Final Project Report

Link to our github repository: https://github.com/renjialan/final_project_python

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather

We wished to calculate trends within the job industries we are interested in. The APIs we originally planned to work with were Discord and Reddit, and we decided we also wanted to web-scrape LinkedIn. We wanted to cross-compare how often people used negative, positive, and neutral wording in regards to specific job industries on specific dates, and planned to count the actual number of jobs that appeared in relation to those job industries. By gathering how often people used certain phrasing surrounding conversations relating to an industry, we could better understand public opinion. We felt this could give us a good overview of general public opinion regarding how well job industries were doing and could help us better understand the scope of our own future within these industries.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather

Ultimately, we had to pivot from our initial goals of measuring the current job market through the lens of existing discourse to instead focus on job postings themselves. After realizing the data we could collect from Discord and Reddit APIs would be unhelpful in this particular regard, we decided to switch gears. We scraped two databases, the News API and the Muse API, and we used BeautifulSoup to fetch information from the third database, Linkedin. From the News API, we collected the time at which an article was posted, a content overview of the article itself, including the title and the first sentence, and the number of keywords the article included broken down by category from our initial list of “positive,” “negative,” or “neutral” words. The goal of this data collection was to determine how often the job market is featured in current news. Then, to analyze how many jobs there are actually in said market, we gathered the job title, company, location of the company, and links to apply to the job from LinkedIn postings. To strengthen these findings further, we then gathered the job title, the time at which it was posted to determine recency, the experience level required, and the job category from Muse postings.

3. The problems that you faced

One major issue with web scraping LinkedIn was that the website typically prompts users to log in to use its resources. Rather than have to maneuver this flow, we began exploring alternative ways to find a URL that could still provide data to parse through without logging in. We found that using an incognito browser and loading the website gave us a URL that could freely bring up jobs regarding any industry, as long as we manipulated keywords in the URL, like location or industry, before inputting it to the browser.

Another primary issue was with Muse API, which did not allow for a breakdown by category or experience level because those were features a user must preselect within filters

instead of appearing automatically tagged through a post. To solve this problem, we decided to prioritize other types of data that would be easier to secure but still work to fit our overall calculations.

We also had trouble figuring out how to best join two of our datasets through a common shared integer key. Our Muse API data and LinkedIn data were both about jobs and job postings, but did not inherently correspond to the same objects of data– one job posting with a certain title on LinkedIn did not necessarily have a corresponding job posting on Muse. Thus, we realized a better way to analyze data was by considering the type of job postings and titles available on both platforms to parse through within the data. We decided to join the items without making the data directly correspond and instead counted how often the word “intern” appeared within the two data sets.

4. The calculations from the data in the database (i.e. a screen shot)

Muse API Calculations:

Screenshot:

```
[Running] python -u "/Users/anjalitandon/Desktop/fall/si 206/final_project_python/muse_calculation.py"
The most common month for a job listing to be posted on Muse is: May
The most common day of the week for a job listing to be posted on Muse is: Thursday
```

File:

Linkedin Calculations:

Screenshot:

```
[Running] python -u "/Users/samyukthavariyam/Library/CloudStorage/OneDrive-TheNewSchool/si_206/final_project_python/linkedin_calculation_and_viz.py"
The top titles for technology related jobs in the U.S. posted on LinkedIn are as follows:[('Operator, Data Entry Junior / Remote', 7), ('Chief Technology Officer', 6), ('Data Entry Clerk Jr (Remote)', 6), ('Director of IT', 6), ('Manufacturing Quality Manager', 6)]
The top locations for technology related jobs in the U.S. posted on LinkedIn are as follows:[('New York, NY', 9), ('San Francisco, CA', 8), ('Arlington, TX', 6), ('Boise, ID', 6), ('Costa Mesa, CA', 6)]
```

File:

News api calculations:

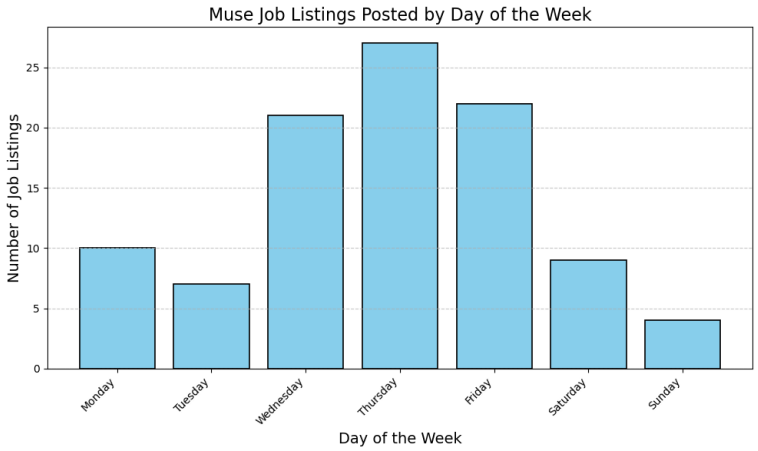
totals_per_day.txt 11-14-14-005.txt

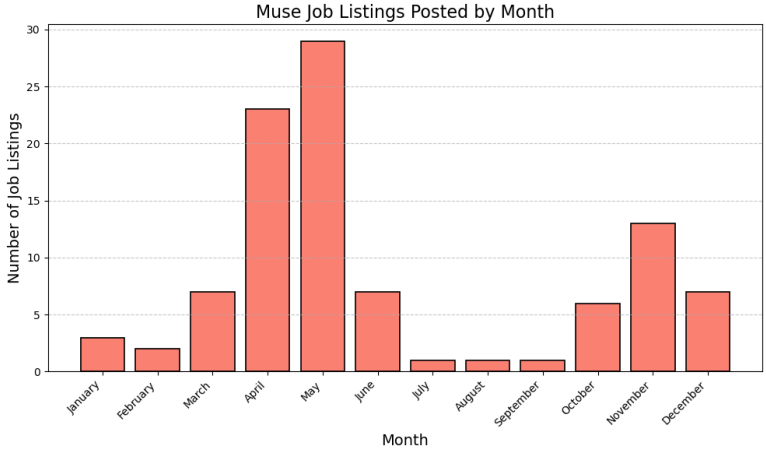
```
2023-12-11: Total Neutral - 12, Total Positive - 6, Total Negative - 0
```

