



LIST OF CONTENTS

Chapter No	Chapter Name	Starting Page Number
1	Introduction To Email Classification System	3
2	Literature Survey for Email Classification	6
3	Design & Analysis	10

CHAPTER 1
INTRODUCTION TO EMAIL CLASSIFICATION SYSTEM

1.1 WHAT IS A CLASSIFICATION SYSTEM

A classification system is a structured framework designed to categorize data into predefined groups or classes based on specific features or patterns. In the context of email processing, such a system analyzes the content of incoming emails and assigns them to relevant categories, such as billing issues, technical support, or account management. The primary goal of a classification system is to streamline workflows, improve efficiency, and ensure that each email is directed to the appropriate department or team for resolution. By leveraging machine learning, deep learning, or rule-based algorithms, these systems can automatically interpret textual data, identify key themes, and make accurate classifications with minimal human intervention. This not only reduces manual effort but also enhances response times and overall customer satisfaction.

Classification systems rely on a combination of data preprocessing, feature extraction, and model training to achieve high accuracy. For instance, in email classification, text data is first cleaned and transformed into numerical representations, such as word embeddings or TF-IDF vectors, which capture the semantic and syntactic nuances of the content. Advanced models, including traditional algorithms like Naïve Bayes or SVM, or sophisticated neural networks like BERT, are then trained on labeled datasets to recognize patterns and predict categories. Additionally, such systems often incorporate safeguards like PII masking to protect sensitive information before processing. The result is a robust, scalable solution that can handle large volumes of emails while maintaining data privacy and operational efficiency.

1.2 INTRODUCTION TO EMAIL CLASSIFICATION SYSTEM

Email classification systems have evolved significantly since the early days of digital communication, where manual sorting was the only option. With the exponential growth of email traffic, businesses and organizations needed automated solutions to efficiently categorize and route messages. Early approaches relied on rule-based systems using keyword matching, but these were limited in handling variations in language. The advent of machine learning (ML) and natural language processing (NLP) revolutionized email classification by enabling systems to learn from data and improve accuracy over time. Today, advanced techniques such as deep learning and transformer-based models (e.g., BERT, GPT) further enhance classification by understanding context, sentiment, and intent, making them indispensable for customer support, spam filtering, and enterprise communication management.

There are multiple ways to build an email classification system, ranging from traditional ML models like Naïve Bayes, Support Vector Machines (SVM), and Random Forests to modern deep learning architectures such as LSTMs and transformer-based models. Rule-based systems, though less flexible, are still useful for simple categorizations, while hybrid approaches combine ML with predefined rules for better adaptability. The choice of method depends on factors like

dataset size, computational resources, and required accuracy. The advantages of automated email classification are numerous: it reduces manual workload, ensures faster response times, improves customer experience, and enhances data security by integrating PII masking. Additionally, businesses gain actionable insights from categorized emails, enabling better decision-making and resource allocation. As AI continues to advance, email classification systems will become even more sophisticated, further optimizing communication workflows across industries.

CHAPTER 2

LITERATURE SURVEY FOR EMAIL CLASSIFICATION

2.1 LITERATURE SURVEY TABLE

PAPER TITLE	PAPER ABSTRACT	PAPER LINK
A COMPARATIVE STUDY OF TEXT CLASSIFICATION ALGORITHMS FOR EMAIL FILTERING	This paper evaluates traditional ML algorithms (Naïve Bayes, SVM, Decision Trees) and deep learning models (LSTM, CNN) for email classification. Results show that ensemble methods and neural networks outperform simpler models in accuracy, especially for large datasets.	Click Here to See Research Paper
BERT FOR EMAIL CLASSIFICATION: A DEEP LEARNING APPROACH	The study fine-tunes BERT (Bidirectional Encoder Representations from Transformers) for email categorization, achieving state-of-the-art results on the Enron dataset. It highlights BERT's ability to capture contextual semantics, outperforming traditional TF-IDF and word2vec-based models.	Click Here to See Research Paper
PRIVACY-PRESERVING EMAIL CLASSIFICATION USING FEDERATED LEARNING	Proposes a federated learning framework to classify emails without centralizing sensitive data. Experiments on client-distributed email datasets show competitive accuracy while preserving user privacy, making it suitable for enterprise applications.	Click Here To See The Research Problem
HIERARCHICAL ATTENTION NETWORKS FOR SPAM AND PHISHING EMAIL DETECTION	Introduces a hierarchical attention network (HAN) to classify malicious emails. The model leverages word- and sentence-level attention mechanisms to improve interpretability and accuracy, achieving 98% F1-score on a phishing email dataset.	Click Here To See The Research Paper
FEW-SHOT LEARNING FOR EMAIL INTENT CLASSIFICATION	Addresses the challenge of limited labeled data by using few-shot learning techniques (e.g., Prototypical Networks). The approach achieves robust performance on niche email categories with minimal training examples.	Click Here To See The Research Paper

2.1 DESCRIPTIONS

A COMPARATIVE STUDY OF TEXT CLASSIFICATION ALGORITHMS FOR EMAIL FILTERING

In the chapter "*A Comparative Study for Email Classification*" by Seongwook Youn and Dennis McLeod, the authors investigate the effectiveness of various machine learning algorithms in detecting email spam. With the increasing volume of email communication, spam filtering has become a critical issue. The study compares four classifiers—Neural Network, Support Vector Machine (SVM), Naïve Bayesian

Classifier, and J48 Decision Tree—across different dataset sizes and feature sets. The classification was binary, distinguishing between spam (1) and non-spam (0), and aimed to assess each model's performance and efficiency under varying conditions.

The results showed that the J48 decision tree classifier outperformed others, particularly when the dataset aligned well with binary tree structures, highlighting the potential of decision tree-based approaches for spam detection. The chapter also draws on foundational literature in the field, referencing prior work such as Naïve Bayesian and memory-based filtering methods (Androutsopoulos et al.), rule learning techniques (Cohen), neural networks for email classification (Cui et al.), and Bayesian filtering approaches (Sahami et al.). This contextual grounding supports the study's conclusions and demonstrates the evolving landscape of email classification research.

BERT FOR EMAIL CLASSIFICATION: A DEEP LEARNING APPROACH

In the paper titled "BERT Model: A Text Classification Technique in NLP," authors Anudeepa Gon, Dr. Gunjan Mukherjee, Dr. Kaushik Chanda, Subhadip Nandi, and Aryabhatta Ganguly from Brainware University explore the application of BERT (Bidirectional Encoder Representations from Transformers) for text classification tasks. They propose a hybrid model that integrates BERT with Convolutional Neural Networks (CNNs) to leverage BERT's ability to capture global contextual information and CNNs' strength in identifying local patterns. The methodology involves tokenizing input text using BERT's tokenizer, obtaining contextual embeddings, applying convolutional layers to extract local features, and then combining these representations for classification tasks such as sentiment analysis and spam detection. The model is fine-tuned on labeled datasets, with certain BERT layers unfrozen to better adapt to specific tasks, and validated to assess performance. [ResearchGate](#)

Key findings from their experiments indicate that the BERT-CNN hybrid model outperforms traditional models in text classification accuracy. The authors highlight the model's effectiveness in capturing both global and local textual features, leading to improved classification performance. They also note the potential for further enhancements through transfer learning and domain adaptation, especially when substantial labeled data is available. The study concludes that integrating BERT with CNNs provides a robust framework for various NLP applications, offering a balance between contextual understanding and feature extraction.

