STOCK MOVEMENT ANALYSIS BASED ON SOCIAL MEDIA SENTIMENT

- **Objectives:-**

  Develop a machine learning model that predicts stock movements by scraping data from social media platforms like Twitter, Reddit, or Telegram. The model should extract insights from user-generated content, such as stock discussions, predictions, or sentiment analysis, and accurately forecast stock price trends.

- **Install Libraries:-**

  In this project, I am analyzing stock market movement based on the sentiment of social media posts using various Python libraries. To begin, I have installed several essential libraries such as **Telethon**, **NLTK**, **Pandas**, **Matplotlib**, **Seaborn**, **WordCloud**, **yfinance**, **TextBlob**, **VADER Sentiment**, **emoji**, **Urlextract**, **Gensim**, and **pyLDAvis**. These libraries are pivotal in collecting, processing, and visualizing data from social media platforms like Twitter. Using **Telethon**, I can scrape social media data, while **NLTK**, **TextBlob**, and **VADER Sentiment** assist in sentiment analysis. **Pandas** is used for data manipulation, **Matplotlib** and **Seaborn** for data visualization, and **WordCloud** helps generate word clouds to visually capture frequent terms. Additionally, **yfinance** allows for stock data retrieval, and **Gensim** and **pyLDAvis** support topic modeling, which is useful for uncovering trends in social media posts related to stock movements. This comprehensive setup enables the analysis of how sentiment around specific stocks correlates with their market performance.

  By leveraging sentiment analysis tools like **TextBlob** and **VADER Sentiment**, I am able to classify social media posts as positive, negative, or neutral. This classification helps in understanding the public's perception of specific stocks. The **Pandas** library is utilized for cleaning and transforming the data, making it easier to analyze and extract useful insights. Visualizations created using **Matplotlib** and **Seaborn** provide an intuitive understanding of the trends and patterns in the data. For sentiment analysis, **NLTK** and **TextBlob** help in tokenizing and lemmatizing the text, while **emoji** and **Urlextract** ensure that special characters and URLs are handled correctly, providing a more accurate sentiment representation.

To better understand the underlying themes in the data, **Gensim** and **pyLDAvis** are employed for topic modeling. This allows for the extraction of topics that are frequently discussed in relation to stocks, providing deeper insights into market sentiment. By integrating **yfinance**, I can retrieve real-time stock data, and correlate this with the sentiment derived from social media. Ultimately, this analysis aims to uncover patterns that might predict stock movement based on the mood and opinions of social media users, offering a unique perspective on market behavior influenced by public sentiment.

- **Data Scraping:-**

1. Setup and Initialization (Connecting to Telegram)

The first step in the code involves setting up the connection to the Telegram API using the Telethon library.

- **API Authentication**: To interact with the Telegram servers, you need an api_id and api_hash that are provided when you register your application with Telegram. These are unique identifiers that authenticate your app.
- **Phone Number**: In order to connect to the Telegram API for the first time, the phone number associated with the account needs to be provided. This step is necessary for Telegram to confirm the identity of the user.
- **Telegram Client**: The TelegramClient class from the Telethon library is used to create the connection to the Telegram servers. It requires the api_id, api_hash, and session_name. The session_name is used to store session data locally, which allows the application to maintain the connection and avoid having to authenticate repeatedly.

This part of the code ensures that the application is authenticated and ready to interact with Telegram. It provides the infrastructure for interacting with specific Telegram channels from which data (messages) will be fetched.

2. Data Cleaning and Preprocessing Functions

Once the connection to Telegram is established, the next step is to clean and preprocess the messages before performing any analysis. This ensures that the data is in a consistent format and is free from unnecessary information, which could otherwise complicate analysis.

- **Cleaning the Message**:
  - **URLs Removal**: The cleaning function starts by removing any URLs from the text. URLs are not relevant for sentiment analysis or stock movement prediction, so they are removed to keep the message content focused.