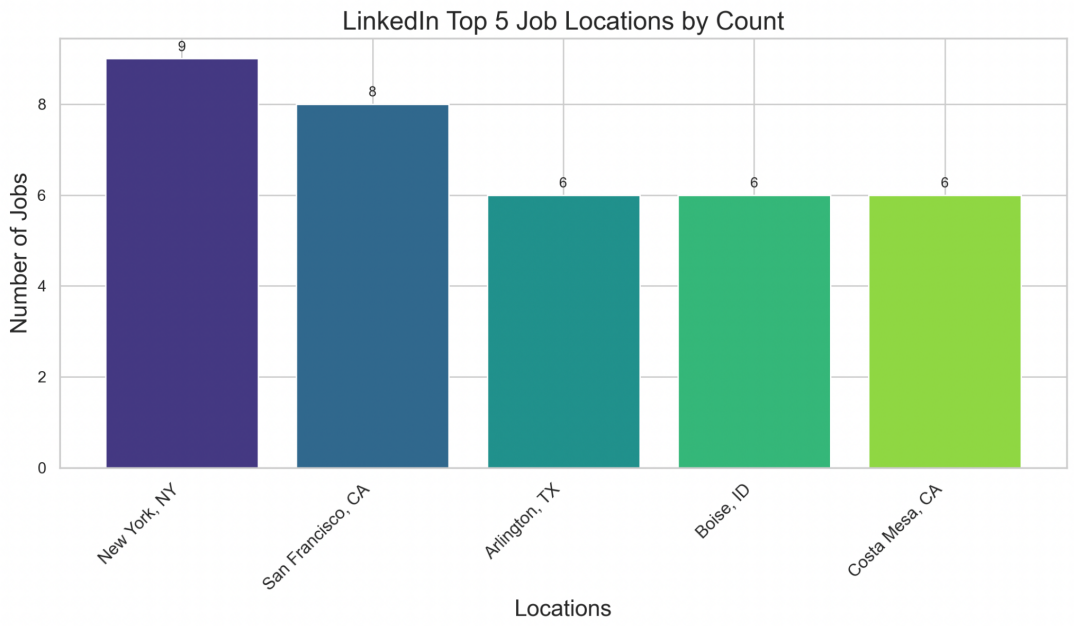
averages_per_day.txt 20-47-49-533.txt

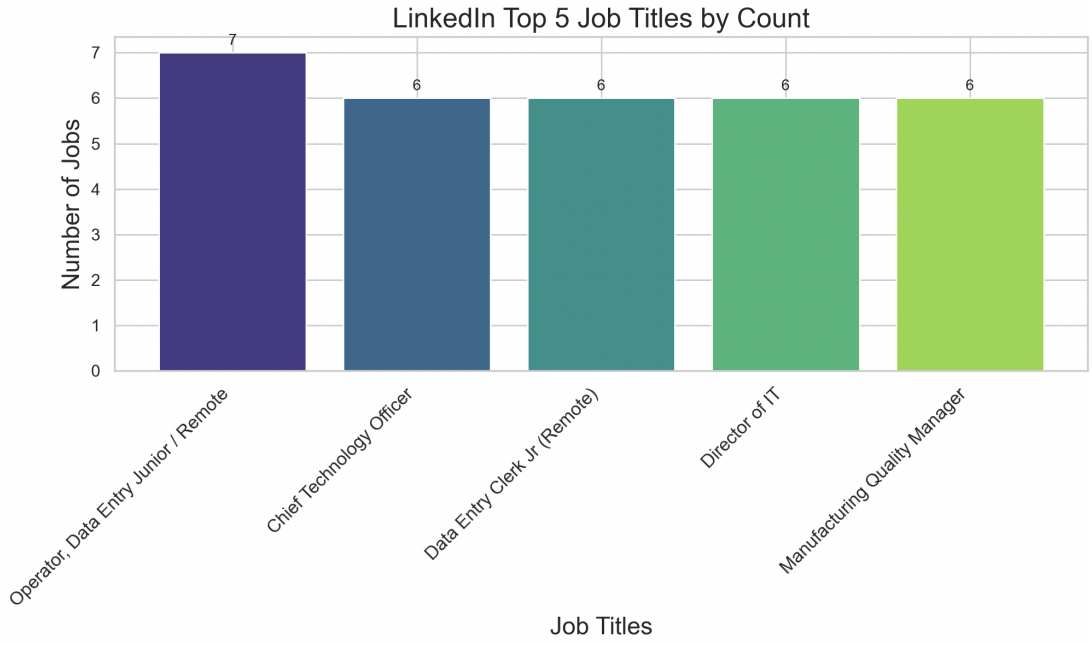
```
2023-12-09: Neutral - 0.2, Positive - 0.12, Negative - 0.04
```

