

Aspect Based Sentiment Analysis



make this prompt better

"I want you to create an aspect based sentiment analysis program that takes this csv and uses its 'Text' column which contains the hospital reviews and I want the program to choose the best 5 aspects after going through all reviews and then use some sort of aspect based sentiment analysis model to score all the aspects and get a final sentiment score and then compare it with the score in 'Total Score' col which is out of 5 to get a corelation between them"



Here's a refined version of your prompt:

"Create an aspect-based sentiment analysis program that processes a CSV file containing hospital reviews. The program should analyze the 'Text' column, which contains the review data, and identify the top 5 most relevant aspects across all reviews. For each of these aspects, perform sentiment analysis using an appropriate aspect-based sentiment model to generate a sentiment score. Finally, compare the aggregated sentiment score for each review with the 'Total Score' column (which is out of 5) to determine the correlation between the sentiment analysis results and the overall review score."

This version is clearer and structured for better understanding.



can you create the program as well



Sure! Here's a Python program using `pandas` for data handling, `nltk` for basic sentiment analysis, and `sklearn` to calculate correlation. You can adjust it to use a more advanced aspect-based sentiment analysis model, like one from `transformers`, if needed. This code assumes you have a CSV file where one column is 'Text' (reviews) and another is 'Total Score' (out of 5).

Here's how you can get started:

Step 1: Install necessary packages

If you haven't installed these libraries, you can do so using the following commands:

```
bash
```

```
pip install pandas nltk scikit-learn
```

Step 2: The Code

python

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk import FreqDist
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import r2_score

# Download nltk data
nltk.download('punkt')
nltk.download('stopwords')

# Load the dataset
df = pd.read_csv('hospital_reviews.csv')

# Function to preprocess text: tokenization, stopword removal
def preprocess_text(text):
    tokens = nltk.word_tokenize(text.lower())
    tokens = [t for t in tokens if t.isalpha()] # Keep only alphabetic tokens
    stop_words = set(stopwords.words('english'))
    tokens = [t for t in tokens if t not in stop_words] # Remove stopwords
    return tokens

# Process the "Text" column
df['tokens'] = df['Text'].apply(preprocess_text)

# Step 1: Find the top 5 aspects
# Using a CountVectorizer to extract frequently mentioned aspects (words)
vectorizer = CountVectorizer(max_features=100)
X = vectorizer.fit_transform(df['Text'])
freq_words = vectorizer.get_feature_names_out()

# Manually choose top 5 aspects (for now we use the top 5 most frequent words)
# In practice, these should be domain-specific (service, staff, cleanliness, etc.)
top_aspects = freq_words[:5]
print(f"Top 5 aspects: {top_aspects}")

# Step 2: Perform sentiment analysis
from nltk.sentiment import SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

def get_aspect_sentiment(text, aspect):
    # Check if the aspect is in the text and get the sentiment score for that sentence
    if aspect in text:
        sentiment = sia.polarity_scores(text)
        return sentiment['compound'] # Return the compound sentiment score (-1 to 1)
    return 0

# Apply sentiment analysis on each aspect
for aspect in top_aspects:
    df[f'{aspect}_sentiment'] = df['Text'].apply(lambda x: get_aspect_sentiment(x, aspect))

# Step 3: Calculate final sentiment score (average of aspect sentiment scores)
df['final_sentiment_score'] = df[[f'{aspect}_sentiment' for aspect in top_aspects]].mean(axis=1)

# Step 4: Compare with 'Total Score' and find correlation
df['Total Score'] = df['Total Score'].astype(float)
correlation = df['final_sentiment_score'].corr(df['Total Score'])
print(f"Correlation between final sentiment score and Total Score: {correlation}")

# Optional: Calculate R-squared for a linear relationship
r2 = r2_score(df['Total Score'], df['final_sentiment_score'])
print(f"R-squared between final sentiment score and Total Score: {r2}")
```

```
# Export results to a new CSV file
df.to_csv('sentiment_analysis_results.csv', index=False)
```

Explanation:

1. **Preprocessing:** Tokenizes reviews and removes stopwords.
2. **Aspect Selection:** Uses `CountVectorizer` to find the most frequent words as placeholders for aspects. You can replace this with domain-specific aspects such as 'service', 'staff', etc.
3. **Sentiment Analysis:** Uses `nlk`'s `SentimentIntensityAnalyzer` to calculate sentiment scores for sentences containing each aspect.
4. **Correlation:** Computes correlation between the final sentiment score (average sentiment for the top 5 aspects) and the 'Total Score'.

Next Steps:

- **Refine Aspects:** Replace the frequent words with actual domain-specific aspects like "service", "staff", "cleanliness", etc.
- **Advanced Sentiment Analysis:** Consider using pre-trained models like BERT for better aspect-based sentiment analysis using the `transformers` library.

Let me know if you need further improvements or guidance on running the code!



