```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import re
from collections import Counter
import nltk
from nltk.tokenize import word_tokenize

        #1.1
# Load CSV files
trump_tweets = pd.read_csv('data/trump_encoded.csv')
clinton_tweets = pd.read_csv('data/clinton_encoded.csv')
# Add candidate column
trump_tweets['candidate'] = 'Donald Trump'
clinton_tweets['candidate'] = 'Hillary Clinton'
# Combine DataFrames
combined_df = pd.concat([trump_tweets, clinton_tweets], ignore_index=True)
# Display the first few rows to verify
combined_df.head()
    #1.2 (first downloading the lexicon and cleaning data)
!wget https://raw.githubusercontent.com/aditeyabaral/lok-sabha-election-twitter-analysis/master/NRC-Emotion-Lexicon-
Wordlevel-v0.92.txt

nrc_lexicon = pd.read_csv("NRC-Emotion-Lexicon-Wordlevel-v0.92.txt", sep="      ", header=None, names=["word", "emotion",
"association"])
nrc_lexicon = nrc_lexicon[nrc_lexicon["association"] == 1].drop(columns=["association"])
print(nrc_lexicon.head())

# Function to clean text
def clean_text(text):
    # Remove mentions (@username) and hashtags (#hashtag)
    text = re.sub(r'@[\w]+', '', text)  # Remove mentions
    text = re.sub(r'#[\w]+', '', text)  # Remove hashtags
    # Remove non-alphabetical characters (punctuation, numbers, etc.)
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Only keep alphabets and spaces
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()
    return text

# Download the 'punkt' tokenizer and 'punkt_tab'
nltk.download('punkt')
nltk.download('punkt_tab')
# 1.2.1 Create a Sentiment Analysis Function
    # Fix the sentiment analysis function and apply it to cleaned text
def get_sentiment_counts(tweet_text, lexicon):
    # Tokenize the tweet text into words
    tokens = word_tokenize(str(tweet_text).lower())  # Lowercasing for matching

    # Initialize a dictionary to store sentiment counts
    sentiment_counts = {emotion: 0 for emotion in lexicon['emotion'].unique()}

    # Check each token in the tweet text
    for token in tokens:
        # Check if the token is in the NRC Emotion Lexicon
        emotion_matches = lexicon[lexicon['word'] == token]
        # For each matching emotion, increment the corresponding count
        for _, row in emotion_matches.iterrows():
            sentiment_counts[row['emotion']] += 1  # Increment the count for the emotion

    return sentiment_counts

# 1.3.1 Group by candidate and generate summary statistics for each emotion category
# Load data
trump_tweets = pd.read_csv('data/trump_encoded.csv')
clinton_tweets = pd.read_csv('data/clinton_encoded.csv')
# Add candidate column
trump_tweets['candidate'] = 'Donald Trump'
clinton_tweets['candidate'] = 'Hillary Clinton'
# Combine DataFrames
combined_df = pd.concat([trump_tweets, clinton_tweets], ignore_index=True)
# Clean text and perform sentiment analysis
def clean_text(text):
