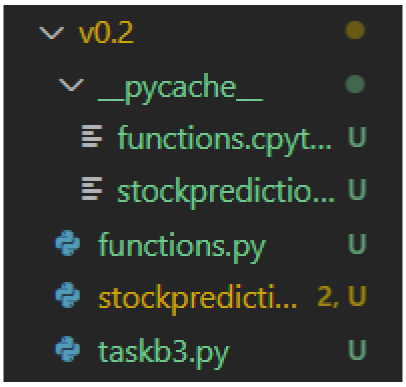
COS30018 Intelligent Systems – Task B3 Report

Ryan Chessum 102564760

This week I had to make 2 different functions to display financial stock data.

To start off, I made a new folder to put the code for this weeks work in.

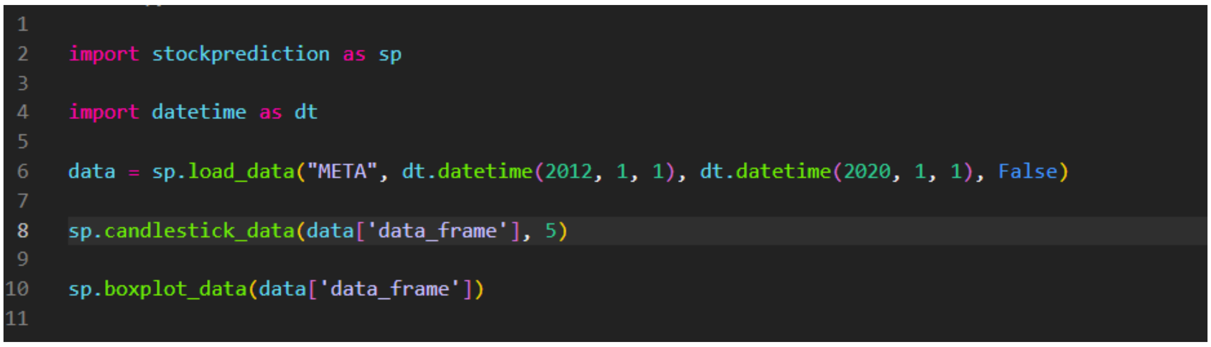


```
v0.2
  __pycache__
    functions.cpyt...        U
    stockpredictio...        U
  functions.py               U
  stockpredicti...        2, U
  taskb3.py                  U
```

Functions has shuffle in unison in it. I thought it would be good to have a file that held small functions like that but I never really ended up using it for anything else.

Stockprediction has the load data function from last week along with the 2 new functions I made this week.

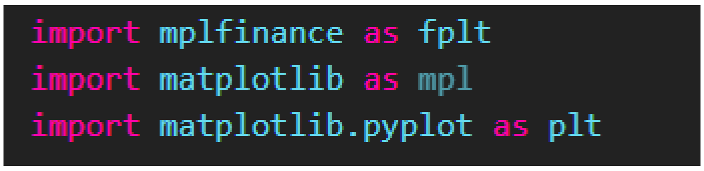
Taskb3 is the file to be run. It loads the data into a variable then passes it into the other two functions to create the graphs.



```

import stockprediction as sp

import datetime as dt

data = sp.load_data("META", dt.datetime(2012, 1, 1), dt.datetime(2020, 1, 1), False)

sp.candlestick_data(data['data_frame'], 5)

sp.boxplot_data(data['data_frame'])

```

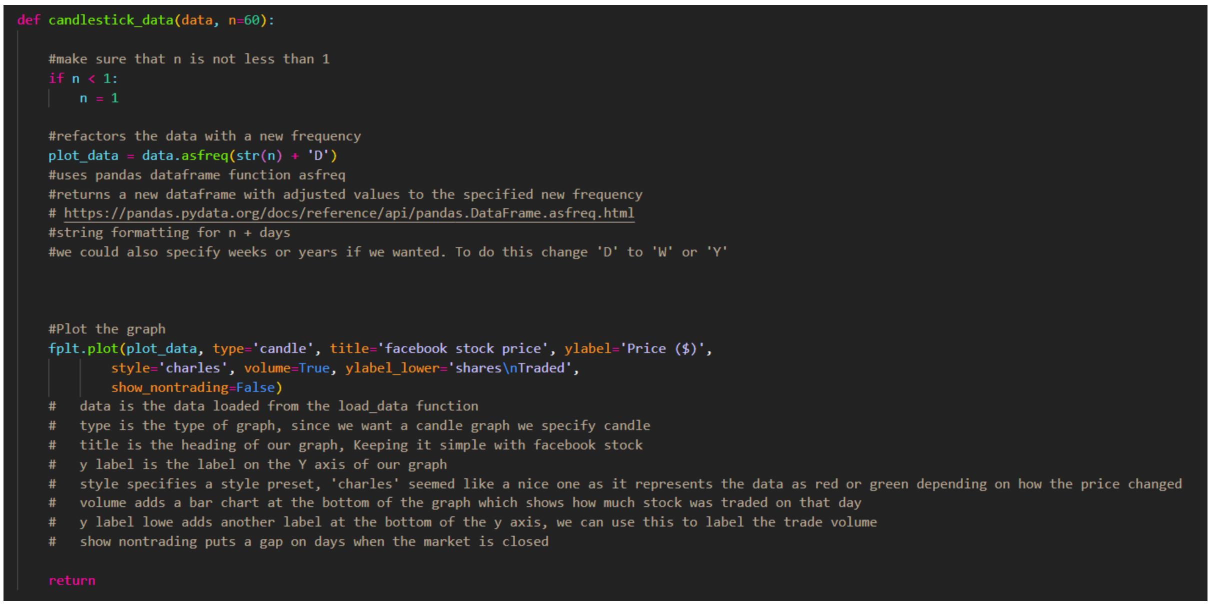
I had an interesting problem, where the function couldn't find the Facebook ticker. Even though it worked in previous weeks, I need to use the new ticker 'META' to load facebooks stock data now as facebook now has a new company name.

