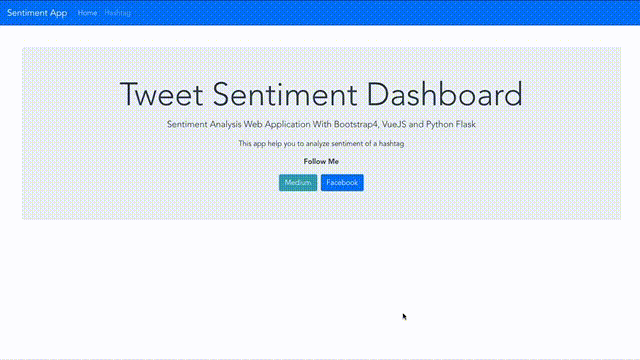
[Date]

|  |  |
| --- | --- |
| Student Name |  |
| Student Id |  |
| Subject |  |

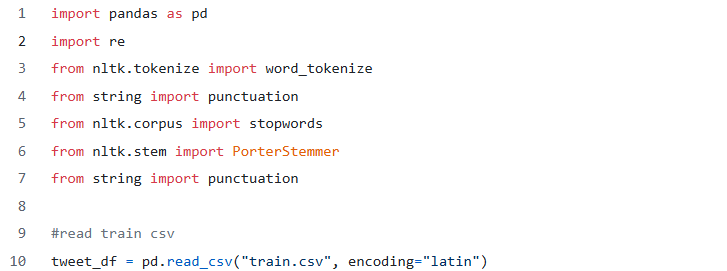
**Introduction:**

Flask and Vue.js are popular among developers globally. Flask is a well-known Python framework, while Vue.js is a prominent JavaScript framework. When designing an application, there are often two key components: the backend and the frontend. For this project, I used Flask to create the backend application and Vue.js to build the frontend application.



This program collects tweets using specified hashtags, analyzes their sentiment, and displays the findings on an interactive dashboard. The idea is to give a seamless way to view the overall sentiment of tweets on a certain topic or event in real time.  
  
To accomplish this, the development method included three major steps:  
  
Sentiment Analysis Model: The initial stage was to create a comprehensive sentiment analysis model that could identify tweets as positive, negative, or neutral. This included data preparation, training the model on labeled datasets, and fine-tuning its accuracy.  
The backend application was built with the Flask framework, a lightweight and adaptable Python tool. Flask handled the fundamental functionality, such as retrieving tweets from the Twitter API, sending them to the sentiment analysis model, and storing the processed data for display.

Frontend Development: The frontend application was built with Vue.js, a popular JavaScript framework. Vue.js created a simple and dynamic interface for users to examine sentiment analysis findings. The dashboard shows real-time updates, making it simple to analyze the information.  
  
The combination of Flask with Vue.js results in a responsive and efficient application that bridges data processing and user interaction.

**Building a Sentiment Analysis Model.**  
  
The sentiment analysis model was built with Jupyter Notebook, a powerful tool for data preparation and model construction. Naive Bayes was used as the classification method since it is simple and efficient at categorizing tweet emotions.  
  
The training dataset, train.csv, can be downloaded from Kaggle. To prepare the data, a notebook called Preparing.ipynb was generated. This notebook handles operations like loading the necessary Python modules and retrieving the training data from a CSV file. The pretreatment processes guarantee that the data is clean and ready to train the model.  
  
This methodology efficiently integrates simple tools and methodologies to create a sentiment analysis model capable of identifying tweets as good, negative, or neutral.  
  
  
  
  
**Data Preparation and Cleaning**  
Pandas was used to maintain the dataset in DataFrame format, while preprocessing and cleaning the twitter data was done with libraries such as re and NLTK. Regular expressions (re) helped eliminate unnecessary characters, URLs, and special symbols, while NLTK aided with tokenization and stopword removal.  
  
To confirm that the dataset was loaded correctly, the tweet\_df.head() method was regularly used to preview the DataFrame's initial few rows. This step ensured that the data was properly imported and prepared for future preparation and analysis.



The itemID variable was deemed superfluous while training the sentiment analysis model, thus just the Sentiment and SentimentText columns were employed. The Sentiment column is the target variable, with a value of 0 indicating a negative sentiment and a value of 1 representing a good attitude.  
  