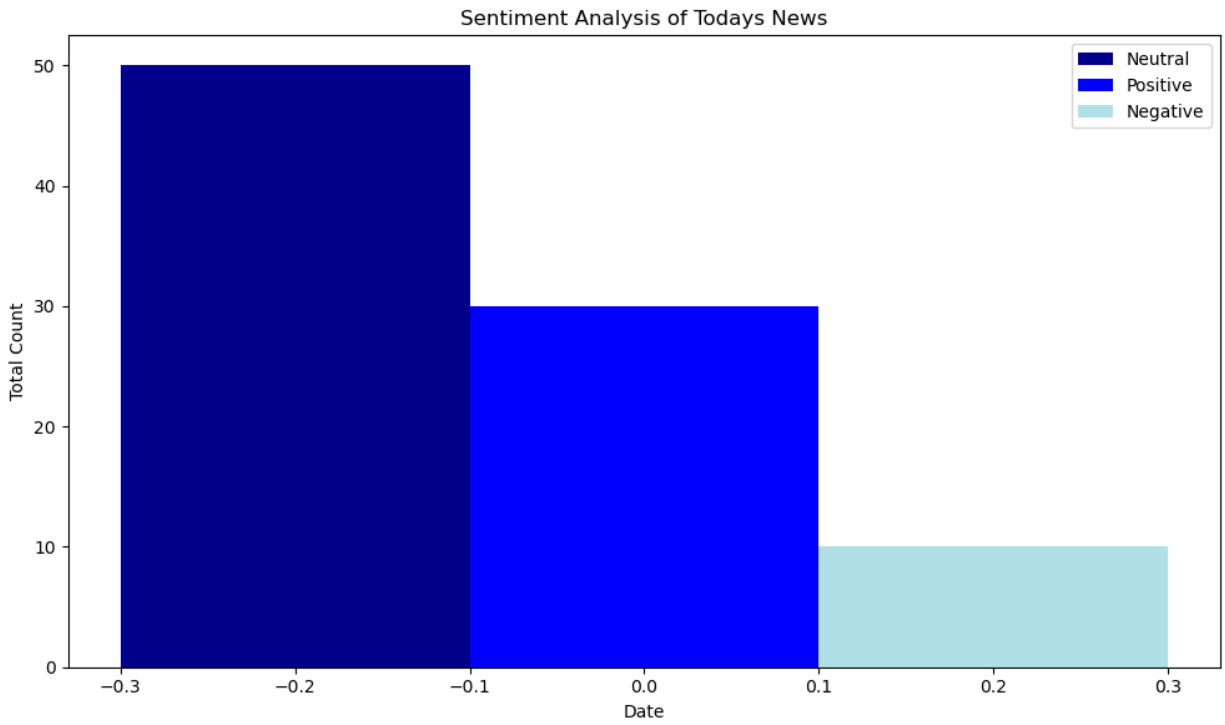
5. The visualization that you created (i.e. screen shot or image file)


Muse Job Listings Posted by Day of the Week
Number of Job Listings
0
5
10
15
20
25
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day of the Week


Muse Job Listings Posted by Month
Number of Job Listings
0
5
10
15
20
25
30
January
February
March
April
May
June
July
August
September
October
November
December
Month


LinkedIn Top 5 Job Locations by Count
Number of Jobs
0
2
4
6
8
9
8
6
6
6
New York, NY
San Francisco, CA
Arlington, TX
Boise, ID
Costa Mesa, CA
Locations


LinkedIn Top 5 Job Titles by Count
Number of Jobs
0
1
2
3
4
5
6
7
7
6
6
6
6
Operator, Data Entry Junior / Remote
Chief Technology Officer
Data Entry Clerk Jr (Remote)
Director of IT
Manufacturing Quality Manager
Job Titles



6. Instructions for running your code

For news api, the api key can be obtained from https://newsapi.org/ and we stored it in our config.py file. This should be the only api key you need. Then, begin with running the news_main.py, linkedin_main.py, and muse_main.py so that the news_database.db, linkedin_jobs.db, and muse_jobs.db are generated. From there, run data_migration.py so the unified_database.db is generated with all the data joined together. After that, run the news_calculation.py and news_viz.py, the linkedin_calculation_and_viz.py, and the muse_calculation.py and muse_viz.py files for the calculated results and associated graphs for each API. Finally, run linkedin_muse_joined.py to calculate and visualize the difference between the two job posting sites in the number of internship listings out of total listings.

7. Documentation for each function that you wrote. This includes describing the input and output for each function

news_main.py:

- Count_words function:
  - Input: content, words
  - Output: total number of content and words in string

- Main:
  - Input: database, search queries, word counts
  - Output: database updates

- Description:
    - These functions essentially integrate news data fetching, sentiment analysis, and database management into a single process, making it a comprehensive solution for news content analysis and storage.

news_db_manager.py:

- Create_connection function:
    - Input: a database file, db_file
    - Output: connection object to database

- Create_table function:
    - Input: conn (SQLite Connection Object) - The database connection.create_table_sql (String) - SQL query to create a table.
    - Output: Creates a table in the database. No return value.