    # Remove mentions (@username) and hashtags (#hashtag)
    text = re.sub(r'@[\w]+', '', text)  # Remove mentions
    text = re.sub(r'#[\w]+', '', text)  # Remove hashtags
    # Remove non-alphabetical characters (punctuation, numbers, etc.)
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Only keep alphabets and spaces
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip().lower()
    return text
# Load lexicon
nrc_lexicon = pd.read_csv("NRC-Emotion-Lexicon-Wordlevel-v0.92.txt",
                          sep="  ",
                          header=None,
                          names=["word", "emotion", "association"])
nrc_lexicon = nrc_lexicon[nrc_lexicon["association"] == 1].drop(columns=["association"])
def get_sentiment_counts(tweet_text, lexicon):
    # Clean the text
    cleaned_text = clean_text(str(tweet_text))

    # Use simple split as the TA confirmed is acceptable
```

```
    words = cleaned_text.split()

    # Initialize counts for each emotion
    sentiment_counts = {emotion: 0 for emotion in lexicon['emotion'].unique()}

    # Count emotions for each word, allowing duplicates
    for word in words:
        # Find matching emotions for this word
        matches = lexicon[lexicon['word'] == word]
        for _, row in matches.iterrows():
            sentiment_counts[row['emotion']] += 1

    return sentiment_counts
# First, create a cleaned text column
combined_df['cleaned_text'] = combined_df['text'].apply(clean_text)
# Apply the sentiment analysis function to cleaned text
sentiment_counts = combined_df['cleaned_text'].apply(lambda tweet: get_sentiment_counts(tweet, nrc_lexicon))
# Convert the dictionary of sentiment counts into separate columns
sentiment_df = pd.DataFrame(list(sentiment_counts))
# Merge the sentiment columns with the original DataFrame
merged_df_with_sentiments = pd.concat([combined_df, sentiment_df], axis=1)
# Define emotion categories
emotion_categories = ['anger', 'anticipation', 'disgust', 'fear', 'joy',
                      'sadness', 'surprise', 'trust', 'positive', 'negative']
# 1.3.1 Group by candidate and generate summary statistics for each emotion
print("1.3.1 Summary statistics for emotions by candidate:")
emotion_stats_by_candidate = merged_df_with_sentiments.groupby('candidate')[emotion_categories].describe()
print(emotion_stats_by_candidate)

# 1.3.2 Group by candidate and each emotion category, then compute summary statistics
# Load data
trump_tweets = pd.read_csv('data/trump_encoded.csv')
clinton_tweets = pd.read_csv('data/clinton_encoded.csv')
# Add candidate column
trump_tweets['candidate'] = 'Donald Trump'
clinton_tweets['candidate'] = 'Hillary Clinton'
# Combine DataFrames
combined_df = pd.concat([trump_tweets, clinton_tweets], ignore_index=True)
# Clean text function
def clean_text(text):
    # Remove mentions (@username) and hashtags (#hashtag)
    text = re.sub(r'@[\w]+', '', text)  # Remove mentions
    text = re.sub(r'#[\w]+', '', text)  # Remove hashtags
    # Remove non-alphabetical characters (punctuation, numbers, etc.)
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Only keep alphabets and spaces
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip().lower()
    return text
# Load lexicon
nrc_lexicon = pd.read_csv("NRC-Emotion-Lexicon-Wordlevel-v0.92.txt",
                          sep="  ",
                          header=None,
                          names=["word", "emotion", "association"])
nrc_lexicon = nrc_lexicon[nrc_lexicon["association"] == 1].drop(columns=["association"])
def get_sentiment_counts(tweet_text, lexicon):
    # Clean the text
    cleaned_text = clean_text(str(tweet_text))

    # Use simple split as the TA confirmed is acceptable
    words = cleaned_text.split()

    # Initialize counts for each emotion
    sentiment_counts = {emotion: 0 for emotion in lexicon['emotion'].unique()}

    # Count emotions for each word, allowing duplicates
    for word in words:
        # Find matching emotions for this word
        matches = lexicon[lexicon['word'] == word]
        for _, row in matches.iterrows():
            sentiment_counts[row['emotion']] += 1

    return sentiment_counts
# First, create a cleaned text column
combined_df['cleaned_text'] = combined_df['text'].apply(clean_text)
# Apply the sentiment analysis function to cleaned text
sentiment_counts = combined_df['cleaned_text'].apply(lambda tweet: get_sentiment_counts(tweet, nrc_lexicon))
# Convert the dictionary of sentiment counts into separate columns
sentiment_df = pd.DataFrame(list(sentiment_counts))
# Merge the sentiment columns with the original DataFrame
merged_df_with_sentiments = pd.concat([combined_df, sentiment_df], axis=1)
# List of all emotion categories
emotion_categories = ['anger', 'anticipation', 'disgust', 'fear', 'joy',
                      'sadness', 'surprise', 'trust', 'positive', 'negative']
# 1.3.2 Group by candidate and emotion category, compute summary stats for engagement
print("1.3.2 Summary statistics for engagement metrics by candidate and emotion:")
engagement_metrics = ['favorite_count', 'retweet_count']
# Analyze all emotions, but highlight two selected ones for deeper analysis
for emotion in emotion_categories:
    # Create bins for emotion counts
    merged_df_with_sentiments[f'{emotion}_level'] = pd.cut(
        merged_df_with_sentiments[emotion],
        bins=[-1, 0, 2, float('inf')],
        labels=['None', 'Low', 'High']
    )
```

```
    # Group by candidate and emotion level, calculate engagement stats
    stats = merged_df_with_sentiments.groupby(['candidate', f'{emotion}_level'])[engagement_metrics].describe()
    print(f"
Engagement metrics for {emotion}:")
    print(stats)
# Based on the analysis, select two emotions for visualization in 1.4
selected_emotions = ['fear', 'disgust']
print(f"
Selected emotions for visualization in 1.4: {selected_emotions}")
print("1. Fear: Selected because it showed significant differences between candidates' tweets.")
print("2. Disgust: Selected as a complementary negative emotion that reveals interesting patterns in engagement metrics.")