The data was then cleaned and preprocessed to achieve the best model performance. Regular expressions were used to remove extraneous letters, punctuation, and URLs. To standardize the language, NLTK was used for tokenization, stopword removal, and lowercase formatting. These preprocessing methods reduced noise in the dataset, allowing the model to focus on important patterns in the text data.

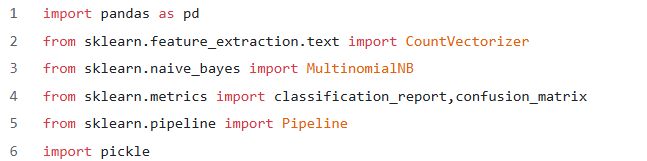


Lines 7 to 11 of the preProcessTweet function are used to lowercase the tweet, remove punctuation from negation, remove the web URL, remove the user mention, and finally remove the hashtag. The next step is to stem each word in a tweet, add NOT\_ as described above, and then remove stop words.   
  
Use this syntax to apply the preProcessTweets function to our dataframe. Use the head() function to examine the new column (clean\_text), which contains preprocessed tweets.



The above syntax will run for a time based on our hardware specs, and then save it to a new CSV file, eliminating the need to call the preprocess function again.

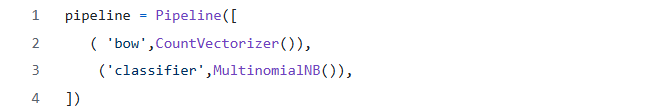
**Train the Classification Model**  
  
The Naive Bayes approach was used to train the sentiment classification model, which was then implemented in Scikit-Learn. This method was chosen for its simplicity and efficacy in text categorization tasks.  
  
A second notebook, modeling.ipynb, was established just for training the model. The first step in this notebook was to import all of the required libraries and dependencies. This resulted in a more ordered workflow and expedited the model-building and evaluation process.  
  
Using Scikit-Learn, the cleaned and preprocessed data was divided into training and testing sets to assess the model's performance. The Naive Bayes classifier was then trained on the training data, using the Sentiment and SentimentText columns to reliably predict tweet sentiments.



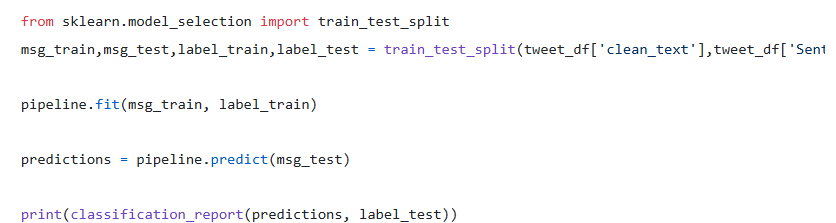
Convert the object to String/Unicode.

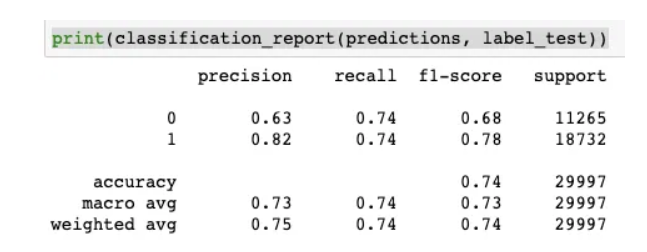


Then we created a pipeline to automate the training process. This pipeline is used to generate a bag-of-words model and then train the classifier.

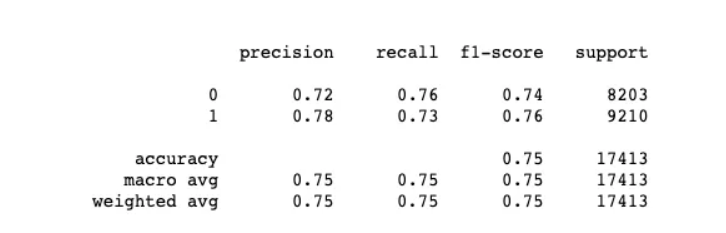


Split the data into train and test sets, using 70:30 proportions, and then train the data using the previously created pipeline. Print the classification report to determine our model's accuracy, recall, and precision.

