Sentiment Analysis Project Documentation

1. Project Overview

This project implements a sentiment analysis system with a Flask-based REST API backend and a React frontend. The core sentiment analysis is performed using a model hosted on Google Colab, while the local system handles user interactions, data storage, and result presentation.

2. Functional Requirements

Here's the revised content with all hash symbols (#) and asterisks (*) removed:

2.1 REST API
- Requirement: Design and deploy a REST API that receives text input and returns sentiment analysis results.
- Implementation: A Flask-based API (senti_ui_api.py) was created to handle text input, communicate with the Google Colab model, and return results.

2.2 Sentiment Analysis
- Requirement: Analyze input text using specified machine learning models.
- Implementation: The system uses a TensorFlow model hosted on Google Colab for sentiment analysis, focusing on education and emotion scores.

2.3 Data Logging
- Requirement: Log received text and returned results to a database.
- Implementation: SQLite database (sentiment_analysis.db) is used to store input text and analysis results.

2.4 Web UI
- Requirement: Display analysis results and history in a web interface.
- Implementation: A React-based frontend (sentiment-analysis-ui) provides an intuitive interface for text input, result display, and history viewing.

2.5 Containerization
- Requirement: Containerize the REST API and web UI for deployment.
- Implementation: Docker containers for both the Flask API and React UI are created using Dockerfiles and orchestrated with docker-compose.yml.

3. Non-Functional Requirements

3.1 Performance
- Requirement: The system should provide timely responses to user requests.
- Implementation:
  - Asynchronous communication with Google Colab minimizes wait times.
  - Efficient database queries ensure quick retrieval of historical data.

3.2 Scalability

- Requirement: The system should be able to handle increased load.
- Implementation:
  - Docker containerization allows for easy scaling of the API and UI components.
  - Separation of concerns (API, UI, ML model) allows for independent scaling of each component.

3.3 Maintainability

- Requirement: The codebase should be easy to understand and modify.
- Implementation:
  - Clear separation of backend (senti_api.py, senti_ui_api.py) and frontend (React components) logic.
  - Use of popular frameworks (Flask, React) ensures long-term maintainability.
  - Docker configuration files (Dockerfile, docker-compose.yml) are provided for consistent environments.

3.4 Security

- Requirement: The system should handle data securely.
- Implementation:
  - CORS configuration in Flask API to control access.
  - Environment variables used for sensitive configuration (e.g., Google Colab URL).
  - SQLite database is not exposed directly to the internet.

3.5 Usability

- Requirement: The system should be user-friendly and intuitive.
- Implementation:
  - React frontend provides a clean, responsive interface.
  - Clear display of sentiment analysis results with visual indicators (e.g., color-coded scores).
  - Easy-to-navigate history of past analyses.

4. Implementation Details

4.1 Backend (Flask API)

- senti_api.py: Handles core API functionality.
- senti_ui_api.py: Manages communication between the frontend, database, and Google Colab model.
- SQLite database for data persistence.

4.2 Frontend (React)

- React components for text input, result display, and history viewing.
- Responsive design for various device sizes.
- Integration with backend API using Axios for HTTP requests.

4.3 Machine Learning Model

- TensorFlow model hosted on Google Colab.
- Analyzes text for education relevance and emotional content.