# Part 2: Correlation Analysis
# 2.1 Research Question: Is there a relationship between emotion categories
# and the number of retweets or favorites?
# This code assumes that parts 1.1-1.4 have been run first
# and that the merged_df_with_sentiments variable exists
# Define emotion categories and engagement metrics
emotion_categories = ['anger', 'anticipation', 'disgust', 'fear', 'joy',
                      'sadness', 'surprise', 'trust', 'positive', 'negative']
engagement_metrics = ['favorite_count', 'retweet_count']
# Calculate correlation between emotion counts and engagement metrics
print("
Research Question: Is there a relationship between emotion categories and engagement?")
correlation_df = merged_df_with_sentiments[emotion_categories + engagement_metrics].corr()
# Extract the correlations between emotions and engagement metrics
emotion_engagement_corr = correlation_df.loc[emotion_categories, engagement_metrics]
print("
Correlation between emotions and engagement metrics:")
print(emotion_engagement_corr)
# Create a heatmap to visualize the correlations
plt.figure(figsize=(10, 8))
sns.heatmap(emotion_engagement_corr, annot=True, cmap='coolwarm', vmin=-0.2, vmax=0.2)
plt.title('Correlation between Emotions and Engagement Metrics', fontsize=16)
plt.tight_layout()
plt.savefig('output/emotion_engagement_correlation.png')
# Analyze correlations separately for each candidate
fig, axes = plt.subplots(1, 2, figsize=(20, 8))
candidates = merged_df_with_sentiments['candidate'].unique()
for i, candidate in enumerate(candidates):
    # Filter data for this candidate
    candidate_df = merged_df_with_sentiments[merged_df_with_sentiments['candidate'] == candidate]

    # Calculate correlations
    candidate_corr = candidate_df[emotion_categories + engagement_metrics].corr()
    candidate_emotion_engagement = candidate_corr.loc[emotion_categories, engagement_metrics]

    # Plot heatmap
    sns.heatmap(candidate_emotion_engagement,
                annot=True,
                cmap='coolwarm',
                vmin=-0.2,
                vmax=0.2,
                ax=axes[i])
    axes[i].set_title(f'Correlations for {candidate}', fontsize=14)
plt.tight_layout()
plt.savefig('output/emotion_engagement_correlation_by_candidate.png')
"""
The heatmap analysis reveals slight relationships between emotional expressions in tweets and engagement, specifically
favorite and retweet
counts. Tweets expressing emotions such as disgust and negativity are slightly slightly correlated with higher engagement,
suggesting
people may respond more actively to emotionally charged content, specifically negatively charged. Tweets characterized by
anticipation, joy,
surprise, or positive emotions exhibit minor negative correlations, implying these emotions might be marginally less
effective at driving
engagement. Emotions such as anger, fear, sadness, and trust don't seem to have an impact on engagement. Overall, while these
correlations
are small, they indicate that tweets with a negative or intense sentiment, particularly with emotions of disgust, may be
somewhat more
effective in eliciting likes and retweets.
"""

# 1.4 Visualizing Results
# Load data
trump_tweets = pd.read_csv('data/trump_encoded.csv')
clinton_tweets = pd.read_csv('data/clinton_encoded.csv')

# Add candidate column
trump_tweets['candidate'] = 'Donald Trump'
clinton_tweets['candidate'] = 'Hillary Clinton'

# Combine DataFrames
combined_df = pd.concat([trump_tweets, clinton_tweets], ignore_index=True)

# Clean text function
def clean_text(text):