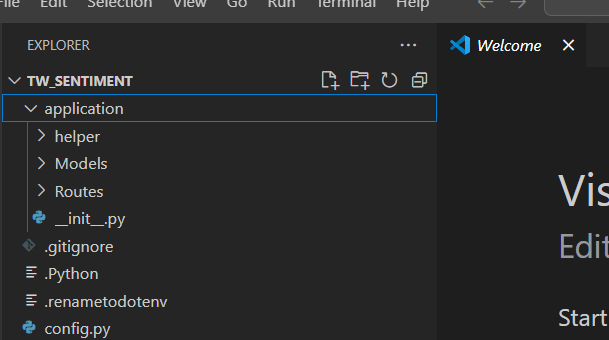




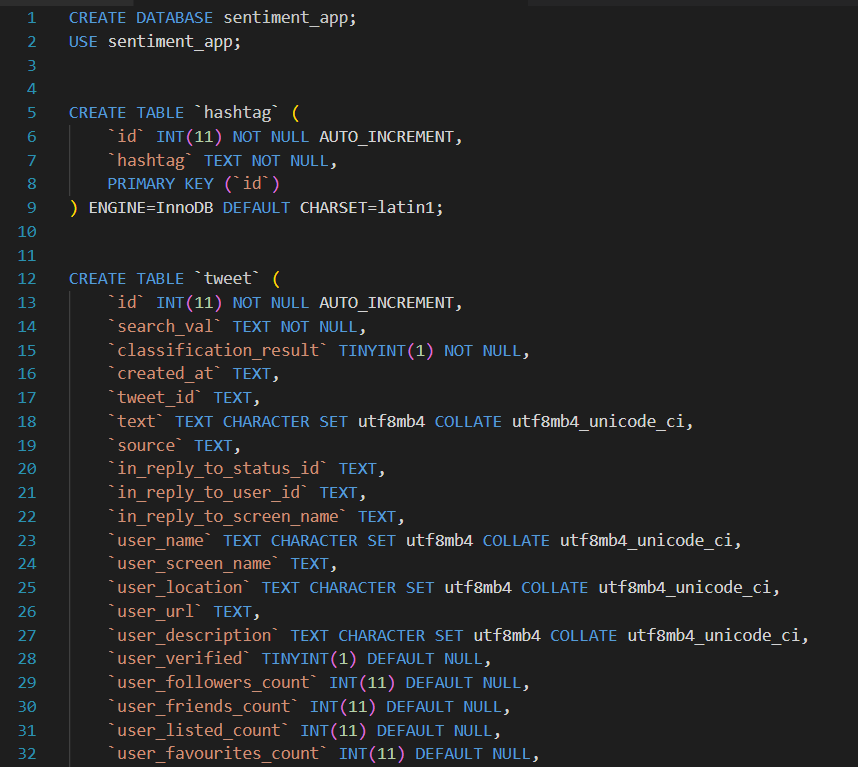
We try to optimize the model. Looking deeper into train.csv, We saw that good tweets outnumber negative tweets by a large margin. In some positive sentiment data to equalize the two moods (50:50).   
  
The classification report comes after that.