```
[nlk_data] Downloading package punkt to /home/rishav/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /home/rishav/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
Traceback (most recent call last):
  File "/home/rishav/.local/lib/python3.10/site-packages/pandas/core/indexes/base.py", line 3805,
in get_loc
    return self._engine.get_loc(casted_key)
  File "index.pyx", line 167, in pandas._libs.index.IndexEngine.get_loc
  File "index.pyx", line 196, in pandas._libs.index.IndexEngine.get_loc
  File "pandas/_libs/hashtable_class_helper.pxi", line 7081, in
pandas._libs.hashtable.PyObjectHashTable.get_item
  File "pandas/_libs/hashtable_class_helper.pxi", line 7089, in
pandas._libs.hashtable.PyObjectHashTable.get_item
KeyError: 'Text'
```

The above exception was the direct cause of the following exception:

```
Traceback (most recent call last):
  File "/mnt/c/Users/rishav/data/major_project/main.py", line 25, in <module>
    df["tokens"] = df["Text"].apply(preprocess_text)
  File "/home/rishav/.local/lib/python3.10/site-packages/pandas/core/frame.py", line 4102, in
__getitem__
    indexer = self.columns.get_loc(key)
  File "/home/rishav/.local/lib/python3.10/site-packages/pandas/core/indexes/base.py", line 3812,
in get_loc
```

```
raise KeyError(key) from err
KeyError: 'Text'
```



The error you're encountering indicates that the column `'Text'` does not exist in your DataFrame. This could happen if:

1. The column name in the CSV file is different (e.g., `'Review'`, `'Comment'`, or something similar).
2. There might be a typo or case sensitivity issue (e.g., `'text'` vs `'Text'`).

Steps to Fix:

1. **Inspect the Column Names:** Check the actual column names in your CSV file to see if the column name is different. You can do this by printing the column names.

Here's a quick way to inspect your DataFrame:

```
python

print(df.columns)
```

This will show you all the column names in the DataFrame. Look for the one that contains the hospital reviews and replace `'Text'` with the correct column name.

2. **Update the Code:** Once you know the correct column name, update the code accordingly.

For example, if the column name is `'Review'`, modify the line:

```
python

df["tokens"] = df["Text"].apply(preprocess_text)
```

to:

```
python

df["tokens"] = df["Review"].apply(preprocess_text)
```

Let me know what you find, and I can assist further!



i want the aspects to be domain specific like food, cleanliness, doctors, service etc and i want the correlation to be show in each row, like the totalScore (1 to 5) which is the rating of the hospital compared to our score which is the sentiment score given based on the aspects

modify the code accordingly

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import r2_score

# Download nltk data
nltk.download("punkt")
nltk.download("stopwords")

# Load the dataset
df = pd.read_csv("dataset_v1.xlsx")

# Function to preprocess text: tokenization, stopword removal
def preprocess_text(text):
    tokens = nltk.word_tokenize(text.lower())
    tokens = [t for t in tokens if t.isalpha()] # Keep only alphabetic tokens
    stop_words = set(stopwords.words("english"))
    tokens = [t for t in tokens if t not in stop_words] # Remove stopwords
    return tokens

# Process the "Text" column
df["tokens"] = df["text"].apply(preprocess_text)

# Step 1: Find the top 5 aspects
# Using a CountVectorizer to extract frequently mentioned aspects (words)
vectorizer = CountVectorizer(max_features=100)
X = vectorizer.fit_transform(df["tokens"])
freq_words = vectorizer.get_feature_names_out()

# Manually choose top 5 aspects (for now we use the top 5 most frequent words)
# In practice, these should be domain-specific (service, staff, cleanliness, etc.)
top_aspects = freq_words[:5]
print(f"Top 5 aspects: {top_aspects}")

# Step 2: Perform sentiment analysis
from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()

def get_aspect_sentiment(text, aspect):
    # Check if the aspect is in the text and get the sentiment score for that sentence
    if aspect in text:
        sentiment = sia.polarity_scores(text)
        return sentiment["compound"] # Return the compound sentiment score (-1 to 1)
```

```

return 0

# Apply sentiment analysis on each aspect
for aspect in top_aspects:
    df[f"{aspect}_sentiment"] = df["text"].apply(
        lambda x: get_aspect_sentiment(x, aspect)
    )

# Step 3: Calculate final sentiment score (average of aspect sentiment scores)
df["final_sentiment_score"] = df[
    [f"{aspect}_sentiment" for aspect in top_aspects]
].mean(axis=1)

# Step 4: Compare with 'Total Score' and find correlation
df["totalScore"] = df["totalScore"].astype(float)
correlation = df["final_sentiment_score"].corr(df["totalScore"])
print(f"Correlation between final sentiment score and Total Score: {correlation}")

# Optional: Calculate R-squared for a linear relationship
r2 = r2_score(df["totalScore"], df["final_sentiment_score"])
print(f"R-squared between final sentiment score and totalScore: {r2}")

# Export results to a new CSV file
df.to_csv("sentiment_analysis_results.csv", index=False)

```



Here's an updated version of your code that incorporates domain-specific aspects like `food`, `cleanliness`, `doctors`, `service`, etc., and computes correlation row-by-row between the `totalScore` (1 to 5) and the sentiment scores for each aspect.

Updated Code:

python

```

import pandas as pd
import nltk
from nltk.corpus import stopwords
from sklearn.metrics import r2_score

# Download nltk data
nltk.download("punkt")
nltk.download("stopwords")

# Load the dataset
df = pd.read_csv("dataset_v1.csv") # Adjust based on your actual file format, use
pd.read_excel() if necessary

# Function to preprocess text: tokenization, stopwords removal
def preprocess_text(text):
    tokens = nltk.word_tokenize(text.lower())
    tokens = [t for t in tokens if t.isalpha()] # Keep only alphabetic tokens
    stop_words = set(stopwords.words("english"))
    tokens = [t for t in tokens if t not in stop_words] # Remove stopwords

```

```

    return tokens

# Process the "Text" column
df["tokens"] = df["text"].apply(preprocess_text)

# Define domain-specific aspects manually
top_aspects = ['food', 'cleanliness', 'doctors', 'service', 'staff']

# Perform sentiment analysis
from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()

def get_aspect_sentiment(text, aspect):
    # Check if the aspect is in the text and get the sentiment score for that sentence
    if aspect in text:
        sentiment = sia.polarity_scores(text)
        return sentiment["compound"] # Return the compound sentiment score (-1 to 1)
    return 0

# Apply sentiment analysis on each aspect
for aspect in top_aspects:
    df[f"{aspect}_sentiment"] = df["text"].apply(lambda x: get_aspect_sentiment(x,
    aspect))

# Calculate final sentiment score (average of aspect sentiment scores)
df["final_sentiment_score"] = df[[f"{aspect}_sentiment" for aspect in
top_aspects]].mean(axis=1)

# Step 4: Compare sentiment score with 'totalScore' for each review and calculate
correlation row-by-row
def calculate_row_correlation(row):
    return row["final_sentiment_score"] * 5 # Scaling sentiment score (range -1 to 1) to
match the totalScore scale (1-5)

df["sentiment_vs_totalScore"] = df.apply(calculate_row_correlation, axis=1)

# Calculate R-squared for each row to show the match between the sentiment score and
totalScore
df["correlation_diff"] = abs(df["totalScore"] - df["sentiment_vs_totalScore"])

# Export results to a new CSV file
df.to_csv("sentiment_analysis_results_with_correlation.csv", index=False)

# Print example results to confirm success
print(df[["text", "totalScore", "final_sentiment_score", "sentiment_vs_totalScore",
"correlation_diff"]].head())

```

Changes and Enhancements:

- 1. Manual Aspect Selection:** I have hardcoded domain-specific aspects such as `'food'`, `'cleanliness'`, `'doctors'`, `'service'`, and `'staff'` into the `top_aspects` list. These can be expanded or refined based on the dataset.
- 2. Aspect Sentiment Analysis:** Sentiment analysis is now done for these specific aspects, and the sentiment score for each aspect is calculated for every review.
- 3. Row-by-Row Correlation:**
 - A new column `sentiment_vs_totalScore` is created, scaling the sentiment score (which is between -1 and 1) to a 1-to-5 scale to match the `totalScore` scale.
 - The difference between the `totalScore` and the scaled sentiment score (`correlation_diff`) is calculated for each row.