- Description: These functions collectively manage the database setup and data insertion for a news content analysis system, including creating necessary tables and handling data entries for sources and posts.

news_api_client.py:

- fetch_news function:
    - Input: keyword, a string, used to query the News API
    - Output: connection object to database

- Description: this function retrieves a list of news articles related to a given keyword from the past month using the News API, returning this list or an empty list in case of an error.

news_viz.py:

- Function: fetch_total_sentiment_data(db_path)
    - Input: db_path (String) - The file path to the SQLite database.
    - Output: A list of tuples containing the date (timestamp), total neutral count, total positive count, and total negative count of news sentiments grouped by date.

- Function: plot_total_sentiment_data(data)
    - Input: data (List of Tuples) - The data retrieved from the database, consisting of dates and total counts of neutral, positive, and negative sentiments.
    - Output: None. This function generates and displays a bar graph visualizing the total counts of neutral, positive, and negative sentiments for each date in the provided data.

- Function: main()
    - Input: None. This function does not take any external inputs.

- Output: None. The main function orchestrates the data retrieval and plotting process by calling fetch_total_sentiment_data and plot_total_sentiment_data functions. It results in the visualization of the sentiment analysis data.

news_calculation.py:

- connect_to_db(db_path)
  - Input: db_path (String) - The file path to the SQLite database.
  - Output: Returns a connection object to the specified SQLite database.

- calculate_averages_and_totals(conn)
  - Input: conn (SQLite Connection Object) - The connection object to the SQLite database.
  - Output: Returns two lists of tuples. The first list contains the average neutral, positive, and negative counts for each date. The second list contains the total neutral, positive, and negative counts for each date.

- main()
  - Input: None. This function does not take any external inputs.
  - Output: None. This function orchestrates the process of connecting to the database, calculating averages and totals, and writing these results to two separate text files (averages_per_day.txt and totals_per_day.txt). It also prints a confirmation message after completing these tasks.

data_migration.py:

- fetch_data_from_db(db_path, query): Retrieves data from the specified SQLite database using the provided SQL query and returns it as a list of rows.

- insert_data_into_unified_db(db_path, data, insert_query): Inserts data into the specified SQLite database using the provided SQL insert query.

- create_table_in_unified_db function:
  - Input: db_path, create_table_sql
  - Output: Creates a table in the specified SQLite database using the provided SQL command.

- Fetch_data_from_db function:
  - Input: (db_path, query)
  - output: Retrieves data from the specified SQLite database using the provided SQL query and returns it as a list of rows.

- Insert_data_into_unified_db function:
  - Input: (db_path, data, insert_query)

    - Output: Inserts data into the specified SQLite database using the provided SQL insert query.

- Description: The script includes calls to these functions for creating and manipulating tables for LinkedIn, Muse, and News API data within a unified database. Data is fetched from source databases (linkedin-jobs.db, muse_jobs.db, news_database.db) and inserted into the unified_database.db for further processing and analysis.

linkedin_main.py:

- Get_or_insert function:
    - Input:
        - cursor: cursor object connected to the linkedin-jobs.db database
        - Table: string, the name of the table in the database
        - Column: string, the name of the column in the table
        - Value: the value being looked up/ inserted into the specified column
    - Output: Returns the ID of the record retrieved or the last inserted row ID if a new record is inserted.

- Linkedin_scraper function:
    - Input:
        - Webpage: string, the base URL of the LinkedIn job search pages.
        - Conn: SQLite connection object to the database
        - Cursor: SQLite cursor objects used for executing SQL queries
        - Jobs_to_fetch: integer, the total number of job postings to fetch. Default value is 25.
    - Output: None
    - Description: the function scrapes job information from LinkedIn job search pages and stores the relevant information in a SQLite database.

- Main:
    - Input: None
    - Output: None
    - Description: It establishes a connection to an SQLite database, creates necessary tables if they don't exist, and then initiates the LinkedIn job scraper to fetch job postings from the specified LinkedIn job search page. It also involves connecting to a database, defining tables, and invoking the LinkedIn scraper function to fetch job postings based on the provided URL. The jobs_to_fetch variable determines the number of job postings to fetch per run.

