Sentiment Analysis in Climate-FocusedNew Energy Solutions

Introduction:

Sentiment analysis, also known as opinion mining, is a natural language processing technique that categorizes text data as positive, negative, or neutral based on the opinions, attitudes, or emotions conveyed by the words used.
Sentiment analysis is a crucial tool in the climate sector for assessing public opinion and emotions towards new energy solutions, technologies, and policies aimed at mitigating climate change and transitioning to cleaner energy sources, providing valuable insights into stakeholder responses.

Need of the Project:

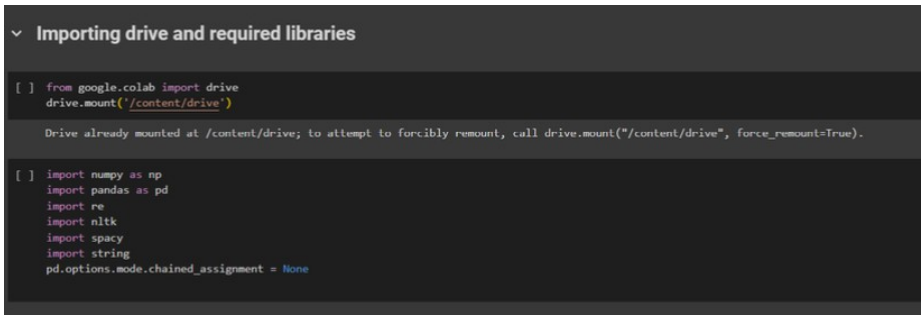
The urgency to combat climate change has led to a shift towards sustainable energy solutions. Understanding public sentiment is crucial for policymakers, businesses, and advocates to tailor strategies, policies, and communication efforts effectively. Sentiment analysis helps identify societal attitudes, concerns, and acceptance levels, preventing challenges, ensuring smooth adoption, and fostering a collective commitment to climatechange.

Methodology:

- **Data- Extraction**: We extracted comments from approximately 80-90 Youtube Videos, from popular news channels namely BBC, Fox,CNN,etc, using the official Youtube API and used it s our dataset for this project.

```python
import csv
import random
from googleapiclient.discovery import build


# Your API key obtained from the Google Cloud Console
api_key = 'AIzaSyApB7X4dUaFZ3gn2w5aqMyGnCim6t-XgSc'

# List of YouTube video IDs
video_ids = ['PEr2C3Jzkns', 'A7LP-rMnmFU' ]

# Create a YouTube Data API client
youtube = build('youtube', 'v3', developerKey=api_key)

# List to store all comments with their associated video IDs
all_comments = []

# Fetch comments for each video
for video_id in video_ids:
    # Call the API to fetch comments for the video
    next_page_token = None
    comments = []
    while True:
        response = youtube.commentThreads().list(
            part='snippet',
            videoId=video_id,
            maxResults=100,  # Maximum number of comments to fetch per page
            pageToken=next_page_token,
            textFormat='plainText'
        ).execute()

        # Extract comments from the response
        for item in response['items']:
            comment = item['snippet']['topLevelComment']['snippet']['textDisplay']
            comments.append(comment)

        # Check if there are more pages of comments
        next_page_token = response.get('nextPageToken')
        if not next_page_token:
            break

    # Shuffle and select a random subset of comments
    random_comments = random.sample(comments, min(100, len(comments)))

    # Append to the list
    all_comments.extend(random_comments)

# Write comments to a CSV file
with open('youtube_comments.csv', 'w', newline='', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['Comment'])  # Write header row
    writer.writerows([[comment] for comment in all_comments])
```

- **Pre-Processing:** The dataset for sentiment analysis was prepared using a pre-processing pipeline. Libraries like **pandas** and **NLTK** were imported, and the dataset containing comments was loaded. The comments were converted into strings,**empty comments were removed**, and a l**anguage filtering mechanism** was used to focus on English-language comments. **Lowercasing** and **lemmatization** techniques were applied to standardize the text, and emojis were transformed into textual representations. **HTML tags and URLs were removed** to eliminate irrelevant information. A **spelling correction step** was integrated to enhance text coherence andaccuracy. This meticulous pre-processing workflow ensures clean, standardized text data for sentiment analysis.

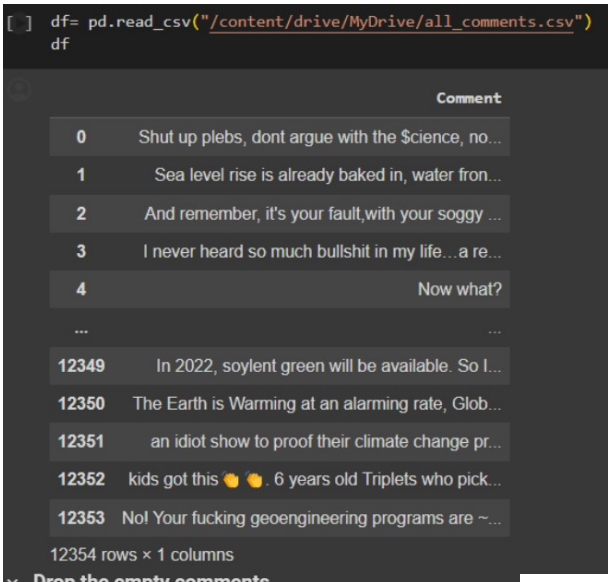


Importing drive and required libraries

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] import numpy as np
    import pandas as pd
    import re
    import nltk
    import spacy
    import string
    pd.options.mode.chained_assignment = None
```



```
[ ] df= pd.read_csv("/content/drive/MyDrive/all_comments.csv")
    df
```

	Comment
0	Shut up plebs, dont argue with the $cience, no...
1	Sea level rise is already baked in, water fron...
2	And remember, it's your fault,with your soggy ...
3	I never heard so much bullshit in my life…a re...
4	Now what?
...	...
12349	In 2022, soylent green will be available. So I...
12350	The Earth is Warming at an alarming rate, Glob...
12351	an idiot show to proof their climate change pr...
12352	kids got this👏👏. 6 years old Triplets who pick...
12353	No! Your fucking geoengineering programs are ~...

12354 rows × 1 columns

Drop the empty comments

