Distance Arbitrage aka “Distarb”

Pobody’s Nerfect!

By Diego, MSD 2022

Table of Contents

[Introduction 3](#_Toc121072536)

[Background:  What's wrong with existing solutions? 4](#_Toc121072537)

[Design Choices 5](#_Toc121072538)

[Organisation and Design 6](#_Toc121072539)

[Class diagram 6](#_Toc121072540)

[Libraries 7](#_Toc121072541)

[Results 8](#_Toc121072542)

[Landing Page of the app 8](#_Toc121072543)

[User input 9](#_Toc121072544)

[Edit Distance 11](#_Toc121072545)

[Senate Scraper 13](#_Toc121072546)

[Error Checking 14](#_Toc121072547)

[Successes 16](#_Toc121072548)

[Failures 16](#_Toc121072549)

[Conclusions 18](#_Toc121072550)

[Works Cited 20](#_Toc121072551)

# Introduction

This project is the realisation of an idea I had years ago: what if you make stock picks based off of user input mistakes, such as fat fingering or misreading the stock which they are trying to purchase. To do this, I would need to find a way to

1. Keep a list of stocks/companies available on the NYSE.

2. Find the edit distance of those companies of those companies based on both substitution/deletion, as well as accounting for the physical distance of those keys on the keyboard.

3. And finally as a fun aside, look at the stocks which members of the senate are buying and selling and visualising the value of those companies after executing the buy/sell order.

# Background:  What's wrong with existing solutions?

The current problem with existing solutions is that there is an issue of cost, as well as an overabundance of features. By their nature, they exist for the purpose of making the user money. The economy is driven to make more wealth, and the money poured into technology to make this “better” is in the trillions of dollars. A single Bloomberg Terminal, for example, is considered the gold standard of real-time stock tracking technology can have a per user cost of US$20,000 per year (Martel, 2022). Koyfin, a free alternative to the Bloomberg Terminal does not offer any sort of charting ability to the user. Which anyone who has looked at a stock’s performance will tell you that is a major deficiency. In addition, current programs offered by companies for stock tracking may provide tons of tools a user may want, but also provide many that a user might not need/want/care about. However, I have yet to see one like mine that is betting on humans being, well, human and making a mistake. Trying to figure out what ways a human may err is an interesting problem to me.

# Design Choices

For the sake of my sanity, I decided to use as much python 3 as I could. The fact that it has so many libraries, made it an obvious choice as a place to start. For the Graphical User Interface, my project relies on PyQt5. This is a high-level Python3 API that is built on C++, making it a fast and easy to use option for designing a GUI. In tandem with QT Designer, making a clean (opinions may vary) GUI is a breeze. This however would be an issue I will discuss further in the conclusion section.

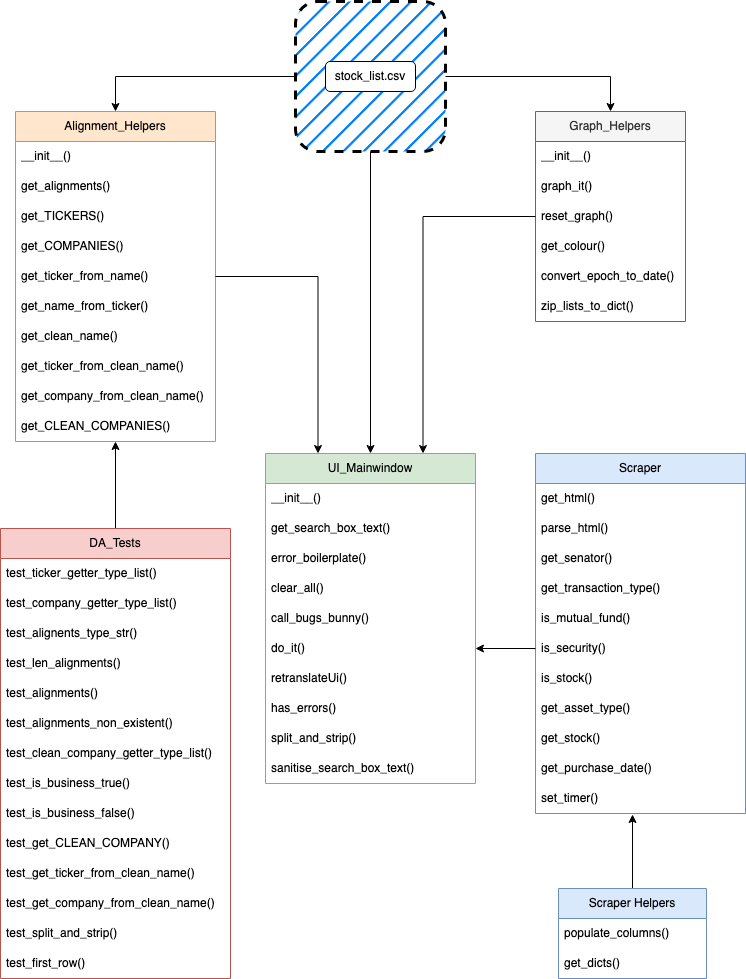
The stock data comes from yfinance. This API gets its data from Yahoo Finance, and is open source. It returns Pandas dataframes, which make it easy to use and extract data from. The problem with it however, is that the web scraping aspect of it is dependent on Yahoo Finance not changing the div/class/etc names. of their HTML.