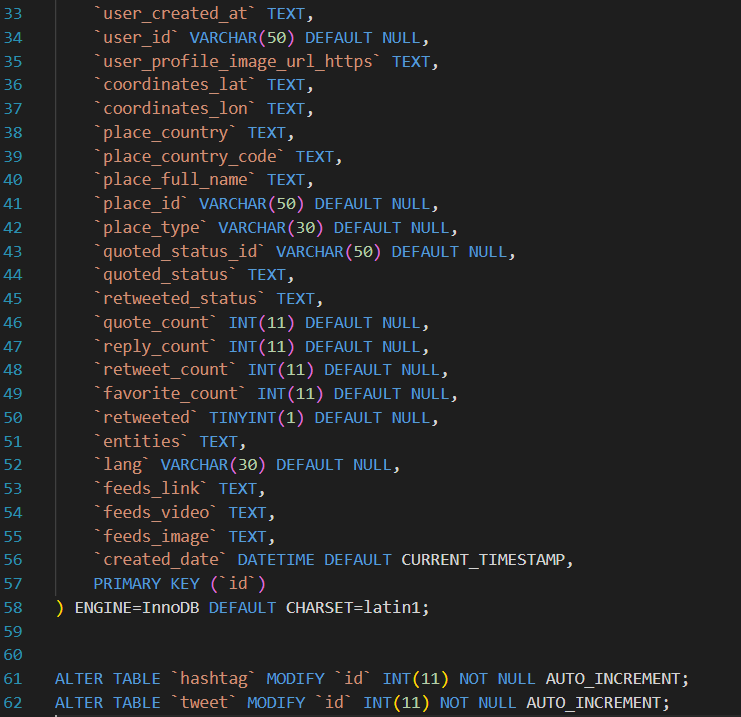


Building the backend application with Flask.  
  
Following the development of the classification model, the next stage was to design the backend program, which would manage the server-side activities of the website. The backend's key functions are database management, tweet crawling, and sentiment categorization processing.  
  
The backend was constructed as a RESTful server with Flask to give APIs and services to the frontend. As a consequence, the application's output is in JSON format, with no visual components, allowing for easy data transfer between the backend and frontend.  
  
This project was developed in a virtual environment in accordance with Python's application development best practices. Using a virtual environment allows you to isolate dependencies and avoid problems with other projects on the same machine.  
  
Although this was created by someone with a PHP experience rather than Python, Flask's simplicity and versatility made it a perfect choice for developing the backend application.

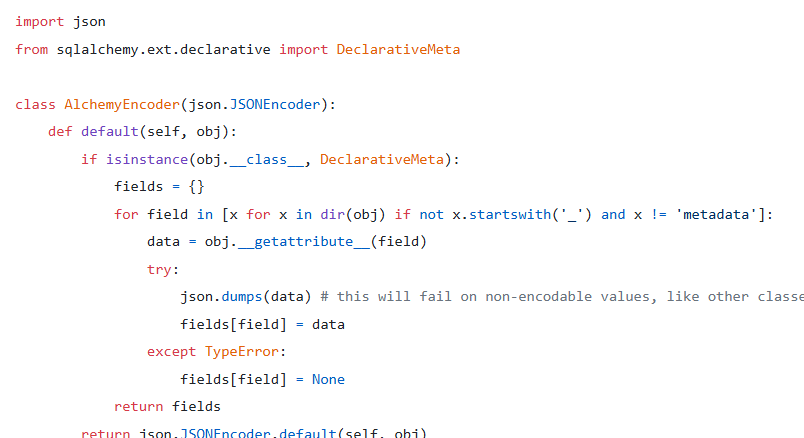
  
  
  
  
**Backend Folder Structure and Package Installation**  
  
The backend application is organized into a hierarchical folder structure. The main program folder includes the following subfolders and files:  
  
Helper: This class contains utility methods that may be used to help with activities such as data preparation or API queries.  
Models: Contains database models for managing and organizing data structures.  
Routes: Contains route definitions for API endpoints.  
\_\_init\_\_.py: Sets up the Flask application and combines the necessary components.  
When we build up the Python virtual environment, other directories like bin, include, and lib are also created automatically.  
  
To configure the backend, numerous Python packages were installed using pip to guarantee that all necessary features were available. The essential packages contain the following:  
Flask is a lightweight web framework for constructing backend applications.   
Flask-CORS: Manages Cross-Origin Resource Sharing (CORS) to provide safe communication between backend and frontend.   
Flask-SQLAlchemy: Implements Object-Relational Mapping (ORM) for database interactions.   
nltk: Enables text preparation and word processing.   
PyMySQL: Serves as a MySQL connection for Python.   
requests: simplifies HTTP request processing.   
tweepy: Provides an interface to the Twitter API.

**SQL script for the backend**  
  
The database management system used in the backend application is MySQL (particularly MariaDB). The database is called sentiment\_app and acts as a central repository for tweets, sentiment analysis results, and other important information.  
  