4. **Final Results:** The final results include the raw review, ``totalScore``, the calculated sentiment score (``final_sentiment_score``), the scaled sentiment score (``sentiment_vs_totalScore``), and the correlation difference (``correlation_diff``).
5. **Output:** The output is saved as ``sentiment_analysis_results_with_correlation.csv``, which contains all the calculated data for further analysis.

Optional Adjustments:

- If the aspect-based sentiment analysis needs to be more advanced, you could integrate a pre-trained model from ``transformers`` to handle multi-aspect sentiment scoring.
- Fine-tune the scaling or transformation used to calculate ``sentiment_vs_totalScore`` depending on how closely you want it to reflect the 1-to-5 ``totalScore``.

Let me know if you need further modifications!



can you calculate the correlation as a percentage as well



To calculate the correlation as a percentage, you can compare the sentiment-based score (scaled from -1 to 1 and then adjusted to a 1-to-5 scale) against the ``totalScore`` on a per-row basis. You can express the difference between the two scores as a percentage of the maximum possible score (5 in this case).

Here's how you can update the code to include this percentage-based correlation:

Updated Code with Correlation Percentage:

python

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from sklearn.metrics import r2_score

# Download nltk data
nltk.download("punkt")
nltk.download("stopwords")

# Load the dataset
df = pd.read_csv("dataset_v1.csv") # Adjust based on your actual file format, use
pd.read_excel() if necessary

# Function to preprocess text: tokenization, stopwords removal
def preprocess_text(text):
    tokens = nltk.word_tokenize(text.lower())
    tokens = [t for t in tokens if t.isalpha()] # Keep only alphabetic tokens
    stop_words = set(stopwords.words("english"))
    tokens = [t for t in tokens if t not in stop_words] # Remove stopwords
    return tokens

# Process the "Text" column
df["tokens"] = df["text"].apply(preprocess_text)
```



```
# Define domain-specific aspects manually
top_aspects = ['food', 'cleanliness', 'doctors', 'service', 'staff']

# Perform sentiment analysis
from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()

def get_aspect_sentiment(text, aspect):
    # Check if the aspect is in the text and get the sentiment score for that sentence
    if aspect in text:
        sentiment = sia.polarity_scores(text)
        return sentiment["compound"] # Return the compound sentiment score (-1 to 1)
    return 0

# Apply sentiment analysis on each aspect
for aspect in top_aspects:
    df[f"{aspect}_sentiment"] = df["text"].apply(lambda x: get_aspect_sentiment(x,
    aspect))

# Calculate final sentiment score (average of aspect sentiment scores)
df["final_sentiment_score"] = df[[f"{aspect}_sentiment" for aspect in
top_aspects]].mean(axis=1)

# Step 4: Compare sentiment score with 'totalScore' for each review and calculate
correlation row-by-row
def calculate_row_correlation(row):
    return row["final_sentiment_score"] * 5 # Scaling sentiment score (range -1 to 1) to
match the totalScore scale (1-5)

df["sentiment_vs_totalScore"] = df.apply(calculate_row_correlation, axis=1)

# Calculate the correlation difference as a percentage for each row
def calculate_correlation_percentage(row):
    diff = abs(row["totalScore"] - row["sentiment_vs_totalScore"])
    return (1 - (diff / 5)) * 100 # Calculate percentage: lower difference means higher
correlation

df["correlation_percentage"] = df.apply(calculate_correlation_percentage, axis=1)

# Calculate R-squared for each row to show the match between the sentiment score and
totalScore
df["correlation_diff"] = abs(df["totalScore"] - df["sentiment_vs_totalScore"])