PRIVACY-PRESERVING EMAIL CLASSIFICATION USING FEDERATED LEARNING

In the article titled "Evaluation of Federated Learning in Phishing Email Detection," the authors investigate the application of federated learning (FL) for detecting phishing emails. They address the challenge of data privacy in machine learning by utilizing FL, which allows multiple clients to collaboratively train a model without sharing their raw data. The study involves implementing a federated learning framework using the Flower framework and evaluating its performance on a phishing email dataset. The authors compare the results of the federated model with a centralized model to assess the effectiveness of FL in this context.

The key findings indicate that the federated learning model achieves comparable performance to the centralized model while preserving data privacy. The study demonstrates that FL can be an effective approach for phishing email detection, especially in scenarios where data privacy is a concern. The

authors conclude that federated learning offers a promising solution for collaborative training of machine learning models in sensitive applications like email security, without compromising user data privacy.

HIERARCHICAL ATTENTION NETWORKS FOR SPAM AND PHISHING EMAIL DETECTION

In the paper titled "Email Spam Detection Using Hierarchical Attention Hybrid Deep Learning Method," authors Sultan Zavrak and Seyhmus Yilmaz from Duzce University present a novel approach to enhancing email spam detection. Their method integrates Convolutional Neural Networks (CNNs), Gated Recurrent Units (GRUs), and attention mechanisms to form a hybrid deep learning model. This architecture is designed to focus selectively on pertinent parts of email text during training, enabling the extraction of meaningful and generalizable features through hierarchical representation. A significant contribution of their study is the incorporation of cross-dataset evaluation, which assesses the model's performance on datasets different from those it was trained on, thereby evaluating its generalizability.

The experimental results demonstrate that this hybrid model outperforms existing attention-based techniques in spam detection tasks. By leveraging temporal convolutions, the model benefits from flexible receptive field sizes, enhancing its ability to capture contextual information within emails. The authors conclude that their approach not only improves detection accuracy but also offers robustness across various datasets, making it a promising solution for real-world email spam filtering applications.

FEW-SHOT LEARNING FOR EMAIL INTENT CLASSIFICATION

In the paper titled "Advancing Text Classification: A Systematic Review of Few-Shot Learning Approaches," authors Amani Aljehani, Syed Hamid Hasan, and Usman Ali Khan from King Abdulaziz University provide a comprehensive review of recent advancements in few-shot learning (FSL) techniques applied to text classification tasks. Recognizing the challenges posed by limited labeled data, especially in niche domains and low-resource languages like Arabic, the authors examine 32 studies that explore various FSL methodologies. These methodologies include metric-based, model-based, and optimization-based approaches, as well as the utilization of transfer learning and pre-trained language models. The review emphasizes the significance of these techniques in enabling models to learn effectively from minimal data, thereby addressing the data scarcity issue prevalent in many text classification scenarios.

The authors also delve into the specific challenges associated with applying FSL to Arabic text classification, such as morphological complexities and the scarcity of annotated datasets. They highlight the importance of developing versatile models capable of generalizing across diverse textual domains while maintaining high performance. Furthermore, the review discusses the real-world impacts of FSL in text classification and suggests future research directions to overcome existing limitations. By providing this systematic analysis, the paper serves as a valuable resource for researchers and practitioners aiming to advance the field of text classification through few-shot learning techniques.

CHAPTER 3

DESIGN & ANALYSIS

3.1 INTRO TO GIVEN TASK

The assignment titled "**Email Classification for Support Team**" focuses on developing a system that can automatically categorize incoming support emails into predefined categories like Billing Issues, Technical Support, and Account Management. A key requirement of the project is to ensure that all personally identifiable information (PII) such as names, email addresses, phone numbers, and card details are masked before processing, without using large language models (LLMs) for this task. The project seeks to balance automation with data privacy, aiming to process emails intelligently while safeguarding sensitive information.

To complete this assignment, you are required to carry out several steps: start with data collection and preprocessing using the provided dataset, mask the PII using methods like Named Entity Recognition (NER), machine learning models, or regex (excluding LLMs), and store original data securely. Then, select an appropriate classification model — either a traditional ML model, a deep learning model, or an LLM — and train it to classify emails. Following this, integrate these steps into a pipeline that masks, classifies, and then demasks the email data. The entire system must be exposed via an API that accepts an email as input and returns both the masked email and its predicted category.

The expected outcome is a fully functional and deployed API (e.g., on Hugging Face Spaces) that strictly adheres to the given JSON format for outputs. The submission must include clean, well-commented code, documentation, and a short report explaining the approach, challenges, and results. The evaluation will be based on API deployment, code quality, test case coverage, and how well the output format is maintained. This project not only tests your machine learning and NLP skills but also emphasizes robust API design and data protection practices.

3.2 HOW THE TASK IS SOLVED

- First of all I downloaded the data from the google drive through the link mentioned in the pdf.
- After that I started Analyzing the data and preprocessing the data.
- After Data Preprocessing I Performed Exploratory Data Analysis.
- During Data Preprocessing I Implemented Some NLP Text Processing Techniques. So, I Used Vectorizers to vectorize email data. And Encoded type column with ordinal encoding.
- Finally I Implemented more than 10+ classification algorithms from scikit learn to create a machine learning model. Downloaded ML Model, Vectorizer, Scaler in .pkl format.