- **Removing Mentions, Hashtags, and Newlines**: Mentions (like @username), hashtags (like #stocks), and newlines are stripped from the message. Mentions and hashtags are typically not needed for analysis, especially if you are focusing on the content of the message itself. Removing newlines makes the text more uniform and easier to process.
- **Removing Emojis**: Emojis are removed by converting them into text using the demojize() function from the emoji library. Emojis are usually non-informative for sentiment analysis, so transforming them into their textual representations (e.g., 😊 becomes :smiling_face:) ensures that the message can be processed in a more meaningful way.
- **Converting to Lowercase**: Converting all text to lowercase ensures consistency. For example, "Good" and "good" would be treated as the same word, helping improve the accuracy of text analysis later.

- **Extracting Hashtags**: Hashtags are often used in social media to group posts related to a specific topic. The code defines a function to extract all hashtags from a message. This function scans the message for any words beginning with # and collects them. Hashtags could provide valuable insight into the context of the message and can be used for further analysis, such as grouping messages by topic or analyzing sentiment trends around certain hashtags.

  These functions ensure that the messages are cleaned, consistent, and free of irrelevant elements that would complicate sentiment analysis.

3. Fetching Data from Telegram and Saving to CSV

After setting up the connection and cleaning functions, the script proceeds to fetch data from a specific Telegram channel and save the cleaned information into a structured format.

- **Fetching Messages:**
  - Once the client is connected to Telegram, the script fetches messages from a specific channel using the get_messages() function. In this case, it is set to fetch up to 5000 messages from the channel named 'stocks'. This function retrieves the raw messages, which may contain a mix of different types of content (text, media, links, etc.).
  - The messages are fetched and stored in a list. Each message is an object that contains information like the message ID, date, text content, and more.
- **Processing Each Message:**
  - For every message fetched, the script checks if the message has any text content. It then proceeds to clean and preprocess the message using the cleaning function described earlier.
  - The script also extracts hashtags from each message using the function defined previously. Hashtags provide context about the message, such

as whether it's related to a specific stock or financial term.
    - After cleaning and extracting relevant data (text, hashtags, message length), the information is stored in a structured format (a dictionary) and added to a list. This list is eventually used to create a Pandas DataFrame, which organizes the data into tabular form, making it easier to work with and analyze.
- **DataFrame and Data Cleaning:**
    - A Pandas DataFrame is created from the list of messages. DataFrames are powerful data structures used for handling and analyzing data, allowing you to perform operations like filtering, sorting, and cleaning.
    - Missing values are handled by dropping any rows where the text is empty or invalid. Duplicates are also removed, ensuring that only unique messages remain.
    - The DataFrame is reset to ensure that the indices of the rows are continuous, which is important for ensuring that the data is well-organized.
- **Saving Data:**
    - Finally, the cleaned data is saved to a CSV file using the Pandas to_csv() function. This file contains the processed messages with information such as the message ID, date, text, length, and hashtags. The CSV format makes it easy to export the data for further analysis or machine learning tasks.

      By fetching, cleaning, and organizing the data in this way, the script prepares the data for later use in sentiment analysis or other predictive modeling techniques related to stock movements based on social media posts.

- **<u>Challenges Encountered in the Scraping Process:-</u>**

1. **Challenge: API Authentication and Connectivity Issues**
    - **Problem**: The first step in using Telegram's API is to authenticate and establish a connection to the platform using api_id, api_hash, and the phone number. This process can sometimes be tricky due to incorrect credentials or network issues.
    - **Resolution**: To resolve these issues, the script ensures that the correct api_id, api_hash, and the phone number are used, which are obtained by registering an application with Telegram. Also, the script prompts for user authentication if required, to handle any first-time login scenarios. Once the client is connected, it remains persistent, avoiding the need for repeated authentication
      .

2. **Challenge: Handling Large Amounts of Data**
    - **Problem**: When scraping messages from Telegram, the amount of data can quickly become overwhelming, especially when fetching a large number of messages (e.g., 5000 messages as in this case). Fetching a large dataset may lead to performance issues, such as long processing times, memory overflow, or server errors.
    - **Resolution**: To mitigate this, the script fetches a limited number of messages (limit=5000) in manageable chunks to avoid overwhelming the system. Additionally, the data is processed and cleaned incrementally, which helps in reducing the memory load. For extremely large datasets, implementing pagination and fetching data in smaller batches would further enhance the performance and avoid potential memory overload.