    # Remove mentions (@username) and hashtags (#hashtag)
    text = re.sub(r'@[\w]+', '', text)  # Remove mentions
    text = re.sub(r'#[\w]+', '', text)  # Remove hashtags
    # Remove non-alphabetical characters (punctuation, numbers, etc.)
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Only keep alphabets and spaces
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip().lower()
    return text
```

```
# Load lexicon
nrc_lexicon = pd.read_csv("NRC-Emotion-Lexicon-Wordlevel-v0.92.txt",
                          sep="  ",
                          header=None,
                          names=["word", "emotion", "association"])
nrc_lexicon = nrc_lexicon[nrc_lexicon["association"] == 1].drop(columns=["association"])

def get_sentiment_counts(tweet_text, lexicon):
    # Clean the text
    cleaned_text = clean_text(str(tweet_text))

    # Use simple split as the TA confirmed is acceptable
    words = cleaned_text.split()

    # Initialize counts for each emotion
    sentiment_counts = {emotion: 0 for emotion in lexicon['emotion'].unique()}

    # Count emotions for each word, allowing duplicates
    for word in words:
        # Find matching emotions for this word
        matches = lexicon[lexicon['word'] == word]
        for _, row in matches.iterrows():
            sentiment_counts[row['emotion']] += 1

    return sentiment_counts

# First, create a cleaned text column
combined_df['cleaned_text'] = combined_df['text'].apply(clean_text)

# Apply the sentiment analysis function to cleaned text
sentiment_counts = combined_df['cleaned_text'].apply(lambda tweet: get_sentiment_counts(tweet, nrc_lexicon))

# Convert the dictionary of sentiment counts into separate columns
sentiment_df = pd.DataFrame(list(sentiment_counts))

# Merge the sentiment columns with the original DataFrame
merged_df_with_sentiments = pd.concat([combined_df, sentiment_df], axis=1)

# Selected emotions for visualization based on 1.3.1 and 1.3.2 analysis
selected_emotions = ['fear', 'disgust']
print(f"Visualizing engagement metrics for selected emotions: {selected_emotions}")

# Create a figure with 2 subplots
fig, axes = plt.subplots(1, 2, figsize=(18, 6))

# Define explicit colors for candidates
clinton_color = 'blue'
trump_color = 'red'

# Group the data by candidate and calculate total engagement metrics for tweets with these emotions
engagement_by_candidate = {}

for emotion in selected_emotions:
    # Create empty DataFrames to store results
    likes_data = pd.DataFrame(columns=['candidate', 'count'])
    retweets_data = pd.DataFrame(columns=['candidate', 'count'])

    # Calculate engagement metrics for each candidate
    for candidate in merged_df_with_sentiments['candidate'].unique():
        # Get tweets for this candidate that have this emotion
        candidate_tweets = merged_df_with_sentiments[merged_df_with_sentiments['candidate'] == candidate]
        emotion_tweets = candidate_tweets[candidate_tweets[emotion] > 0]

        # Calculate total likes and retweets
        total_likes = emotion_tweets['favorite_count'].sum()
        total_retweets = emotion_tweets['retweet_count'].sum()

        # Store in DataFrames
        likes_data = pd.concat([likes_data, pd.DataFrame({'candidate': [candidate], 'count': [total_likes]})],
ignore_index=True)
        retweets_data = pd.concat([retweets_data, pd.DataFrame({'candidate': [candidate], 'count': [total_retweets]})],
ignore_index=True)

    # Store data for this emotion
    engagement_by_candidate[emotion] = {
        'likes': likes_data,
        'retweets': retweets_data
    }

# Plot likes for selected emotions
x_pos = np.arange(len(selected_emotions))
width = 0.35

# Plot likes for each candidate
clinton_likes = [engagement_by_candidate[emotion]['likes'][engagement_by_candidate[emotion]['likes']['candidate'] == 'Hillary
Clinton']['count'].values[0] for emotion in selected_emotions]
trump_likes = [engagement_by_candidate[emotion]['likes'][engagement_by_candidate[emotion]['likes']['candidate'] == 'Donald
Trump']['count'].values[0] for emotion in selected_emotions]

axes[0].bar(x_pos - width/2, clinton_likes, width, label='Hillary Clinton', color=clinton_color)
axes[0].bar(x_pos + width/2, trump_likes, width, label='Donald Trump', color=trump_color)
axes[0].set_title('Total Likes by Emotion Category', fontsize=14)
axes[0].set_ylabel('Total Likes', fontsize=12)
axes[0].set_xticks(x_pos)
```

```
axes[0].set_xticklabels(selected_emotions)
axes[0].legend()

# Plot retweets for each candidate
clinton_retweets = [engagement_by_candidate[emotion]['retweets'][engagement_by_candidate[emotion]['retweets']['candidate'] ==
'Hillary Clinton']['count'].values[0] for emotion in selected_emotions]
trump_retweets = [engagement_by_candidate[emotion]['retweets'][engagement_by_candidate[emotion]['retweets']['candidate'] ==
'Donald Trump']['count'].values[0] for emotion in selected_emotions]

axes[1].bar(x_pos - width/2, clinton_retweets, width, label='Hillary Clinton', color=clinton_color)
axes[1].bar(x_pos + width/2, trump_retweets, width, label='Donald Trump', color=trump_color)
axes[1].set_title('Total Retweets by Emotion Category', fontsize=14)
axes[1].set_ylabel('Total Retweets', fontsize=12)
axes[1].set_xticks(x_pos)
axes[1].set_xticklabels(selected_emotions)
axes[1].legend()

# Adjust layout and save
plt.tight_layout()
plt.savefig('output/emotion_engagement_comparison.png')
print("Visualization completed and saved to output/emotion_engagement_comparison.png")