The SQL script required to set up the database is provided below:



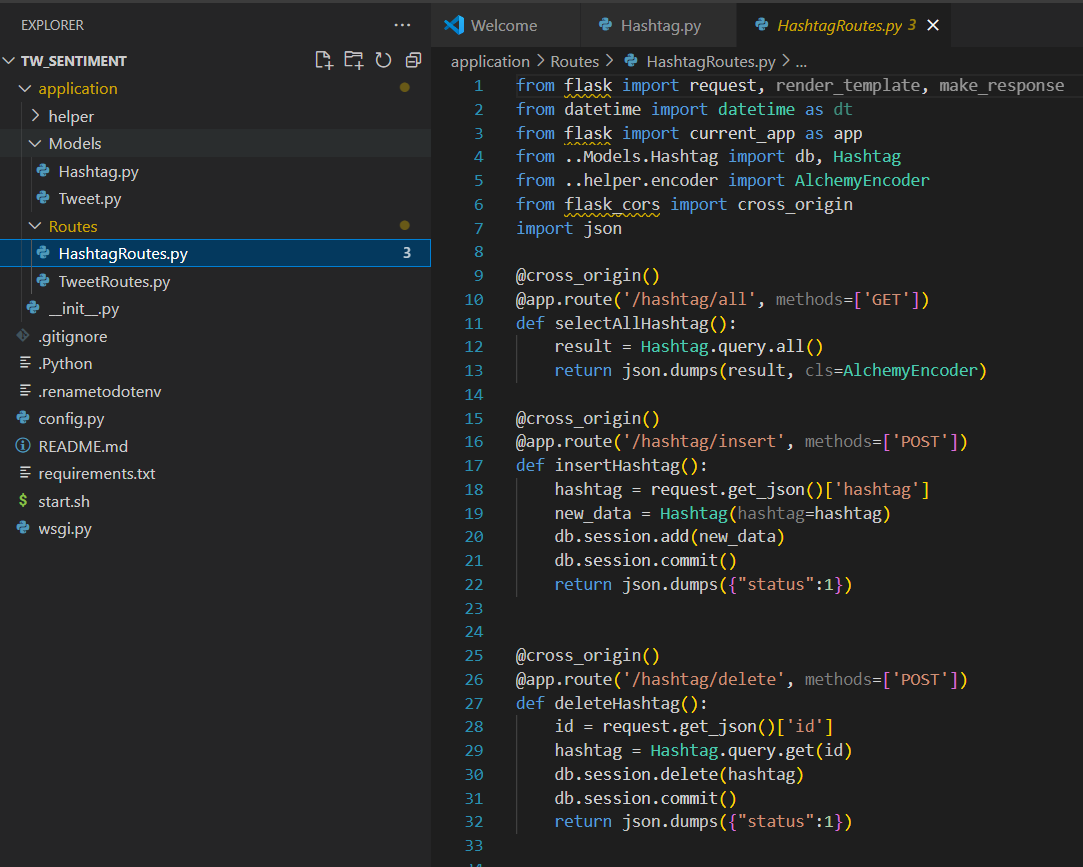


encoder.py is a custom encoder that dumps data collected from SQL Alchemy into JSON format.



The sentiment analysis application's backend contains two primary route files, each of which is responsible for processing unique logic linked to incoming requests. Here's an overview of their functions:  
  
In TweetRoutes.py, the crawlTweet() method (line 69) collects tweet data using the hashtag input from the frontend. This method uses TwitterClassifier.py (found in the helper/folder) to do sentiment categorization on each tweet. The tweets are then recorded in the database.  
  
Functionality: Accepts hashtags from the frontend.  
Crawls up to 300 tweets.  
Performs sentiment categorization on each tweet.  
Saves categorized tweet data to a database.  
Helper/TwitterClassifier.py:

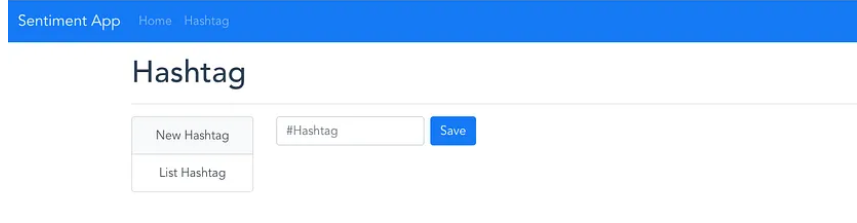
This file contains the categorization logic. The categorization is performed by TwitterClassifier.py on lines 160-162. It employs a trained machine learning model to determine the mood of each tweet.   
  