```
[ ] # Remove empty rows from the DataFrame
    english_comments_df = english_comments_df.dropna()

    # Reset the index of the DataFrame
    english_comments_df = english_comments_df.reset_index(drop=True)

    english_comments_df
```

	Comment
0	Shut up plebs, dont argue with the $cience, no...
1	Sea level rise is already baked in, water fron...
2	And remember, it's your fault,with your soggy ...
3	I never heard so much bullshit in my life…a re...
4	Now what?
...	...
12086	In 2022, soylent green will be available. So I...
12087	The Earth is Warming at an alarming rate, Glob...
12088	an idiot show to proof their climate change pr...
12089	kids got this👏👏. 6 years old Triplets who pick...
12090	No! Your fucking geoengineering programs are ~...

12091 rows × 1 columns

Converting the comment column to string

```
[ ]  df["Comment"] = df["Comment"].astype(str)    ##convert to string
```

Filter out comments that are not primarily in English

```
[ ] # Install the library if not already installed
    !pip install langdetect

    # Import the language detection function
    from langdetect import detect

    def filter_english_comments(df, english_threshold=0.7):

        english_comments = []
        for comment in df['Comment']:
            try:
                # Detect the language of the comment
                lang = detect(comment)
                # Calculate the proportion of English text
                if lang == 'en':
                    english_comments.append(comment)
                else:
                    # Calculate the proportion of English text in mixed-language comments
                    english_ratio = sum(1 for char in comment if char.isascii()) / len(comment)
                    if english_ratio >= english_threshold:
                        english_comments.append(comment)
            except:
                # Handle errors in language detection
                pass
        return df[df['Comment'].isin(english_comments)]

    english_comments_df = filter_english_comments(df)

    Requirement already satisfied: langdetect in /usr/local/lib/python3.10/dist-packages (1.0.9)
    Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from langdetect) (1.16.0)
```

```
]  # Apply the emoticon conversion function to the 'Comment' column in df1
   df1['Comment'] = df1['Comment'].apply(convert_emoticons_to_text)
   # Apply the emoji conversion function to the 'Comment' column in df1
   df1['Comment'] = df1['Comment'].apply(convert_emojis_to_text)

   # Print the DataFrame after conversion
   print(df1)
```

```
                                                 Comment
0      shut up plebs, dont argue with the $cience, no...
1      sea level rise is already baked in, water fron...
2      and remember, it's your fault,with your soggy ...
3      i never heard so much bullshit in my life…a re...
4                                              now what?
...                                                  ...
12086  in 2022, soylent green will be available. so i...
12087  the earth is warming at an alarming rate, glob...
12088  an idiot show to proof their climate change pr...
12089  kid got this:clapping_hands::clapping_hands:. ...
12090  no! your fucking geoengineering program are ~ ...

[12091 rows x 1 columns]
```

```
def preprocess_text(text):
    # Remove "::" and "_" sequences
    text = text.replace("::", " ")
    text = text.replace("_", " ")

    # Add whitespace after removing sequences
    text = text.replace(":", " ")  # Add whitespace after removing ":" characters

    return text

# Apply preprocessing to the 'Comment' column in your DataFrame
df1['Comment'] = df1['Comment'].apply(preprocess_text)
df1
```

	Comment
0	shut up plebs, dont argue with the $cience, no...
1	sea level rise is already baked in, water fron...
2	and remember, it's your fault,with your soggy ...
3	i never heard so much bullshit in my life…a re...
4	now what?
...	...
12086	in 2022, soylent green will be available. so i...
12087	the earth is warming at an alarming rate, glob...
12088	an idiot show to proof their climate change pr...
12089	kid got this clapping hands clapping hands . 6...
12090	no! your fucking geoengineering program are ~ ...

12091 rows × 1 columns

Spelling Correction

```
[ ] !pip install autocorrect
    from concurrent.futures import ProcessPoolExecutor
    import pandas as pd
    from autocorrect import Speller

    # Initialize the spell checker
    spell = Speller(lang='en')

    # Define the function to correct spellings
    def correct_spellings(text):
        return spell(text)

    # Define a function to process each batch
    def process_batch(batch):
        return batch['Comment'].apply(correct_spellings)

    # Define the number of processes to use
    num_processes = 4  # You can adjust this based on your system's resources

    # Split the DataFrame into smaller batches
    batch_size = len(df1) // num_processes
    batches = [df1.iloc[i:i+batch_size] for i in range(0, len(df1), batch_size)]

    # Process batches in parallel
    with ProcessPoolExecutor(max_workers=num_processes) as executor:
        results = executor.map(process_batch, batches)

    # Combine the results
    df_result = pd.concat(results)

    # Update the 'Comment' column in df1 with the corrected values
    df1['Comment'] = df_result

    # Print the DataFrame after correction
    print(df1)
```

Remove HTML tags and URLs

```
[ ] import re

    def remove_html_tags(text):
        """
        Remove HTML tags from the text.

        Args:
            text (str): Input text with HTML tags.

        Returns:
            str: Text with HTML tags removed.
        """
        # Define HTML tag pattern
        html_tag_pattern = re.compile(r'<.*?>')

        # Remove HTML tags using regular expression
        return re.sub(html_tag_pattern, '', text)

    def remove_urls(text):
        """
        Remove URLs from the text.

        Args:
            text (str): Input text with URLs.