Candlestick Function



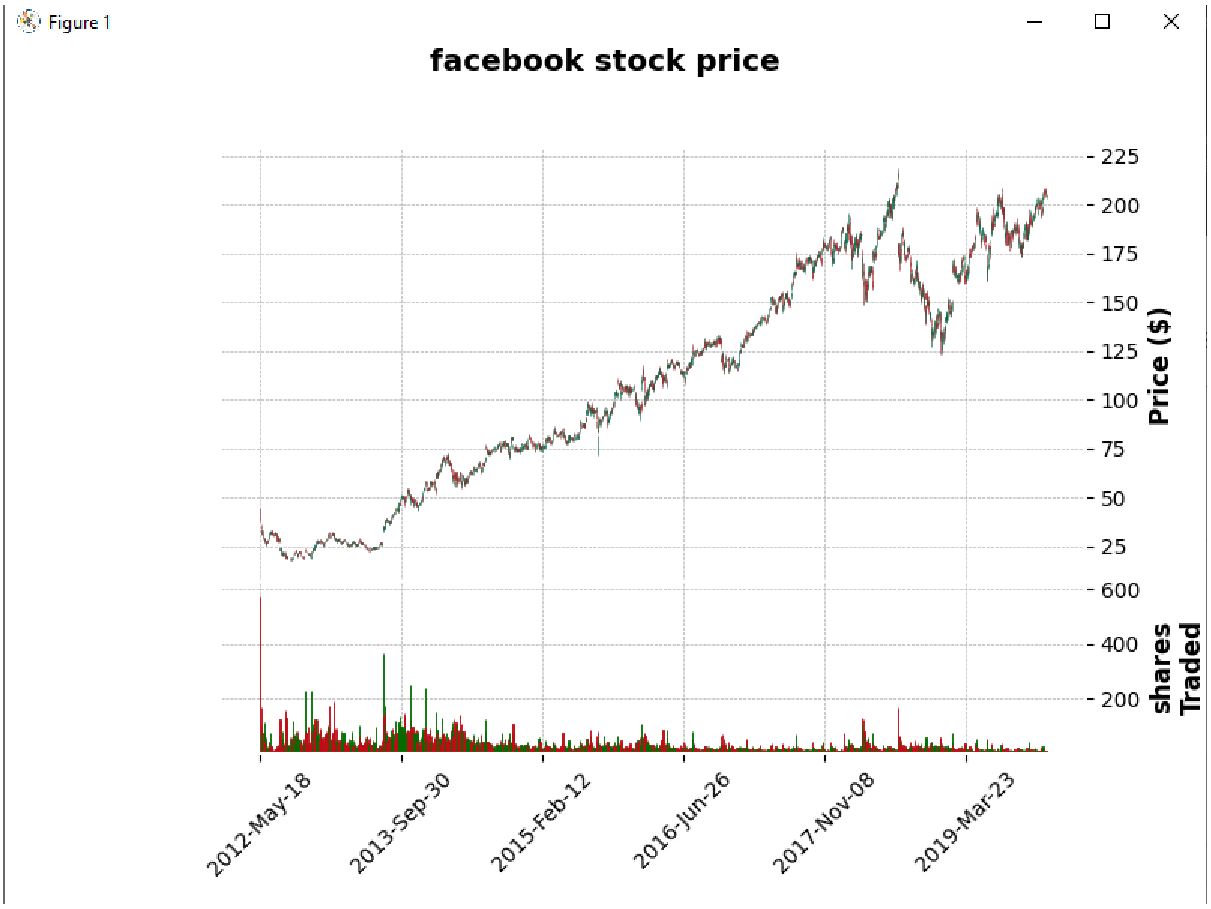
```python
import mplfinance as fplt
import matplotlib as mpl
import matplotlib.pyplot as plt
```

