its performance on the test dataset. You check:
   * Accuracy: How well the model is able to classify messages correctly as either spam or ham.
   * Confusion Matrix: This matrix helps you understand how many messages were correctly or incorrectly classified:
     + True positives (spam correctly identified as spam)
     + True negatives (ham correctly identified as ham)
     + False positives (ham incorrectly classified as spam)
     + False negatives (spam incorrectly classified as ham)
2. Deployment: With a trained and evaluated model, you are now ready to integrate the spam detection system into the ChatLink messaging app. As users send and receive messages, the system will classify each incoming message in real-time. Users will be able to review the classified messages and decide whether they want to mark them as spam or ham. Over time, the system will improve as it learns from new data and user feedback.
3. Analytics and Insights: While developing the system, you also create analytical tools that help you understand the dataset and track the system's performance:
   * You calculate the spam ratio and ham ratio to get an idea of how much spam versus ham is in the dataset.
   * You also track the number of correctly and incorrectly classified messages to understand how well the system is performing.

Outcome: After successfully deploying the spam filter, ChatLink users experience a cleaner inbox. Spam messages are automatically filtered out, and users only see messages that matter to them. This improves the overall user experience and trust in the app.

Reflection: Building a spam filter for ChatLink was a challenging but rewarding task. By using machine learning techniques like Naive Bayes and text preprocessing methods, you were able to develop a system that accurately detects spam and helps protect users from unwanted messages.

List of Analytical Questions for the Code:

1. Total number of messages: How many total messages are there in the dataset?
2. Number of spam messages: How many messages in the dataset are labeled as spam?
3. Number of ham messages: How many messages in the dataset are labeled as ham?
4. Spam message ratio: What is the percentage of spam messages in the dataset?
5. Ham message ratio: What is the percentage of ham messages in the dataset?
6. Accuracy of the classifier: What is the accuracy of the Naive Bayes classifier on the test dataset?
7. Number of correctly classified messages: How many messages were correctly classified by the model (both spam and ham)?
8. Number of incorrectly classified messages: How many messages were incorrectly classified by the model (either as spam or ham)?

**Execution Steps to Follow:**

1. All actions like build, compile, running application,running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
3. This editor Auto Saves the code
4. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the

internal git/repository. Else the code will not be available in the next login.

1. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
2. To setup environment:

You can run the application without importing any packages

1. To launch application:

nlptest.py

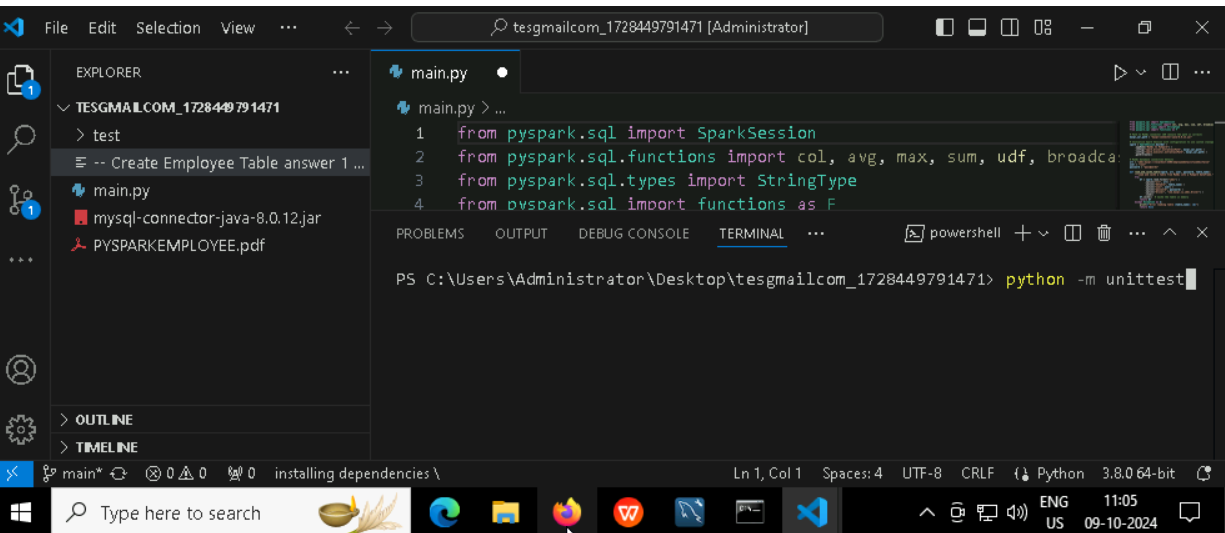
1. To run Test cases:

python -m unittest

Before Final Submission also, you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository for code

