1. Goals of the Project:

Our project aimed to gather and analyze data from Hacker News, a prominent news aggregation platform, leveraging both web scraping techniques and API access.

Primary objectives included:

- Extracting data from Hacker News website using web scraping (Beautiful Soup) to capture information about top stories, including titles, points, authors, comment counts, and story URLs.
- Accessing the Hacker News API to collect structured data about stories, comments, and user activity, enhancing the depth and diversity of our dataset.

Planned Data Gathering:

1. **From Hacker News Website:**
    - Utilizing web scraping techniques to extract information about top stories, including titles, points, authors, comment counts, and story URLs.
2. **From Hacker News API:**
    - Accessing the API to fetch structured data about stories, comments, and users.
    - Gathering the list of comment IDs for each story to facilitate fetching comment text and author details.
    - Scraping comments for each story to analyze user engagement, including the text of each comment and the author of the comment.

2. Achieved Goals of the Project:

In our project, we successfully achieved the following goals, working with the Hacker News website and API to gather comprehensive data for analysis:

Data Gathering and Storage:

- **Data Collection:**
    - Gathered data from both the Hacker News website and API, encompassing details about top stories, comments, and user activity.
    - Collected information such as story titles, points, authors, comment counts, comment texts, and comment authors.
- **Database Storage:**
    - Stored the collected data into a SQLite database for efficient organization and retrieval.
    - Created separate tables for stories and comments, with appropriate columns to store relevant data.
    - Established a relationship between the two tables, linking comments to their respective parent stories through foreign key constraints.

Summary:

Through the combined use of beautiful soup and API access, we successfully collected a diverse range of data from the Hacker News platform. The gathered data encompassed essential

information about top stories, user engagement through comments, and user activity on the platform. These achievements laid the foundation for our subsequent data analysis and visualization tasks, ensuring that the collected data is stored efficiently and is ready for further exploration.

3. Challenges Faced During the Project:

Throughout the project, we encountered several challenges that required careful consideration and problem-solving. Here are some of the key issues we faced:

1. **Handling Deleted Comments in API Responses:**
    - One significant challenge arose when fetching comments from the Hacker News API. We observed that some comments were deleted but still had entries in the API response. These deleted comments lacked the 'text' key, which caused errors during data processing.
    - To address this issue, we implemented error handling mechanisms, such as try-except statements, to handle cases where the 'text' key is missing in comment responses. This ensured that our data retrieval process remained robust and could handle unexpected variations in API responses.
2. **Preventing Duplicate Story Entries in Web Scraping:**
    - During web scraping of the Hacker News website, we encountered the possibility of duplicate story entries being saved into our database. This occurred when the same story was displayed on multiple pages of the website.
    - To mitigate this issue, we implemented logic to check for duplicate story IDs before inserting data into the database. Using try-except statements, we ensured that only unique story IDs were saved, preventing duplication and maintaining the integrity of our database.
3. **Ensuring Consistency and Reliability of Data:**
    - Another challenge we faced was ensuring the consistency and reliability of the gathered data. Due to the dynamic nature of online platforms like Hacker News, the structure and content of web pages and API responses could vary over time.
    - To address this challenge, we regularly monitored and tested our data retrieval processes, adapting our code as needed to accommodate changes in website layout or API response formats. Additionally, we implemented data validation checks to identify and handle any anomalies or inconsistencies in the collected data.