        Returns:
            str: Text with URLs removed.
        """
        # Define URL pattern
        url_pattern = re.compile(r'https?://\S+|www\.\S+')

        # Remove URLs using regular expression
        return re.sub(url_pattern, '', text)

    # Apply the functions to remove HTML tags and URLs from the 'Comment' column in your DataFrame
    df1['Comment'] = df1['Comment'].apply(remove_html_tags)
    df1['Comment'] = df1['Comment'].apply(remove_urls)
    df1
```

	Comment
0	shut up plebs, dont argue with the $cience, no...
1	sea level rise is already baked in. water fron...

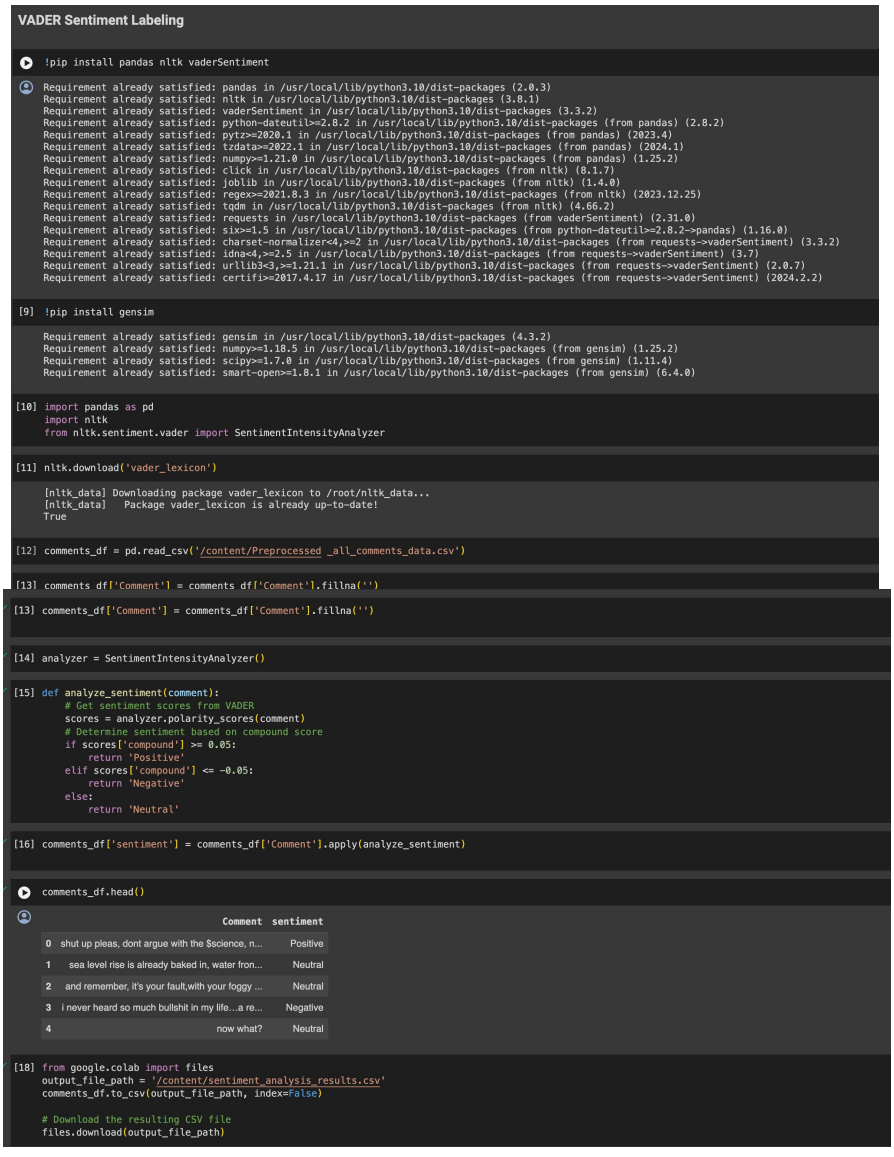
Chat word conversion

```
[ ] import pandas as pd

    # Custom chat words dictionary
    chat_words_dict = {
        "lol": "laugh out loud",
        "brb": "be right back",
        "btw": "by the way",
        "idk": "I don't know",
        "imo": "in my opinion",
        "imho": "in my humble opinion",
        "omg": "oh my god",
        "rofl": "rolling on the floor laughing",
        "ttyl": "talk to you later",
        "wtf": "what the heck",
        "afk": "away from keyboard",
        "bfn": "bye for now",
        "gtg": "got to go",
        "lmao": "laughing my ass off",
        "np": "no problem",
        "ty": "thank you",
        "yw": "you're welcome",
        "bs": "bullshit",
        "b" : "bullshit"
    }

    # Function to expand chat words
    def expand_chat_words(text):
        words = text.split()
        expanded_text = [chat_words_dict.get(word.lower(), word) for word in words]
        return ' '.join(expanded_text)


    # Apply the expand_chat_words function to the 'Comment' column in df1
    df1['Comment'] = df1['Comment'].apply(expand_chat_words)

    # Print the DataFrame after conversion
    print(df1)
```

	Comment
0	shut up plebs, dont argue with the $cience, no...
1	sea level rise is already baked in. water fron...

- **Sentiment Analysis:** The project used three distinct methods for sentiment analysis: VADER, TextBlob, and pattern sentiment labeling. **VADER** is a lexicon and rule-based tool that captures sentiment intensity and nuances in text, allowing for fine-grained analysis of sentiments ranging from highly positive to extremely negative. **TextBlob**, a simpler yet effective approach, uses machine learningalgorithms and linguistic rules to provide polarity scores for text, indicating whether the sentiment expressed is positive, negative, or neutral. **Pattern sentiment labeling**, part of the Pattern library, extends sentiment analysis capabilities by analyzing syntactic structures and linguistic patterns within text. By comparing the outputs of these methods, we gained insights into their strengths and limitations. **VADER excelled in capturing sentiment intensity and nuances in informal text, TextBlob was suitable for quick analyses, and pattern sentiment labeling provided deeper insights into context-specific sentiment expressions.** By triangulating the outputs, we provided a more comprehensive understanding of public sentiment towards climate-focused new energy solutions, enabling informed decision-making and strategic planning.

Vader Sentiments Labelling:



VADER Sentiment Labeling

```
!pip install pandas nltk vaderSentiment
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: vaderSentiment in /usr/local/lib/python3.10/dist-packages (3.3.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.25.2)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from vaderSentiment) (2.31.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (2024.2.2)
```

```
[9] !pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages (4.3.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.25.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.11.4)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from gensim) (6.4.0)
```

```
[10] import pandas as pd
     import nltk
     from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```
[11] nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
True
```

```
[12] comments_df = pd.read_csv('/content/Preprocessed _all_comments_data.csv')
```

