**Project Report: Email Sentiment & Priority Analyzer**

**Submitted by:** K Yaswanth Naga Sai  
**Date:** March 24, 2025  
**Project Repository:** <https://github.com/YASWANTH1976/EmailAnalyzer>

**1. Introduction**

**1.1 Project Overview**

The Email Sentiment & Priority Analyzer is a web application I created to help people manage their emails better. I built this app using Python and a framework called Flask. The app looks at emails and tells you things like whether the email sounds positive or negative (sentiment), how urgent it is (priority), whether it’s formal or casual (tone), and what category it belongs to (like Technical or General). It also checks if an email might be spam, gives a short summary of the email, and suggests a reply. I added a login system so each user can have their own account and see their past email analyses. The app looks modern because I used Tailwind CSS for the design, and users can download their results as CSV, PDF, or text files.

I started this project because I wanted to learn how to build a complete web app, from the backend (where the logic happens) to the frontend (what the user sees). I also wanted to make something useful that could help people save time by focusing on the most important emails first.

**1.2 Objectives**

I had a few goals for this project:

* Create a web app that can analyze emails using natural language processing (NLP).
* Add a login system so users can have their own accounts and see their history.
* Make the app easy to use with a nice, modern design.
* Let users analyze multiple emails at once and save the results in different formats.
* Improve how the app decides if an email is urgent by using better rules.
* Make sure the app works properly and can be shared online later.

**1.3 Scope**

This project includes:

* Analyzing emails to find their sentiment, priority, tone, and categories.
* Checking for spam or phishing emails (like ones that say “win a prize”).
* Summarizing emails and suggesting replies based on the email’s tone.
* Adding a login system so each user has their own account and history.
* Designing a clean and modern interface using Tailwind CSS.
* Letting users export their results as CSV, PDF, or text files.
* Adding features like searching, sorting, and showing stats (like how many emails are high priority).

**2. System Design**

**2.1 How the App Works**

I built the app using Flask, a Python framework that makes it easy to create web applications. The app has two main parts: the frontend and the backend. The frontend is what users see—it’s made of HTML pages that I styled with Tailwind CSS to look modern. The backend is the part that does all the work, like analyzing emails and saving data. I used Flask to connect the frontend and backend.

The app also uses a database called SQLite to store user information (like their username and password) and their email analysis history. When a user does something, like selecting emails to analyze, the frontend sends a request to the backend. The backend processes the request, does the analysis, and sends the results back to the frontend to show on the page.

**2.2 Tools and Technologies**

I used these tools to build the app:

* **Python and Flask**: Flask is the framework I used to build the web app. It helps with things like creating pages and handling user requests.
* **NLTK**: This is a Python library I used to analyze the sentiment of emails. It has a tool called VADER that can tell if an email sounds positive, negative, or neutral.
* **Flask-Login and Flask-SQLAlchemy**: These are Flask tools I used to add the login system and manage the database.
* **ReportLab**: A Python library I used to create PDF files for exporting results.
* **HTML and Tailwind CSS**: HTML makes the structure of the pages, and Tailwind CSS makes them look nice with colors and layouts.
* **SQLite**: A simple database to store user accounts and their email analysis history.
* **Git and GitHub**: I used Git to keep track of my changes and GitHub to store my project online.

**2.3 Main Parts of the App**

* **Login System**: Users need to sign up and log in to use the app. Their username and password are saved in the database.
* **Email Analysis**: The app can analyze emails from a file called emails.txt or from a .txt file the user uploads. It looks at the email’s subject and body to figure out its sentiment, priority, and other details.
* **Database**: I created two tables in the database: one for users (to store their username and password) and one for analysis history (to save each user’s past email analyses).
* **User Interface**: The app has a clean design with a navigation bar at the top, a main section for analyzing emails, and sections for viewing results, exporting data, and seeing past analyses.