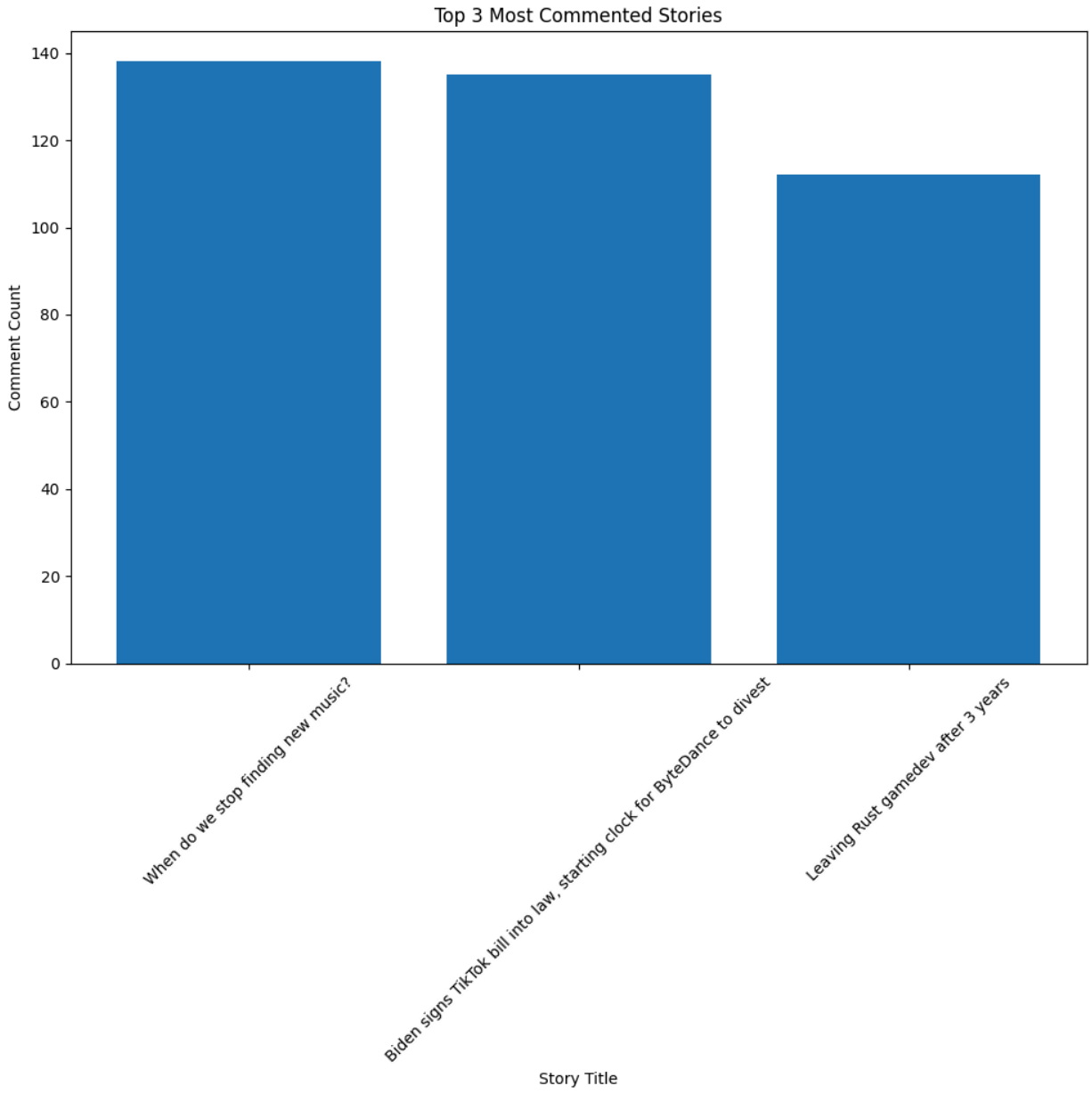
4. Data Calculations

The following calculations were performed using the data stored in the database:

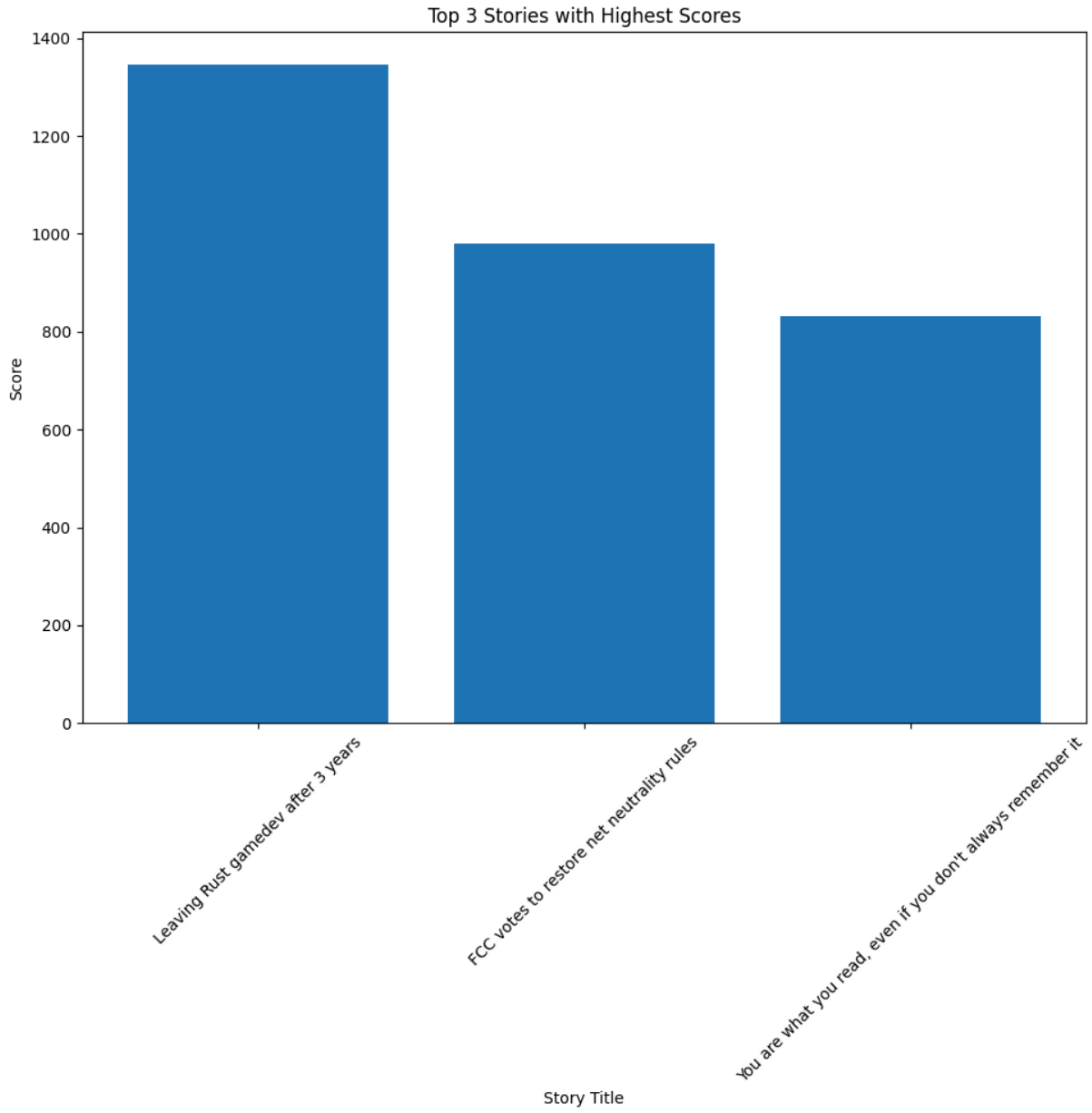
- **Top 3 stories with the highest scores:**
    - Story ID: 40172033, Title: Leaving Rust gamedev after 3 years, Score: 1345
    - Story ID: 40160429, Title: FCC votes to restore net neutrality rules, Score: 980
    - Story ID: 40151952, Title: You are what you read, even if you don't always remember it, Score: 833
- **Stories with comments on the story with the highest score:**
    - Story ID: 40172033, Title: Leaving Rust gamedev after 3 years, Score: 1345
        - Comment ID: 40172273, Comment Text: I would love to be able to bypass the orphan rule for internal crates., Comment Author: maximilianburke
        - Comment ID: 40172314, Comment Text: The thing that the Rust community thinks sets them apart (their community), is really the thing holding them back., Comment Author: syndicatedjelly
- **Authors and their story counts:**
    - Author: thunderbong, Story Count: 3
    - Author: PaulHoule, Story Count: 3
    - Author: vyrotek, Story Count: 2
    - Author: tosh, Story Count: 2
    - Author: todsacerdoti, Story Count: 2
    - Author: rntn, Story Count: 2
    - Author: kryster, Story Count: 2
    - Author: fanf2, Story Count: 2
    - Author: bookofjoe, Story Count: 2
    - Author: zdw, Story Count: 1