The star of the show is the clavier library by Max Halford (Halford, n.d.). It was an implementation of the algorithm I needed, that takes both physical distance of keys as well as edit distance of two strings. I did a quick look around the internet and didn’t see where else this had been implanted and for what purpose.

And to manage all those packages, I used Anaconda. This provided a simple way to download/manage packages, as well as provide a place I can upload environments to the internet to help facilitate troubleshooting package management for users. A subject which is often a thorn in the side of anyone who has used python.

# Organisation and Design

# Class diagram

**

# Libraries

While this is not an exhaustive list of libraries that I used, these are the ones that were of the most use to me in the development of Distance Arbitrage.

Yfinance – making API calls to Yahoo Finance to get stock data

Clavier -- to get the keyboard aware edit distances.

BeautifulSoup -- for web scraping.

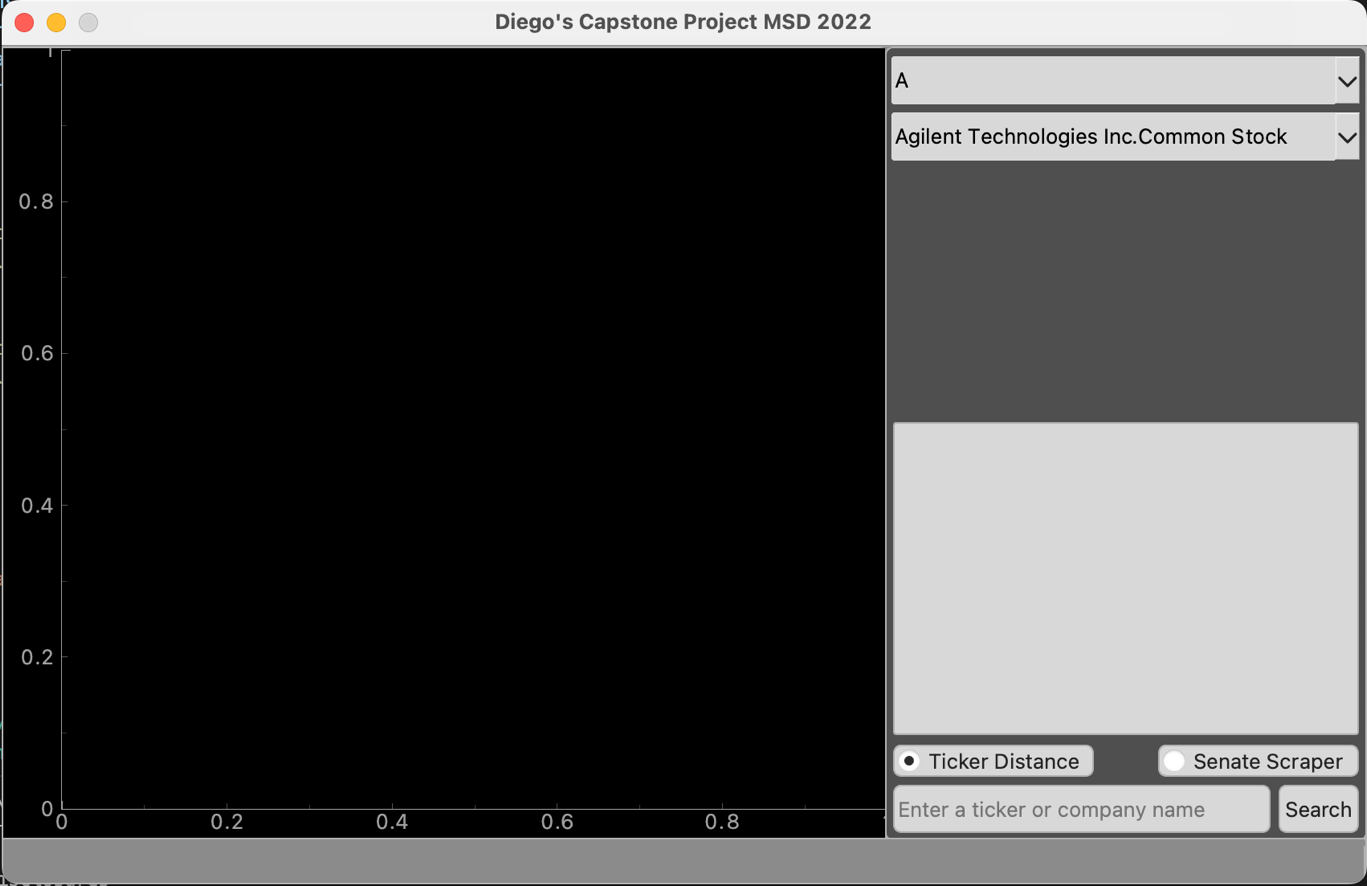
Pandas -- for data manipulation.

Urllib -- for web requests.

Pendulum -- for date manipulation.