# Export results to a new CSV file
df.to_csv("sentiment_analysis_results_with_correlation_percentage.csv", index=False)

# Print example results to confirm success
print(df[["text", "totalScore", "final_sentiment_score", "sentiment_vs_totalScore",
"correlation_diff", "correlation_percentage"]].head())
```

Key Changes:

1. Correlation Percentage:

- The function `calculate_correlation_percentage` computes the correlation as a percentage by taking the difference between the `totalScore` and the scaled sentiment score (`sentiment_vs_totalScore`), dividing by the maximum score (5), and subtracting this from 1 to get a percentage (with lower differences yielding higher percentages).
- The result is stored in the new column `correlation_percentage`.

2. Final Output:

- The final CSV file will contain the `correlation_percentage` for each row, showing how well the sentiment-based score aligns with the hospital's `totalScore`, expressed as a percentage.

Example Output:

The output will include columns such as:

- ``totalScore``: The actual rating for the hospital (1-5).
- ``final_sentiment_score``: The calculated sentiment score based on the aspects.
- ``sentiment_vs_totalScore``: The scaled sentiment score to match the range of the ``totalScore``.
- ``correlation_diff``: The absolute difference between the ``totalScore`` and the sentiment score.
- ``correlation_percentage``: The correlation as a percentage, where higher values (closer to 100%) indicate a stronger correlation between the sentiment score and the totalScore.

This will give a clearer and more interpretable measure of the correlation for each row.

Let me know if you need further customization!



create a report for this which includes all the info about the things used in this code



Here's a detailed report that explains the code, the concepts, and the tools used in performing the aspect-based sentiment analysis and correlation calculation.

Aspect-Based Sentiment Analysis and Correlation Report

1. Overview

This report describes an aspect-based sentiment analysis program built to analyze hospital reviews and compare sentiment scores based on key aspects (e.g., food, cleanliness, doctors, service, staff) with the ``totalScore`` (ratings provided by users). The program processes textual data, performs sentiment analysis, and computes the correlation between the user-provided rating and the sentiment-based score for each review.

2. Dataset Description

The dataset is assumed to contain hospital reviews and a ``totalScore`` column representing user ratings on a scale of 1 to 5. The primary column used in this analysis is:

- **Text**: Contains the hospital reviews written by users.
- **totalScore**: A numerical column representing the hospital rating (1 to 5).

3. Tools and Libraries Used

a. Python

The entire program is built using Python due to its versatility and strong support for text processing, data analysis, and machine learning.

b. Pandas

Pandas is a Python library used for data manipulation and analysis. It provides fast, flexible, and expressive data structures like `DataFrame`, which are used to load and process tabular data.

c. NLTK (Natural Language Toolkit)

NLTK is a powerful Python library used for working with human language data. It is particularly useful for text preprocessing, tokenization, and sentiment analysis. The key components used from NLTK are:

- **nltk.corpus.stopwords**: A predefined list of common words (e.g., "and", "the", "is") that do not contribute much to sentiment or meaning in a sentence and are removed in the preprocessing phase.
- **nltk.sentiment.SentimentIntensityAnalyzer**: A tool used to perform sentiment analysis on text. It uses the VADER model to return a compound sentiment score for each review, which is a number between -1 (most negative) and 1 (most positive).

d. Scikit-learn

Scikit-learn is a library used for machine learning and data analysis. In this project, it's used for:

- **r2_score**: A metric used to calculate the R-squared score, indicating the goodness of fit of the sentiment score to the `totalScore`.

4. Program Workflow

a. Data Preprocessing

1. Tokenization:

- The `nltk.word_tokenize()` function splits the text into individual words (tokens). This helps in analyzing the reviews on a word-by-word basis.

2. Stopword Removal:

- Words like "the", "and", and "is" (which are common and carry little meaning) are removed using the NLTK's stopwords list to ensure the focus remains on meaningful words.

3. Aspect Identification:

- The program identifies domain-specific aspects manually, such as:
 - **food**
 - **cleanliness**
 - **doctors**
 - **service**
 - **staff** These are the key areas users may talk about in reviews. The program then performs sentiment analysis for each aspect.

b. Sentiment Analysis

- **SentimentIntensityAnalyzer** is used to compute the sentiment score for each aspect in a review. It returns a compound score between -1 (most negative) and 1 (most positive).