5. Visualization

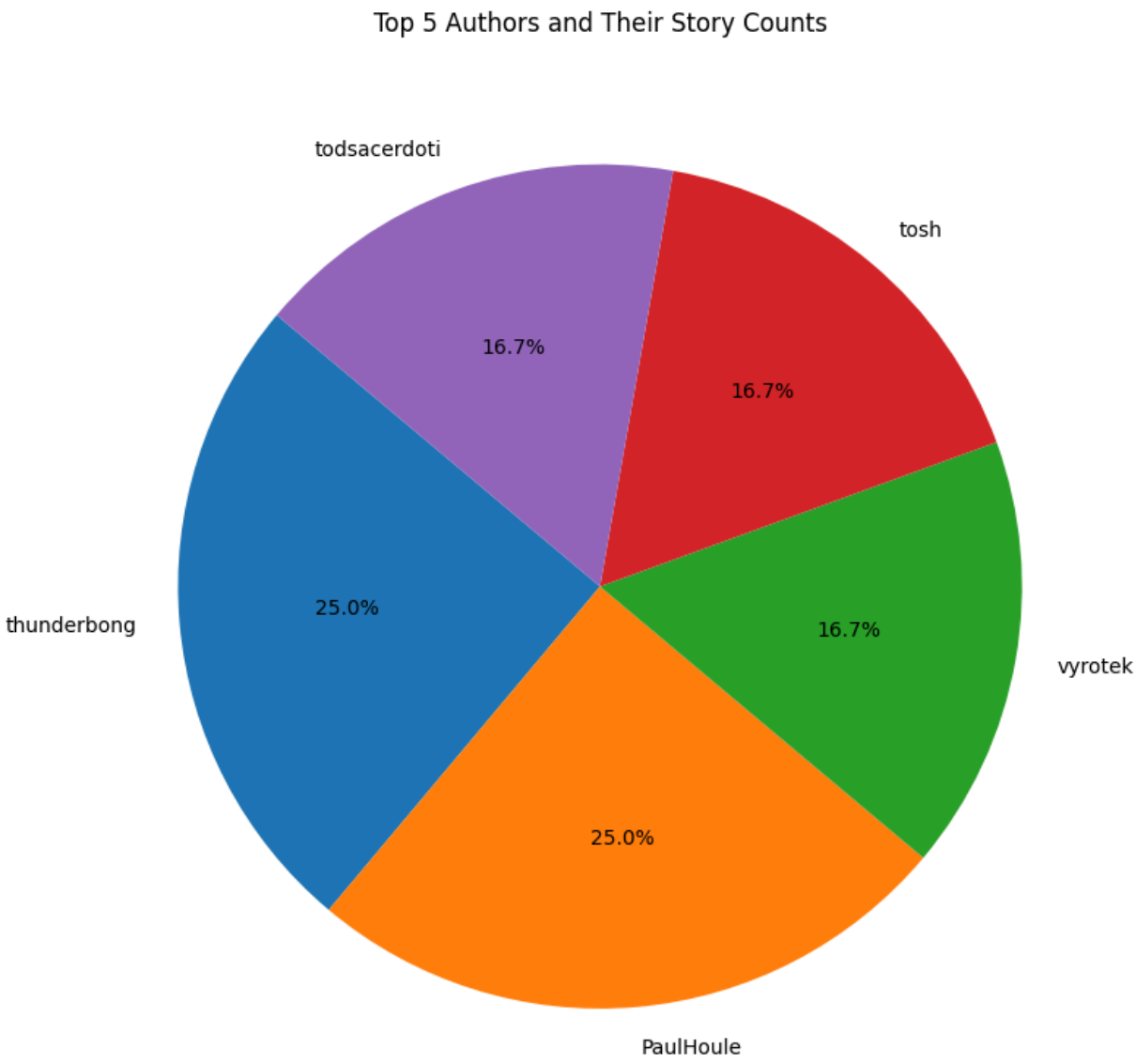
Top 3 Most commented stories:



Top 3 Stories with Highest Scores:



Top 5 Authors and their Story count



6. Instructions for Running the Code

1. Ensure that Python 3.x is installed on your system.
2. Install the required libraries by running the following command:
    a. Requests
    b. beautifulsoup4
    c. Matplotlib
3. **fetch_data.py**:
    a. Execute this file first to fetch and store the data into the database.
    b. Follow any prompts or input any required parameters as instructed by the program.

c. Wait for the program to fetch and store the data. Progress messages will be displayed in the terminal.
d. Once the program finishes execution, the data will be stored in the SQLite database.

4. **process_data.py**:
    a. After the data has been fetched and stored, execute this file to perform calculations and write the results to a text file.
    b. The program will read data from the database, perform calculations, and generate a text file with the calculated results.
    c. Once the program finishes execution, review the generated text file for the calculated results.
5. **visualization.py**:
    a. Finally, execute this file to create visualizations for the calculated data from the database.
    b. The program will read data from the database, create visualizations, and save them as image files.

7. Documentation

Fetch_data.py:

fetch_html(url):

- This function fetches the HTML content of a webpage from the specified URL.
- It uses the requests.get() method to send a GET request to the URL.
- If the request is successful (status code 200), it returns the HTML content of the webpage.
- If the request fails or the status code is not 200, it returns None.

fetch_kids(story_id):

- This function fetches information about comments associated with a story from the Hacker News API.
- It constructs the API endpoint URL for fetching comments using the provided story_id.
- Upon successful retrieval, it returns a JSON object containing information about the comments.
- If the request fails, it returns None.

fetch_comments(kids, parent_id, conn):

- This function fetches individual comments associated with a story from the Hacker News API.
- It iterates through the list of comment IDs (kids) associated with a parent story.
- For each comment ID, it constructs the API endpoint URL and sends a GET request to fetch comment data.

- If the request is successful, it inserts the comment data into the database using the insert_comment() function.
- If the request fails, it prints a failure message.

create_database():

- This function creates an SQLite database named 'Data_base.sqlite3' and defines two tables: 'stories' and 'comments'.
- The 'stories' table stores information about stories, including ID, title, URL, score, author, and comment count.
- The 'comments' table stores information about comments, including ID, parent ID (foreign key referencing stories), text, and author.
- If the database or tables already exist, it does not recreate them.
- Returns the SQLite connection object for database operations.

insert_story(story, conn):

- This function inserts information about a story into the 'stories' table of the SQLite database.
- It takes a dictionary containing story data (story) as input and inserts the data into the database.
- If the insertion fails due to a duplicate story ID, it prints a message indicating that the story already exists.
- Commits the transaction to the database upon successful insertion.

insert_comment(comment, parent_id, conn):

- This function inserts information about a comment into the 'comments' table of the SQLite database.
- It takes a dictionary containing comment data (comment), along with the parent story ID, as input.
- If the insertion fails due to a deleted comment (missing text key), it prints a message indicating that the comment is deleted.
- Commits the transaction to the database upon successful insertion.

parse_html(html, conn):

- This function parses the HTML content of a webpage to extract relevant information about stories and comments.
- It uses BeautifulSoup library to parse the HTML and extract story titles, URLs, scores, authors, and comment counts.