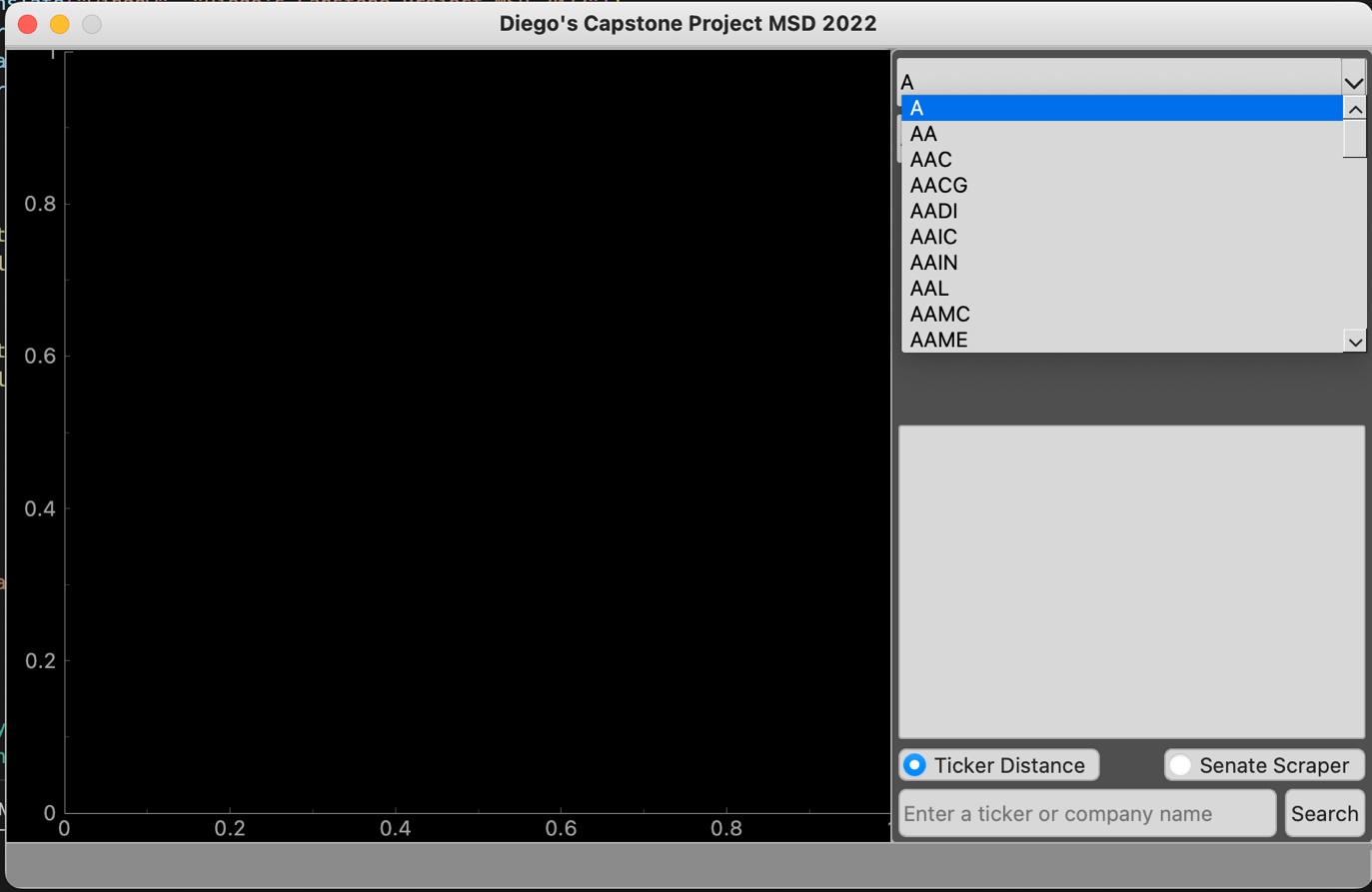
# Results

# Landing Page of the app

**

The program runs with a simple interface allowing the user to quickly and easily select the functionality they desire. The default is the ticker edit distance because that is as I see it, the primary use for the app.

