Charting Platform Python

Goal

Build a Python program capable of displaying and plotting charts, allowing users to invest, and using AI to make informed decisions to help users invest. The charting function allows users to see any stock in the market charted with relevant, up to date data in seconds. The investing function allows users to invest in their favorite stocks with paper money while providing AI trained insights to help them make better decisions (based on Tree Regression Model). The Analytics mode lets users see advanced portfolio analytics as the days go by, letting them improve their portfolio management skills as the days go by.

I've been learning Python in my college computer science course, and I've wanted to apply to something I'm very passionate about — stocks. The project will also give me experience with using Tkinter to run graphics and will give me experience with using an outside API.

Overview

11-11-2021

The game will be developed in Python. It will take place in a Tkinter window (using Carnegie Mellon University's version of Tkinter called cmu_112_graphics). The console will be able to take user input from clicks to text input through inbuilt functions. The program will be developed to make sure users don't break it with error messages. The AI will be based on Tree Regression from the scikitlearn module on Python.

The project will follow a pipeline with goals as follows:

Date	Goals
11/13 (TP0)	- Learn how to use the requests module in Python - Learn how the pandas module in Python - Get the UI for plotting set up
11/18 (TP1)	- UI w/plots - candlestick plot + bar graph **-** panning/zooming features - Paper Trader UI setup - text entry and changing ability
11/23 (TP2) - Minimum Viable Product	- Panning/Zooming for graphs on click - Game UI completely done and working - Portfolio Page display - Buy and Sell pages with dynamic balance - Analytics page working in real time

Date	Goals
12/1 (TP3)	- Integrate advanced features into the game - Integrate screeners using Yahoo API - Integrate polynomial regression into the charts - Integrate search bar into plotting - Integrate moving averages into graphing - Integrate AI Buy/Sell into Buy page

***Update (11-13-2021)**

I reached TP0 in time! I spent the last two days researching and learning how to use the requests module in Python to query the IEX Cloud API that I'll be using to plot my charts and get all of my data from! I also learned how to convert it all into a Pandas for later use to create moving averages and run some linear and polynomial regression.

***Update (11-18-2021)**

TP1 time has arrived and I was able to finish everything on my list! Although it has been a hectic couple of days, I was able to set up all of my UI elements, which took a longer time than expected due to the fact that I had to calculate where to put all of them on the screen of the Tkinter app. Finally, I was able to create a basic plotting algorithm as well! It works fairly well, but I think I will need to make some edits down the line to make it as robust as I possibly can.

***Update (11-23-2021)**

For this week, I was able to make some more edits to my charting page by allowing a user to chart multiple different charts in the same run of the app without having to start it over again. I also created the portfolio and allowed users to see the stocks they purchased through the newly created Buy and Sell pages. Finally, a portfolio analytics page was also created to provide the user with portfolio analytics to better understand their portfolio.

***Update (12-01-2021)**

I finally finished the project! I added some more advanced functionality into the app to better help users with buying and selling. First of all, I was able to integrate a search bar (albeit with some Tkinter module errors). I also implemented moving averages using Pandas and was able to create polynomial regression using a Sci Kit Learn module in Python. Finally, I was able to train a Tree-Regressor-based AI on the dataset I am using and was able use it give users buy and sell "AI Helper" alerts.

Technical Info

Stack/Tools:

- **Python**

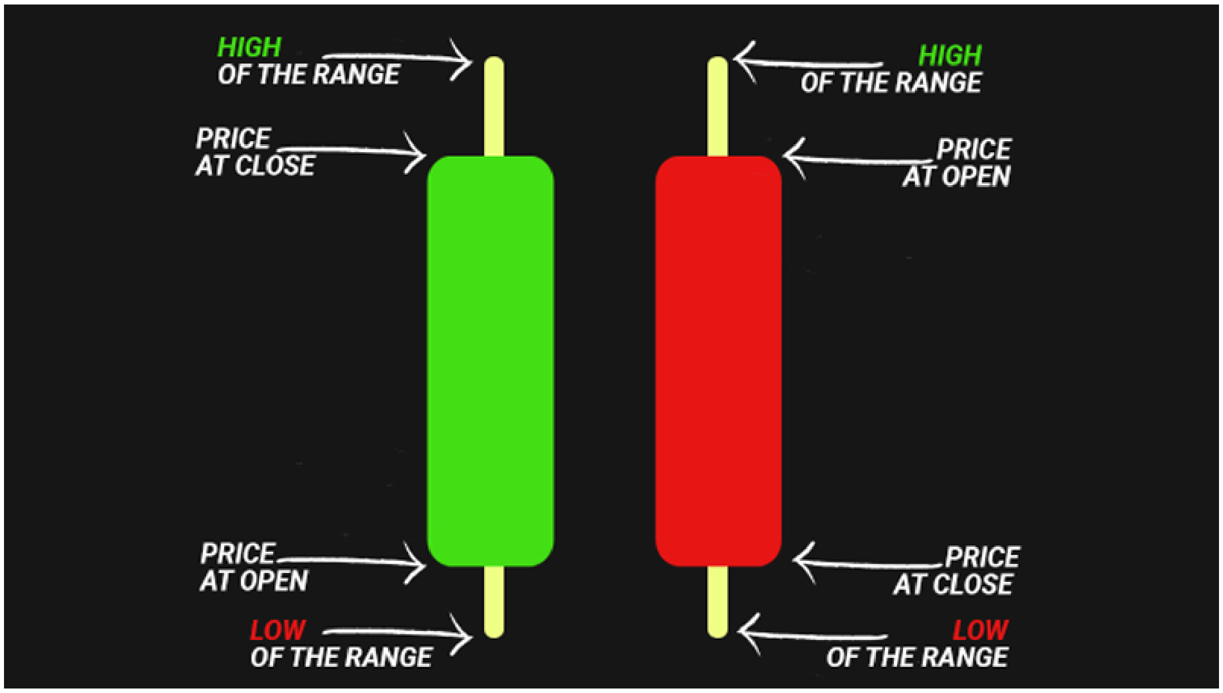
  I decided to use Python since I am learning the language in my college CS course. A project like this will hopefully open me up to using Python for more stock-related projects in the future such as Discord bots performing stock analysis instantaneously using web hooks to show images from the internet. One of the major benefits of using Python is the community and tools built for it. Whenever I had issues with something Python related, API related, or otherwise, I was able to find help, videos, and examples taught specifically in Python.

- **Visual Studio Code**

I normally use Visual Studio for my college Python course, and I decided to use it here to build my app from the ground up. The only other tool that I had was the cmu_112_graphics module that was provided as a Python file which I imported.

Background Knowledge

How to read a candlestick chart:



Credit: InvestorsObserver.com

Plotting Algorithm (V1):

The algorithm I chose to use to plot my points on the board was a self-created algorithm that translates price points into 2d coordinates on a coordinate grid that is then displayed on the app. It works by calculating the change of the bottom and top of the candle stick along with their wicks. Then, after calculating the number of pixels per day, I can multiply the height (in dollars) from the bottom of the board and convert it to a y-value. The X-value is a linear value that changes based on the number of candles being displayed.

Steps →

1. Get first 100 data values in a dictionary
2. Find the difference between the top of the graph an the bottom of the graph in Tkinter pixels
3. Calculate the difference in the stock's price from the low of the values to the high of the values
4. Find the move in pixels per dollar in the real world by dividing the graph height by the difference in the stock
5. Calculate the number of pixels needed to move for each wick (half of the width of the candlestick)
6. Run the getTicks function (self created) to get OHLC Values
    a. The getTicks function initializes the same variables above and initializes a list that will contain the OHLC values for each day in the dataset

    b. Take each O,H,L, and C value and calculate the y-value in computer space by taking the computer space y-value of the bottom of the graph and subtracting the difference of the O,H,L, or C value in dollars from the low of the stock in dollars multiplied by the move in pixels per dollar to find the y-value in computer space.
    c. add all of these O,H,L, and C computer-space y-values to a list, which itself is later added to the first initialized list
    d. This above process is repeated for every day in the data and returned as a list
7. Then plot the candles based on whether they are green or red for the day and plot their wicks as well (the lines extended both ways from the middle of the candle body)