The TW-sentiment project's folder structure organizes its backend components. For better visualization, please see the completed folder structure below:



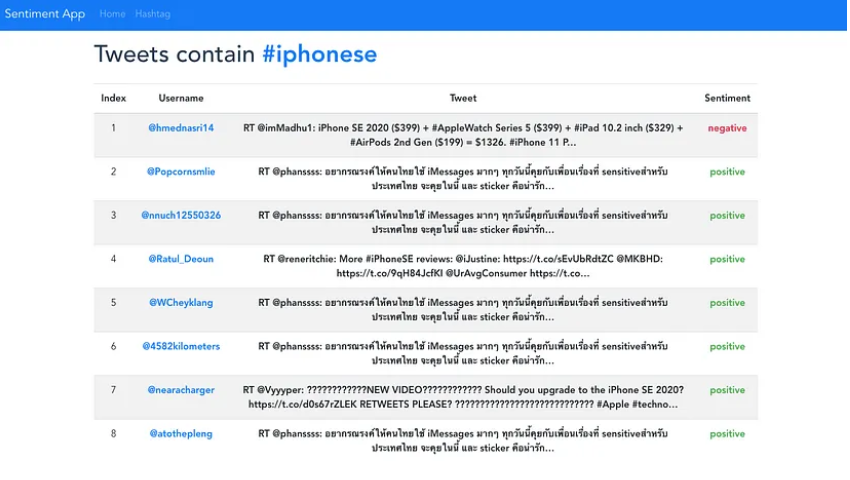
The following is a list of URLs used in the backend to provide services to the frontend. These APIs allow the frontend to communicate with the server, retrieve tweet data, do sentiment analysis, and manage hashtags.  
  
GET: /tweet/all?hashtag={hashtag}&page={page}  
  
Retrieves a paginated collection of tweets with the supplied hashtag.  
Query parameters include hashtag (the hashtag to search for) and page (pagination page number).  
GET: /tweet/count?hashtag={hashtag}  
  
Retrieves the total number of tweets for the specified hashtag.  
Query parameter: hashtag (the hashtag to count tweets with).  
GET: /tweet/detail?id={id}  
  
Retrieves comprehensive information on a given tweet using its id.  
Query parameter: id (the tweet's unique identification).  
GET: /tweet/sentiment?hashtag={hashtag}  
  
Provides sentiment analysis findings for tweets using the provided hashtag.

Query parameter: hashtag (to analyze). GET /tweet/toptweet?hashtag={hashtag}.   
  
Retrieves the most popular tweet using the specified hashtag.   
Query parameter: hashtag (the hashtag to look for in the top tweets).   
GET /tweet/daytoday?tag={hashtag}   
  
Returns daily statistics (such as sentiment trends or tweet counts) for the specified hashtag.   
Query Parameter: hashtag (the hashtag for which to track daily trends).   
POST, tweet, or crawl using # {hashtag}.   
  
Starts the crawling operation to collect tweets based on the supplied hashtag.   
Query parameter: hashtag (the hashtag to search for in tweets).   
Method: POST, which initiates an activity (crawling tweets).   
GET /hashtag/all.   
  
Returns a list of all stored hashtags in the system.   
POST /hashtag/Insert   
  
Introduces a new hashtag to the system.   
Method: POST, which conducts an insertion action.   
Post /hashtag/delete.   
Deletes the hashtag from the system.   
Method: POST, which conducts a delete operation.