- The program calculates an **average sentiment score** for all aspects in a review. This score is referred to as the ``final_sentiment_score``.

c. Aspect-Based Sentiment Calculation

- For each review, the sentiment score for each aspect (e.g., food, cleanliness, service) is calculated.
- The sentiment scores for all aspects are averaged to create a final sentiment score (``final_sentiment_score``).

d. Correlation and Scaling

- The sentiment score (``final_sentiment_score``) is scaled to match the range of the ``totalScore`` (1 to 5). This is done by multiplying the sentiment score by 5 (since sentiment scores range from -1 to 1).
- A column ``sentiment_vs_totalScore`` is created to hold this scaled sentiment score.
- The correlation difference (``correlation_diff``) is calculated as the absolute difference between the user-provided ``totalScore`` and the sentiment-based score.

e. Correlation as Percentage

- The program computes the **correlation percentage** for each review. This indicates how close the sentiment score is to the user-provided ``totalScore``. The formula used is:

$$\text{correlation percentage} = \left(1 - \frac{\text{difference}}{5}\right) \times 100$$

The correlation percentage gives a clearer interpretation of how well the sentiment-based score correlates with the totalScore, with 100% representing a perfect match.

5. Output

- **sentiment_analysis_results_with_correlation_percentage.csv**: The final output is saved in a CSV file that contains the following columns:
 - **Text**: The original hospital review.
 - **totalScore**: The user's rating of the hospital (1-5).
 - **final_sentiment_score**: The average sentiment score derived from the aspects.
 - **sentiment_vs_totalScore**: The sentiment score scaled to match the totalScore range (1-5).
 - **correlation_diff**: The absolute difference between the ``totalScore`` and the scaled sentiment score.
 - **correlation_percentage**: The correlation as a percentage, with higher values indicating a stronger match between the sentiment score and the totalScore.

6. Conclusion

This program performs an aspect-based sentiment analysis on hospital reviews by breaking down the reviews into key aspects (such as food, cleanliness, service), calculating sentiment scores for these

aspects, and comparing the final sentiment-based score with user-provided ratings (`totalScore`). The program offers a detailed analysis by calculating both the absolute difference between scores and a percentage-based correlation, giving insights into how well user sentiments align with their provided ratings.

This approach can be used in various domains beyond healthcare, wherever user reviews contain domain-specific aspects.

7. Future Improvements

- **Aspect Extraction:** Instead of manually defining the aspects, the program could be improved by automatically extracting domain-specific aspects from the text using techniques like topic modeling.
- **Advanced Sentiment Models:** For more accurate sentiment analysis, pre-trained models like BERT or other transformer-based models could be integrated to capture more nuanced sentiment.

Let me know if you need further information or modifications to this report!



create a prompt which can be fed to an ai flowchart creator for this code



Here's a concise and clear prompt that can be used to generate a flowchart for the code:

Prompt for AI Flowchart Creator:

"Create a flowchart for a program that performs aspect-based sentiment analysis on hospital reviews and calculates a correlation with user-provided scores. The flowchart should include the following steps:

1. **Start**
2. **Input Data:** Load the dataset containing hospital reviews (Text) and user ratings (TotalScore).
3. **Preprocess Text:**
 - Tokenize the reviews (split text into words).
 - Remove stopwords (common, irrelevant words).
4. **Define Aspects:** Identify domain-specific aspects such as food, cleanliness, doctors, service, and staff.
5. **Sentiment Analysis:**
 - For each aspect, perform sentiment analysis using a pre-trained sentiment analyzer.
 - Calculate the sentiment score for each aspect in the review.
6. **Average Sentiment Score:** Compute the final sentiment score as the average of all aspect sentiment scores for each review.
7. **Scale Sentiment Score:** Scale the sentiment score to match the rating scale (1 to 5).
8. **Calculate Correlation:**
 - Compare the scaled sentiment score with the user's rating (TotalScore).

- Calculate the difference between the scores.
 - Compute a correlation percentage based on the difference.
9. **Output:** Save the results including the original review, TotalScore, sentiment score, and correlation percentage to a CSV file.
10. **End"**
-

This prompt should provide a clear structure for creating a flowchart that visually represents the logic and flow of the code.

ChatGPT can make mistakes. Check important info.