Unfortunately, this AI does not perform very well. Some moves are made seemingly-randomly due to the **horizon effect**. This occurs when the algorithm reaches a max tree depth of 3 (the "horizon") and can't tell if a high-valued move was actually a bad strategic choice. Another issue is the tree depth itself — 3 levels is too low, and by itself the AI takes too much time to move. I plan to fix these issues in **V2.**

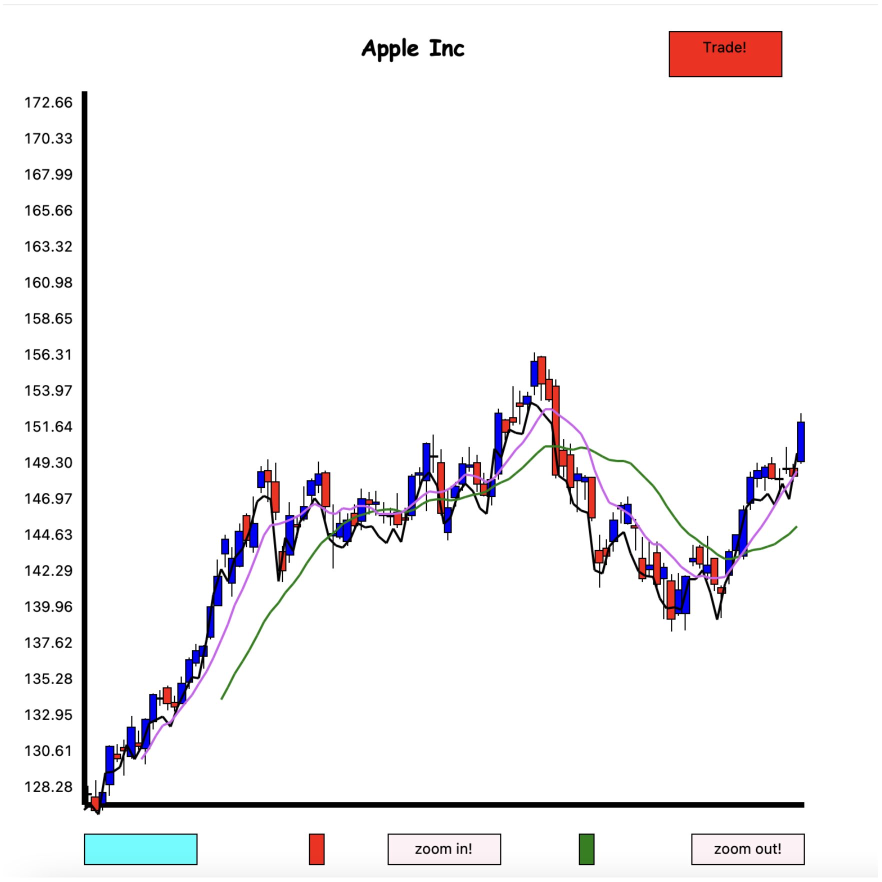
Plotting Algorithm (V2 - Advanced):

In the second version of the algorithm, I added provisions for the zooming of the data through the click of a button by changing the actual length of the dataset temporarily when the zoom button is clicked and I made the dataset to be accessed as a global variable (an "app" variable as it's called in the graphics package), so I could facilitate different stocks in the same run of the app from the user. I also used the same algorithm above to perform polynomial regression on the data and plot it as a line in addition to 2 moving averages.

