Web scraper logic:

Current Ver: 1.0.0

Web scraping method:

* I send a search query using the google search python library, the query is dynamically generated.
* Current search query: “Company\_Name” AI usage within the company, I then return the top 6 results and explore those websites.
  + This is just for testing on a wide variety of websites before the query is narrowed specifically for foundational models.
* I have set up a stand-alone proxy service, each request is randomly tunneled through one proxy from a set so the website and google does not restrict the request number.
* The query is initially sent through the simple requests library for speed, if the request is denied or restricted it switches to a second method, this uses selenium to create its own browser and sends the request through that. Selenium is a lot slower, but it is usually never blocked.
* The Selenium browser also searches with “Ublock” and “Bypass paywalls” browser extensions, these addons strip out ads and removes subscription blocks respectively.
  + Note that I have not used these ad-ons when using requests so a lot of junk is scraped, will be fixed in the next release.
* If any request is blocked, the code hits a timeout function and attempts the request again in 10 seconds after changing the proxy.
  + Note that this works currently but with how I understand google proxy detection works this is not a finalized solution, more work will need to be done.
* There is a mixture of different error handling methods within the code, corresponding to various HTTPS return conditions.
* An initial cleaning is done on the website text data using Beautiful Soup removing hyperlinks and other obvious junk data returning only text.
* Total run time for a single query is about 3 seconds, so pretty fast.

Text extraction:

* Given the text returned by the web scraping method per website.
* I first tokenize the text using nltk , I do some initial cleaning removing unnecessary words and spacing issues.
* If a sentence in the text contains a certain target word, I would extract that given sentence.
* Current target words: ["Virtual Assistant", "GPT", "ChatGPT", "AI"]
  + Note that this is again just for testing, the list of target words will have to be improved and biased specifically to foundational models.
* Once the list of sentences is extracted, I use the T5-small text summarizer to return a dynamic summary of the article now concentrating on the AI aspects of the article.

The company, URL and the summary are now extracted and written into a table.

I have the run the code for the top 50 companies from fortune 500 companies list.

Improvements for Ver 1.1.0

* Tuning the target words and query to extract more relevant websites.
* Adding a new column that predicts the exact foundational model being used.
  + This will be extracted directly from a mixture of the summary and the main website text.
* Little more automation to be done.