# User input



Graphical user interface, text, application

Description automatically generated

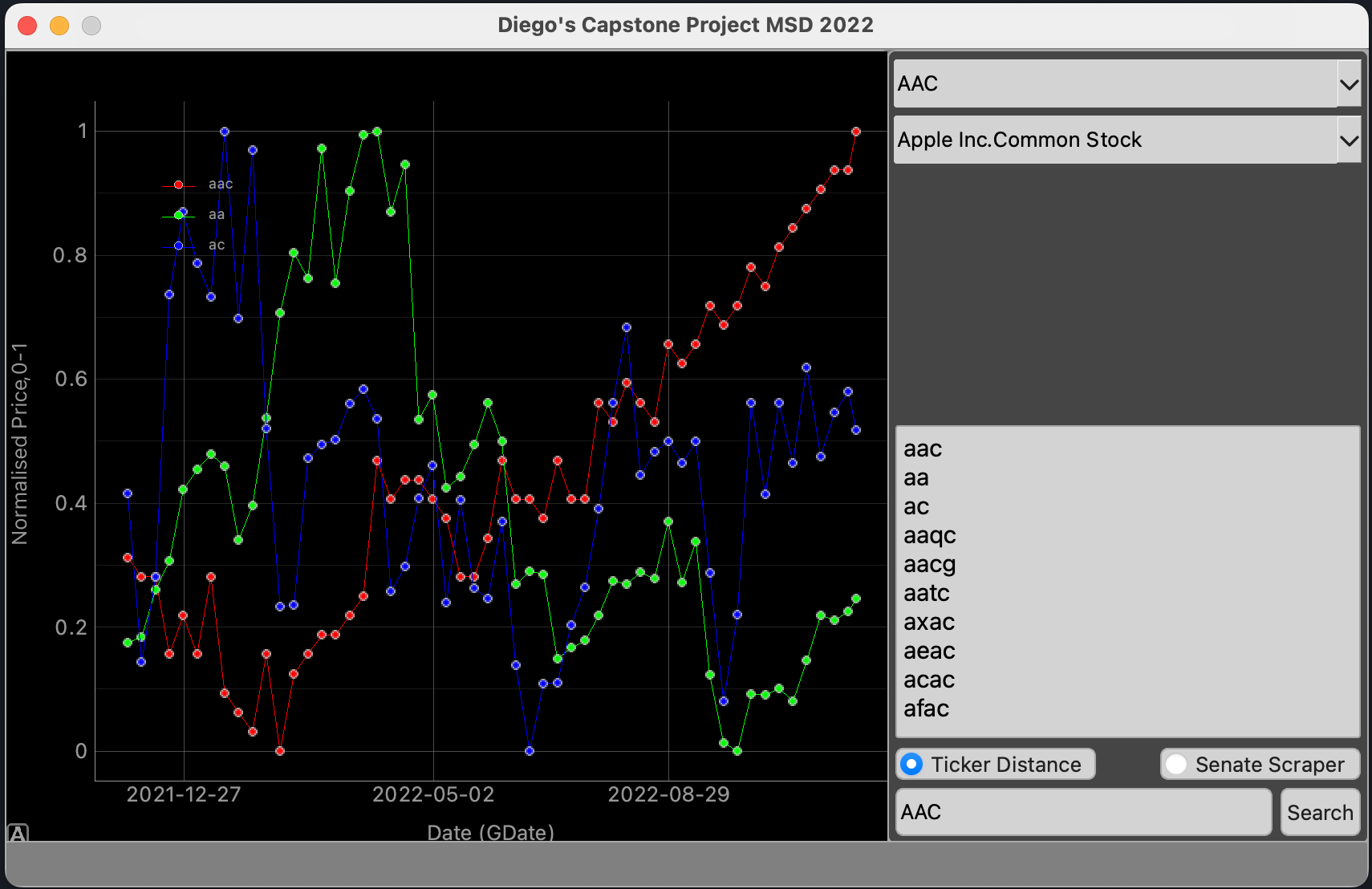
Graphical user interface, application

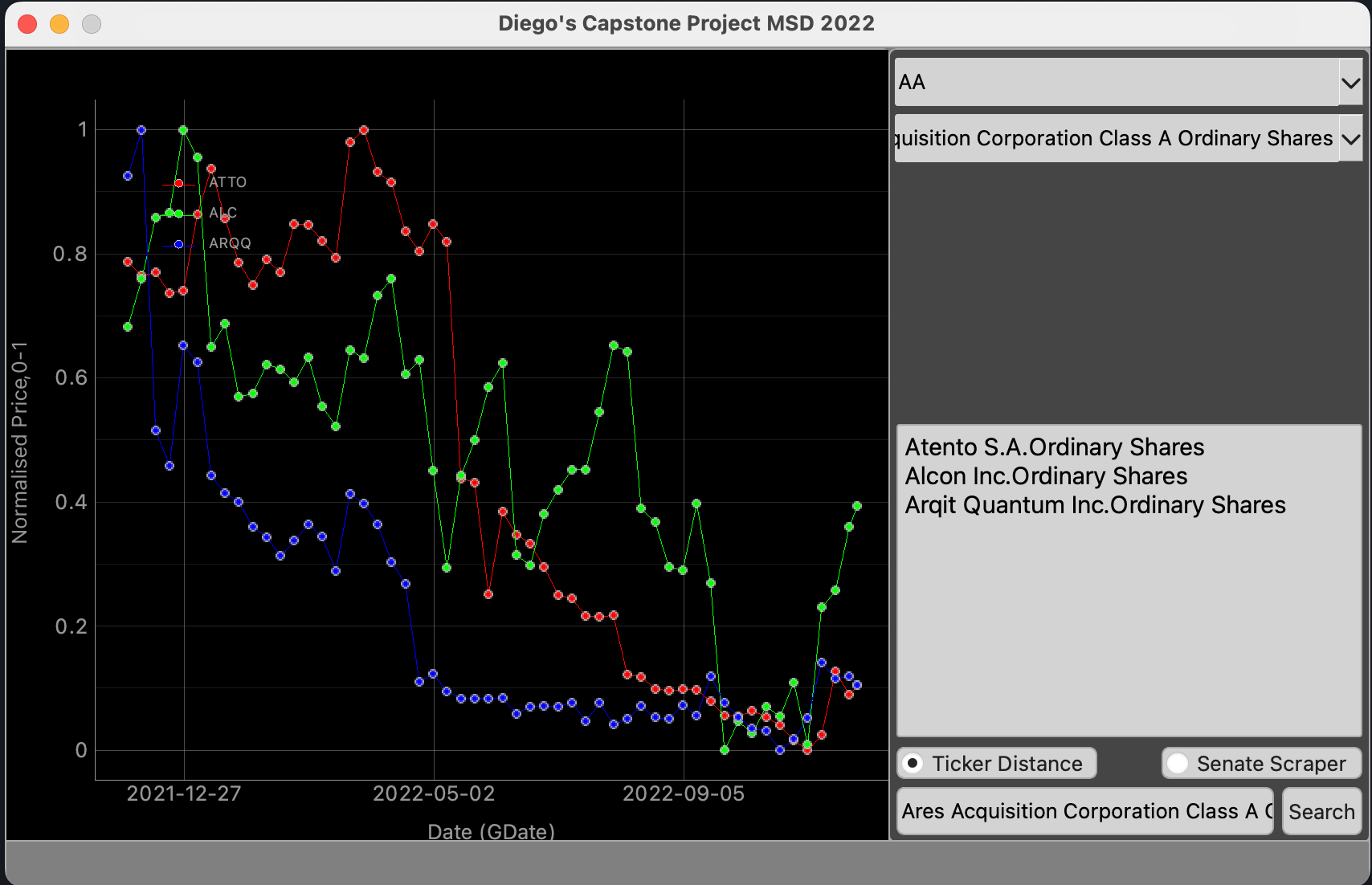
Description automatically generated

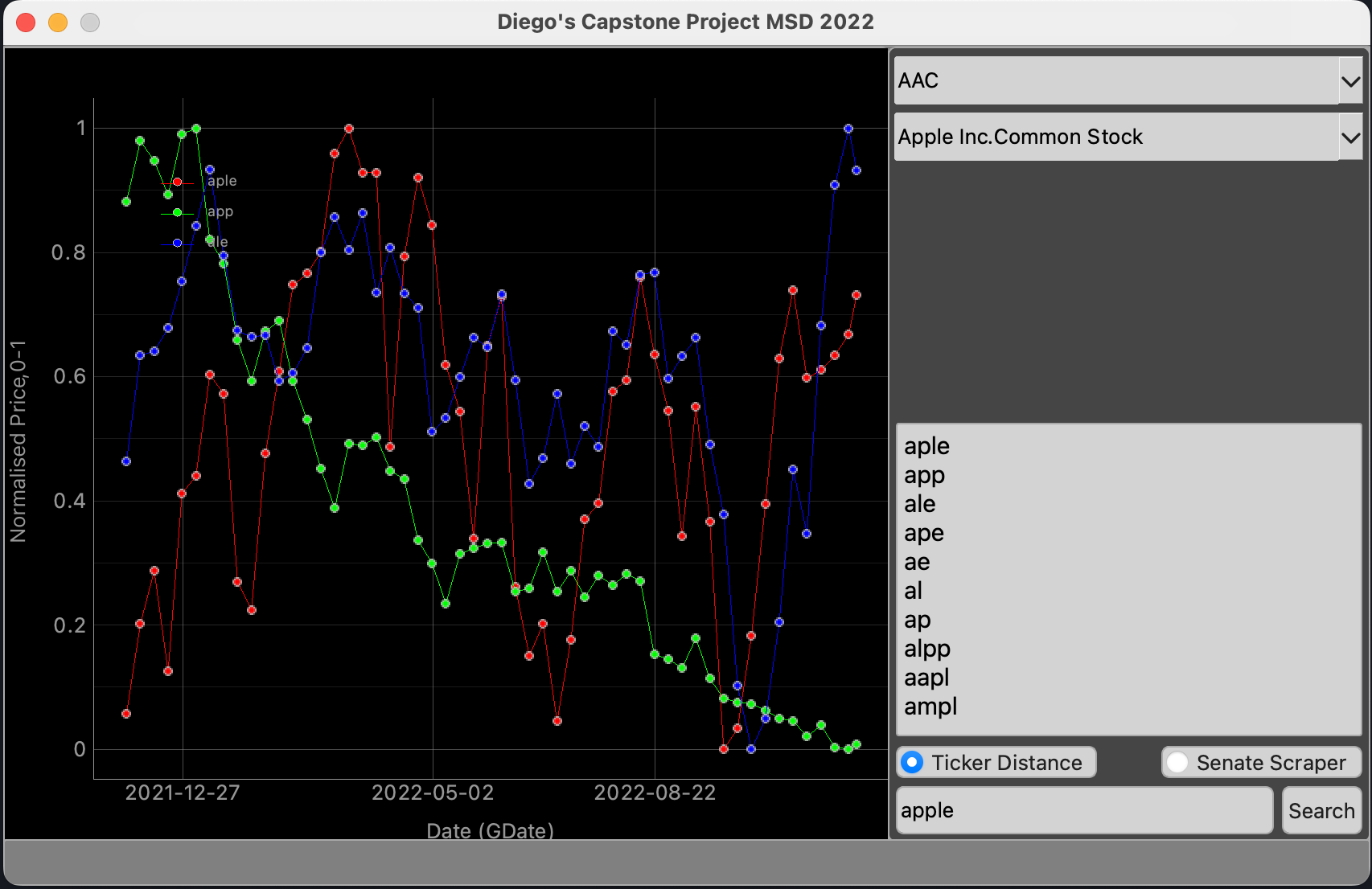
The user is able to enter in their searches in two ways:

1. The user can use the dropdown menus at the top right of the application, and scroll through a list of stock tickers on the NYSE, or the registered company names. I wanted this option because as is the core of this program, mistakes can be made when user input is concerned, and removing typing from the equation when possible is a reasonable solution to the problem.
2. The user is able to manually enter a ticker/company name. This is so the user can search for companies that are not yet traded on the exchange that are about to IPO, as well as for companies that have a legal name vs colloquial name. Before Facebook became Meta, the stock ticker was FB, legally they were TheFacebook, Inc., then Facebook Inc., and colloquially we just call them Facebook. Giving the user the ability to search for what they feel is the most common name of a company adds more flexibility.