During the process of scraping data from Telegram channels, several challenges were encountered. Data inconsistencies and missing information were common, requiring careful handling to ensure the quality of the dataset. Non-relevant content, such as URLs, mentions, and emojis, needed to be cleaned to focus on the core text. Extracting hashtags from the text proved important for identifying key topics, but it required attention to different formats. Telegram API restrictions and rate-limiting posed obstacles, which were managed by adjusting request intervals. Finally, text data had to be cleaned and preprocessed to remove noise, ensuring it was in a consistent format before saving and exporting it for further analysis.

- **<u>Data Analysis:-</u>**

1. **Sentiment Analysis with VADER (Valence Aware Dictionary and sEntiment Reasoner):**
    - **VADER Sentiment Analyzer**: The first step involves importing the necessary libraries and using the **VADER sentiment analyzer** from the Natural Language Toolkit (NLTK) to analyze the sentiment of Telegram messages. VADER is specialized in analyzing social media text and works well with short, informal messages like those in Telegram.
    - **Sentiment Classification**: For each message, the sentiment is analyzed, and a **polarity score** is computed, representing the overall sentiment of the message. The compound score indicates the strength of the sentiment.
        - **Positive Sentiment**: When the compound score is greater than or equal to 0.05.
        - **Negative Sentiment**: When the compound score is less than

or equal to -0.05.

- **Neutral Sentiment**: When the score falls between -0.05 and 0.05.

- **Handling Missing Text**: Any missing or null values in the message text are replaced with an empty string to ensure the sentiment analysis doesn't fail due to missing data.

2. Extracting Stock Mentions from Messages:

- **Keyword Dictionary**: A dictionary is created that maps various stock ticker symbols (e.g., 'AAPL' for Apple, 'TSLA' for Tesla) and market-related terms (e.g., 'bullish', 'bearish') to meaningful descriptions. This helps identify whether a message is discussing a particular stock or market sentiment.
- **Extracting Mentions**: The next step is to scan each message for mentions of these predefined stock symbols or market-related keywords. The function scans each message for the presence of these keywords using regular expressions, which allow for case-insensitive matching of whole words.
- **Counting Mentions**: Once mentions are extracted, their frequency is counted across all messages. The result shows how often each stock or trend is mentioned.

3. Data Visualization:

- **Visualizing Mention Frequency:** The script uses Seaborn and Matplotlib to create bar plots showing the frequency of mentions for various stocks or market-related terms. This gives a visual representation of which stocks or trends are being discussed the most within the dataset.
- **Graph Annotations:** The bars in the plot are annotated with their respective values, allowing for easy interpretation of the chart. This helps users quickly understand which stocks or trends are the most popular or frequently discussed.

4. Topic Modeling with LDA (Latent Dirichlet Allocation):

- **Preprocessing the Text:** The text is first cleaned by removing non-alphabetic characters and converting everything to lowercase. It then tokenizes the text and removes common stopwords (words that don't carry significant meaning, such as 'the', 'and', etc.).
- **LDA Model:** The LDA (Latent Dirichlet Allocation) model is used to discover the underlying topics in the Telegram messages. It creates a set of topics based on the words present in the messages. The num_topics parameter controls how many topics are extracted. Each topic is represented as a list of words that frequently appear together in the dataset.
- **Visualization:** PyLDAvis is used to visualize the topics extracted

by the LDA model. This tool allows for an interactive exploration of topics, helping to understand which words are most strongly associated with each topic. The topics are displayed in a way that makes it easy to identify patterns in the data.

5. Time Series Analysis of Sentiment and Mentions:

- **Resampling the Data:** The messages are grouped by date (resampled) to calculate daily sentiment trends and mention counts. This allows us to observe how the sentiment (positive, negative, or neutral) and the volume of stock mentions change over time.
- **Trend Visualization:** Two separate plots are created**:**
  - **Sentiment Trends:** A line plot showing how the average daily sentiment polarity (positive, negative, neutral) changes over time.
  - **Mention Trends:** A line plot showing the total number of mentions for stock and market-related terms each day.
- **Normalized Visualization:** For comparative analysis, both sentiment and mentions can be visualized together on the same plot, with mention counts normalized to a range between 0 and 1. This helps in comparing the trends directly.