3.3 DETAILED INFO ON TASK IMPLEMENTATION

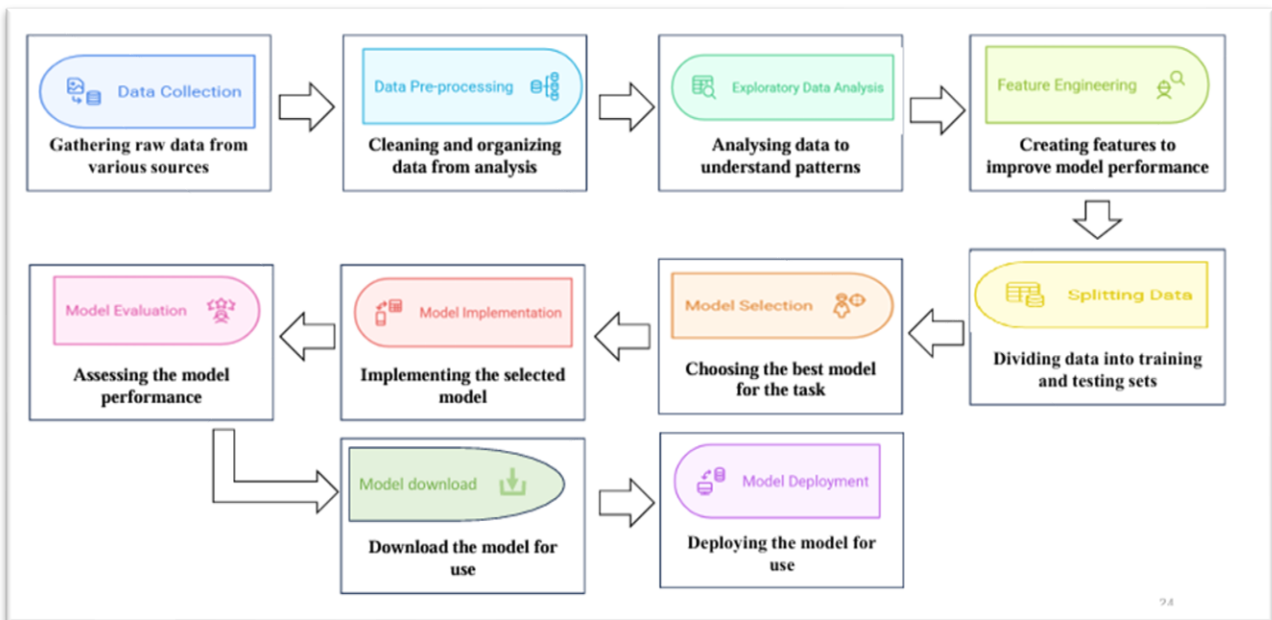


Fig 3.1 : Proposed Architecture Diagram

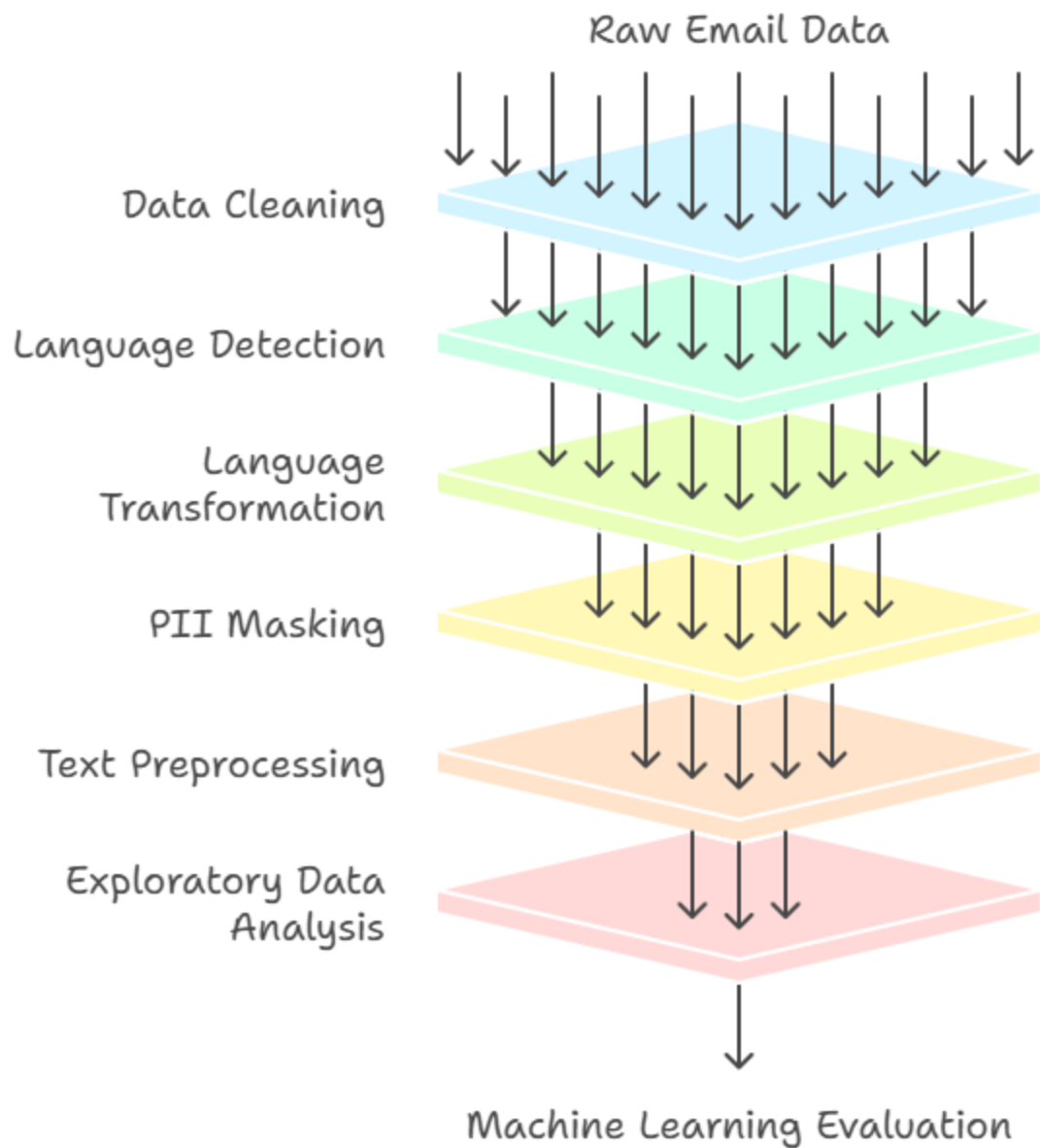
Fig 3.1 says the stages that are implemented to complete the assignment. The stages including collecting data from the drive link given in the pdf. Analyzing data, cleaning data and transforming data etc. A detailed steps are given below that are implemented in each stage.

Data Collection: [combined_emails_with_natural_pii.csv - Google Drive](#) [source : link given in pdf]

READING DATA: using pandas readied the data and created data frame, and analyzed basic information like column names, column dtypes, data frame shape and size etc.

DATA PREPROCESSING: After knowing some basic information about data set, performed the following data preprocessing operations.

Data Preprocessing and Analysis Funnel



Made with  Napkin

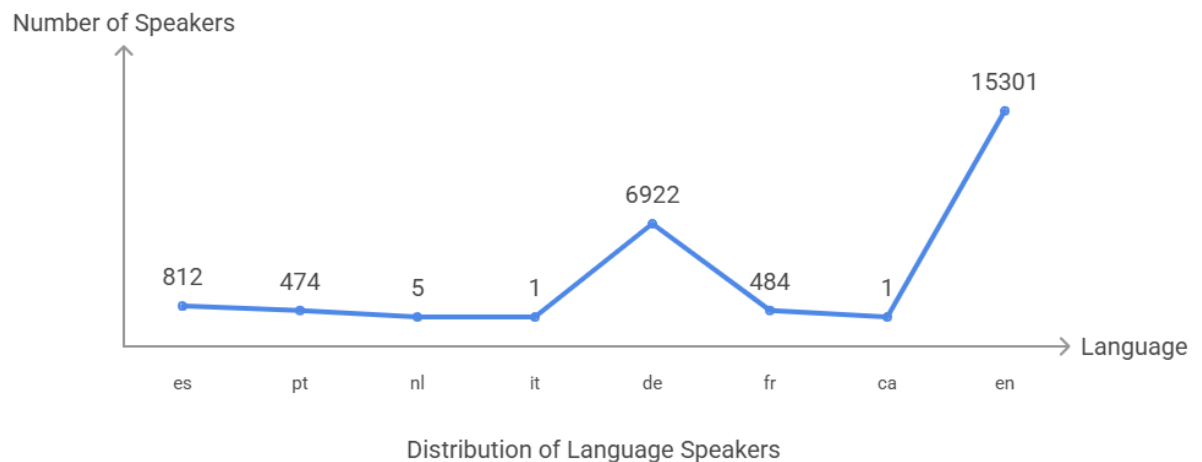
Fig 3.2 : Steps Performed For Email Classification Sysytem

- **Removed Duplicated Rows:** No duplicate rows found

- **Removed Missing Values:** No Missing Values Found
- **Analyzed language used in email column:** Noticed that email contains values written in English but their meaning would be in different languages. Like writing French content in English words where the words are only in English but the meaning is in French language. Below is the detained list.