```
[13] comments_df['Comment'] = comments_df['Comment'].fillna('')
```

```
[13] comments_df['Comment'] = comments_df['Comment'].fillna('')
```

```
[14] analyzer = SentimentIntensityAnalyzer()
```

```
[15] def analyze_sentiment(comment):
         # Get sentiment scores from VADER
         scores = analyzer.polarity_scores(comment)
         # Determine sentiment based on compound score
         if scores['compound'] >= 0.05:
             return 'Positive'
         elif scores['compound'] <= -0.05:
             return 'Negative'
         else:
             return 'Neutral'
```

```
[16] comments_df['sentiment'] = comments_df['Comment'].apply(analyze_sentiment)
```

```
comments_df.head()
```

	Comment	sentiment
0	shut up pleas, dont argue with the $science, n...	Positive
1	sea level rise is already baked in, water fron...	Neutral
2	and remember, it's your fault,with your foggy ...	Neutral
3	i never heard so much bullshit in my life…a re...	Negative
4	now what?	Neutral

```
[18] from google.colab import files
     output_file_path = '/content/sentiment_analysis_results.csv'
     comments_df.to_csv(output_file_path, index=False)

     # Download the resulting CSV file
     files.download(output_file_path)
```

TextBlob Sentiment Labelling:

TextBlob Sentiment Labeling

```
[19] !pip install textblob
```

```
Requirement already satisfied: textblob in /usr/local/lib/python3.10/dist-packages (0.17.1)
Requirement already satisfied: nltk>=3.1 in /usr/local/lib/python3.10/dist-packages (from textblob) (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (1.4.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (4.66.2)
```

```
[20] import pandas as pd
     from textblob import TextBlob
```

```
# Load the data
data = pd.read_csv('/content/Preprocessed _all_comments_data.csv')
```

```
[ ] data['Comment'] = data['Comment'].fillna('')
```

```
# Define a function to classify comments based on polarity scores
def classify_sentiment(text):
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    if polarity > 0:
        return 'Positive'
    elif polarity < 0:
        return 'Negative'
    else:
        return 'Neutral'

# Apply the function to the dataset and create a new column 'sentiment'
data['Sentiment'] = data['Comment'].apply(classify_sentiment)

# Save the results as a CSV file
data.to_csv('sentiment_analysis_results1.csv', index=False)

print("Sentiment analysis results saved as sentiment_analysis_results1.csv")
```

```
Sentiment analysis results saved as sentiment_analysis_results1.csv
```

Pattern Sentiment Labelling:

Pattern Sentiment Labeling

```
!pip install pattern
```

```
Requirement already satisfied: pattern in /usr/local/lib/python3.10/dist-packages (3.6)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from pattern) (0.18.3)
Requirement already satisfied: backports.csv in /usr/local/lib/python3.10/dist-packages (from pattern) (1.0.7)
Requirement already satisfied: mysqlclient in /usr/local/lib/python3.10/dist-packages (from pattern) (2.2.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from pattern) (4.12.3)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from pattern) (4.9.4)
Requirement already satisfied: feedparser in /usr/local/lib/python3.10/dist-packages (from pattern) (6.0.11)
Requirement already satisfied: pdfminer.six in /usr/local/lib/python3.10/dist-packages (from pattern) (20231228)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pattern) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pattern) (1.11.4)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (from pattern) (3.8.1)
Requirement already satisfied: python-docx in /usr/local/lib/python3.10/dist-packages (from pattern) (1.1.0)
Requirement already satisfied: cherrypy in /usr/local/lib/python3.10/dist-packages (from pattern) (18.9.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from pattern) (2.31.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->pattern) (2.5)
Requirement already satisfied: cheroot>=8.2.1 in /usr/local/lib/python3.10/dist-packages (from cherrypy->pattern) (10.0.1)
Requirement already satisfied: portend>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from cherrypy->pattern) (3.2.0)
Requirement already satisfied: more-itertools in /usr/local/lib/python3.10/dist-packages (from cherrypy->pattern) (10.1.0)
Requirement already satisfied: zc.lockfile in /usr/local/lib/python3.10/dist-packages (from cherrypy->pattern) (3.0.post1)
Requirement already satisfied: jaraco.collections in /usr/local/lib/python3.10/dist-packages (from cherrypy->pattern) (5.0.1)
Requirement already satisfied: sgmllib3k in /usr/local/lib/python3.10/dist-packages (from feedparser->pattern) (1.0.0)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk->pattern) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk->pattern) (1.4.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk->pattern) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk->pattern) (4.66.2)
Requirement already satisfied: charset-normalizer>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from pdfminer.six->pattern) (3.3.2)
Requirement already satisfied: cryptography>=36.0.0 in /usr/local/lib/python3.10/dist-packages (from pdfminer.six->pattern) (42.0.5)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from python-docx->pattern) (4.11.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->pattern) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->pattern) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->pattern) (2024.2.2)
```

```
[ ] # Load the data
    data1 = pd.read_csv('/content/Preprocessed _all_comments_data.csv')

[ ] import pandas as pd
    from pattern.en import sentiment

[ ] data1['Comment'] = data1['Comment'].fillna('')

[ ] def classify_sentiment(text):
        polarity, subjectivity = sentiment(text)
        if polarity > 0:
            return 'Positive'
        elif polarity < 0:
            return 'Negative'
        else:
            return 'Neutral'