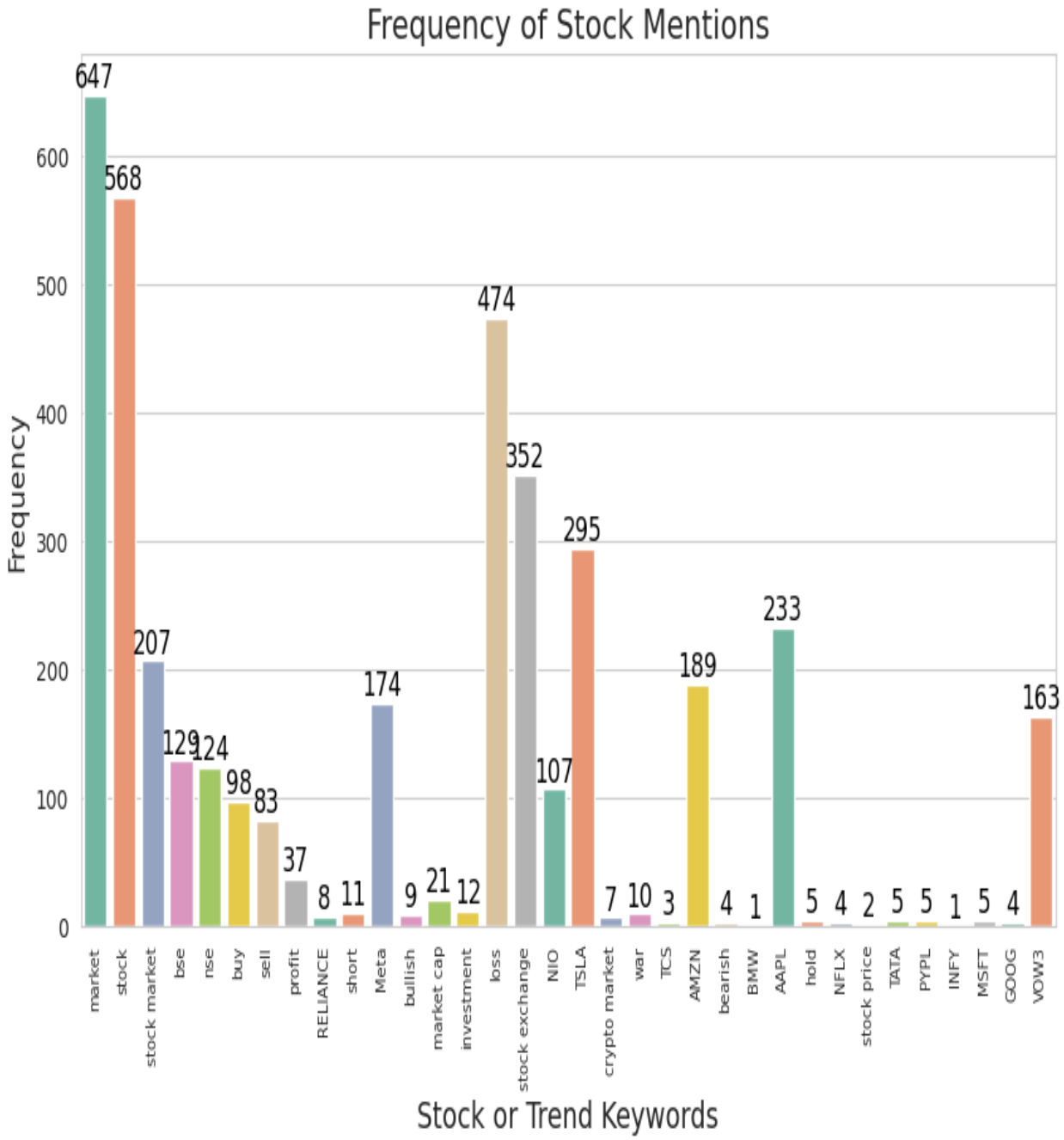
6. Stock Market Data with Yahoo Finance (yfinance):