**To run the application**

* **Python nlptest.py**



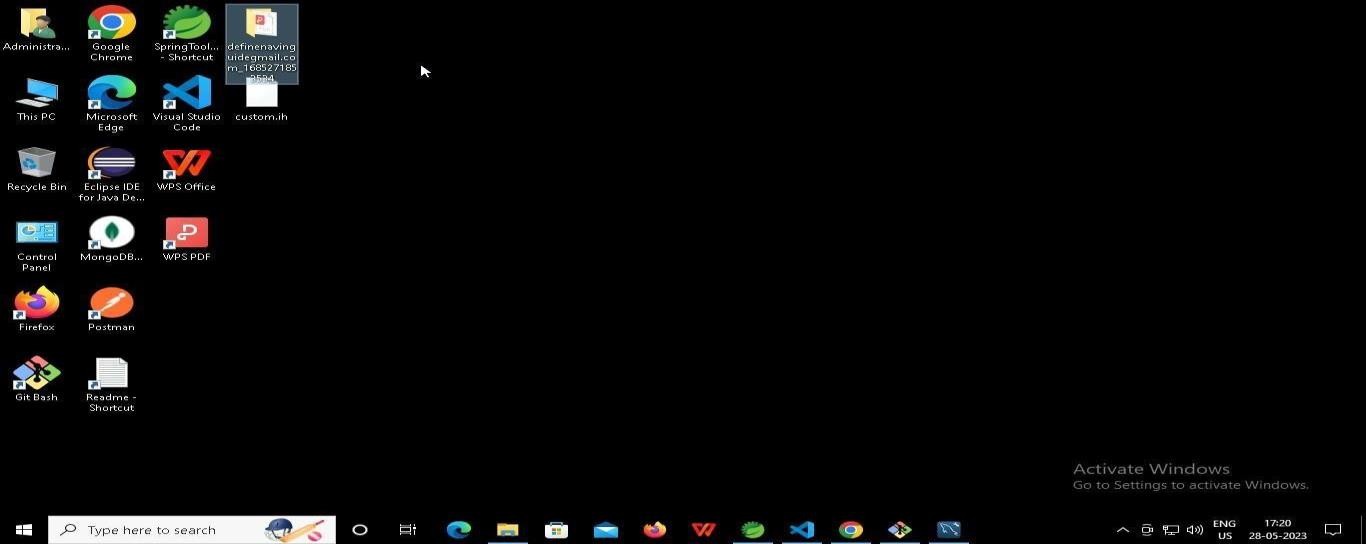
**To run the testcase**

* **Python -m unittest**

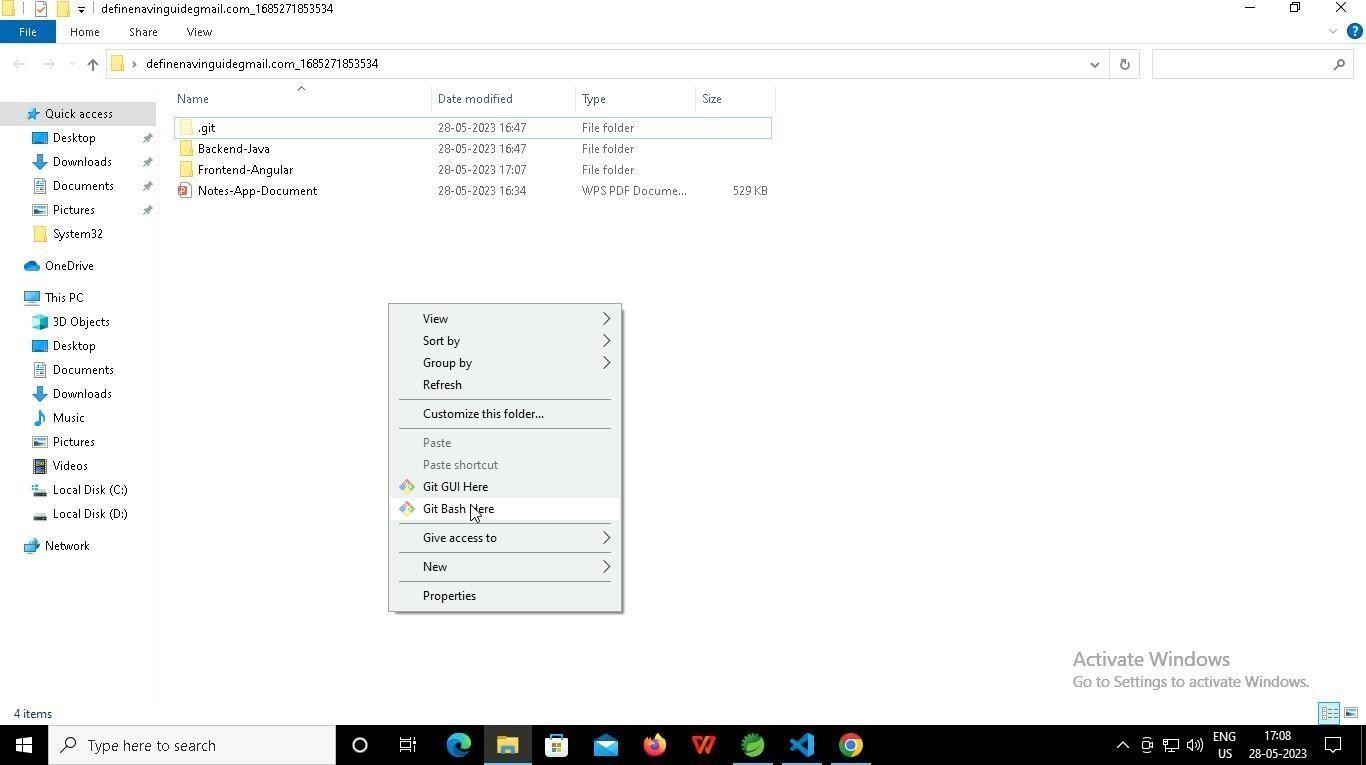
**Screenshot to push the application to github**

#### You can run test cases as many numbers of times and at any stage of Development, to check how many test cases are passed/failed and accordingly refactor your code.

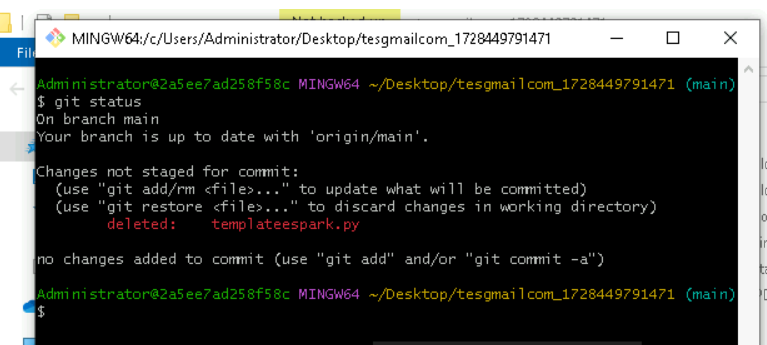
* 1. **Make sure before final submission you commit all changes to git**. For that open the project folder available on desktop



* + 1. **Right click in folder and open Git Bash**



* + 1. **In Git bash terminal, run following commands**
    2. **git status**



* + 1. **git add .**

A black screen with yellow and purple text

Description automatically generated

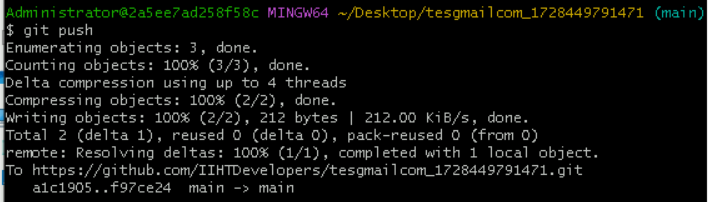
* + 1. git commit -m “First commit”

(You can provide any message every time you commit)

A screenshot of a computer

Description automatically generated

* + 1. **git push**



### After you have pushed your code Finally click on the final submission button

### 

### You should see a screen like this you will have to wait for the results . after getting this page you can leave the system

### A blue screen with white text Description automatically generated

### -----x-----