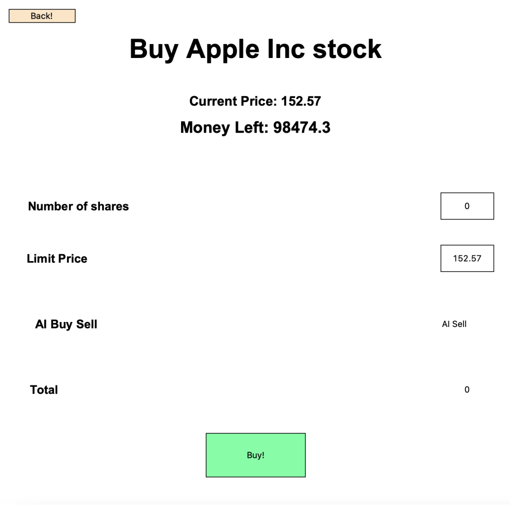
Resources:

- *https://stackoverflow.com/questions/3615375/number-of-days-between-2-dates-excluding-weekends*
- *https://www.geeksforgeeks.org/python-implementation-of-polynomial-regression/*
- *https://www.youtube.com/watch?v=hOLSGMEEwlI*
- https://www.cs.cmu.edu/~112/notes/notes-animations-part4.html

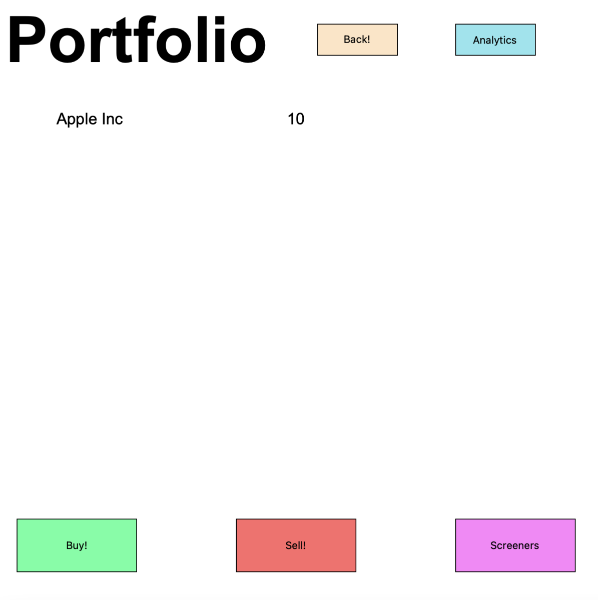
Gallery



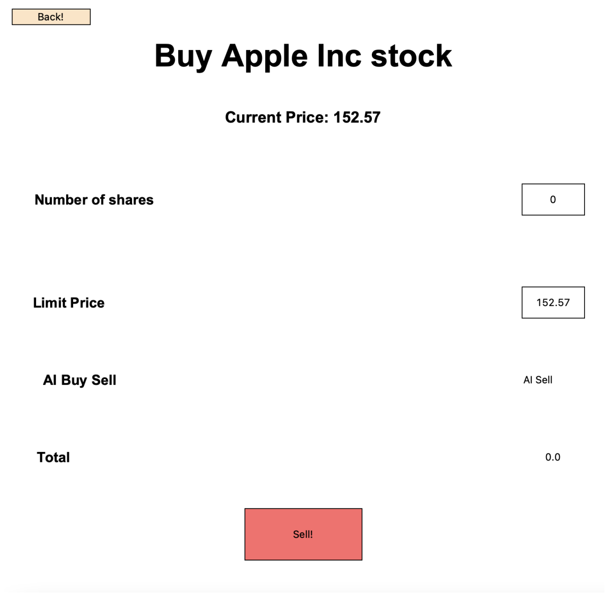
The main charting platform



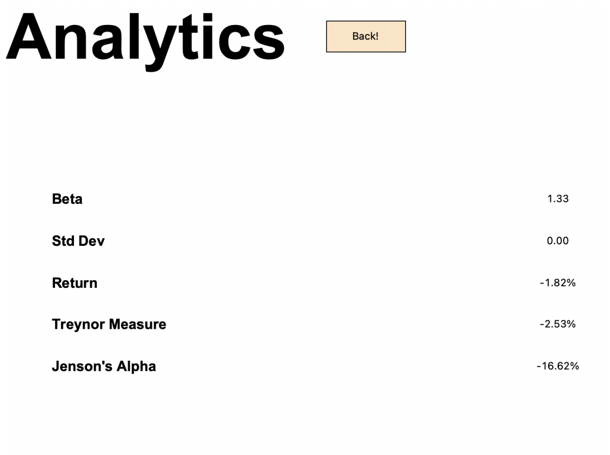
Buy Screen



Portfolio Page



Sell Screen



Beta	1.33
Std Dev	0.00
Return	-1.82%
Treynor Measure	-2.53%
Jenson's Alpha	-16.62%