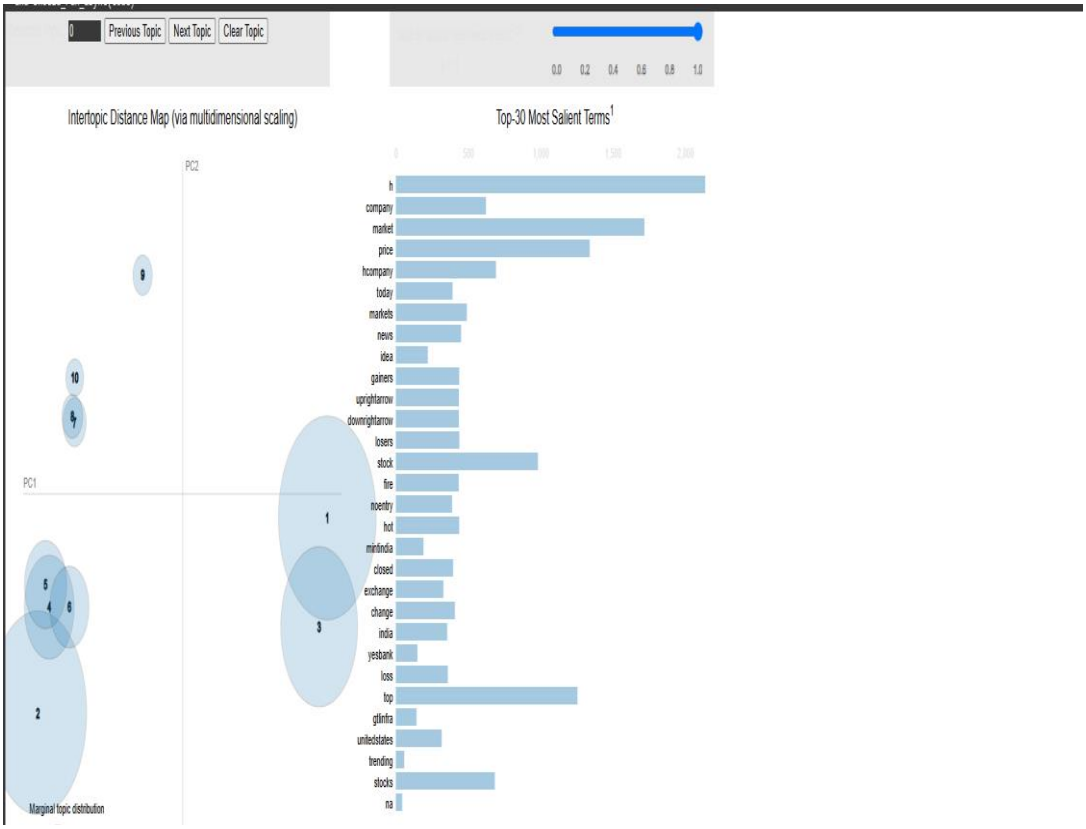
- **Stock Tickers:** A list of stock symbols is defined, which are the focus of the analysis (e.g., Apple, Tesla, Google). These tickers are important for understanding the context of the Telegram messages, as the analysis is centered around stock-related discussions.
- **Data Collection:** The yfinance library is available to fetch stock data, allowing you to compare Telegram message trends with real stock market data. Although this step is indicated in the code, it is not fully implemented in the provided code block.