**3. How I Built the App**

**3.1 Project Files**

Here’s how I organized my project:

text

CollapseWrapCopy

EmailAnalyzer/

│

├── app.py # The main Python file with all the app logic

├── emails.txt # A file with sample emails to test the app

├── requirements.txt # A list of Python libraries the app needs

├── templates/ # A folder with all the HTML pages

│ ├── base.html # The main layout for all pages

│ ├── index.html # The main page where users analyze emails

│ ├── login.html # The login page

│ └── signup.html # The signup page

├── users.db # The database file (created when the app runs)

├── .gitignore # A file to tell Git which files to ignore

└── README.md # A file explaining how to use the app

**3.2 Key Features I Added**

1. **Login System**:
   * I used Flask-Login to create a login system. When a user signs up, their username and password are saved in the database. When they log in, the app checks if the username and password match what’s in the database.
   * If the login is successful, the user is taken to the main page. If not, they see an error message saying “Invalid username or password.”
2. **Analyzing Emails**:
   * I wrote a function called analyze\_email in app.py that looks at an email’s subject and body. It does the following:
     + **Sentiment**: Uses NLTK’s VADER to decide if the email is positive, negative, or neutral based on the words used.
     + **Priority**: Checks for certain words to decide how urgent the email is. For example, if the email has words like “urgent” or “asap,” it’s marked as high priority. If it has words like “no rush,” it’s low priority.
     + **Tone**: If the email has the word “dear,” I marked it as formal; otherwise, it’s casual.
     + **Categories**: If the email mentions “error,” I labeled it as Technical; otherwise, it’s General.
     + **Spam/Phishing**: If the email says “win a prize,” I flagged it as potential spam.
     + **Summary**: I took the first 50 characters of the email body to create a short summary.
     + **Reply Suggestion**: If the email is positive, the app suggests “Thank you for your email.” If it’s negative, it suggests “I’m sorry to hear about your issue.”
3. **Analyzing Multiple Emails at Once**:
   * I added a feature where users can select more than one email to analyze at the same time. On the main page, there’s a dropdown menu where users can hold Ctrl and click multiple emails. When they click “Analyze,” the app processes all the selected emails together.
4. **Saving Analysis History**:
   * Every time a user analyzes an email, the app saves the results in the database under their account. On the main page, there’s a section called “Analysis History” that shows all the emails they’ve analyzed, along with the date and time.
5. **Modern Design with Tailwind CSS**:
   * I wanted the app to look professional, so I used Tailwind CSS. I added a navigation bar at the top with a logout button, and I made the analysis results table look nice with colors. For example, high-priority emails are shown in red, medium in yellow, and low in green. I also added a section for analytics that uses cards to show stats like how many emails are positive or high priority.
6. **Exporting Results**:
   * Users can download their analysis results in three formats: CSV (for spreadsheets), PDF (for printing), or text (for simple notes). I used ReportLab to create the PDF files, and I wrote code to format the data properly for each type.
7. **Search, Sort, and Analytics**:
   * I added a search bar where users can type a keyword (like “urgent”) to find emails in their results. They can also sort the results by priority, sentiment, or tone.
   * The analytics section shows stats like the total number of emails analyzed, how many are positive/negative/neutral, and how many are high/medium/low priority.

**3.3 How I Developed the App**

I worked on the project in stages:

* **Stage 1: Setting Up the Basics**:
  + I started by setting up Flask and creating the login system. I also added the basic email analysis feature using NLTK.
* **Stage 2: Adding More Features**:
  + I added the ability to analyze multiple emails, improved the priority detection by adding more keywords, and saved the analysis history for each user.
  + I also added the export feature and the search/sort options.
* **Stage 3: Improving the Design**:
  + I redesigned the app using Tailwind CSS to make it look better. I added colors, a navigation bar, and a better layout for the analytics section.
* **Stage 4: Testing and Fixing Issues**:
  + I tested the app on my computer and fixed problems, like when I couldn’t log in because the database was empty. I also made sure all the features worked correctly.

**4. Problems I Faced**

1. **Login Not Working**:
   * At first, I kept getting an “Invalid username or password” error when trying to log in. I realized the database didn’t have my user because I had deleted it while testing.
   * To fix this, I checked the database using a tool called SQLite and saw that there were no users. I deleted the database file (users.db) and restarted the app, which created a new database. Then I signed up again, and the login worked.
2. **Selecting Multiple Emails**:
   * I wanted users to be able to analyze more than one email at a time, but the dropdown menu only let me select one email. I learned that I needed to add the multiple attribute to the dropdown in the HTML, and then I changed the backend code to handle a list of selected emails.
3. **Making the App Look Good**:
   * The app didn’t look very nice at first—it was just plain HTML with no styling. I decided to use Tailwind CSS because it’s easy to use and makes the app look modern. It took some time to learn how to use Tailwind, but I was able to add colors, buttons, and a better layout.
4. **Saving History for Each User**:
   * I wanted each user to see their own analysis history, but at first, the history was the same for everyone. I fixed this by adding a new table in the database to store the history and linking it to the user’s ID.
5. **Deciding Email Priority**:
   * The app wasn’t very good at figuring out if an email was urgent or not. I improved this by adding more words to check for, like “asap” for high priority and “no rush” for low priority. I also made sure that if an email has a low-priority word, it stays low even if it has a high-priority word.