OUTPUT : Detecting emails Language: 100%| [REDACTED] | 24000/24000 [01:58<00:00, 202.75it/s]

{'es': 812, 'pt': 474, 'nl': 5, 'it': 1, 'de': 6922, 'fr': 484, 'ca': 1, 'en': 15301}



Made with Napkin

Fig 3.3 : Email Content Language Count In Column email

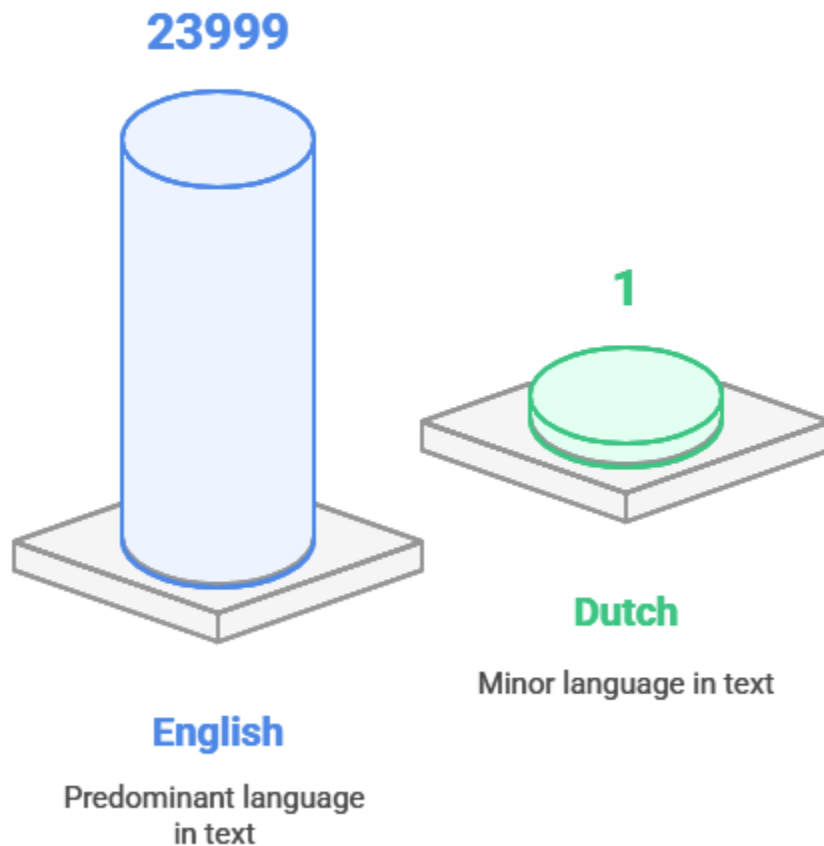
- **Transformed All Content into pure English:** using python programming language converted all email content into English. This step alone took more than 2 hours to execute.

```

BEFORE TRANSLATION :
Detecting emails Language: 100%|██████████| 24000/24000 [01:47<00:00, 224.13it/s]
{'en': 23989, 'nl': 3, 'de': 6, 'fr': 2}
Translating emails: 100%|██████████| 24000/24000 [01:57<00:00, 205.10it/s]
Detecting emails Language: 100%|██████████| 24000/24000 [01:47<00:00, 222.41it/s]
AFTER TRANSLATION :
{'en': 23999, 'nl': 1}

```

Comparison of Language Counts



Made with Napkin

Fig 3.4 : Language Detected For Content in column email After Performing Translation

- **Implementing PII Masking :** Implemented PII Masking which follows the specified rules
 - Full Name ("full_name")
 - Email Address ("email")
 - Phone number ("phone_number")
 - Date of birth ("dob")
 - Aadhar card number ("aadhar_num")
 - Credit/ Debit Card Number ("credit_debit_no")
 - CVV number ("cvv_no")
 - Card expiry number ("expiry_no")
- **Some Basic Text Preprocessing Steps Performed in NLP:**
 1. Lower Casing

2. Removing HTML Tags
3. Encoding emoji's
4. Removing Punctuations
5. Removing Stop Words
6. Tokenization
7. Stemming
8. Lemmatization

Steps to Effective Text Preprocessing



Fig 3.5 : Text Preprocessing Steps On Masked Data

- **Performed Exploratory Data Analysis**
 1. Visualized Value counts of each categories in type column
 2. Visualizing word cloud graphs for each category present in type column.
- **Feature Engineering :**
 1. Performed Vectorization Operations
- **Implemented Train Test Split with training size as 80%**
- **Implemented 10+ classification algorithms**
- **Evaluated each machine learning algorithm through metrics like accuracy and classification report.**

3.4: WHAT HAPPEN WHEN YOU INPUT AN EMAIL FOR CLASSIFICATION

- 1 Detecting the language used in email
- 2 Translating Into English language if detected language is not in English
- 3 Implementing PII Masking
- 4 Text Preprocessing
- 5 Vectorization
- 6 Prediction

Steps to Classify Email



Fig 3.6 : Output Process

CHAPTER 4

CODE IMPLEMENTATION & RESULTS

4.1 CODE FOR MODEL TRAINING

[You can access more code here](#)

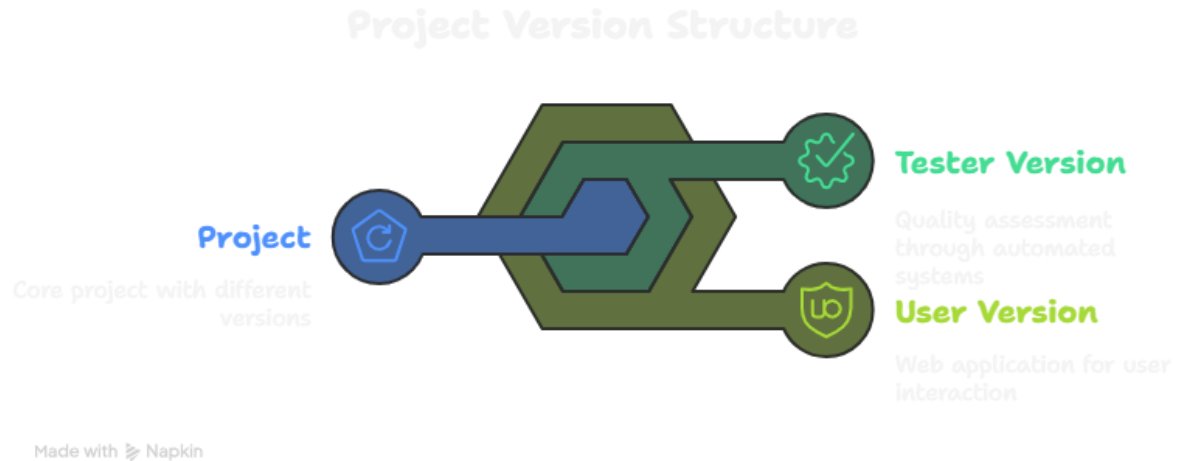
4.2 CODE USED FOR PRODUCTION

TESTER VERSION: <https://huggingface.co/spaces/Rohith25Jan/email-classification>

USER VERSION: <https://huggingface.co/spaces/Rohith25Jan/email-classification-with-ui>

