

<b>Sentiment Analysis API with LLM Integration .....</b>	<b>2</b>
1.1 Code implementation:.....	2
1.2 How Structured Response is Implemented:.....	6
1.3 Analysis of Results:.....	6
1.4 Potential Improvements:.....	7
1.5 Additional insights and observations:.....	7

# Sentiment Analysis API with LLM Integration

The objective is to develop a Flask-based API that analyzes customer evaluations, allows file uploads in CSV and XLSX formats, and integrates a Large Language Model (LLM) with the Groq API to do sentiment analysis. In order to return sentiment data (positive, negative, and neutral) in percentage format, we organized the response.

The strategy adhered to these crucial steps:

- **File Handling:** Both CSV and XLSX file formats are processed via the Flask API, which also allows file uploads.
- **Review Extraction:** If there is a column in the file called "Review," the reviews are taken out of it.
- **Sentiment Analysis:** The Groq API is contacted for each review in order to do sentiment analysis; the outcomes are then combined.
- **Reaction:** The percentage of positive, negative, and neutral sentiments is provided in a structured JSON response that contains the sentiment analysis results.

## 1.1 Code implementation:

Firstly, what I did was create a virtual environment before installing the necessary dependencies within the environment. The following code, `python -m venv myenv`, Myenv is the name of the virtual environment.

Now, using pip, we install the following libraries, such as Pandas, Flask, and Groq, within the virtual environment.

```

from flask import Flask, request, jsonify
import pandas as pd
import os
import groq # Import Groq library
from groq import Groq, Client # Import Groq library
from dotenv import load_dotenv, find_dotenv
import json # Import json library

|

app = Flask(__name__)

```

Fig. 1.1: Importing main libraries

On Fig. 1.1 above, we started by importing all the necessary libraries needed for our project, and later on, we created an app for Flask.

```

def analyze_sentiment(review_text):
    try:
        client = Groq(
            # This is the default and can be omitted
            api_key=os.environ.get("NEW_GROQ_API_KEY"),
        )

        chat_completion = client.chat.completions.create(
            messages=[
                {
                    "role": "user",
                    "content": f"perform sentiment analysis on the input text: {review_text} I should just see number the number"
                }
            ],
            model="llama3-8b-8192",
        )

        ### Here I am cleaning the data to ease the sentiment analysis and to get the number of positive, negative, and neutral
        collected_response = str(chat_completion.choices[0].message.content).lower()
        collected_response = collected_response[collected_response.find('positive:'):collected_response.find('neutral:')+10]

```

Fig. 1.2: Implementing sentiment analysis using GroqAPI Part 1

From Fig. 1.2 above, first we have to try and catch situations to handle unforeseen errors in case of API failure. So I obtained my API key from the Groq official website and stored it in an environmental variable. Then following the documentation, I created a query that takes in input as the review\_text and provides sentiment analysis for the text.

Then I later had to clean the data to extract the most relevant information from the query response provided by the API.

```

store = {} ### we use a dict n loop to extract the various numbers from the response
length = ''
for i in collected_response:
    try:
        int(i) # Try to convert the character to an integer
        length += str(i)
        if len(length) == 1: store['positive'] = i# Append the integer to the list
        if len(length) == 2: store['negative'] = i# Append the integer to the list
        if len(length) == 3: store['neutral'] = i# Append the integer to the list
    except ValueError:
        pass
print(store)
return store # Expected to return {'positive': x, 'negative': y, 'neutral': z}

except groq.APIConnectionError as e:
    print("The server could not be reached")
    print(e.__cause__) # an underlying Exception, likely raised within httpx.
except groq.RateLimitError as e:
    print("A 429 status code was received; we should back off a bit.")
except groq.APIError as e:
    print("A (variable) status_code: int code was received")
    print(e.status_code)
    print(e.response)

```

Fig. 1.3: Implementing sentiment analysis using GroqAPI Part 2

Fig. 1.3 above provides a continuation for implementing sentiment analysis. We can see the possible exceptions that are being handled in case of code failure, and the corresponding values for the positive, negative, and neutral are stored in a dictionary for further processing.

```

# Function to extract reviews from CSV or XLSX
def extract_reviews(file_path):
    # Load the file using pandas
    if file_path.endswith('.csv'):
        df = pd.read_csv(file_path)
    elif file_path.endswith('.xlsx'):
        df = pd.read_excel(file_path)

    # Assuming the reviews are stored in a column named 'review' (adjust as needed)
    if 'Review' not in df.columns:
        raise ValueError("File does not contain 'review' column")

    # Return the list of reviews
    return df['Review'].tolist()

```

Fig. 1.4: Extraction and Handling of Data from CSV or XLSX Files.

So Fig. 1.4 simply demonstrates how we utilize a well-known library called Pandas for data extraction and processing. So we later on convert the extracted data into a list to be used within the code.

```

# Set the maximum size for file uploads if needed (optional)
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024 # 16 MB limit

# Route to upload files
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({"error": "No file part"}), 400

    file = request.files['file']

    if file.filename == '':
        return jsonify({"error": "No selected file"}), 400

    if file and (file.filename.endswith('.csv') or file.filename.endswith('.xlsx')):
        file_path = os.path.join('/tmp', file.filename)
        file.save(file_path)
        print('all good')
        try:
            reviews = extract_reviews(file_path)
            positive, negative, neutral = 0, 0, 0

            # Perform sentiment analysis for each review

```

Fig 1.5: Implementing flask application for response handling, part 1

Here we create the route to upload our files (i.e., csv, xlsx) using the POST method for our API, which handles the response.