# Part 3: Open Ended Exploration - Hashtag Analysis
# This exploration examines the hashtags used by Trump and Clinton during the election,
# testing the hypothesis that Trump used more actionable and negative hashtags than Clinton,
# and analyzing the emotional content and engagement of tweets containing these hashtags.

# Research question: Did Trump use more actionable and negative hashtags than Clinton?
# Analysis method: Frequency analysis of hashtags, categorization of hashtag types,
# emotion profile analysis, and engagement metrics comparison.

# Note:
# - Fixed 'mcincle' to 'rncincle' in hashtag extraction
# - Combined 'maga' and 'makeamericagreatagain' as one hashtag theme
# - Combined 'votetrump' and 'trump2016' as one hashtag theme
# - Categorized hashtags as 'Event-related', 'Action-oriented', 'Negative/Attack', or 'Other'

# Extract hashtags from original tweets
def extract_hashtags(text):
    # Use regex to find all hashtags
    hashtags = re.findall(r'#(\w+)', str(text))

    # Normalize specific hashtags
    normalized = []
    for tag in hashtags:
        tag = tag.lower()
        # Fix mcincle to rncincle
        if tag == 'mcincle':
            normalized.append('rncincle')
        else:
            normalized.append(tag)

    return normalized

# Normalize and combine similar hashtags
def normalize_hashtags(hashtags):
    normalized = []
    for tag in hashtags:
        # Combine MAGA-related hashtags
        if tag in ['maga', 'makeamericagreatagain']:
            normalized.append('maga/makeamericagreatagain')
        # Combine Trump campaign hashtags
        elif tag in ['votetrump', 'trump2016']:
            normalized.append('votetrump/trump2016')
        else:
            normalized.append(tag)
    return normalized

# Function to get hashtag emotion data - for combined hashtags
def get_hashtag_emotions(df, hashtag):
    # For combined hashtags, split and check for either one
    if '/' in hashtag:
        tag1, tag2 = hashtag.split('/')
        mask = (df['text'].str.contains(f'#{tag1}', case=False, na=False) |
                df['text'].str.contains(f'#{tag2}', case=False, na=False))
    else:
        mask = df['text'].str.contains(f'#{hashtag}', case=False, na=False)

    hashtag_tweets = df[mask]

    if len(hashtag_tweets) == 0:
        return None

    # Calculate average emotion scores
    emotion_scores = hashtag_tweets[emotion_categories].mean()

    # Calculate average engagement
    engagement = {
        'favorite_count': hashtag_tweets['favorite_count'].mean(),
        'retweet_count': hashtag_tweets['retweet_count'].mean(),
        'tweet_count': len(hashtag_tweets)
    }

    return pd.Series({**emotion_scores, **engagement})
```

```
# Categorize hashtags
def categorize_hashtag(tag):
    if tag in event_hashtags:
        return 'Event-related'
    elif tag in action_hashtags:
        return 'Action-oriented'
    elif tag in negative_hashtags:
        return 'Negative/Attack'
    else:
        return 'Other'

# Analysis:
"""
Though It would've been intriguing to see whether or not there was any interaction between engagement levels and the expected outcome of the election, I felt it was flawed for a multitude of reasons. The First of that is that the sample is way too small. Sure there are 12000 lines of tweets, but to truly understand whether or not it is a predictor of elections (something that already changes so much in the 4 years between elections) you would need a much larger sample and that's limited to just 4 elections, one of which, was while it was still a relatively small app. Instead of analyzing election outcomes, I decided to focus on whether or not there was an increased engagement with certain hashtags, which hashtags were used the most by each set of candidate related tweets, and the emotion surrounding a given hashtag. From the 6 charts (3 types, 1 for each candidate) I put together I was able to make 3 conclusions. The first is that Hillary related tweets used almost exclusively event-related hashtags, with the exception of "#imwithher" and "#tbt," which is in extreme contrast to Trump related tweets who used more actionable and targeted hashtags, like "#crookedhillary," "#draintheswamp," and "#americafirst" all of which spoke down on Hillary's campaign. The second conclusion I was able to make was that Hillary related tweets with hashtags were generally much more emotional than Trump related tweets. The final conclusion is that Trump's choice to shift toward more targeted and actionable messaging generally worked from a marketing and engagement perspective on social media as we see with the tweets. Trump related tweets across the board had more engagement with his most used hashtags than Hillary, having significantly higher favorite and retweet counts.