    # Apply the function to the dataset and create a new column 'sentiment'
    data1['sentiment'] = data1['Comment'].apply(classify_sentiment)

    data1.head()
```

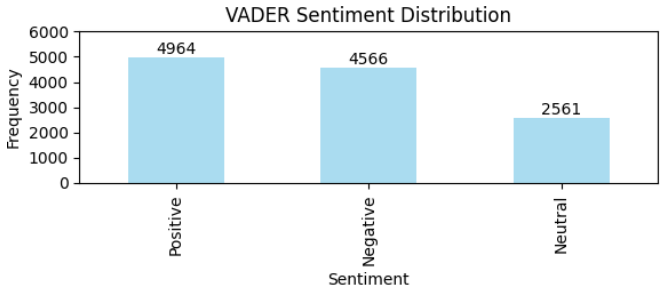
	Comment	sentiment
0	shut up pleas, dont argue with the $science, n...	Neutral
1	sea level rise is already baked in, water fron...	Neutral
2	and remember, it's your fault,with your foggy ...	Negative
3	i never heard so much bullshit in my life...a re...	Negative
4	now what?	Neutral

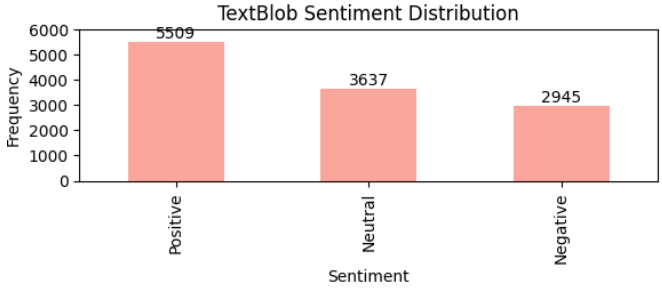
```
[ ] # Save the results as a CSV file
    data1.to_csv('sentiment_analysis_results2.csv', index=False)

    print("Sentiment analysis results saved as sentiment_analysis_results2.csv")

    Sentiment analysis results saved as sentiment_analysis_results2.csv
```

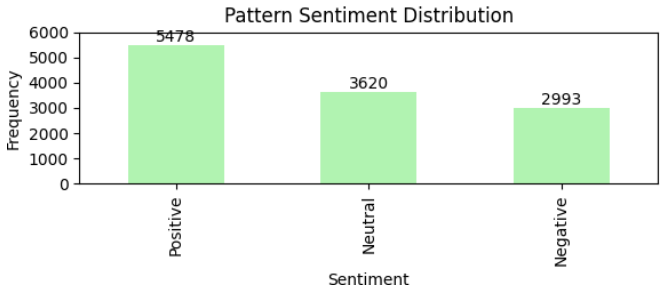
Output and Visualisation:

Plot of Sentiment Label Distribution:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
vader_results = pd.read_csv('sentiment_analysis_results.csv')
textblob_results = pd.read_csv('sentiment_analysis_results1.csv')
pattern_results = pd.read_csv('sentiment_analysis_results2.csv')

# Plot the distribution of sentiment labels
plt.figure(figsize=(6, 8))  # Reduce figure size to fit all plots

# Function to add labels on top of bars
def add_value_labels(ax):
    for p in ax.patches:
        ax.annotate(
            f'{p.get_height():.0f}',  # Display value as an integer
            (p.get_x() + p.get_width() / 2, p.get_height()),  # Position label at the top of the bar
            ha='center',  # Center the label horizontally
            va='bottom'   # Position label above the bar
        )

# Plot VADER sentiment distribution
plt.subplot(3, 1, 1)
vader_counts = vader_results['sentiment'].value_counts()
ax1 = vader_counts.plot(kind='bar', color='skyblue', alpha=0.7)
plt.title('VADER Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Frequency')
plt.ylim(0, 6000)  # Set y-axis range from 0 to 6000
plt.yticks([0, 1000, 2000, 3000, 4000, 5000, 6000])
add_value_labels(ax1)

# Plot TextBlob sentiment distribution
plt.subplot(3, 1, 2)
textblob_counts = textblob_results['Sentiment'].value_counts()
ax2 = textblob_counts.plot(kind='bar', color='salmon', alpha=0.7)
plt.title('TextBlob Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Frequency')
plt.ylim(0, 6000)  # Set y-axis range from 0 to 6000
plt.yticks([0, 1000, 2000, 3000, 4000, 5000, 6000])
add_value_labels(ax2)
```







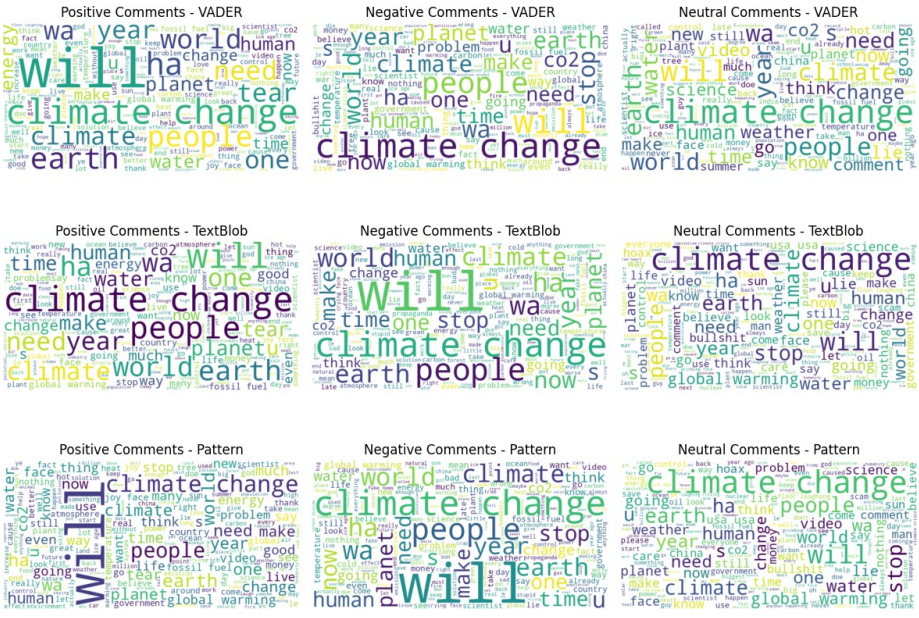
Identification of conflicting sentiment labels:

```python
import pandas as pd

# Load sentiment analysis results from CSV files into DataFrames
vader_results = pd.read_csv('sentiment_analysis_results.csv')
textblob_results = pd.read_csv('sentiment_analysis_results1.csv')
pattern_results = pd.read_csv('sentiment_analysis_results2.csv')

# Identify comments with conflicting sentiment labels between VADER and TextBlob, VADER and Pattern, TextBlob and Pattern
conflicting_comments = []

# Iterate through each row and check for conflicting sentiment labels
for index in range(len(vader_results)):
    # Get sentiments from each method
    vader_sentiment = vader_results.iloc[index]['sentiment']
    textblob_sentiment = textblob_results.iloc[index]['Sentiment']
    pattern_sentiment = pattern_results.iloc[index]['sentiment']

    # Check if there is a conflict between the methods
    if (vader_sentiment != textblob_sentiment) or (vader_sentiment != pattern_sentiment) or (textblob_sentiment != pattern_sentiment):
        # If there's a conflict, store the comment and sentiments
        conflicting_comments.append({
            'Comment': vader_results.iloc[index]['Comment'],
            'VADER Sentiment': vader_sentiment,
            'TextBlob Sentiment': textblob_sentiment,
            'Pattern Sentiment': pattern_sentiment
        })

# Display conflicting comments
print("Conflicting Comments:")
for conflict in conflicting_comments:
    print(f"Comment: {conflict['Comment']}")
    print(f"VADER Sentiment: {conflict['VADER Sentiment']}")
    print(f"TextBlob Sentiment: {conflict['TextBlob Sentiment']}")
    print(f"Pattern Sentiment: {conflict['Pattern Sentiment']}")
    print("----------------------------------------")
```

```
Streaming output truncated to the last 5000 lines.
Comment: are there enough raw material for renewables?
VADER Sentiment: Neutral
TextBlob Sentiment: Negative
Pattern Sentiment: Negative
----------------------------------------
Comment: just stop flying man struggling light skin tone . use zoom or take a train.
VADER Sentiment: Negative
TextBlob Sentiment: Positive
Pattern Sentiment: Positive
----------------------------------------
Comment: we do not have the time with renewables after 20yrs of wind & solar energy failure. climate change and co2 can be reduced quickly
VADER Sentiment: Negative
TextBlob Sentiment: Positive
Pattern Sentiment: Positive
----------------------------------------
Comment: sa code green
VADER Sentiment: Neutral
TextBlob Sentiment: Negative
Pattern Sentiment: Negative
----------------------------------------
Comment: more hymn and catechism from old been on our apparent new global religion of climate change. face with rolling eyes keeping with
VADER Sentiment: Negative
TextBlob Sentiment: Positive
```