- For each story, it inserts the story data into the 'stories' table and fetches comments if available.
- If comments are available, it iterates through them and inserts them into the 'comments' table.
- Populates the SQLite database with data extracted from the HTML content of the webpage.

Main Function:

- The main function prompts the user to enter a page number between 2 and 5 to fetch data from Hacker News.
- It fetches HTML data from the website, creates an SQLite database, parses the HTML, and stores the data in the database.
- Prints a success or failure message based on the outcome of the data retrieval process.
- Serves as the entry point for executing the code to fetch and store data from Hacker News.

Process_data.py:

calculate_and_write_to_file():

- This function performs SQL queries on the SQLite database to select relevant data from tables and calculate certain statistics.
- It retrieves the top 3 stories with the highest scores, stories with comments on the story with the highest score, and the count of stories for each author.
- The calculated results are then written to a text file named 'calculated_results.txt'.
- The function connects to the SQLite database, executes SQL queries, writes results to the text file, and closes the database connection.

Main Function:

- The main function serves as the entry point for executing the calculate_and_write_to_file() function.
- It checks if the script is being run directly (__name__ == "__main__") and calls the calculate_and_write_to_file() function accordingly.

Input:

- No direct input is provided to the function. It relies on the SQLite database ('Data_base.sqlite3') to perform calculations.

Output:

- The function generates a text file named 'calculated_results.txt' containing the calculated data.
- The text file includes the top 3 stories with the highest scores, stories with comments on the story with the highest score, and authors with their respective story counts.

Visualisation.py

fetch_data(query):

- This function connects to the SQLite database ('Data_base.sqlite3'), executes the provided SQL query, fetches the data returned by the query, and then closes the database connection.
- It is designed to retrieve data from the database based on the specified query.

Input:

- query: A string containing the SQL query to be executed on the database.

Output:

- data: A list of tuples containing the fetched data from the database. Each tuple represents a row of data returned by the SQL query.

Main Function:

- The main part of the code checks if the script is being run directly (__name__ == "__main__") and executes a series of SQL queries to fetch data from the database.
- It then calls the fetch_data() function to retrieve the data based on the specified queries.

create_bar_plot(x_labels, y_values, title, x_label, y_label, filename):

- This function creates and saves a bar plot using Matplotlib.
- It accepts input parameters such as x-axis labels, y-axis values, plot title, x-axis label, y-axis label, and the filename for saving the plot.

Input:

- x_labels: A list of strings representing the labels for the x-axis.
- y_values: A list of numerical values representing the heights of the bars on the y-axis.
- title: A string representing the title of the plot.
- x_label: A string representing the label for the x-axis.
- y_label: A string representing the label for the y-axis.
- filename: A string representing the filename (including extension) for saving the plot.

Output:

- The function saves the generated bar plot as an image file with the specified filename.

create_pie_chart(labels, sizes, title, filename):

- This function creates and saves a pie chart using Matplotlib.
- It accepts input parameters such as labels, sizes (proportions), title, and filename for saving the plot.

Input:

- labels: A list of strings representing the labels for each segment of the pie chart.
- sizes: A list of numerical values representing the proportions of each segment.
- title: A string representing the title of the pie chart.
- filename: A string representing the filename (including extension) for saving the plot.

Output:

- The function saves the generated pie chart as an image file with the specified filename.

8.Resources Used:

1. Python Documentation: The official Python documentation was consulted for information on language syntax, standard library modules, and best practices.
2. SQLite Documentation: The official SQLite documentation was referenced for information on SQL syntax and SQLite-specific features.
3. Matplotlib Documentation: The official Matplotlib documentation was consulted for information on creating various types of plots and customizing plot attributes.
4. Requests Documentation: The official Requests documentation was referred to for information on making HTTP requests and handling responses.
5. BeautifulSoup Documentation: The official BeautifulSoup documentation was used to understand the library's functionality for parsing HTML data.
6. Hacker News API Documentation: The documentation for the Hacker News API was reviewed to understand how to interact with the API and fetch data from it.