"""

##############################################
# === Part 3 Exploration #1 (Paolo) ===
##############################################

# This exploration examines the hashtags used by Trump and Clinton during the election,
# testing the hypothesis that Trump used more actionable and negative hashtags than Clinton,
# and analyzing the emotional content and engagement of tweets containing these hashtags.

# Research question: Did Trump use more actionable and negative hashtags than Clinton?
# Analysis method: Frequency analysis of hashtags, categorization of hashtag types,
# emotion profile analysis, and engagement metrics comparison.

# Note:
# - Fixed 'mcincle' to 'rncincle' in hashtag extraction
# - Combined 'maga' and 'makeamericagreatagain' as one hashtag theme
# - Combined 'votetrump' and 'trump2016' as one hashtag theme
# - Categorized hashtags as 'Event-related', 'Action-oriented', 'Negative/Attack', or 'Other'

# Extract hashtags from original tweets
def extract_hashtags(text):
    # Use regex to find all hashtags
    hashtags = re.findall(r'#(\w+)', str(text))

    # Normalize specific hashtags
    normalized = []
    for tag in hashtags:
        tag = tag.lower()
        # Fix mcincle to rncincle
        if tag == 'mcincle':
            normalized.append('rncincle')
        else:
            normalized.append(tag)

    return normalized

# Normalize and combine similar hashtags
def normalize_hashtags(hashtags):
    normalized = []
    for tag in hashtags:
        # Combine MAGA-related hashtags
        if tag in ['maga', 'makeamericagreatagain']:
            normalized.append('maga/makeamericagreatagain')
        # Combine Trump campaign hashtags
        elif tag in ['votetrump', 'trump2016']:
            normalized.append('votetrump/trump2016')
        else:
            normalized.append(tag)
    return normalized

# Function to get hashtag emotion data - for combined hashtags
def get_hashtag_emotions(df, hashtag):
    # For combined hashtags, split and check for either one
    if '/' in hashtag:
        tag1, tag2 = hashtag.split('/')
        mask = (df['text'].str.contains(f'#{tag1}', case=False, na=False) |
                df['text'].str.contains(f'#{tag2}', case=False, na=False))
    else:
        mask = df['text'].str.contains(f'#{hashtag}', case=False, na=False)

    hashtag_tweets = df[mask]

    if len(hashtag_tweets) == 0:
        return None

    # Calculate average emotion scores
```

```
    emotion_scores = hashtag_tweets[emotion_categories].mean()

    # Calculate average engagement
    engagement = {
        'favorite_count': hashtag_tweets['favorite_count'].mean(),
        'retweet_count': hashtag_tweets['retweet_count'].mean(),
        'tweet_count': len(hashtag_tweets)
    }

    return pd.Series({**emotion_scores, **engagement})

# Categorize hashtags
def categorize_hashtag(tag):
    if tag in event_hashtags:
        return 'Event-related'
    elif tag in action_hashtags:
        return 'Action-oriented'
    elif tag in negative_hashtags:
        return 'Negative/Attack'
    else:
        return 'Other'

# Analysis:
"""
Though It would've been intriguing to see whether or not there was any interaction between engagement levels and the expected
outcome of the election, I felt it was flawed for a multitude of reasons. The First of that is that the sample is way too
small. Sure there are 12000 lines of tweets, but to truly understand whether or not it is a predictor of elections (something
that already changes so much in the 4 years between elections) you would need a much larger sample and that's limited to just
4 elections, one of which, was while it was still a relatively small app. Instead of analyzing election outcomes, I decided
to focus on whether or not there was an increased engagement with certain hashtags, which hashtags were used the most by each
set of candidate related tweets, and the emotion surrounding a given hashtag. From the 6 charts (3 types, 1 for each
candidate) I put together I was able to make 3 conclusions. The first is that Hillary related tweets used almost exclusively
event-related hashtags, with the exception of "#imwithher" and "#tbt," which is in extreme contrast to Trump related tweets
who used more actionable and targeted hashtags, like "#crookedhillary," "#draintheswamp," and "#americafirst" all of which
spoke down on Hillary's campaign. The second conclusion I was able to make was that Hillary related tweets with hashtags were
generally much more emotional than Trump related tweets. The final conclusion is that Trump's choice to shift toward more
targeted and actionable messaging generally worked from a marketing and engagement perspective on social media as we see with
the tweets. Trump related tweets across the board had more engagement with his most used hashtags than Hillary, having
significantly higher favorite and retweet counts.