Word cloud visualisation:

```python
from wordcloud import WordCloud
import pandas as pd
import matplotlib.pyplot as plt

# Load sentiment analysis results from CSV files into DataFrames
vader_results = pd.read_csv('sentiment_analysis_results.csv')
textblob_results = pd.read_csv('sentiment_analysis_results1.csv')
pattern_results = pd.read_csv('sentiment_analysis_results2.csv')

# Filter comments based on sentiment labels for VADER
positive_vader_comments = vader_results[vader_results['sentiment'] == 'Positive']['Comment'].dropna().str.cat(sep=' ')
negative_vader_comments = vader_results[vader_results['sentiment'] == 'Negative']['Comment'].dropna().str.cat(sep=' ')
neutral_vader_comments = vader_results[vader_results['sentiment'] == 'Neutral']['Comment'].dropna().str.cat(sep=' ')

# Filter comments based on sentiment labels for TextBlob
positive_textblob_comments = textblob_results[textblob_results['Sentiment'] == 'Positive']['Comment'].dropna().str.cat(sep=' ')
negative_textblob_comments = textblob_results[textblob_results['Sentiment'] == 'Negative']['Comment'].dropna().str.cat(sep=' ')
neutral_textblob_comments = textblob_results[textblob_results['Sentiment'] == 'Neutral']['Comment'].dropna().str.cat(sep=' ')

# Filter comments based on sentiment labels for Pattern
positive_pattern_comments = pattern_results[pattern_results['sentiment'] == 'Positive']['Comment'].dropna().str.cat(sep=' ')
negative_pattern_comments = pattern_results[pattern_results['sentiment'] == 'Negative']['Comment'].dropna().str.cat(sep=' ')
neutral_pattern_comments = pattern_results[pattern_results['sentiment'] == 'Neutral']['Comment'].dropna().str.cat(sep=' ')

# Generate word clouds for VADER results
wordcloud_positive_vader = WordCloud(width=800, height=400, background_color='white').generate(positive_vader_comments)
wordcloud_negative_vader = WordCloud(width=800, height=400, background_color='white').generate(negative_vader_comments)
wordcloud_neutral_vader = WordCloud(width=800, height=400, background_color='white').generate(neutral_vader_comments)
```