# Edit Distance

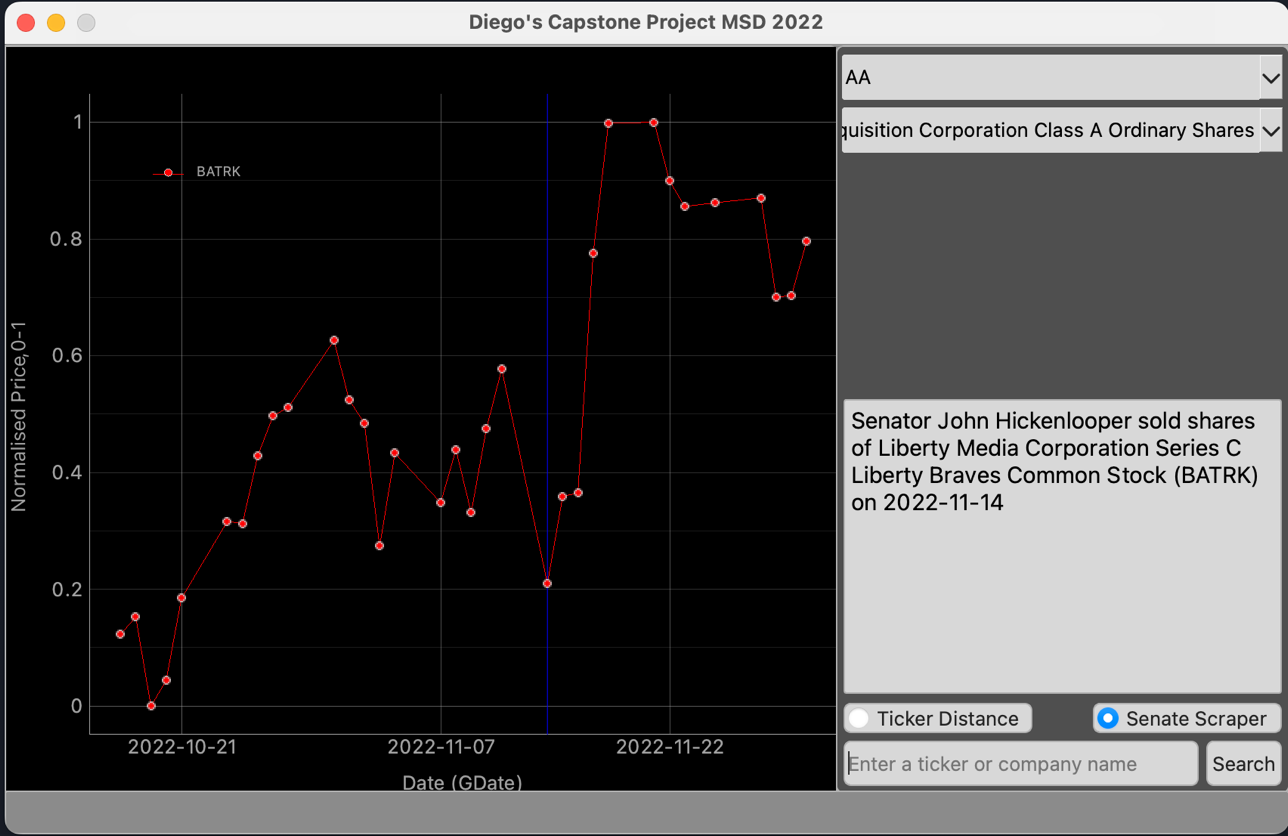






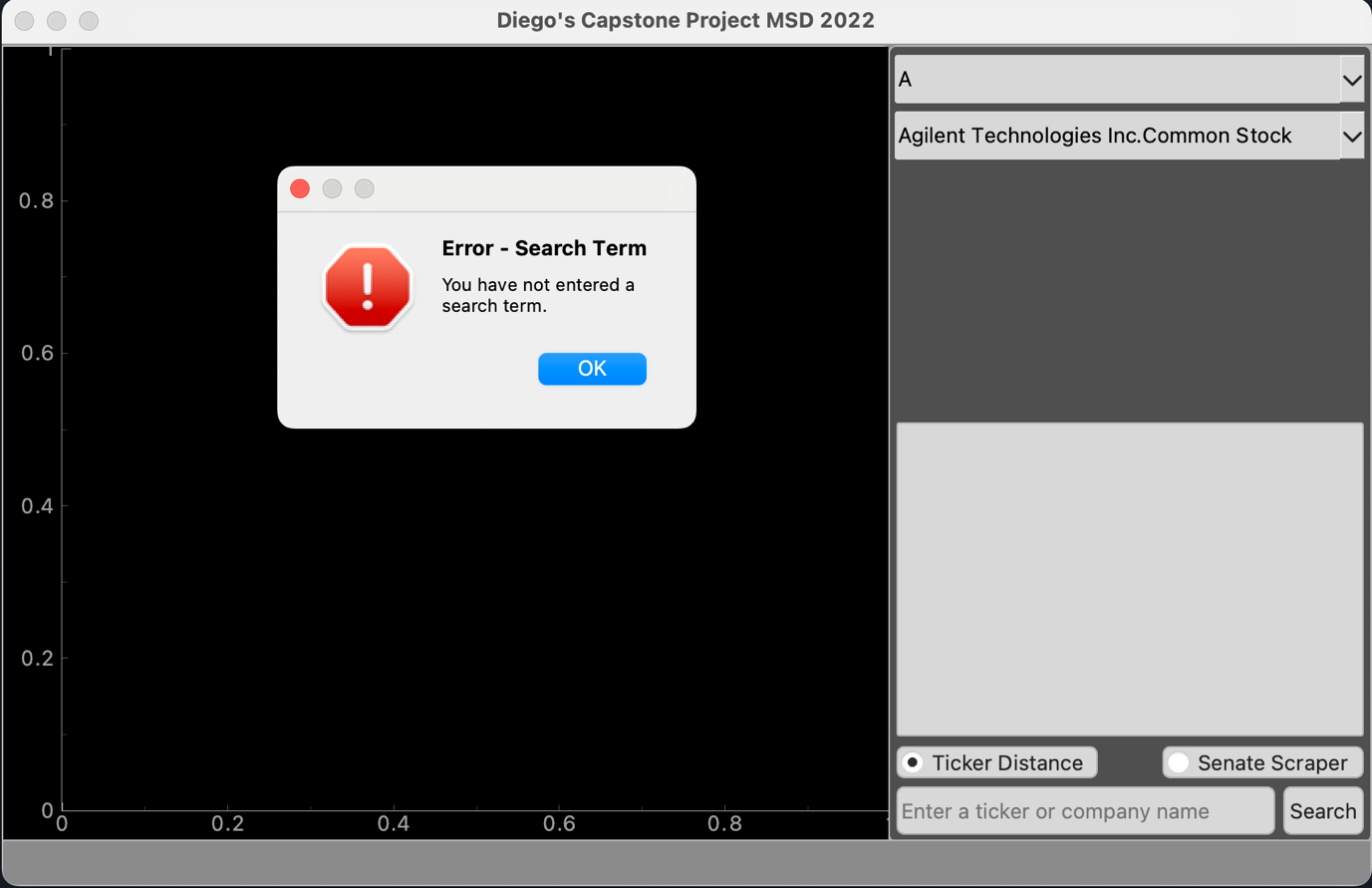
When the user enters their search term, the application tells the user the best matches based on a combination of edit and Euclidean distance of the stock and keyboard keys, and then graphs the normalised price data for the stock they searched for as well as the top two results. I chose to normalise the data because a detail I forgot to consider at first is the possibility that stocks may be wildly different prices. A silly oversite, but one that needed to be solved to make the app more user friendly and clear. With normalised prices, all the stock prices can be displayed cleanly in the window.