4.4 Containerization
- Docker containers for API and UI.
- docker-compose.yml for orchestrating multi-container setup.

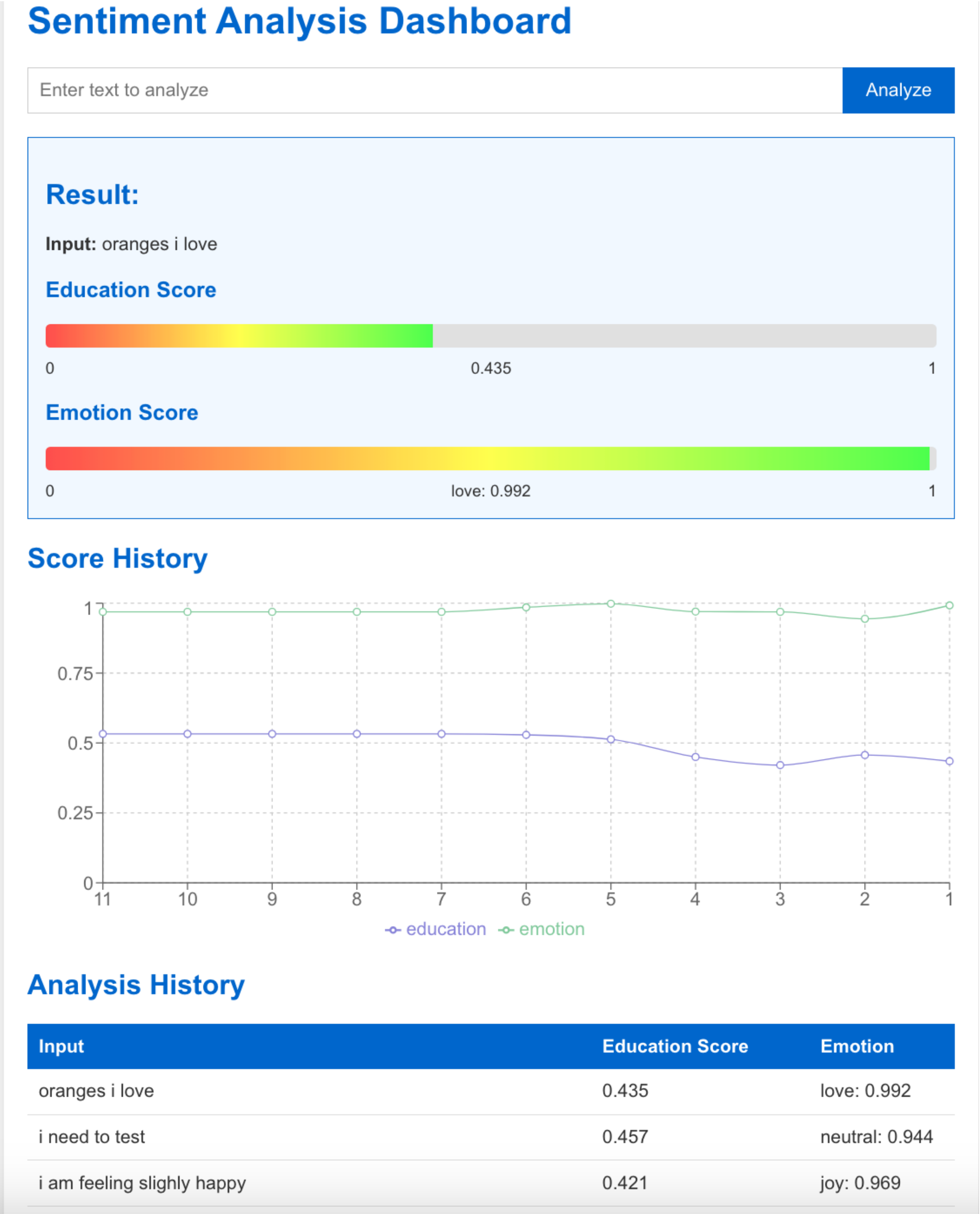
5. Testing and Quality Assurance

- Unit tests for API endpoints.
- Integration tests for database operations.
- Manual testing of the UI for usability and responsiveness.
- Docker build and run tests to ensure containerization works as expected.

6. Deployment Instructions

1. Ensure Docker and Docker Compose are installed.
2. Clone the repository.
3. Navigate to the project root.
4. Run docker-compose -f dockerfile/docker-compose.yml up --build
5. Access the UI at http://localhost:3000 and the API at http://localhost:5001

8. Screenshot of Web-UI



Sentiment Analysis Dashboard

Result:

Input: oranges i love

Education Score

0 0.435 1

Emotion Score

0 love: 0.992 1

Score History

Analysis History

Input	Education Score	Emotion
oranges i love	0.435	love: 0.992
i need to test	0.457	neutral: 0.944
i am feeling slighly happy	0.421	joy: 0.969

7. Future Enhancements

- Implement user authentication for personalized history.
- Add more sophisticated sentiment analysis models.
- Enhance the UI with more detailed visualizations of sentiment trends.
- Implement caching to improve performance for repeated analyses.

8. Conclusion

This project successfully meets both the functional and non-functional requirements set forth. It demonstrates a robust, scalable, and user-friendly sentiment analysis system that leverages modern web technologies and cloud-based machine learning. The use of Docker ensures easy deployment and scalability, while the clear separation of concerns in the architecture allows for future enhancements and maintainability.