Analytics Page

Development

Code commit history can be found in the GitHub repo link:

```
https://github.com/mehersujayk/ChartingPlatform
```

Program Flow:

1. The game opens in a console window and the user is shown 2 options: Charts and Portfolio
2. The user is prompted to enter their chosen screen (Charts, Portfolio).
   a. **Charts**
      i. The default chart, AAPL (Apple), is shown
      ii. The user can then zoom in to show a closer in view of the candles, zoom out to go back, move forward a day in the dataset (which causes the entire chart to move forward), or move back a day.
      iii. The user is given a choice to pursue a new chart by clicking the blue button or go to the portfolio to buy or sell shares of the stock being charted
         1. If the new stock button is pressed, the user inputs a new ticker (MSFT for example) and presses the button with MSFT on it to get the MSFT chart

2. If the portfolio button is pressed, the user is shown a view of their investments and is given the option to buy, sell, or go to portfolio analytics

iv. After a given "turn" is over, the user can go back to the charts page and advance a day to see their investments change

v. The game ends when the user reaches the end of the dataset, when they are greeted with an "end of dataset" message

b. **Portfolio**

i. The user starts from step 2 of iii above and continues with the same flow

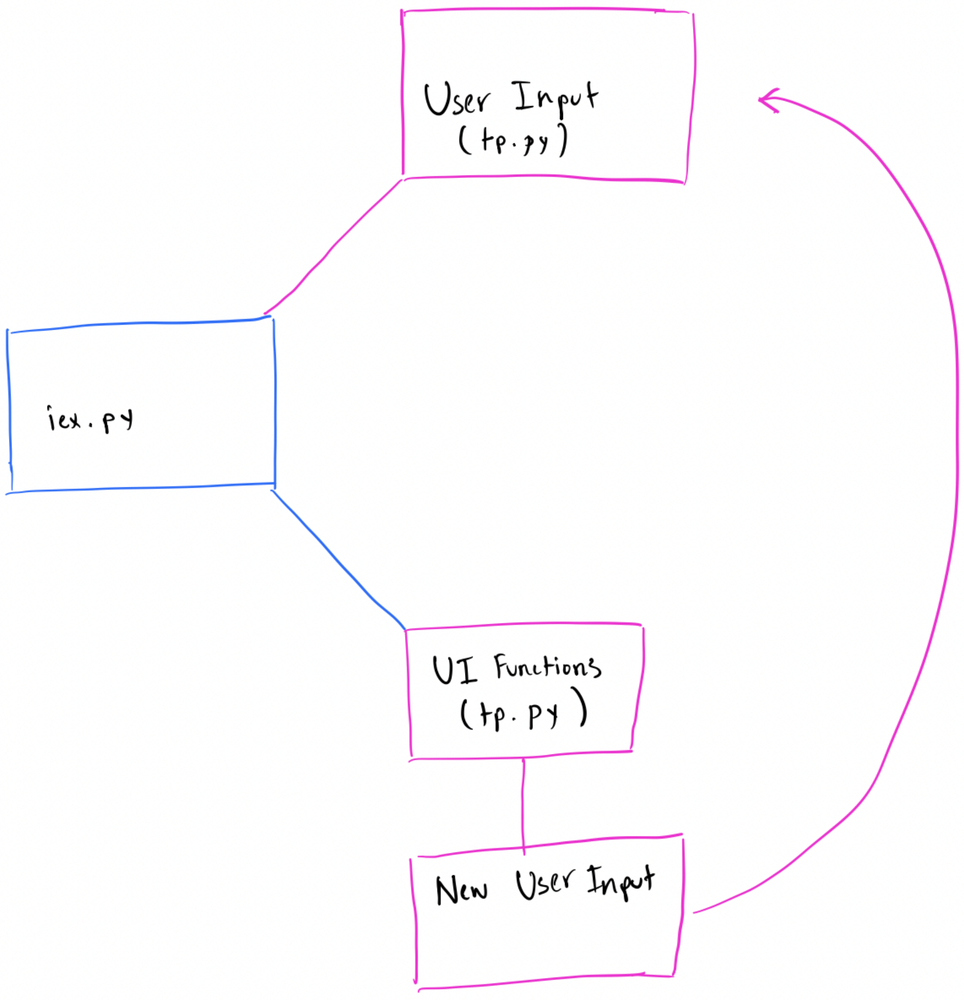
Making the Base Systems:

```
def getTicks(app, arr):
    #These are a bunch of variables used to calculate the pixel difference
    #per dollar of the stock
    graphDiff = app.graphBottom - app.graphTop
    stockDiff = app.stockHigh - app.stockLow
    pixelDiffPerDollarMove = graphDiff / stockDiff #change this if tkinter can't graph decimals
    OHLCValues = []
    prices = arr

    #calculates the pixel difference for open, high, low, and close
    #and adds them to a list
    for date in prices:
        dollarDiff = prices[date][0] - prices[date][3]
        pixelDiff = pixelDiffPerDollarMove * dollarDiff
        open = app.graphBottom - (prices[date][0] - app.stockLow) * pixelDiffPerDollarMove
        high = app.graphBottom - (prices[date][1] - app.stockLow) * pixelDiffPerDollarMove
        low = app.graphBottom - (prices[date][2] - app.stockLow) * pixelDiffPerDollarMove
        close = app.graphBottom - (prices[date][3] - app.stockLow) * pixelDiffPerDollarMove
        priceLst = [open, high, low, close]


        OHLCValues.append(priceLst)



    #return the OHLC values
    return OHLCValues
```