COCLUSION

STEP BY STEP GUIDE ON HOW TO USE THIS PROJECT



This project comes with two versions which are known as

1. Tester version
2. User version

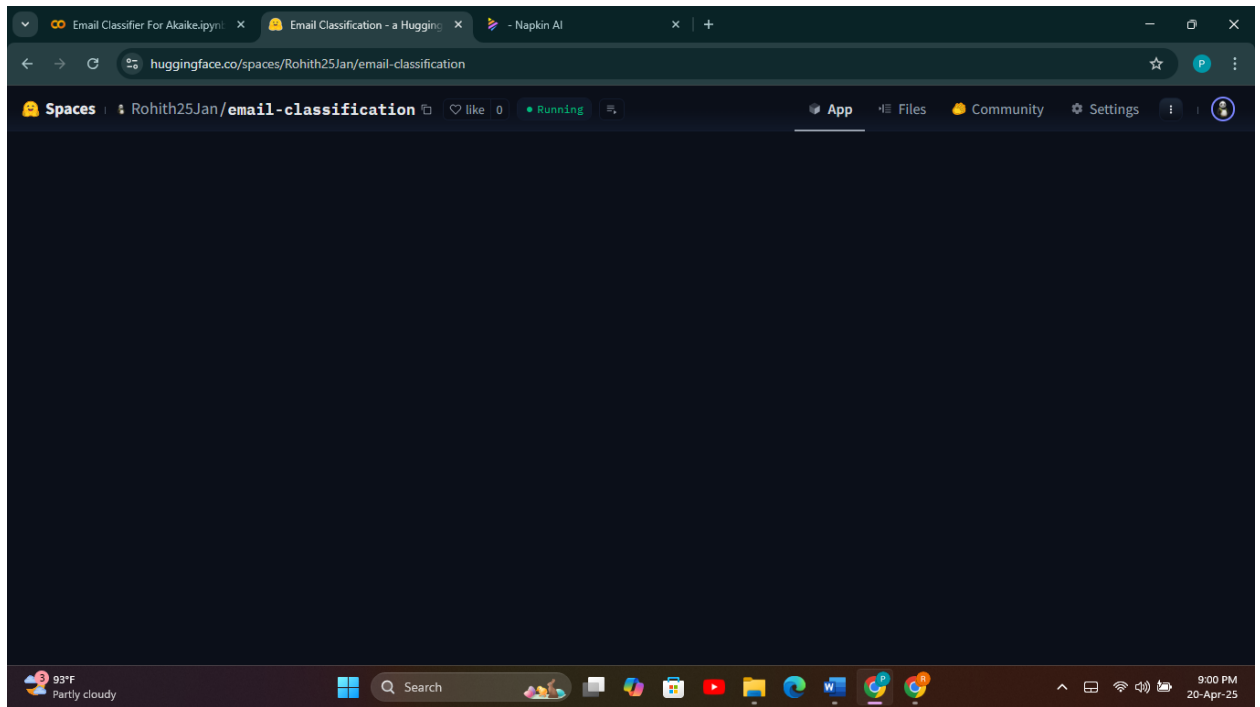
Tester version is used for assessing the quality of this project through the system or any automated grading system. Whereas user version comes with a web site application with front end, so that user can use this web application.

You can access any version with the below given links : Tester Version : User version :

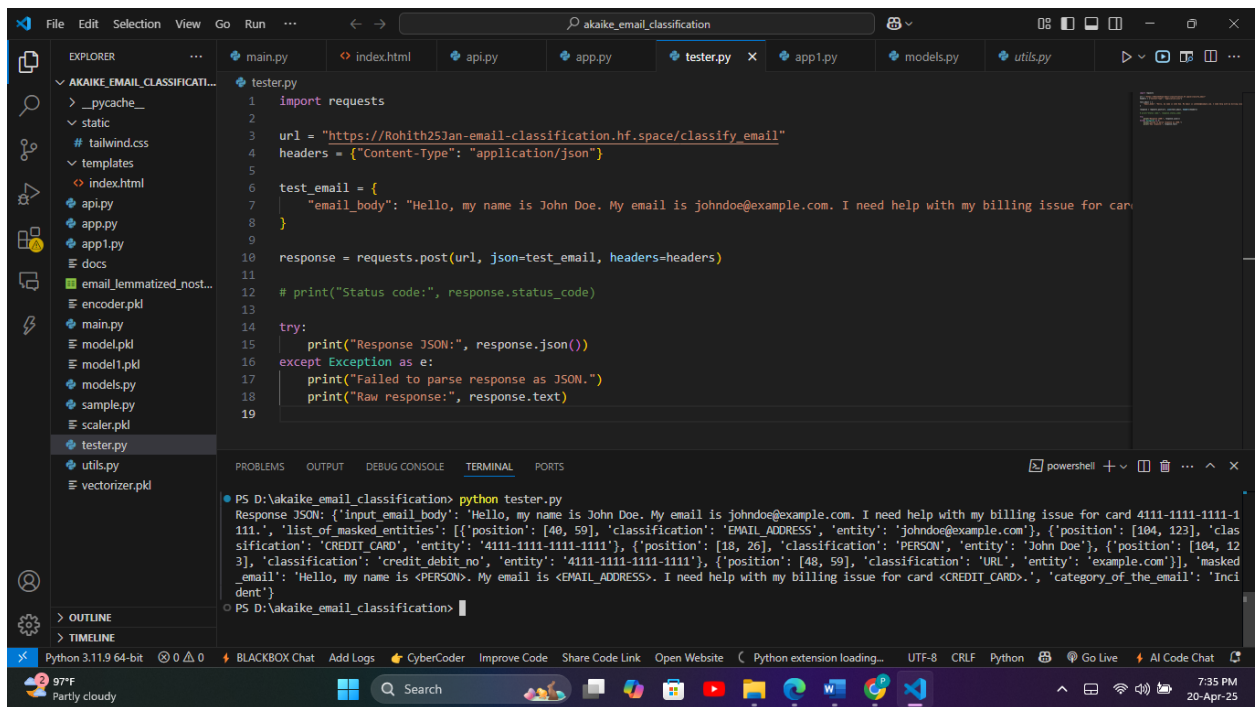
I used hugging face Spaces as a deployment platform with docker image.