"""

##############################################
# === Part 3 Exploration #2 (Sydney) ===
##############################################
"""
# The previous exploration looked at engagement with certain hashtags associated with Trump and Clinton during the 2016
election.
# To further understand the amount of engagement with these hashtags, I decided to make a graph that visualizes the
engagement with the
# top five hashtags over time. This way, we can see when certain hashtags were created, and when they were most popular.

# My research question is: Do certain hashtags have more engagement during certain weeks of the election?
# This could give insights into more specific questions, such as:  is there more engagement with hashtags relating to debates
during debate weeks?

# My hypothesis is that engagement with hashtags will fluctuate over time, with greater variance in engagement for hashtags
associated
# with Hillary Clinton compared to those associated with Donald Trump. This is because Clinton's top hashtags are more event-
driven,
# leading to spikes and drops in engagement during certain times, while Trump's hashtags function more as slogans and
maintain more consistent engagement.

# Analysis method: I created a time-series graph that visualizes the engagement of the five most popular hashtags over time,
# highlighting when each hashtag emerged and peaked.

# The visualization validates my hypothesis, confirming that hashtags vary in engagement
# over the course of the election, and specifically that Trump's hashtags seem to be more popular for longer periods of time.
# For example the orange line on the graph for the Trump tweets, which was for the hashtag "#Makeamericagreatagain" spans for
nearly the
# entire period of time that the data examined. Notably, it is his campaign slogan. Similarly, the "#Imwithher" hashtag from
the Hilary tweets,
# represented by the purple line, which was the most similar to a slogan out of her top five hashtags, has the longest period
of engagement
# out of the five. Another insight we can make from this visualization is that engagement as a whole with certain hashtags
spiked in late October
# and early November for both candidates. This makes sense, due to the proximity to the election. I also had it confirm the
sentiment (positive
# or negative) of the hashtag to see if there was a difference, but it turns out the most popular hashtags were all positive.

#The code for the time series graph is below:
"""

##############################################
# === Load Data ===
##############################################
trump_df = pd.read_csv('data/trump_encoded.csv')
clinton_df = pd.read_csv('data/clinton_encoded.csv')
trump_df['candidate'] = 'Donald Trump'
clinton_df['candidate'] = 'Hillary Clinton'
df = pd.concat([trump_df, clinton_df], ignore_index=True)

# Convert 'created_at' to datetime
df['created_at'] = pd.to_datetime(df['created_at'], errors='coerce')

# Calculate engagement for each tweet
```

```
df['engagement'] = df['favorite_count'] + df['retweet_count']

##############################################
# Use hashtagged tweets with sentiment analysis
##############################################

# --- Clean text function (already in your code) ---
def clean_text(text):
    text = re.sub(r'@[\w]+', '', text)
    text = re.sub(r'#[\w]+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip().lower()
    return text

# --- Load NRC Emotion Lexicon ---
nrc = pd.read_csv('NRC-Emotion-Lexicon-Wordlevel-v0.92.txt', sep="      ", header=None,
                  names=["word", "emotion", "association"])
nrc = nrc[nrc['association'] == 1].drop(columns=['association'])

# --- Sentiment Function (raw counts) ---
def get_sentiment_counts(text, lexicon):
    tokens = word_tokenize(text)
    counts = {emotion: 0 for emotion in lexicon['emotion'].unique()}
    for token in tokens:
        matches = lexicon[lexicon['word'] == token]
        for _, row in matches.iterrows():
            counts[row['emotion']] += 1
    return counts

# --- Filter hashtagged tweets ---
df['has_hashtag'] = df['text'].str.contains(r'#\w+')