The main algorithm for calculating the computer y-value from the dollar amount is shown above. Using "app" variables (app.etc), I am able to define pseudo instance variables that I can use throughout an instance of the particular Tkinter window that is being created. The getTicks method eventually supplies data to the drawCandlesticks method shown below that actually draws the candlesticks on the app window.

```
def drawCandlesticks(app, canvas):
    graphDiff = app.graphBottom - app.graphTop
    stockDiff = app.stockHigh - app.stockLow
    pixelDiffPerDollarMove = graphDiff / stockDiff
    OHLCValues = getTicks(app, app.prices)

    #This code is adapted from https://www.geeksforgeeks.org/python-implementation-of-polynomial-regression/
    X = app.dataClone.iloc[:, 1].values.reshape(-1, 1)  # values converts it into a numpy array
    Y = app.dataClone.iloc[:, 2].values.reshape(-1, 1)  # -1 means that calculate the dimension of rows, but have 1 column
    poly = PolynomialFeatures(degree = 4)
    X_poly = poly.fit_transform(X)

    poly.fit(X_poly, Y)
    lin2 = LinearRegression()
    lin2.fit(X_poly, Y)
    YPolyPred = lin2.predict(poly.fit_transform(X))


    w = app.width
    firstHundred = getTicks(app, app.firstHundredDict)

    numDays = get_workdays(app.a, app.b) - app.holidays
    #print(numDays)
    xaxisWidthInPixels = app.graphRight - app.graphLeft
    pixelsPerDay = xaxisWidthInPixels/len(app.firstHundred)
    wickPixels = pixelsPerDay / 2

    #drawing a candlestick

    #top wick


    #body
    #using the Open and Close values from OHLC Values, I created the bodies
    #at the correct places in the chart and moved the candlesticks the perfect
    #amount by calculating the pixels per day in the x direction and adding it
    #to the x coordinate

    #wicks
    #using the high, low, close, and open values, I constructed wicks for the
    #stock that represent how high and how low the price reached before it
    #it settled at its close price


    for day in range(len(firstHundred)):
        if firstHundred[day][3] >= firstHundred[day][0]:

            canvas.create_rectangle((w//10) + (pixelsPerDay * day),
                                    firstHundred[day][3],
                                    (w//10) + (pixelsPerDay * day) + pixelsPerDay,
                                    firstHundred[day][0],
                                    fill = 'red')

            #top wick
            canvas.create_line((w//10) + (pixelsPerDay * (day)) + wickPixels,
                               firstHundred[day][1],
                               (w//10) + (pixelsPerDay * (day)) + wickPixels,
                               firstHundred[day][0])

            #bottom wick
            canvas.create_line((w//10) + (pixelsPerDay * (day)) + wickPixels,
                               firstHundred[day][3],
                               (w//10) + (pixelsPerDay * (day)) + wickPixels,
                               firstHundred[day][2])

        if firstHundred[day][3] < firstHundred[day][0]:
            canvas.create_rectangle((w//10) + (pixelsPerDay * day),
                                    firstHundred[day][3],
                                    (w//10) + (pixelsPerDay * day) + pixelsPerDay,
                                    firstHundred[day][0],
                                    fill = 'blue')

            #top wick
            canvas.create_line((w//10) + (pixelsPerDay * (day)) + wickPixels,
                               firstHundred[day][1],
                               (w//10) + (pixelsPerDay * (day)) + wickPixels,
                               firstHundred[day][3])

            #bottom wick
            canvas.create_line((w//10) + (pixelsPerDay * (day)) + wickPixels,
                               firstHundred[day][0],
                               (w//10) + (pixelsPerDay * (day)) + wickPixels,
                               firstHundred[day][2])
```