STEP BY STEP GUIDE ON HOW TO USE TESTER VERSION

1. Note : Use this link (<https://huggingface.co/spaces/Rohith25Jan/email-classification>) to go the tester version.



- 2.
3. Actually, tester version is made for assessing the quality of project and validating output structure mentioned in the pdf.
4. So, when you redirected to the web site, you may not see any thing except one blank web page which is in Running State.
5. So, you no need to visit the website if you want to test this project.
6. But, to test the project you need to download the tester.py file present in Python Files Directory ---> Email Classifier For Testing ---> tester.py
7. After downloading the tester.py, open it using any code editor or IDLE. I used vs code. You can see the image for refernce



```
1 import requests
2
3 url = "https://Rohith25Jan-email-classification.hf.space/classify_email"
4 headers = {"Content-Type": "application/json"}
5
6 test_email = {
7     "email_body": "Hello, my name is John Doe. My email is johndoe@example.com. I need help with my billing issue for card 4111-1111-1111-1111."
8 }
9
10 response = requests.post(url, json=test_email, headers=headers)
11
12 # print("Status code:", response.status_code)
13
14 try:
15     print("Response JSON:", response.json())
16 except Exception as e:
17     print("Failed to parse response as JSON.")
18     print("Raw response:", response.text)
19
```

```
PS D:\akaik_email_classification> python tester.py
Response JSON: {'input_email_body': 'Hello, my name is John Doe. My email is johndoe@example.com. I need help with my billing issue for card 4111-1111-1111-1111.', 'list_of_masked_entities': [{'position': [40, 50], 'classification': 'EMAIL_ADDRESS', 'entity': 'johndoe@example.com'}, {'position': [104, 123], 'classification': 'CREDIT_CARD', 'entity': '4111-1111-1111-1111'}, {'position': [18, 26], 'classification': 'PERSON', 'entity': 'John Doe'}, {'position': [104, 123], 'classification': 'credit_debit_no', 'entity': '4111-1111-1111-1111'}, {'position': [48, 59], 'classification': 'URL', 'entity': 'example.com'}], 'masked_email': 'Hello, my name is <PERSON>. My email is <EMAIL_ADDRESS>. I need help with my billing issue for card <CREDIT_CARD>.', 'category_of_the_email': 'Incident'}
```

- 8.
9. So, You can keep the code as it is, i will explain the code in short. The code uses request model and it sends a POST request to the **Tester version** website, which is made using FastAPI. So, all the processing will be done using the .py files present in **Python Files Directory/Email Classifier For Testing**. Returns a json object. which will be fetched using request.response. If successfull you will get 200 server response.
10. If you want to send your email_body, just change the value for the key calld 'email_body' in test_email dictionary. Refer the below image.



```
6 test_email = {
7     "email_body": "Hello, my name is John Doe. My email is johndoe@example.com. I need help with my billing issue for card 4111-1111-1111-1111."
8 }
```

- 11.
12. You can give email_body in **any language** but you have to give it as a string datatype which may be single line or multi line string.
13. The output for the above email_body is shown below

```

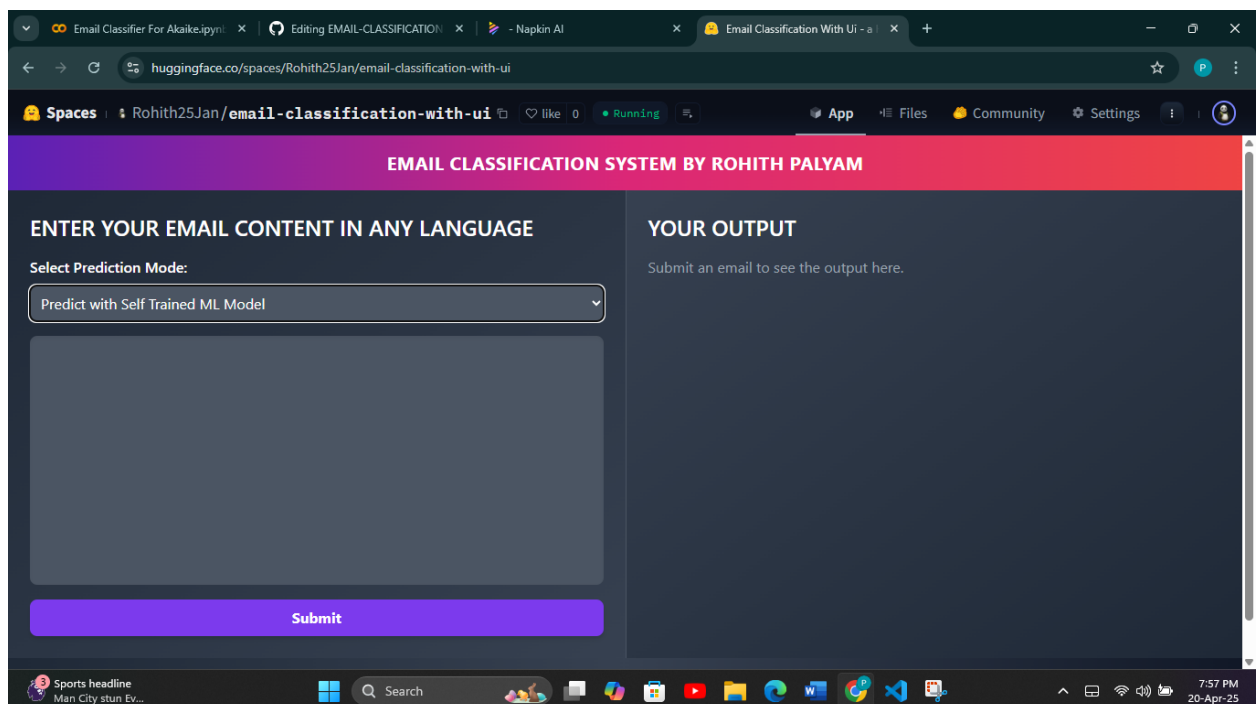
14. PS D:\akaik_email_classification> python tester.py
Response JSON: {'input_email_body': 'Hello, my name is John Doe. My email is johndoe@example.com. I need help with my billing issue for card 4111-1111-1111-1111.', 'list_of_masked_entities': [{'position': [40, 59], 'classification': 'EMAIL_ADDRESS', 'entity': 'johndoe@example.com'}, {'position': [104, 123], 'classification': 'CREDIT_CARD', 'entity': '4111-1111-1111-1111'}, {'position': [18, 26], 'classification': 'PERSON', 'entity': 'John Doe'}, {'position': [104, 123], 'classification': 'credit_debit_no', 'entity': '4111-1111-1111-1111'}, {'position': [48, 59], 'classification': 'URL', 'entity': 'example.com'}], 'masked_email': 'Hello, my name is <PERSON>. My email is <EMAIL_ADDRESS>. I need help with my billing issue for card <CREDIT_CARD>.', 'category_of_the_email': 'Incident'}

```

- As You can see the out put is obtained but it is the dictionary form. So to visualize the out put in a pretiest way and to use the application with front end, you can use User Version.