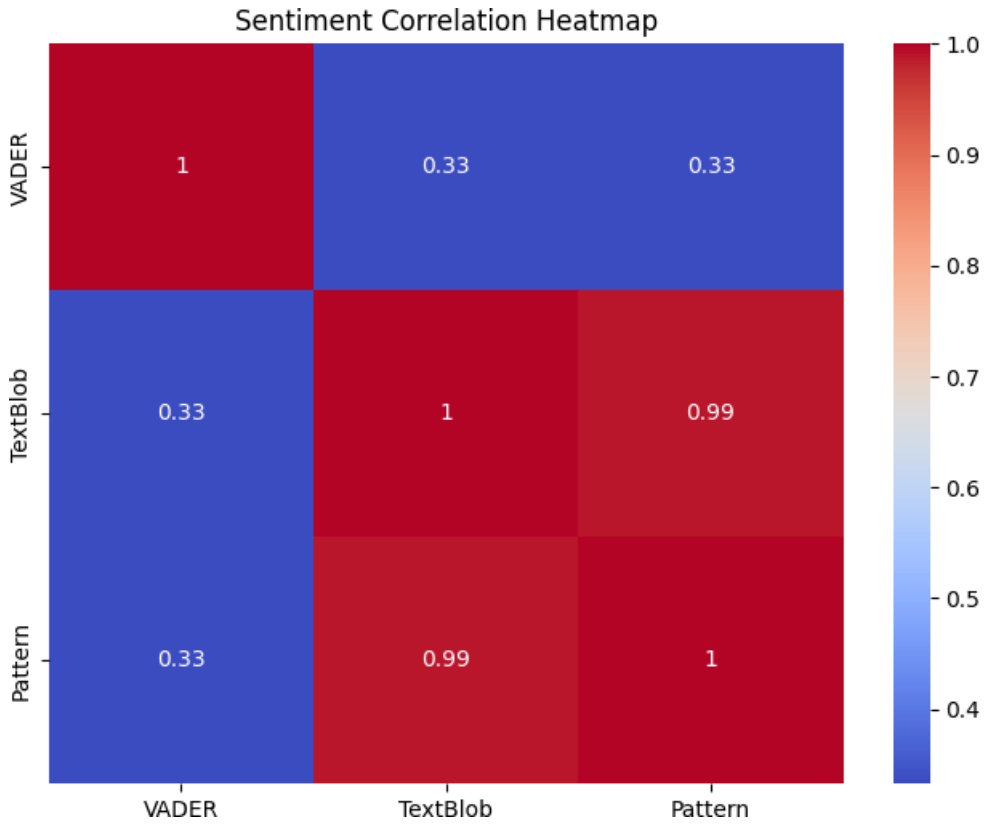


Sentiment correlation Analysis:

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load sentiment analysis results from CSV files into DataFrames
vader_results = pd.read_csv('sentiment_analysis_results.csv')
textblob_results = pd.read_csv('sentiment_analysis_results1.csv')
pattern_results = pd.read_csv('sentiment_analysis_results2.csv')

# Map sentiment labels to numerical scores
sentiment_mapping = {'Positive': 1, 'Negative': -1, 'Neutral': 0}
vader_results['sentiment_score'] = vader_results['sentiment'].map(sentiment_mapping)
textblob_results['sentiment_score'] = textblob_results['Sentiment'].map(sentiment_mapping)
pattern_results['sentiment_score'] = pattern_results['sentiment'].map(sentiment_mapping)

# Merge DataFrames on a common column (e.g., index or another column)
merged_results = pd.concat([vader_results['sentiment_score'], textblob_results['sentiment_score'], pattern_results['sentiment_score']], axis=1)
merged_results.columns = ['VADER', 'TextBlob', 'Pattern']

# Calculate correlation matrix
correlation_matrix = merged_results.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Sentiment Correlation Heatmap')
plt.show()
```

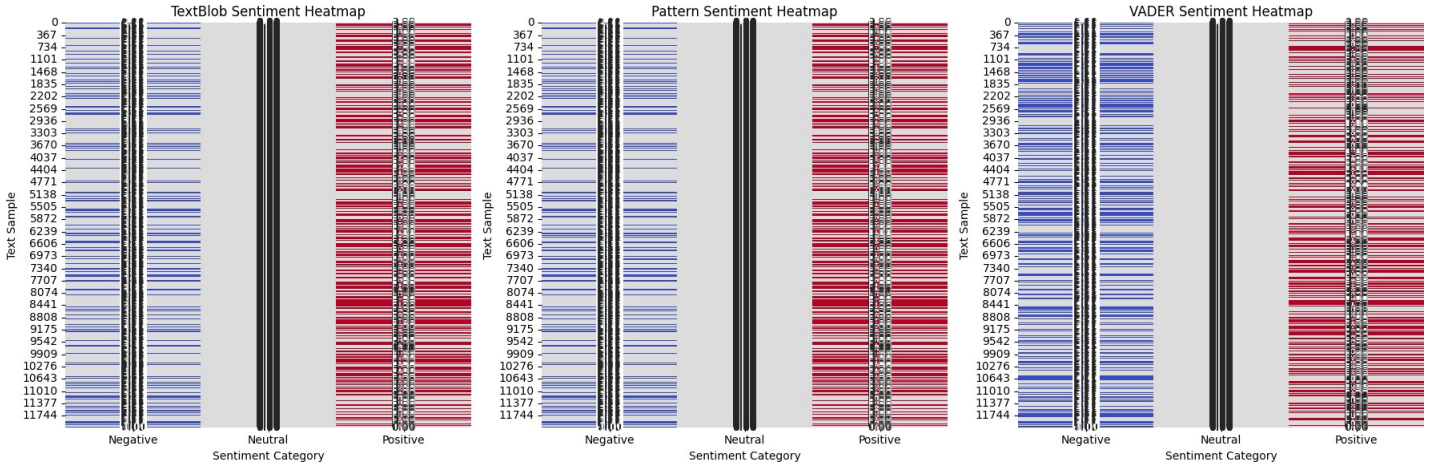
(variable) pattern_results: DataFrame

