**Project description:** Political Tweet Analyzer compares and visualizes the frequency at which major politicians from both parties mention a specified keyword. The user can choose a specific politician to see the individual trends (of mentioning the keyword) and see similar tweets relating to the politician's tweets about the keyboard.

**Competitive analysis:** There are quite a few of Tweet analyzers that graph some aspect of tweets — positivity, sentiment, spread, or just tweet frequency of a specified user. These tweet sentiment analyzers have been used in some research papers linguistically analyze the content of political tweets, so my program is somewhat similar in that it focuses on what words the politicians tweet, but it won't analyze the meaning of these tweets. For example, a politician may allude to the election but not use the word "election," and my program wouldn't count that in its analysis. I also haven't seen any programs compare a specific keyword's frequency between different political users, so that is also a potential difference.

**Structural plan:**

* tweet\_scraping.py is for the Tweet scraping part. Anything that uses snscrape or Tweepy goes in this file, so it does the behind-the-scenes work for the app.
* political\_visualization\_vis.py handles the cmu\_112\_graphics portion, or the visualization part. It stores the list of politicians and uses them to draw various graphics (plots, buttons, etc.)
* politicians.py is just to store my custom objects, Politician and Button. I find it easier to have a separate file for this instead of cramming it in political\_visualization\_vis.py
* main.py is just the main file that handles the running of the app
* twitter\_auth\_data.json has my authorization keys that tweet\_scraping.py needs to use to access Twitter API

**Algorithmic plan:**

* Button sizes: The more counts, the bigger the button, but I don't want the buttons to start overlapping, so I find the greatest count in a separate function; whichever button has the greatest count fills up the "cell" (each cell size = divide the canvas into 12). Then I loop through the other politicians' counts and calculate the radius by using each count's proportion to the greatest count.
* Individual plots: I store the button that was pressed in an app variable, then loop through 5-day increments from 30 days ago (30 days ago, 25, 20, all the way to the present day) and store the counts of the relevant tweets (with given politician and keyword) in a list. Because this list has cumulative frequencies, I loop over the list to find the differences and append them to a new list (so it's no longer frequencies). I find the max value so I know how to scale my y-axis, then draw that on the y axis. Then I loop through each value in the new list to plot each point, with the y-values being proportional to the max y value.
* Similar tweets by keyword: For each tweet object in the set of relevant tweets (matching a given politician and keyword), I use Tweepy to access the mentions and quoted tweet data and find the user who was mentioned/quoted the most, then search with this repeated mentioned/quoted user as the query (with the same keyword). If there is no most-quoted/mentioned user, I randomly get a mentioned/quoted user and search their tweets with the keyword.

**Timeline plan:** I hope to finish the comparison aspect (with the different buttons) and the individual plots by TP1, then finish the similar tweet algorithm part by TP2, then do the detailing (profile pictures for the buttons, etc.) by TP3.

**Version control plan:** I'm using GitHub for version control and pushing with every big change or new feature (i.e. buttons, plots)

A screenshot of a cell phone

Description automatically generated

**Module list:**

* snscrape
* Tweepy (wrapper for official Twitter API)
* cmu\_112\_graphics

TP 2 update

1. Caching (pre-scraping) tweets in a JSON file to speed things up

* Scraping tweets from the past year and storing them in a JSON file
* When app is run, app.updateJson() checks the last scraped date and scrapes new tweets if necessary and stores them back in the file

2. Different modes for different screens

3. Similarity mode

* Get list that matches user and keyword from a date
* Get a random tweet from the list and loop through the words to search for potential keywords (longer than some length, title case)
* Find new keyword from these potential keywords by getting the keyword with the greatest frequency