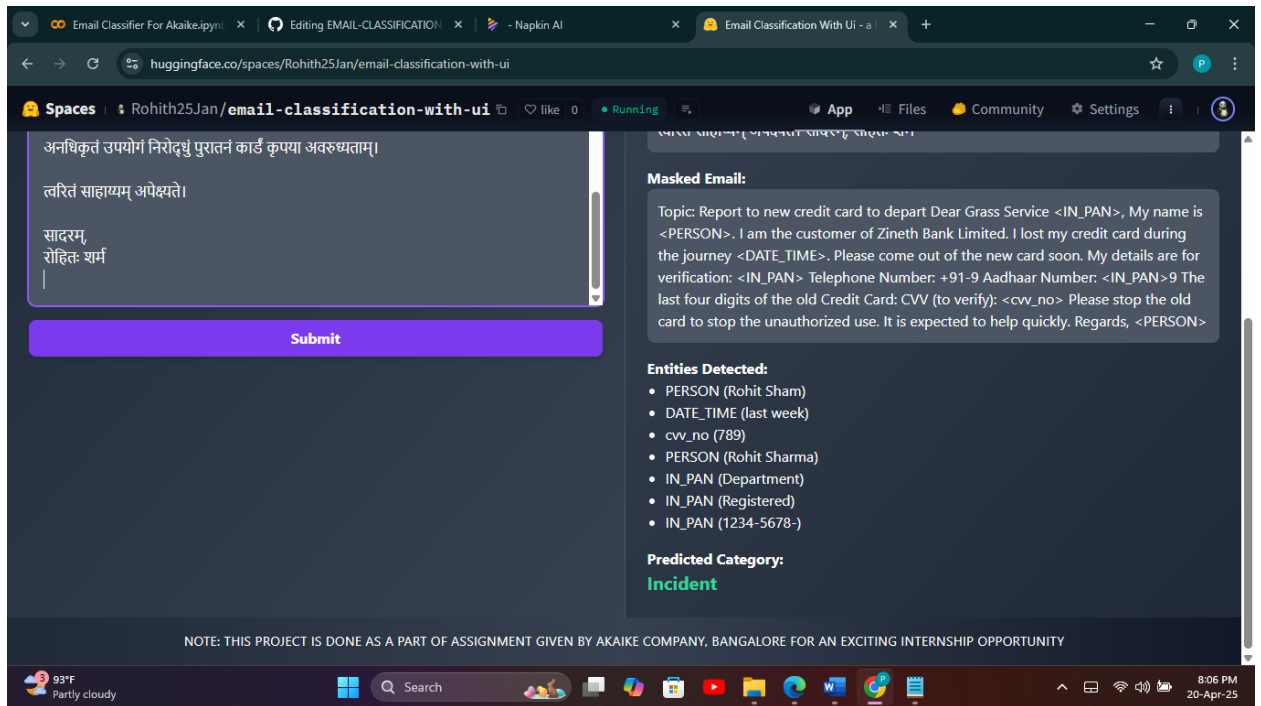
STEP BY STEP GUIDE ON HOW TO USE USER VERSION

- go to the website : <https://huggingface.co/spaces/Rohith25Jan/email-classification-with-ui>
- you will see the user interface like shown in the below figure



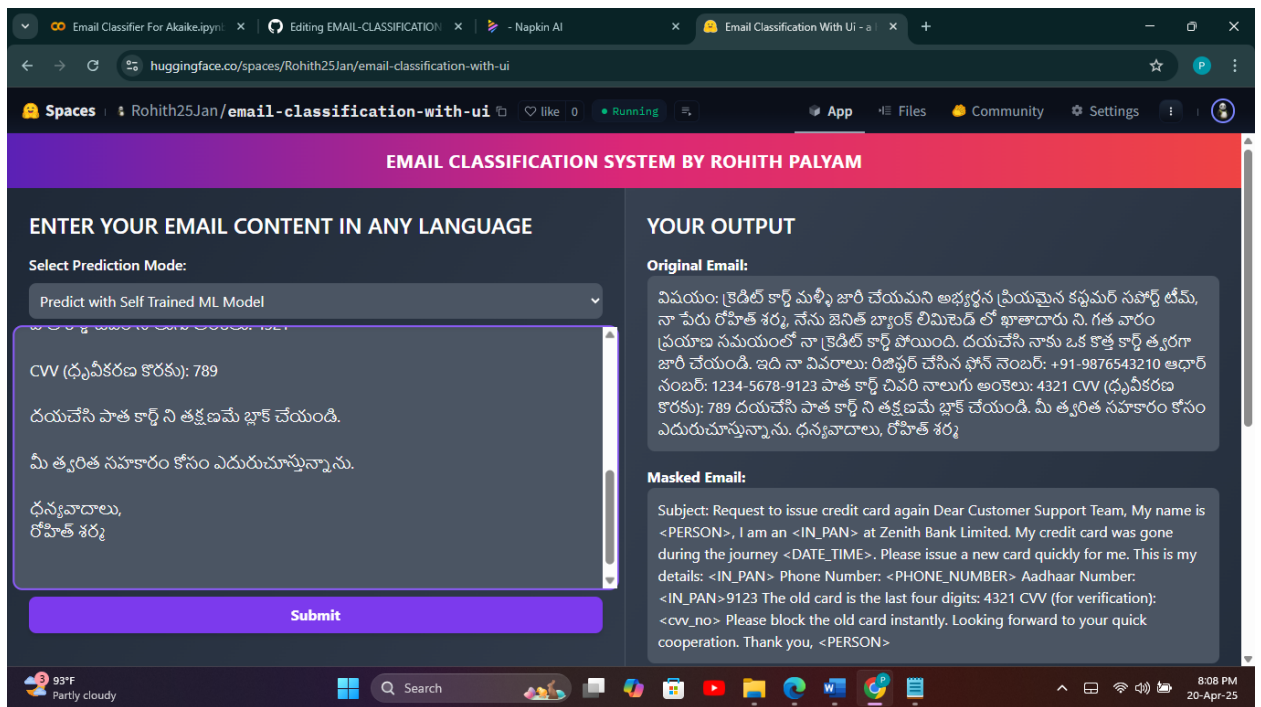
- You can see one drop down, one text area and one submit button.
- So, you can classify the emails in two approaches. one is using self trained ML model trained by me and you can download those models in Models Directory through the link files given. Another approach is using a populer open source llm called 'laama 3.3-70b-versatile' through Groq cloud.
- If you want to classify emails using LLM, then you need an API key which is absolutely free to call model. You can get that API from Groq Cloud which is one the fastest AI Inference.
- So, let's classify the email using self trained ML Model.
- I passed email in the teaxt area specified, and then clicked submit. so output is obtained, which is shown in below figure.