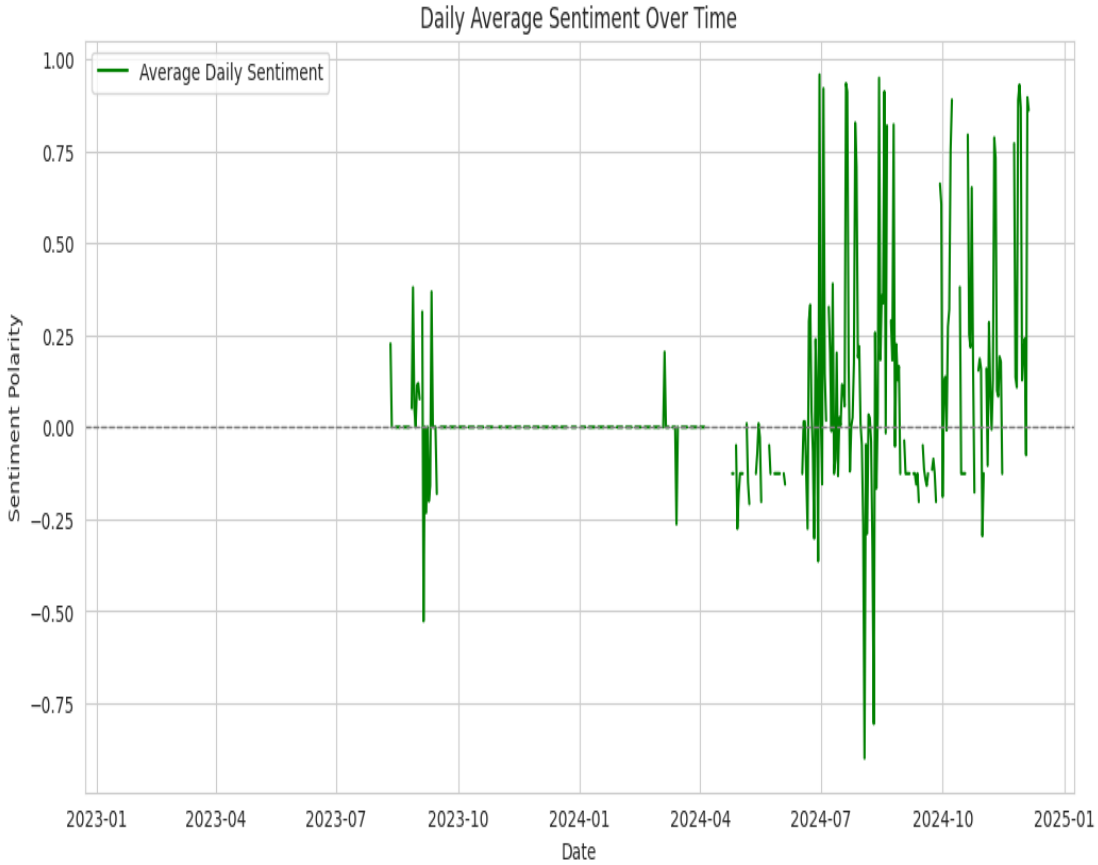
7. Final Output and Data Export:

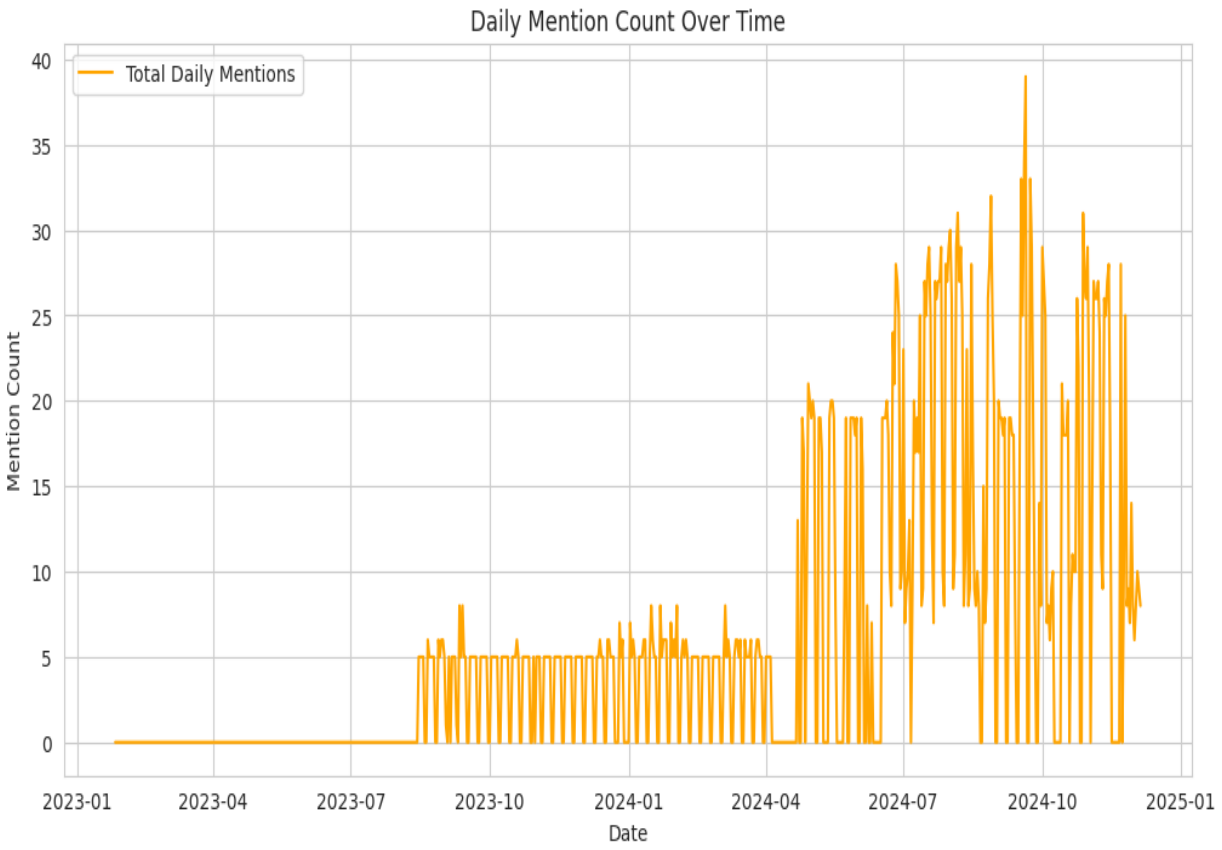
- **Visuals:** After plotting all the relevant visualizations, the script saves and displays the graphs. For example, the plot for keyword mentions frequency is saved as a PNG file for later use.
- **Displaying Data:** Finally, the processed data, including sentiment classifications, mention counts, and topic modeling outputs, are displayed and exported. This ensures that the results can be analyzed further, either in the notebook or by exporting