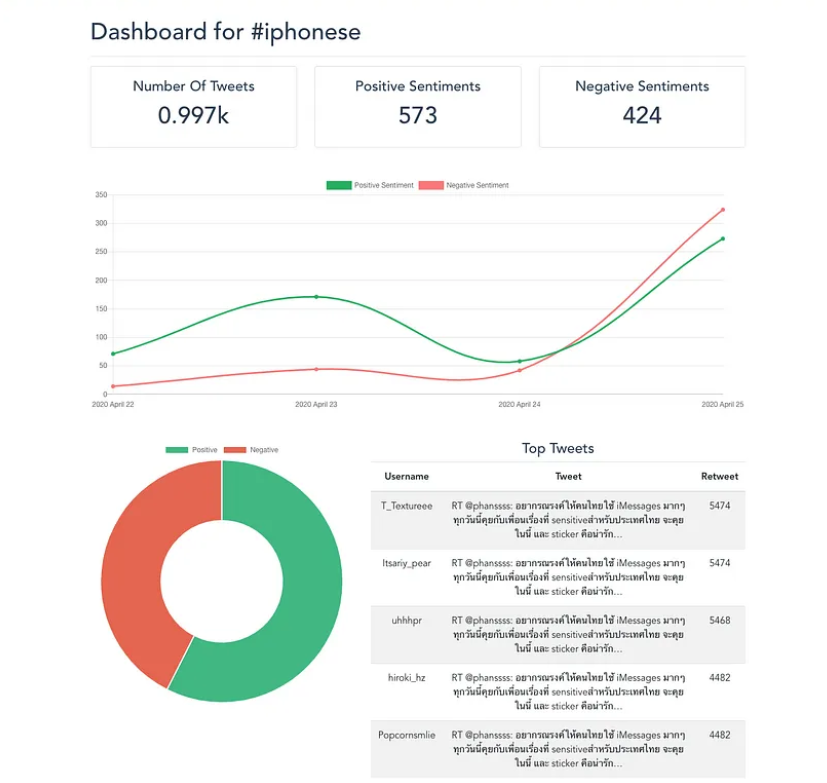
The default appearance of the Hashtag section shows a list of hashtags, with five action buttons for each one.  
  
Fetch Data Button: This button starts the process of obtaining tweets related to the specified hashtag. When selected, the program searches the last 300 tweets for the chosen hashtag. The maximum amount of tweets may be set in the backend application (TW-sentiment), and a streaming function can be added to allow for real-time tweet retrieval.  
  
The New Hashtag and Edit menus use the same component to create and alter hashtags.



View Tweets. The button is used to display tweets with a hashtag.



Many of the tweets retrieved may be in languages other than English. To fix this, consider improving the model or changing the crawler setup to better accommodate non-English tweets.  
  
The Delete Button removes a hashtag, while the Dashboard Button takes you to the Dashboard View for further insights and analysis.



The dashboard shows crucial parameters for a hashtag, such as total tweets, good feelings, and negative sentiments. It includes a line chart to show daily sentiment movements and a pie chart to emphasize sentiment proportions. Furthermore, the dashboard displays the most retweeted tweet for the specified hashtag.

**Results and Discussion**

Using the Naive Bayes technique, the sentiment analysis model correctly identified tweets as positive or negative. The training procedure proved that Naive Bayes is an effective and lightweight strategy for this classification issue, producing consistent results while requiring low CPU resources. The preprocessing techniques, such as cleaning and tokenization, improved the model's accuracy by reducing noise and standardizing the input data.

In the backend application, the Flask framework allowed for quick API request handling and sentiment model integration. The crawlTweet() function successfully extracted up to 300 tweets every request, categorizing and storing them in a database. This restriction guarantees that data is processed in a reasonable manner while yet allowing for adjustments to the threshold or real-time streaming for dynamic tweet analysis.

The dashboard offered a user-friendly interface for visualizing outcomes. Charts and graphs were used efficiently to communicate key indicators such as total tweet count, positive and negative sentiment proportions, and daily sentiment trends. The inclusion of the most retweeted tweet provides more information for hashtag performance and user interaction. However, it was discovered that many tweets were written in languages other than English, which may have an impact on model accuracy. Optimizing the model or structuring the crawling process to accommodate multilingual data might increase classification results.  
  
Overall, the integration of the sentiment analysis model, backend, and dashboard creates a streamlined sentiment monitoring pipeline. The findings highlight the model's usefulness and scalability for social media analysis. Future development might include increasing language support, upgrading categorization algorithms, and integrating real-time streaming for live sentiment tracking. This method can help companies and scholars properly assess public opinion patterns on social media platforms.