Linkedin_calculations_and_viz.py:

- Calculate_top_job_titles:
    - Input:
        - Cursor: cursor object used to execute SQL queries.
        - Limit: an optional parameter (default value is 5) specifying the maximum number of top job titles to retrieve

    - Output:
        - Top_job_titles: a list of tuples containing the top job titles and their corresponding job counts, ordered by count in descending order
    - Description: calculates and retrieves the top job titles for tech-related jobs in the U.S. posted on LinkedIn. It executes a SQL query to join the ‘titles’ and ‘job’ tables, grouping the results by job title and counting the occurrences. The result is ordered by the count of job postings in descending order and limited to the specific number of top titles. It then prints the top job titles to the console and returns the same information.
- Calculate_top_locations:
    - Input:
        - Cursor: cursor object used to execute SQL queries.
        - Limit: an optional parameter (default value is 5) specifying the maximum number of top locations to retrieve.
    - Output:
        - Top_locations: a list of tuples containing the top job locations and their corresponding job counts, ordered by count in descending order.
    - Description: calculates and retrieves the top job locations for tech-related jobs in the U.S. posted on LinkedIn. It executes a SQL query to join the ‘locations’ and ‘jobs’ tables, filtering for locations in “city, state” format, grouping the results by location, and counting the occurrences. The result is ordered by the count of job postings in descending order and limited to the specific number of top locations. It then prints the top job titles to the console and returns the same information.
- Plot_top_job_titles:
    - Input:
        - Data: a list of tuples, where each tuple represents a job title and its associated job count.
    - Output:
        - Visualization: a bar plot displaying the top job titles, where the x-axis represents job titles, the y-axis represents the number of jobs, and each bar is color-coded based on Viridis color palette. Job counts are also displayed at the top of each bar for better readability.
    - Description: Generates a bar plot to visualize the top job titles for tech-related jobs in the U.S. posted on LinkedIn. It uses the Seaborn and Matplotlib libraries to create an aesthetically pleasing visualization.
- Plot_top_locations:
    - Input:
        - Data: a list of tuples, where each tuple represents a job location in “city, state” format and its associated job count.
    - Output:
        - Visualization: a bar plot displaying the top job locations, where the x-axis represents job locations, the y-axis represents the number of jobs, and each bar is color-coded based on the Viridis color palette. Job counts are also on the top of each bar for better readability.

        - Description: Generates a bar plot to visualize the top job locations for tech-related jobs in the U.S. posted on LinkedIn. It uses the Seaborn and Matplotlib libraries to create an aesthetically pleasing visualization.
- Main:
    - Input: None
    - Output:
        - Visualization: this function generates 2 visualizations– two bar graphs where each bar is color coded based on the Viridis color palette. Job counts are displayed on the top of each bar.
            - Top Job Titles Plot: a bar plot showcasing the top job titles for tech-related jobs in the U.S. posted on LinkedIn. The x-axis represents job titles, and the y-axis represents the number of jobs.
            - Top job locations plot: a bar plot displaying the top job locations for tech-related jobs in the U.S. posted on LinkedIn. The x-axis represents job locations in “city, state” format, and the y-axis represents the number of jobs.
    - Description: Connects to the SQLite database ‘linkedin-jobs.db’, retrieves data using the specified queries, and generates visualizations based on the calculated results. This function handles the entire process by calling various other functions responsible for calculations and visualizations.

Muse_main.py:

- Get_last_processed_page function:
    - Input: cursor – cursor object connected to the muse_jobs.db database
    - Output: returns last processed page number as an integer if it exists, or if there is no record in the table of an id = 1, then returns 0
    - Description: retrieves the last processed page number from the job_postings table so that the 10-page search limit can be maintained in data collection for the muse_jobs.db

- Update_last_processed_page function:
    - Input: cursor, page_number
    - Output: N/A
    - Description: updates the last processed page number by inserting or replacing a data value in the job_postings table id=1 (if there is data with id=1, it’s replaced with this function, if not a new record is simply inserted)

Linkedin_muse_joined.py:

- Calculate_percentage:
    - Input:
        - Conn: SQLite database connection object
        - Table: name of the table in the SQLite database containing job titles
        - Title_column: column name in the specified table containing job titles
    - Output:

        - Percentage: the function returns a float representing the percentage of job titles containing the term ‘intern’ relative to the total job titles. If the total count is zero, the function returns 0.
    - Description: Calculates the percentage of job titles containing the term ‘intern’ in a specified table of the database. It gets the count of job titles containing ‘intern’ and the total count of job titles from the specified table and calculates and returns the percentage of ‘intern’ job titles relative to the total job titles. If the total count is zero, indicating an empty table, the function returns 0.
- Main function:
    - Input: None, the function fetches data from the specified tables and columns internally.
    - Output:
        - Pie-chart: generates a pie chart visualizing the percentage of internship job titles in LinkedIn (‘linkedin_titles’ and Muse (‘muse_job_postings’) and displays the chart.
    - Description: Calculates and visualizes the percentage of internship job titles in two different tables (‘linkedin_titles’ and ‘muse_job_postings’) within a SQLite database (‘unified_database.db’). Utilizes the ‘calculate_percentage’ function to determine the percentage of job titles containing the term ‘intern’ in each table. It then visualizes the results using a pie chart.

8. You must also clearly document all resources you used. The documentation should be of the following form:

Date	Issue Description	Location of Resource	Result
12/2/2023	We initially decided to use Reddit, Discord, and LinkedIn as our APIs but found out they were too complicated for us to handle. So, we used GitHub to look for free APIs.	https://github.com/public-apis/public-apis	We found Muse API and News API that are free, easy to use, and fit our purpose.
12/3/2023	We could not access News API data matching our keywords. We also didn’t know how to use the API key.	News API documentation: https://newsapi.org/docs	Read documentation, stored API key in a config file, and successfully created an API link.
12/3/2023	We could not call News API properly. We got error messages that turned	Office Hours - Michael and https://newsapi.org/	Created a new news API account and found out a way to grab data from API

	out to be due to maxing out API requests		without requesting it too many times
12/4/2023	Forgot how to create tables with SQL and encountered complexities and bugs. Example error message: sqlite3.OperationalError: table jobs has no column named location	https://www.w3schools.com/sql/sql_create_table.asp	Resolved SQL table issue. Also discovered that deleting and recreating tables solved the issue most of the time.
12/4/2023	Once we found the Muse API, we had to figure out how to use it as it was a website none of us had a significant amount of familiarity with.	https://www.themuse.com/developers/api/v2	We relied on this resource a fair amount to determine how to fetch specific data, such as the job posting date.
12/9/2023	We struggled to increment our data by only 25 each time, so we used ChatGPT as a debugging tool to pinpoint the part of our code that wasn't working correctly.	ChatGPT	ChatGPT told us what was wrong with our current code– we had an issue with differentiating rows to fetch versus pages to fetch.
12/9/2023	This is another GitHub repo complexity we didn't know how to resolve. We wanted to override remote repositories with our own repo.	https://stackoverflow.com/questions/6310208/overriding-remote-git-repository-with-my-repository	We successfully overrode the repo as needed.
12/9/2023	We encountered merge conflicts we did not know how to resolve, so we followed a tutorial we found on Google	https://unix.stackexchange.com/questions/181280/how-to-exit-a-git-merge-asking-for-commit-message	The merge conflicts were resolved by using the keyword "--force" a lot
12/9/2023	We did not know how to join data bases so	https://stackoverflow.com/questions/68247	Tables were successfully joined

	had to look for a tutorial. The main resources used were Stack Overflow and ChatGPT.	17/sqlite-how-do-you-join-tables-from-different-databases Prompt for ChatGPT: how to join databases in SQLite	
12/10/2023	Issue was not understanding final project instructions	Solved by asking a bunch of questions on piazza	Questions were resolved
12/10/2023	Merge conflict issues about a file: .DS_Store.	https://stackoverflow.com/questions/18393498/gitignore-all-the-ds-store-files-in-every-folder-and-subfolder	Stored .DS_Store in .gitignore and rm it from commit and resolve merge conflict. Had to do it multiple times