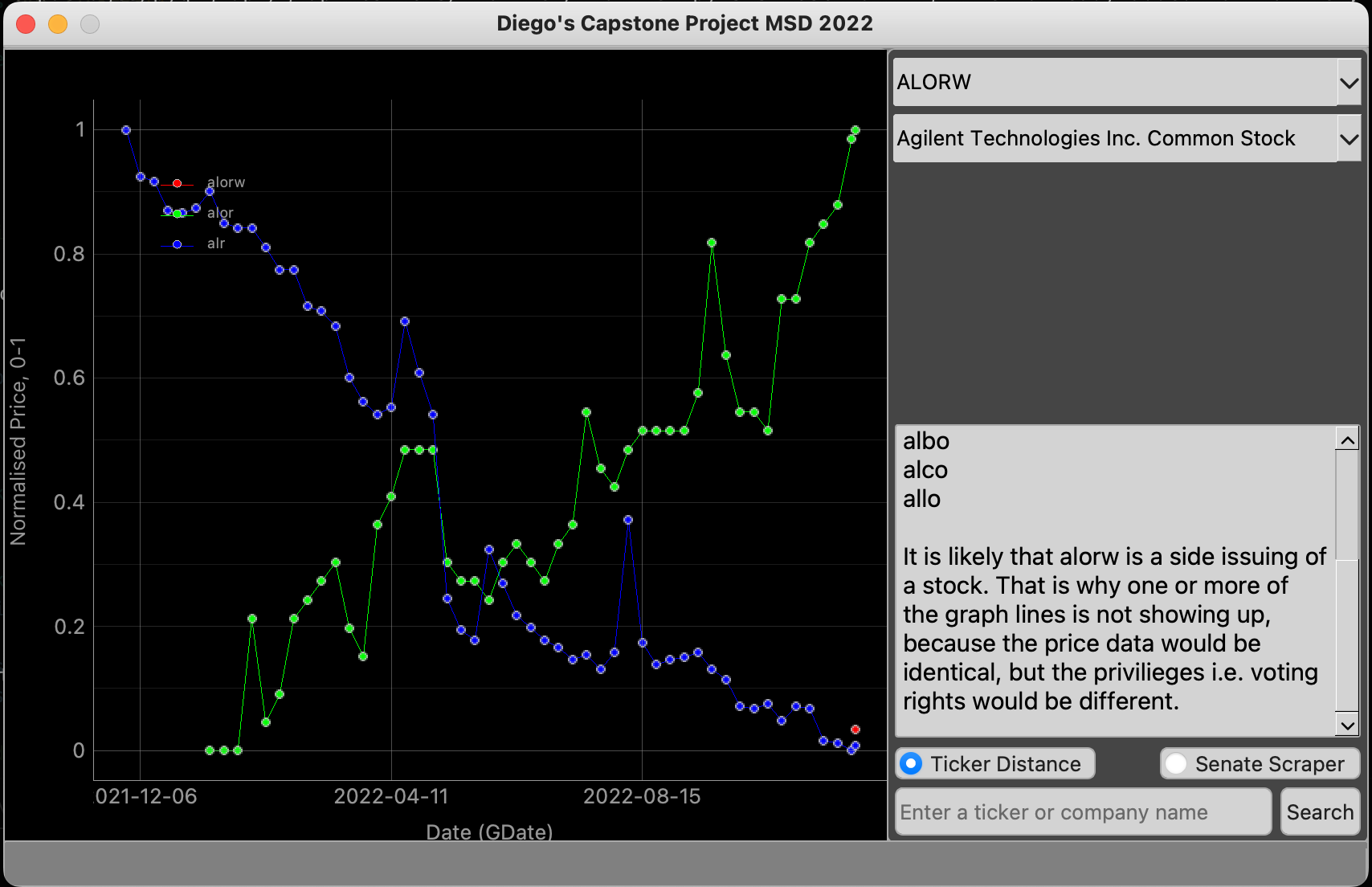
# Senate Scraper



The program has a web scraper that scrapes the SEC’s website. Senators are required by law to disclose what financial transactions they make. Ideally this would help deter them from capitalising on their ability to control the market. This app searches for the most recent stock transaction, and reports to the user the “who”, “what”, and “when” of the transaction. The program stores the entirety of the data from the SEC website, allowing for future versions to perform more data analysis. More of this in the conclusion. Distarb then charts the price data of the stock from four weeks before the purchase, which is marked by the blue line (difficult to see in the screenshot, sorry), until the date the scraper was used.

# Error Checking





A concerted effort was made to make user errors as clear as possible. For the latter, rather than take the time to sanitise my ticker list to look for every possible instance of multiple stock issuances, the program just explains why there are not as many price graphs the user might expect in the case where price data is identical between any number of stocks being checked.

# Successes

I think a lot of things went really well in this project. Python helped in making the code self-documenting, PyQt5 made for a fast GUI because of the underlying code written in C++. I think the app having a simple front end makes it very user friendly too. For the SEC scraper, I was concerned for a time that the SEC website would eventually stop me from scraping. But, because the user likely will only be scraping once, maybe twice, there is enough time between scrapes, that this is not an issue. For further development, the scraper downloads and stores everything from the webpage. So further analysis that may be done on the data can work locally.

# Failures

Something I am not happy with is how the stock legend displays. The legend is in the top left, and depending on the stock activity, it gets muddled in with the price graph. I was unable to find a way to reliably move it away from the lines to be readable. Having the list of edit distances to the side helps somewhat with understanding. But, it is less than ideal. I acknowledge that this issue may be operator error and PyQt5 has functionality for this that I just was not able to find.

Graphical user interface

Description automatically generated

Something else I am not entirely satisfied with is the web scraper is hit or miss for if it will run from the packaged version. I believe that it has something to do with Apple protections of users against apps made by unidentified developers coupled with the nature of the web scraper going out to the internet. I have a workaround using bash and the command line for running the application. The details of how to use it are laid out in the README section “Why doesn’t this work?”. This is less than ideal and rather disappointing. I have also provided a clear walkthrough and links to everything a user might need in the README in the event that all else fails. I am sure that with more time I could create a more elegant i.e. user friendly solution. However, I deem this good faith attempt to be good enough for now.