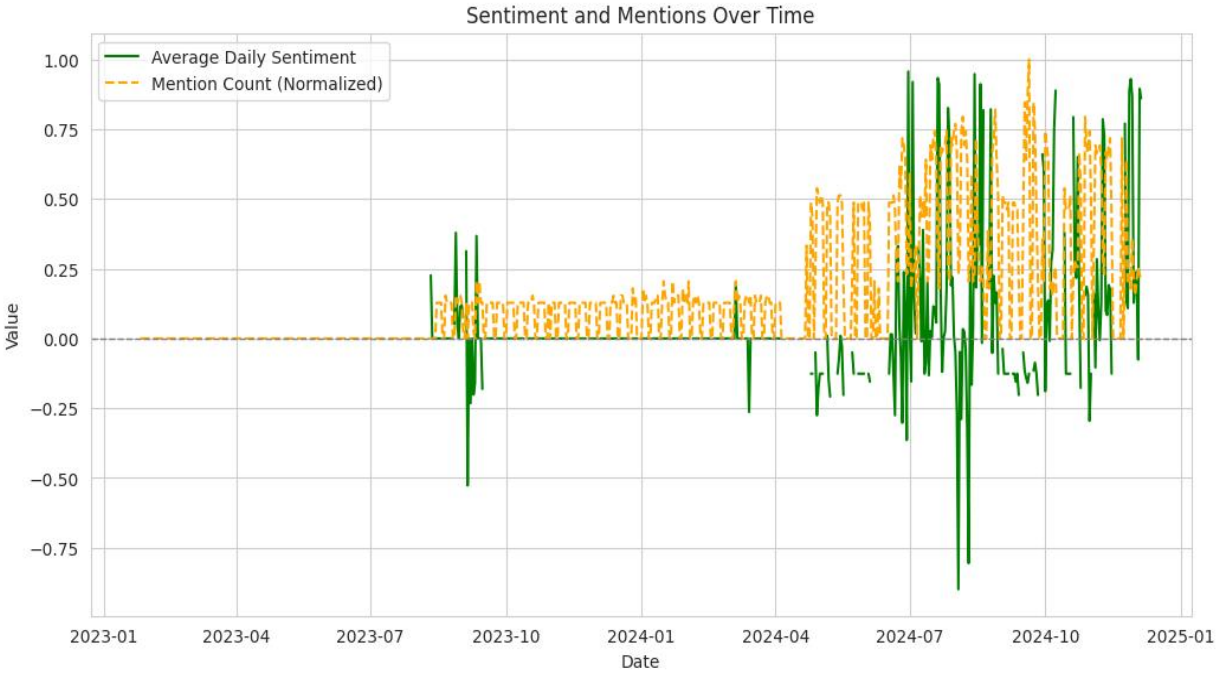
to CSV or other formats.