**Conclusion:**

The sentiment analysis project effectively combines machine learning, backend programming, and frontend visualization to produce a scalable and efficient system for evaluating Twitter data. Using the Naive Bayes classification technique, the model exhibited its capacity to efficiently categorize tweets into positive and negative attitudes, despite very simple preprocessing procedures. The Naive Bayes algorithm's lightweight and efficient nature makes it ideal for this application, especially when dealing with massive amounts of text data.  
  
The backend application, built using Flask, acts as the system's backbone, managing database operations, tweet crawling, and categorization. It has a solid API structure that ensures smooth communication between the model and the frontend application. The integration of critical libraries such as Flask-CORS, Flask-SQLAlchemy, and Tweepy facilitated the easy deployment of backend capabilities. The crawlTweet() method, which can retrieve up to 300 tweets per request, demonstrates the system's scalability while still retaining speed. This limit is customizable, allowing for use cases that need real-time data streaming or bigger datasets.

The frontend dashboard provides a simple interface for visualizing sentiment trends and crucial data. It displays data such as the overall number of tweets, the distribution of positive and negative feelings, and sentiment changes over time using line and pie charts. Furthermore, including the most retweeted tweet provides useful context by showing popular material inside a hashtag. This graphic allows users to efficiently monitor public mood and detect patterns in social media interaction.  
  
One of the most prominent issues encountered throughout the experiment was the presence of non-English tweets, which may have influenced categorization accuracy. Addressing this constraint by improving the model's multilingual capabilities or fine-tuning the crawler configuration to focus on certain languages will considerably boost the system's performance and flexibility.

**References:**

Mathur, D., 2022 Step-by-step guide to building a sentiment analysis app with Django. [online] Medium. Available at: <https://medium.com/@mathur.danduprolu/step-by-step-guide-to-building-a-sentiment-analysis-app-with-django-58f79c69de7a> [Accessed 10 January 2025].

Analytics Vidhya, 2024 Build sentiment analysis application with Flask and Vue.js. [online] Medium. Available at: <https://medium.com/analytics-vidhya/build-sentiment-analysis-application-with-flask-and-vuejs-b607dc1f3604> [Accessed 10 January 2025].

GeeksforGeeks, 2023 Create a simple sentiment analysis web app using Streamlit. [online] Available at: <https://www.geeksforgeeks.org/create-a-simple-sentiment-analysis-webapp-using-streamlit/> [Accessed 10 January 2025].

Project Worlds, 2022 Sentiment analysis ML Flask Python web app project with source code. [online] Available at: <https://projectworlds.in/sentiment-analysis-ml-flask-python-web-app-project-with-source-code/> [Accessed 10 January 2025].

Real Python, 2021 Python NLTK sentiment analysis. [online] Available at: <https://realpython.com/python-nltk-sentiment-analysis/> [Accessed 10 January 2025].

Dataheadhunters, 2020 How to implement Python in e-commerce website analytics. [online] Available at: <https://dataheadhunters.com/academy/how-to-implement-python-in-e-commerce-website-analytics/> [Accessed 10 January 2025].

DataCamp, 2018 Text analytics for beginners using NLTK. [online] Available at: <https://www.datacamp.com/tutorial/text-analytics-beginners-nltk> [Accessed 10 January 2025].

Javatpoint, 2019 Sentiment detector GUI using Tkinter in Python. [online] Available at: <https://www.javatpoint.com/sentiment-detector-gui-using-tkinter-in-python> [Accessed 10 January 2025].

Kirenz, 2021 Text mining and sentiment analysis with NLTK and Pandas in Python. [online] Available at: <https://www.kirenz.com/blog/posts/2021-12-11-text-mining-and-sentiment-analysis-with-nltk-and-pandas-in-python/> [Accessed 10 January 2025].

Analytics Vidhya, 2022. Sentiment analysis using Python. [online] Available at: <https://www.analyticsvidhya.com/blog/2022/07/sentiment-analysis-using-python/> [Accessed 10 January 2025].