**5. Testing and Results**

**5.1 How I Tested the App**

I tested the app on my Windows laptop using Python. Here’s what I checked:

* **Login and Signup**: I made sure I could sign up with a new username and password, log in, and log out without any errors.
* **Email Analysis**: I tested the app with the sample emails in emails.txt and also uploaded my own .txt files to see if the analysis was correct.
* **Multiple Emails**: I selected two emails at once (like “Urgent: Server Down” and “Important: Meeting Tomorrow”) and checked if the app analyzed both correctly.
* **Priority**: I made sure the app marked emails correctly—for example, “Urgent: Server Down” was high priority (in red), and “No Rush: Update Report” was low priority (in green).
* **History**: I analyzed a few emails and checked if they showed up in the “Analysis History” section with the right details.
* **Design**: I opened the app in my browser to make sure the design looked good, with all the colors and layouts working properly.
* **Export**: I downloaded the results as CSV, PDF, and text to make sure they were formatted correctly.
* **Search and Sort**: I searched for the word “urgent” and sorted the results by priority to see if it worked.

**5.2 What I Found**

* The app works well for analyzing emails. It correctly identifies the sentiment, priority, and other details for each email.
* The login system keeps the app secure, and each user can see their own history.
* The new design with Tailwind CSS makes the app look much better, and the colors help users see important information quickly (like red for high-priority emails).
* All the features I added—like analyzing multiple emails, saving history, and exporting results—work without any errors.
* I’m ready to share the app online by deploying it, and I also made a video to show how it works.

**6. What I Can Improve**

1. **Making Passwords Safer**:
   * Right now, the app saves passwords as plain text, which isn’t safe. I can use a Python library to hash the passwords so they’re more secure.
2. **Making the App Faster**:
   * When I analyze emails, the page reloads every time. I can learn how to use something called AJAX to update the page without reloading it, which would make the app feel faster.
3. **Splitting the App into Two Parts**:
   * I can separate the frontend and backend by using a tool like React for the frontend and making the backend a separate API. This would make the app more modern and easier to work on.
4. **Better Email Analysis**:
   * I can use machine learning to make the app better at categorizing emails or spotting spam. For example, I could train a model to recognize more types of emails, like meeting requests or feedback.
5. **Adding Real-Time Features**:
   * If I analyze a lot of emails at once, I can add a progress bar to show how much is done. I’d need to learn how to use WebSockets for this.
6. **Sharing the App Online**:
   * I haven’t put the app online yet, but I can deploy it on a website like Replit so other people can try it.

**7. Conclusion**

I’m really happy with how the Email Sentiment & Priority Analyzer turned out. The app does what I wanted it to do—it analyzes emails, helps users decide which ones are important, and saves their history so they can look back at past analyses. I learned a lot while working on this project, like how to use Flask to build a web app, how to analyze text with NLTK, and how to make a nice-looking interface with Tailwind CSS. I also learned how to fix problems, like when I couldn’t log in because the database was empty.

Even though the app works well, there are still things I can improve, like making passwords safer and adding more advanced features. This project has taught me a lot about building web applications, and I’m excited to keep working on it and learning more.

**8. References**

* I used the Flask website to learn how to build the app: <https://flask.palletsprojects.com/>
* I learned about NLTK from its website: <https://www.nltk.org/>
* I used Tailwind CSS for the design and learned from their site: <https://tailwindcss.com/>
* I got help with the database from the Flask-SQLAlchemy site: <https://flask-sqlalchemy.palletsprojects.com/>
* I learned about user login from the Flask-Login site: <https://flask-login.readthedocs.io/>
* I used ReportLab for PDFs and learned from their site: <https://www.reportlab.com/>
* My project is on GitHub: <https://github.com/YASWANTH1976/EmailAnalyzer>