hashtagged = df[df['has_hashtag']].copy()

# --- Clean and apply sentiment analysis ---
hashtagged['cleaned_text'] = hashtagged['text'].apply(clean_text)
sentiment_counts = hashtagged['cleaned_text'].apply(lambda t: get_sentiment_counts(t, nrc))
sentiment_df = pd.DataFrame(list(sentiment_counts))
# Merge sentiment results back into hashtagged tweets
hashtagged_sentiment = pd.concat([hashtagged.reset_index(drop=True), sentiment_df.reset_index(drop=True)], axis=1)

# Compute net sentiment as positive - negative
hashtagged_sentiment['net_sentiment'] = hashtagged_sentiment['positive'] - hashtagged_sentiment['negative']

# Create a week column (Period) for aggregation
hashtagged_sentiment['week'] = hashtagged_sentiment['created_at'].dt.to_period('W')
hashtagged_sentiment['week_start'] = hashtagged_sentiment['week'].dt.start_time

##############################################
# Extract hashtags and determine top 5 per candidate
##############################################

def extract_hashtags(text):
    tags = re.findall(r'#\w+', text)
    return [tag.lower() for tag in tags]

hashtagged_sentiment['hashtags'] = hashtagged_sentiment['text'].apply(extract_hashtags)

# Explode the hashtags so each hashtag gets its own row
hs_exploded = hashtagged_sentiment.explode('hashtags')
# Remove rows with no hashtag
hs_exploded = hs_exploded[hs_exploded['hashtags'].notna()]

# Determine top 5 hashtags for each candidate by frequency
top_hashtags = {}
for candidate in hs_exploded['candidate'].unique():
    candidate_df = hs_exploded[hs_exploded['candidate'] == candidate]
    top5 = candidate_df['hashtags'].value_counts().head(5).index.tolist()
    top_hashtags[candidate] = top5

print("Top 5 Hashtags per Candidate:", top_hashtags)

# Filter hs_exploded to include only the top hashtags for each candidate
filtered_hs = hs_exploded[hs_exploded.apply(lambda row: row['hashtags'] in top_hashtags[row['candidate']], axis=1)]

##############################################
# Aggregate Engagement & Sentiment by Candidate, Hashtag, and Week
##############################################

agg = filtered_hs.groupby(['candidate', 'hashtags', 'week', 'week_start']).agg({
    'engagement': 'mean',
    'net_sentiment': 'mean'
}).reset_index()

# Compute overall net sentiment for each candidate and hashtag over the entire period
overall_sentiment = filtered_hs.groupby(['candidate', 'hashtags']).agg({
    'net_sentiment': 'mean'
}).reset_index()

# Classify overall sentiment as Positive if net sentiment > 0, else Negative
sentiment_class = {}
for _, row in overall_sentiment.iterrows():
    sentiment_class[(row['candidate'], row['hashtags'])] = "Positive" if row['net_sentiment'] > 0 else "Negative"

##############################################
# Plot: Engagement Over Time for Top Hashtags with Sentiment Annotation
```

```
##############################################

# Create one subplot per candidate
candidates = list(top_hashtags.keys())
n_candidates = len(candidates)
fig, axs = plt.subplots(n_candidates, 1, figsize=(14, 6 * n_candidates), sharex=True)
if n_candidates == 1:
    axs = [axs]

for ax, candidate in zip(axs, candidates):
    candidate_data = agg[agg['candidate'] == candidate]
    for tag in top_hashtags[candidate]:
        tag_data = candidate_data[candidate_data['hashtags'] == tag].sort_values('week')
        overall_label = sentiment_class.get((candidate, tag), "Unknown")
        # Label includes the hashtag and its overall sentiment classification
        ax.plot(tag_data['week_start'], tag_data['engagement'],
                label=f"{tag} ({overall_label})", marker='o', linestyle='-')
    ax.set_title(f"Top 5 Hashtags Engagement & Sentiment Over Time - {candidate}", fontsize=16)
    ax.set_xlabel("Week")
    ax.set_ylabel("Average Engagement")
    ax.legend()
    ax.grid(True)
    ax.xaxis.set_major_locator(mdates.WeekdayLocator(byweekday=mdates.MO, interval=2))
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.setp(ax.get_xticklabels(), rotation=45)

plt.tight_layout()
plt.savefig('output/hashtag_engagement_sentiment_over_time.png')
plt.show()
```