Previous Topic
Next Topic
Clear Topic
Intertopic Distance Map (via multidimensional scaling)
Top-30 Most Salient Terms1
PC1
PC2
Marginal topic distribution
h
company
market
price
hcompany
today
markets
news
idea
gainers
uprightarrow
downrightarrow
losers
stock
fire
noentry
hot
mindindia
closed
exchange
change
india
yesbank
loss
top
gtlinfra
unitedstates
trending
stocks
na


Daily Average Sentiment Over Time
Average Daily Sentiment
Sentiment Polarity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
2024-07
2024-10
2025-01
Date


Daily Mention Count Over Time
Total Daily Mentions
Mention Count
Date
0
5
10
15
20
25
30
35
40
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
2024-07
2024-10
2025-01


Sentiment and Mentions Over Time
Average Daily Sentiment
Mention Count (Normalized)
Value
Date
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
2024-07
2024-10
2025-01

- **Prediction Model:-**

1. Preparing the Data for Prediction

The prepare_prediction_data function is responsible for setting up the data that will be used to train the model. Here's what it does:

- **Selecting Features**: The code selects relevant features (or columns) from the dataset, which are important for predicting the outcome. In this case, the features selected are:
    - polarity: This is a sentiment score, which represents how positive or negative the message is. A higher polarity score indicates more positive sentiment.
    - mention_count: This represents the number of stock-related mentions in the message. It helps capture how focused the message is on stocks or market trends.
- **Target Variable (Movement)**: The target variable, called movement, is created based on the polarity of the message. If the polarity is positive (> 0), it is considered a signal for stock price movement in the upward direction (denoted as 1 for upward movement). Otherwise, the movement is considered as 0, representing no movement or downward movement.

After this, the dataset is split into:

- **Features (X)**: These are the input variables that the model will use to make predictions (in this case, polarity and mention_count).
- **Target (y)**: This is the output variable that the model is trying to predict (movement).

2. Training the Machine Learning Model

The train_prediction_model function focuses on training a machine learning model using the data prepared in the previous step. Here's how the process works:

- **Data Splitting**: The dataset is split into training and testing datasets using the train_test_split method from sklearn. This ensures that the model is trained on a portion of the data and tested on a separate portion.
    - 80% of the data is used for training (X_train, y_train), and 20% is used for testing (X_test, y_test).
- **Feature Scaling**: Before feeding the data into the model, the features are scaled using StandardScaler. Feature scaling ensures that all input features have the same scale (mean of 0 and standard deviation of 1), preventing some features from dominating others due to large numeric

values.

- **Model Training**: The core of the prediction is performed using the **Random Forest Classifier** model. Random Forest is an ensemble learning technique that combines the predictions of multiple decision trees to make a final decision. This helps improve the model's accuracy and robustness.
    - The model is trained using the scaled training data (X_train_scaled, y_train).
- **Making Predictions**: After the model is trained, it is used to predict stock movement on the test dataset (X_test_scaled). The predictions are stored in y_pred.
- **Model Evaluation**: Once predictions are made, the model's performance is evaluated using several metrics:
    - **Accuracy**: The percentage of correct predictions.
    - **Precision**: How many of the predicted positive movements were actually positive.
    - **Recall**: How many of the actual positive movements were correctly predicted by the model.
    - **F1 Score**: The harmonic mean of precision and recall. It gives a balance between precision and recall when they are uneven.

These metrics give an idea of how well the model is performing, especially in terms of predicting positive stock movements.

3. Running the Prediction Process

The main function orchestrates the workflow by calling the functions defined earlier:

- **Prepare Data**: The function first prepares the data by calling prepare_prediction_data.
- **Train Model**: It then trains the machine learning model using train_prediction_model.
- **Display Results**: Finally, it prints out the evaluation metrics (accuracy, precision, recall, F1 score) to give insight into how well the model is performing.