# Sentiment Analysis API Project Documentation

## 1. Approach to Solving the Problem

The goal was to create a sentiment analysis API that accepts CSV or XLSX files containing customer reviews, processes these files to extract the review text, performs sentiment analysis, and returns the results in a structured JSON format. The API needed to integrate with the Groq API for sentiment analysis.

**Steps:**

1. **API Design**:
    - Used Flask to develop a RESTful API.
    - The API accepts both CSV and XLSX file formats, which are handled using the pandas and openpyxl libraries.
    - Implemented basic error handling for invalid file formats and Groq API request failures.
2. **File Handling**:
    - The API extracts customer reviews from the uploaded file by reading the relevant columns using pandas.
3. **Sentiment Analysis**:
    - Integrated Groq API to perform sentiment analysis on the review text.
    - A structured JSON response is returned, which includes sentiment scores for positive, negative, and neutral categories.
4. **Improved Version**:
    - Based on observations about performance, I created an alternative API that uses the TextBlob library, which proved to be faster and simpler to use than the Groq API for sentiment analysis.

## 2. How Structured Response Was Implemented

The API returns a structured response in JSON format, which includes the sentiment scores for positive, negative, and neutral reviews. Here's an example of the response format:

json

```
{
  "positive": 12,
  "negative": 15,
  "neutral": 23
}
```

## 3. Examples of API Usage

**3.1 Example Input:**

A CSV file with the following structure:

Customer Review

"Great product, really loved it!"

"Not satisfied with the service."

"Delivery was okay, but could be faster."

**3.2 Example API Call:**

POST /analyze

Content-Type: multipart/form-data

File: reviews.csv

**3.3 Example API Response:**

json

Copy code

```
{
  "positive": 1,
  "negative": 1,
  "neutral": 1
}
```

## 4. Analysis of Results

After analyzing the results, I observed the following:

- **Groq API**: The sentiment results were accurate but the response time was significantly slower, especially when processing larger batches of reviews.
- **TextBlob**: In contrast, TextBlob produced results much faster, with reasonably accurate sentiment analysis for basic cases, making it a suitable alternative for smaller datasets or when performance is a priority.

## 5. Limitations and Potential Improvements

**Limitations:**

- The **Groq API** was slow when handling multiple requests, which could be a bottleneck in high-traffic scenarios.
- **TextBlob**, while faster, has limitations in understanding complex or ambiguous sentences. It may not be as powerful as more sophisticated machine learning models like Groq.

**Potential Improvements:**

- **Asynchronous Processing**: Implementing async requests could help improve the performance of the API when using the Groq API.
- **Batch Processing**: Instead of processing reviews one by one, we can batch them before sending them to the Groq API to reduce the number of API calls.

## 6. Additional Insights and Observations

During the implementation of the project, I observed that the **TextBlob** library is a faster alternative to Groq when it comes to sentiment analysis for smaller-scale applications.

Therefore, I have also created an improved version of the sentiment analysis API using **TextBlob**, which is more efficient in terms of performance. The improved API will be included in the submission.

The directory structure of the project is as follows:

- **Sentiment_analysis**
    - **APIs**
        - groq_api.py - Implements the Groq-based sentiment analysis API.
        - improved_api.py - Implements the TextBlob-based sentiment analysis API.
    - **APPs**
        - app.py - Runs the web application and interfaces with the Groq API.
        - improved_app.py - Runs the improved version of the web application with TextBlob.
        - **templetes**
            - index.html - Contains the html for frontend app.
        - **static/js**
            - app.js - Contains the frontend JavaScript code for interacting with the backend APIs

.

Additionally, the project includes a requirements.txt file, listing all the required dependencies for both APIs.