9. I given the email content in sanskrit language then we got the classification which is shown in the below image



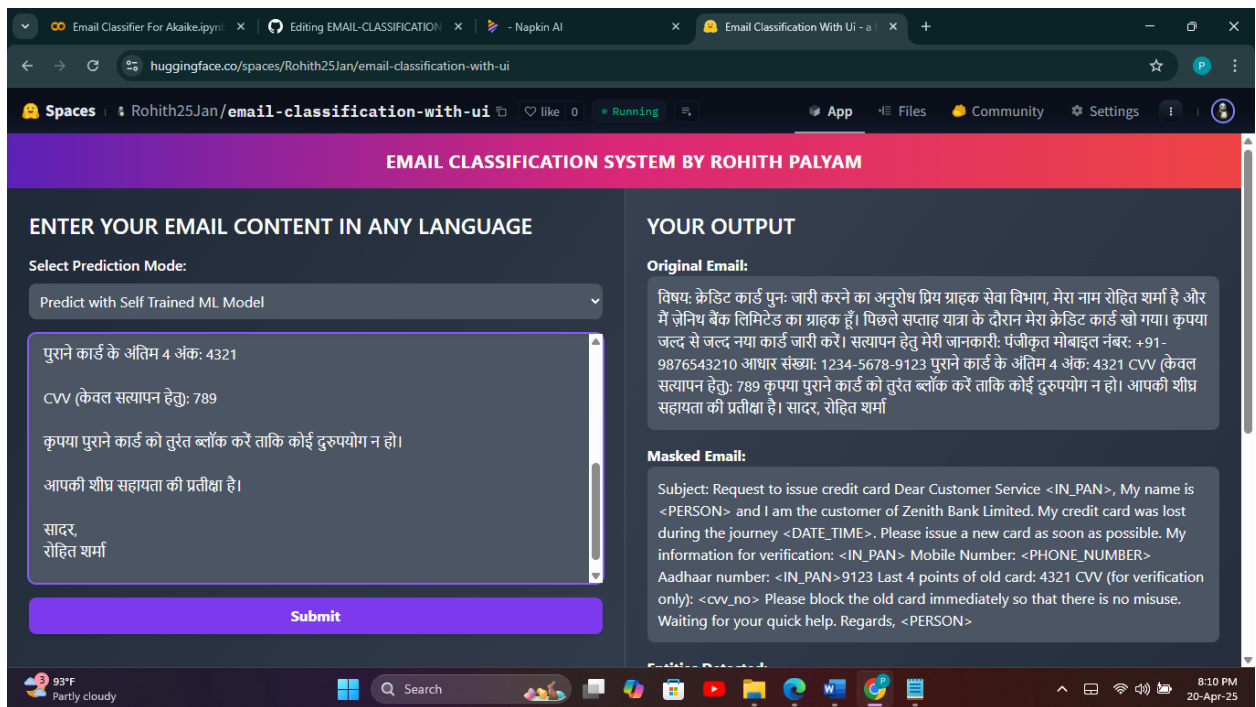
10.

11. This time given content in telugu got the result, shown in the below image



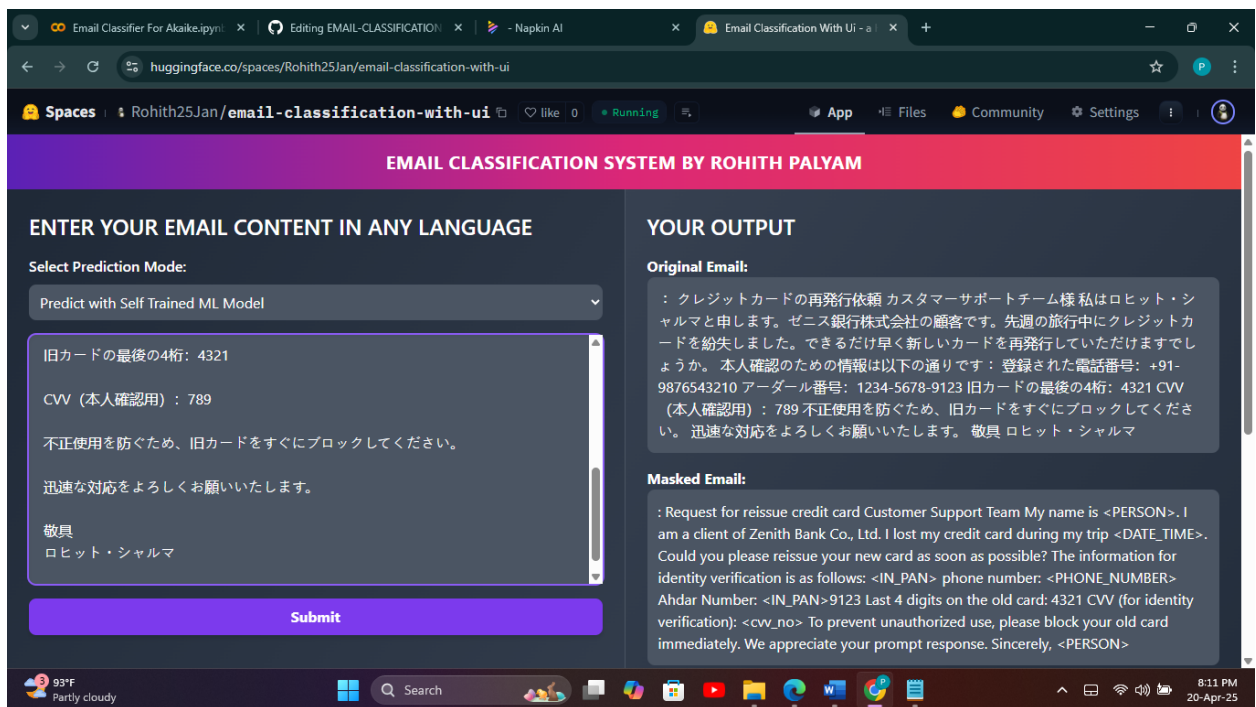
12.

13. This time given content in hindi, got output , shown in the below image



14.

15. This time given content in japanese, got output, which is shown in the below image



16.

17. You can give any language, at backend language translation to english will be done for implementing robust email classification

STEP BY STEP EXPLANATION ON IMPLEMENTATION FOR THIS PROJECT

1. Downloaded the data that is provided through google drive link in pdf.

2. Readed the data set using pandas `pd.read_csv()`
3. Eamined the readed data like dataframes shape, size, columns, dtypes etc
4. Dropped duplicated rows and missing missing values
5. Detected the languages of content present in email using `langdetect` module in python
6. Translated all non english subject content into english subject content using `deep-translator`
7. After, coverting the email content inro english i perofrmed PII masking using `persidio` librery
8. After performing masking, i implemnetd the basic NLP text processing techniques, which is mentioned below
9. Lower casing, Removing HTML Code, Removing Punctuations, Removing Stop words, Encoding Emoji's, Chart word treatment, Tokenization, Stemming & Lemmatization
10. After that performed some Exploratory data analysis on transofrmed data, implemented `value_counts` for type column vislaized thorough bar chart for checking whether it is a balanced data or not, after that implemneted word cloud charts for each category in type column to see which words are dominating in that category.
11. After performing EDA, implemnted TFIDF vectorizer, and implemented more than 20+ classifications algorithms and downloaded vectorizer object and ML Model in pickel format.
12. Also I implemented `CountVectorizer`, then implemnted again 20+ classification algorithms, evaluated ml models performance through metrics like accuracy nd classification report.
13. After careful examination downloaded TFIDF Vectorizer object, Random Forest Classifier Object in .pkl formats.
14. Used FastAPI for POST requests, Rendering templates. Integrated downloaded models in the application.
15. After successfull Testing in Local Environment, Used Hugging Face Spaces And Docker to deploy this project in two variants called Tester Varient & User Varient.

STEP BY STEP EXPLANATION ON 'HOW WE GET OUTPUT WHEN WE GIVEN EMAIL AS INPUT'

1. used FastAPI, created end point at `classify_email` which takes request object that contains `email_body` in json formate.
2. Fetched `email_body` content, passed `email_body` to a function called `classify_email(email_body)`, where that function is present in `api.py` or in some other file.
3. After that, `classify_email()` calls some other functions which performs PII Masking, Cleaning, Classifying etc.
4. The output is given with successful code 200 if success else some other code like 500 etc.