A snippet of the drawCandlesticks function is shown here. This function takes the points from the getTicks function and actually plots the rectangles on the graph as rectangles and lines. Here, we can also see the starts of the polynomial regression line that is being calculated. The polynomial regression values will also be plotted in a similar fashion to the candlesticks themselves, with the line being a compilation of a line of smaller day-by-day lines



tp.py controls the entire plotting and decision making aspects of the app, which include UI elements and buying, selling, and displaying the charts. However, much of the data needed in the showing of these data is obtained through the use of another python file containing the Stock class called iex.py. This iex.py class contains all of the

API calls that provide the data for tp.py to display. Once the data is received, the UI Functions in tp.py kick in, and then await for new use input after which the loop is rerun.

Issues & Fixes:

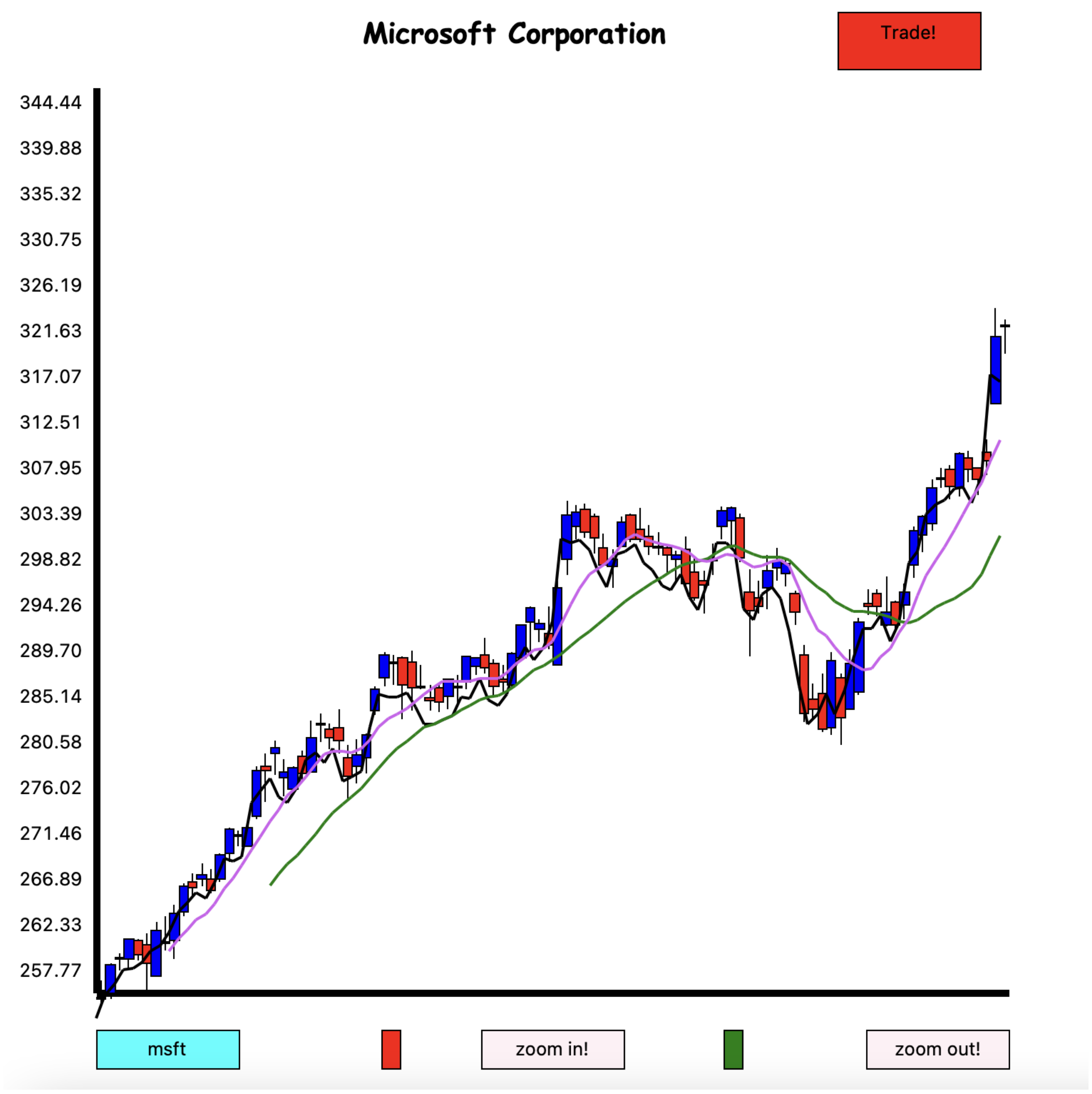
- **API Data** → An issue I had was with API Data. Although I was using IEX Cloud API Data (Some of the best on the market), I wasn't able to manipulate the data the same way that I was able to manipulate Yahoo API data because the dictionary to data frame capabilities in Pandas prevented me form being able to index the data the way it needed to be formatted for the AI/Polynomial regression to work properly. In order to fix this, I actually used the Yahoo API through pandas_datareader, which has a built-in Yahoo Finance stock option.
- **Way too many API calls** → Very early on, I ran out of free data from the IEX API. Although I paid for it, I later learned that I had been consuming vast amounts of data because of my charting feature calling an API request every time it refreshed (multiple times a second). This meant that I was burning through my API requests at a very rapid rate. In order to fix this, I converted a lot of the function-specific variables into app variables that were used to service the entire file. This enabled me to call the API less and save some valuable API calls in addition to speeding up the program!

