Formula 1 Driver Stats Helper

Overview of the application:

This application is an information retrieval system for Formula 1 (F1) races and drivers. It has a web interface for accessing the data related to F1 races and drivers. Users can retrieve detailed information about F1 races for any available year, race schedules, results, and analyses. Users can track and compare performances across different years through this web application.

In addition to race data, the application offers an in-depth view of driver profiles. Users can access individual driver profiles, including biographical information, career statistics, and recent news relevant to the driver. A key functionality of the application is its ability to aggregate and present relevant news articles. This feature integrates different information retrieval techniques to identify and rank news content based on its relevance to the selected driver.

How the application is implemented:

The application can be separated into a few parts: the frontend web GUI, backend web crawling, and relevance scoring. The backend web server initiates first, initializing the web crawler to extract data and subject it to a scoring function for the relevance assessment. Once the scored data is prepared, the frontend web retrieves it via HTTP GET requests. Finally, the received data is displayed on the frontend web GUI for user interaction.

Here is the breakdown of the different major parts of the application:

API:

We utilized the API from *http://ergast.com* and extracted the important race and driver information. This process is performed through an HTTP GET request with the help of the Axios package in node.js**.** With this API: **http://ergast.com/api/f1/{YEAR}.json**, we can change the variable {YEAR} according to the user's input from the website and adjust the data shown on the website accordingly. Since the data is divided into races, we also need to extract information from each race using the for-loop, where we extract race names, circuit names, driver names, constructors, and racers' time. In addition, we can also get drivers' information on this API: **http://ergast.com/api/f1/{YEAR}/drivers.json**, where we extract drivers' basic information, including names, date of birth, nationality, and permanent number.
All these data are displayed in a table in the frontend Web GUI to present to the user on the website.

Web crawlers:

With the **scrape_driver_news()** function, we used the Python package **BeautifulSoup** to perform the web crawling for us. This function is used primarily to web crawl/scrape the recent news posted from the main F1 website (www.formula1.com), and by using the driver's

first/last name, we can categorize each news/article into the drivers accordingly. We encountered two issues while developing this function: when performing web scraping, we could not extract all the pages from the F1 page due to the page number (**https://www.formula1.com/en/latest/all?page=**<page#>). To solve this problem, we use a for-loop to loop through the most recent pages (1~5) to obtain the most recent news about the drivers from the website. The second problem we encountered was to extract helpful information from the website. It took us some time to fully understand the structure of the data returned from the web crawler; the data contains many classes and different Start and End tags. We took many tries to extract the information we needed to present to the user successfully.

Scoring function, scoring():

The scoring function, **scoring()**, is designed to evaluate the relevance of F1 drivers to the news articles that are scraped from the F1 website. Its primary goal is to determine the extent to which a driver is associated with a given news, with a higher score indicating a closer relevance. The function includes different text processing and information retrieval techniques.

The function will pre-process the raw text data, which is the news article. The text pre-processing steps are tokenization, stopword removal, stemming, and lemmatization. Tokenization breaks the text into individual words, while stopword removal eliminates common words that are unlikely to contribute to the relevance assessment. Stemming and lemmatization further refine the words to their base or root forms.

For the weighting part, the term frequency-inverse document frequency (TF-IDF) method is used with a custom title weight multiplier. TF-IDF is a statistical measure used to evaluate how important a word is to a document in a collection of documents. It achieves this by increasing proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus, which helps to control for the fact that some words are generally more common than others.

The custom technique used in this function is the title weight multiplier. The title of a news article often provides a concise and focused summary of its content. Titles are designed to capture the essence of the article, frequently mentioning key individuals. By giving the title a higher weight, the function prioritizes it more, rather than equating title matches with those in the body of the news content. Therefore, if a driver's name appears in the title, it greatly influences the relevance score, reflecting the high likelihood that the article is closely related to that driver. The title is given a weight that inversely correlates with its length relative to the article's total length (title and content combined). Specifically, the weight starts at a baseline of 1 and then scales up based on the title's relative length and the specified title_importance factor. If title_importance is set to 0, the title's weight remains 1. This allows us to have the option to remove the custom weight for the title or increase the weight of the title.

Lastly, the function computes the cosine similarity between the TF-IDF vectors of the drivers' names and the articles. This similarity score reflects how closely related a driver is to each

article. The drivers are then ranked for each article based on these scores. So the most relevant news for each driver can be obtained.

Documentation of the usage of the application:

Package installation:

The setup starts with ensuring that all necessary Python packages (flask, flask_cors, request, nltk, ssl, bs4, pandas, sklearn) are installed on your local computer. This can be done by executing the following commands in your terminal:

- Pip install flask
- Pip install flask_cors
- Pip install request
- Pip install nltk
- Pip install ssl
- Pip install BeautifulSoup

Backend setup:

1. Change to the correct directory by executing `*cd final*`, followed by `*cd backend*` in your terminal
2. Start the application's backend server by running the command `*python main.py*`.
3. After executing the above steps, a "DEBUGGER PIN ******" message will appear in your terminal. At this point, open your web browser and navigate to http://127.0.0.1:5000/api/data.
4. Once the webpage runs, the message "STARTING UP THE SERVER" should be displayed in your terminal. It's important to wait for approximately 5 to 10 minutes at this stage to allow web crawlers to extract the necessary information.
5. Upon successful completion of these steps, the information should be visible in JSON formats on your browser, and the terminal should display "SERVER IS READY TO HANDLE REQUEST". **Remember to keep this terminal open and start a new terminal session for the next steps.**

Frontend Setup:

6. To set up the frontend, navigate to the correct directory with `*cd final*` and then `*cd frontend*`.
7. Verify that you have the correct version of Node.js installed by running `*node -v*` in your terminal. It should return a recent version number.
8. Next, execute `*npm install*` followed by `*npm start*` to install all necessary dependencies and start the frontend of the application.
9. The web page should launch automatically in your default browser if everything is set up correctly. If it does not start automatically, manually navigate to http://localhost:3000/ in your browser.

By following these instructions, you should be able to run the web application on your local machine.

Contribution of each team member:

- Alex Che:
    - Data Processing
    - Web-crawling implementation
    - Demo
    - Documentation
- Xiaoyang Cai:
    - Frontend / Backend Development
    - Connection
    - Demo
    - Documentation
- Sze Chung (Steven) Wong:
    - Text data processing
    - Information retrieval implementation (scoring and ranking function design)
    - Demo
    - Documentation
- Savya:
    - Driver’s biographies crawling
    - Documentation