```
Correlation Matrix:
             VADER  TextBlob   Pattern
VADER     1.000000  0.334421  0.332634
TextBlob  0.334421  1.000000  0.987246
Pattern   0.332634  0.987246  1.000000
```



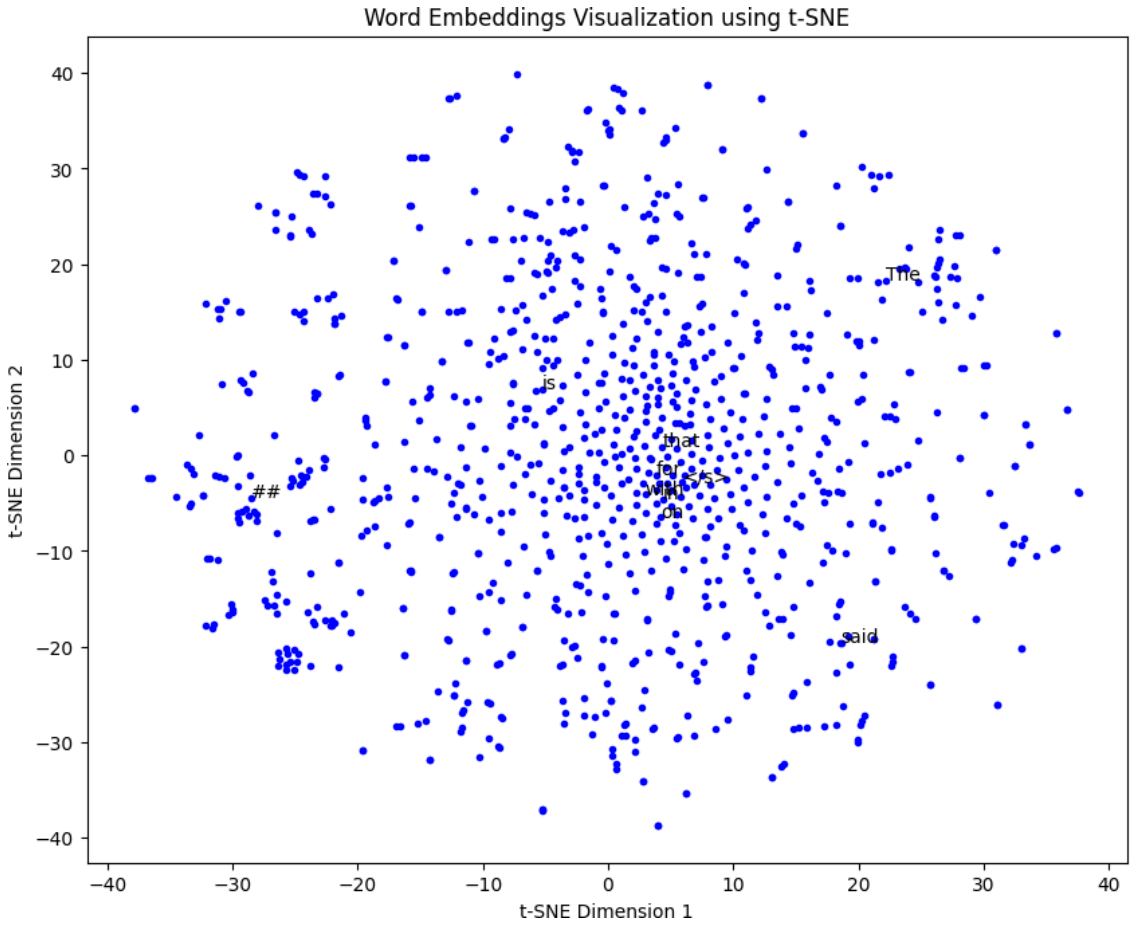
Heatmaps of Sentiment Analysis:

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load TextBlob, Pattern, and VADER sentiment analysis results from CSV files into DataFrames
textblob_results = pd.read_csv('sentiment_analysis_results1.csv')
pattern_results = pd.read_csv('sentiment_analysis_results2.csv')
vader_results = pd.read_csv('sentiment_analysis_results.csv')

# Map TextBlob sentiment labels to numerical scores
sentiment_mapping_textblob = {'Positive': 1, 'Negative': -1, 'Neutral': 0}
textblob_results['sentiment_score'] = textblob_results['Sentiment'].map(sentiment_mapping_textblob)

# Map Pattern sentiment labels to numerical scores
sentiment_mapping_pattern = {'Positive': 1, 'Negative': -1, 'Neutral': 0}
pattern_results['sentiment_score'] = pattern_results['sentiment'].map(sentiment_mapping_pattern)

# Map VADER sentiment labels to numerical scores
sentiment_mapping_vader = {'Positive': 1, 'Negative': -1, 'Neutral': 0}
vader_results['sentiment_score'] = vader_results['sentiment'].map(sentiment_mapping_vader)

# Create pivot tables for TextBlob, Pattern, and VADER results
textblob_pivot = textblob_results.pivot_table(index=textblob_results.index, columns='Sentiment', values='sentiment_score', fill_value=0)
pattern_pivot = pattern_results.pivot_table(index=pattern_results.index, columns='sentiment', values='sentiment_score', fill_value=0)
vader_pivot = vader_results.pivot_table(index=vader_results.index, columns='sentiment', values='sentiment_score', fill_value=0)

# Create a figure for the heatmaps
plt.figure(figsize=(18, 6))

# Plot TextBlob heatmap
if not textblob_pivot.empty:
    plt.subplot(1, 3, 1)
    sns.heatmap(textblob_pivot, cmap="coolwarm", annot=True, fmt=".2f", cbar=False)
    plt.title('TextBlob Sentiment Heatmap')
    plt.xlabel('Sentiment Category')
    plt.ylabel('Text Sample')

# Plot Pattern heatmap
if not pattern_pivot.empty:
    plt.subplot(1, 3, 2)
    sns.heatmap(pattern_pivot, cmap="coolwarm", annot=True, fmt=".2f", cbar=False)
    plt.title('Pattern Sentiment Heatmap')
    plt.xlabel('Sentiment Category')
    plt.ylabel('Text Sample')
```



Visualising Word Embeddings with T-SNE:

```python
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import gensim.downloader as api

# Load the pre-trained Word2Vec model
model = api.load("word2vec-google-news-300")

# Extract a subset of word embeddings and corresponding words
subset_size = 1000
words = model.index_to_key[:subset_size]
embeddings = np.array([model[word] for word in words])

# Apply t-SNE to reduce the dimensionality of word embeddings
tsne = TSNE(n_components=2, random_state=42)
embeddings_2d = tsne.fit_transform(embeddings)

# Plot word embeddings in 2D space
plt.figure(figsize=(10, 8))
plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1], marker='.', color='b')

# Add annotations for a subset of words
num_words_to_annotate = 10
for i in range(num_words_to_annotate):
    plt.annotate(words[i], (embeddings_2d[i, 0], embeddings_2d[i, 1]))

plt.title('Word Embeddings Visualization using t-SNE')
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.show()
```



Conclusion :

This project on sentiment analysis of climate-focused new energy solutions has provided valuable insights into public perception and sentiment towards these initiatives. Using VADER, TextBlob, and pattern sentiment labeling methods,we analyzed textual data to identify trends, patterns, and areas of concern or support. The plots of sentiment label distribution helped visualize sentiment proportions. The findings emphasize the importance of sentiment analysis in gauging public opinion and guiding decision-making processes related to climate-focused new energy solutions. Policymakers, businesses, andadvocates can refine strategies, tailor communications, and drive initiatives that resonate with public sentiment, ultimately advancing the transition towards a more sustainable andresilient future. Continued monitoring and analysis of sentiment trends will be essential for adapting strategies in response to evolving public perceptions and attitudes.