Conclusion & Future

This was one my most fun projects to work on because it is something that I have been wanting to do for a long, long time. I finally was able to do it and it was very very rewarding to see that I had succeeded in a long-term project that I had never undertaken before. I also got to learn a lot more about Python, APIs, and I learned how many considerations go into creating an app of even such a small scale, but it only leaves me more excited to undertake similar, more elaborate projects!

One of the largest challenges I faced in this project was lack of guidance. Sometimes during the project, especially places where I had to work around problems as opposed to solving them, I was left wanting a mentor that had gone through the same problems as me. Because this was a relatively niche project working with a very local graphics package, there was really no one that could help me. I believe that having a more concrete resource could have helped me implement some things that could have been implemented much easier and efficiently.

While I am done with this project for now, I might return later to add further AI optimizations. Currently, the AI is not made by myself, but is sourced from a module. Although this provides a very clean implementation, I want to learn a lot more about AI and it's significance to stocks. In addition, in the future, I plan to add options functionality as well in order to present a more robust trading experience for the user with real-time updating charts and the ability to trade in real-time as opposed to investing with days as your horizon.


Microsoft Corporation
Trade!
344.44
339.88
335.32
330.75
326.19
321.63
317.07
312.51
307.95
303.39
298.82
294.26
289.70
285.14
280.58
276.02
271.46
266.89
262.33
257.77
msft
zoom in!
zoom out!