So from the code we load the data, and each data is processed one after the other, where the result is stored in positive, negative, or neutral.

```

# Perform sentiment analysis for each review
for review in reviews:
    try:
        sentiment = analyze_sentiment(review)
        positive += int(sentiment['positive']) # Add the positive sentiment score to the total
        negative += int(sentiment['negative']) # Add the negative sentiment score to the total
        neutral += int(sentiment['neutral']) # Add the neutral sentiment score to the total
    except Exception as e:
        print(e)

# Calculate the average sentiment scores
positive = positive / (positive+negative+neutral) * 100
negative = negative / (positive+negative+neutral) * 100
neutral = neutral / (positive+negative+neutral) * 100

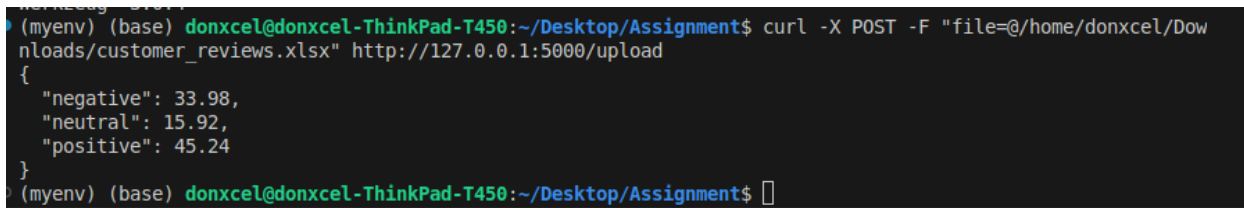
# Return the sentiment scores in percentages
return jsonify({
    "positive": round(positive, 2), # Round to 2 decimal places
    "negative": round(negative, 2), # Round to 2 decimal places
    "neutral": round(neutral, 2) # Round to 2 decimal places
}), 200

except Exception as e:
    return jsonify({"error": str(e)}), 500
else:
    return jsonify({"error": "Invalid file format, only CSV and XLSX are allowed"}), 400

```

Fig. 1.6: Implementing flask application for response handling, part 2

So we get the data gotten from the sentiment analysis function from Groq and further compute it into the API. Now I personally decided to output the result in the percentage format so we can see if we have more positive feedback, negative feedback, or neutral feedback, and the output is returned in JSON format. I utilized curl to handle the API request as shown from the fig below.



```
(myenv) (base) donxcel@donxcel-ThinkPad-T450:~/Desktop/Assignment$ curl -X POST -F "file=@/home/donxcel/Downloads/customer_reviews.xlsx" http://127.0.0.1:5000/upload
{
  "negative": 33.98,
  "neutral": 15.92,
  "positive": 45.24
}
(myenv) (base) donxcel@donxcel-ThinkPad-T450:~/Desktop/Assignment$
```

Fig. 1.7: sample output after execution

## 1.2 How Structured Response is Implemented:

- Each review is processed individually using the **Groq API** to determine sentiment counts (positive, negative, neutral).
- The sentiments are aggregated across all reviews, and percentages are calculated for each sentiment type.
- The API returns a **JSON** response containing the percentage values for positive, negative, and neutral sentiments, rounded to two decimal places.

## 1.3 Analysis of Results:

- **Accuracy:** The accuracy of sentiment analysis depends on the quality of the underlying LLM (Large Language Model) and the Groq API. In this case, sentiment results are aggregated and averaged, giving an overall picture of customer feedback.
- **Limitations:**
  - **Model Dependency:** The sentiment analysis relies entirely on the external Groq API and the specific LLM model used, which may have biases or limitations in understanding complex reviews.

- **Neutral Sentiment Ambiguity:** In some cases, neutral sentiment may be difficult to interpret, especially if the review contains both positive and negative statements.
- **File Format Dependence:** The file must contain a '**Review**' column. Files with different column names for reviews would not work unless adjusted.

#### 1.4 Potential Improvements:

- **Sentiment Confidence:** Introduce a confidence score for each sentiment category to indicate how certain the model is about its classification.
- **Enhanced File Handling:** Allow the user to specify the column containing reviews if it's not named 'Review'.
- **Multiple Language Support:** Add support for reviews written in different languages by detecting and translating text before performing sentiment analysis.
- **Batch Processing:** Optimize the API to handle larger datasets and batch sentiment analysis requests to reduce API overhead.

#### 1.5 Additional insights and observations:

- **API Rate Limits:** If this API were to scale, monitoring the Groq API usage and rate limits would be essential, especially if handling high volumes of customer reviews.
- **Error Handling:** robust error handling was implemented to capture issues related to file formats, missing review columns, and API errors. However, the error messages could be further improved to provide more detailed feedback to the user.