# Conclusions

I believe that I have made the right design choices. An issue I may have later that might change my mind is maintaining the app in the event that the HTML changes on the site that it scrapes. There is not much that I can do to mitigate the amount of work that would need to be done to keep the HTML in the same format, as that is decided on by whoever is developing the website. However, something that I think might be interesting and more user friendly is to make the application a web app. I think that for future versions of this app, I will adapt it to a web app using flask. This will allow me to continue using python libraries and a lot of the code I have written, without worrying as much about packaging the end product.

As mentioned above in *design choices* there are some potential issues regarding PyQt5 in the even that I try and make my app anything other than open source (Although as of now, it is licensed as open source under an MIT license). If I want to monetise my app, I would need to pay a rather steep fee to continue using PyQt5. Given that one of the industry problems I discussed is the cost of such applications, I would be a hypocrite if I tried to charge people a large amount to use it. Generally speaking, I am uncomfortable with the idea of charging any amount for a financial service that is built with helping people in mind. That said, I do have to put food on the table. Reworking Distarb as a webapp would be a good compromise to keep costs down for both the user and myself, and still provide a useful service.

Eventually, I would love to implement machine learning and run millions of tests on thousands of past IPOs against the entirety of the stock tickers list, weighing each key distance and each addition/deletion, and each price point (i.e. whether we are more prone to mistakes on a stock with a higher or lower price point) to performances of similar tickers until my program had determined in what way we humans are most prone to error and could return stocks that fit that profile. The part of this that would be easy is that the code is built to easily adjust weights of adds/deletes in edit distance. However, I do not believe I will be able to make that many API calls using Yfinance in a narrow window that machine learning can make calls while adjusting the weights. That is a problem I can try and solve another time. What I like about this project is that there are heaps of opportunities to try something cool and interesting that I have learned in MSD.

# Works Cited

(n.d.). Retrieved from pypi.org: https://pypi.org/project/yfinance/

Halford, M. (n.d.). *clavier*. Retrieved from github.com: https://github.com/MaxHalford/clavier

*KindPNG*. (n.d.). Retrieved from https://www.kindpng.com/downpng/iRJiiJh\_stonks-meme-transparent-hd-png-download/

Martel, J. (2022, July 05). *investopedia*. Retrieved from https://www.investopedia.com/best-bloomberg-terminal-alternatives-5114517

SAMUELSSON, A. (2017). *Weighting Edit Distance to Improve Spelling Correction in Music Entity Search.* KTH Royal Institute of Technology , Computer Science, Stockholm.

Securities and Exchance Commision. (n.d.). *Latest Senate Stock Disclosures*. Retrieved from https://sec.report/Senate-Stock-Disclosures