For these functions I used matplotlib and matplotlib finance to plot the graphs.



```python
def candlestick_data(data, n=60):

    #make sure that n is not less than 1
    if n < 1:
        n = 1

    #refactors the data with a new frequency
    plot_data = data.asfreq(str(n) + 'D')
    #uses pandas dataframe function asfreq
    #returns a new dataframe with adjusted values to the specified new frequency
    # https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.asfreq.html
    #string formatting for n + days
    #we could also specify weeks or years if we wanted. To do this change 'D' to 'W' or 'Y'



    #Plot the graph
    fplt.plot(plot_data, type='candle', title='facebook stock price', ylabel='Price ($)',
            style='charles', volume=True, ylabel_lower='shares\nTraded',
            show_nontrading=False)
    #   data is the data loaded from the load_data function
    #   type is the type of graph, since we want a candle graph we specify candle
    #   title is the heading of our graph, Keeping it simple with facebook stock
    #   y label is the label on the Y axis of our graph
    #   style specifies a style preset, 'charles' seemed like a nice one as it represents the data as red or green depending on how the price changed
    #   volume adds a bar chart at the bottom of the graph which shows how much stock was traded on that day
    #   y label lowe adds another label at the bottom of the y axis, we can use this to label the trade volume
    #   show nontrading puts a gap on days when the market is closed

    return
```

As candlestick graphs are very useful for displaying financial data, matplotlib finance makes this very easy for us. We can use a plot function and specify our graph to be a candle graph.

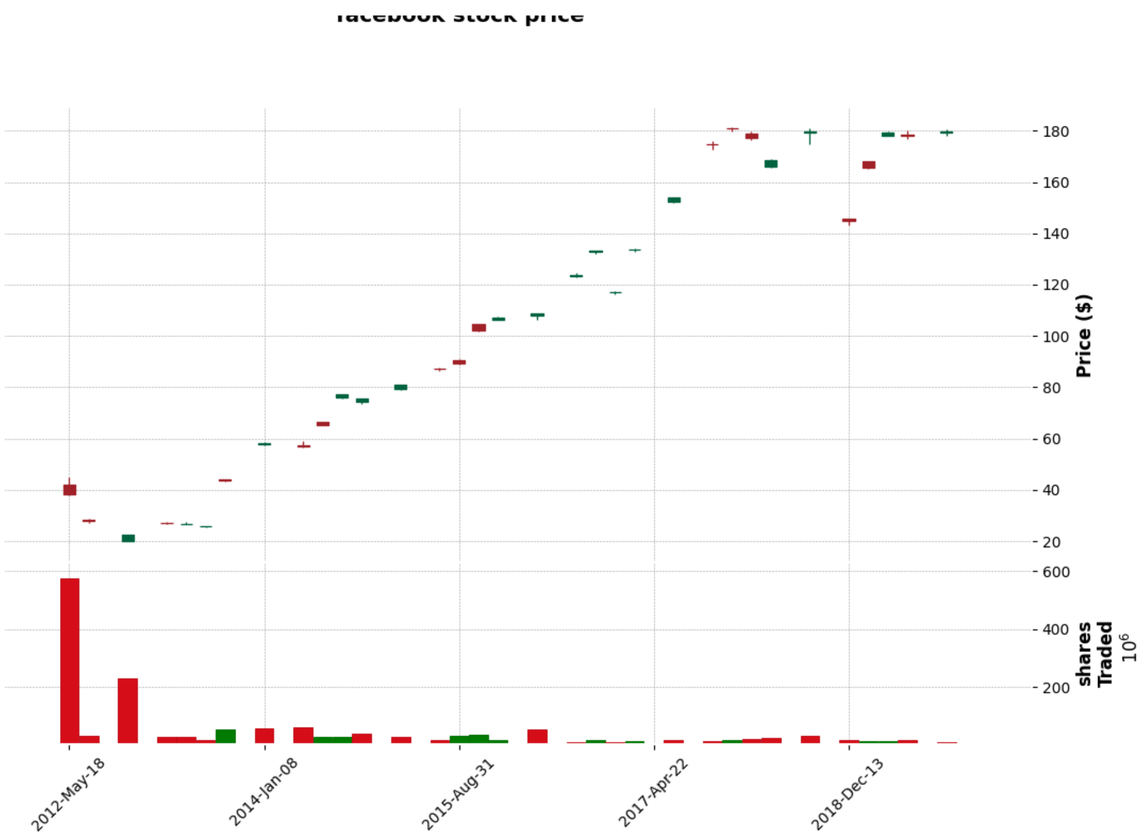


One issue however is that if we are plotting a large sample size of data, it is very hard to see on the graph.

To fix this there is a parameter to specify how many days are represented by each candlestick.

```
plot_data = data.asfreq(str(n) + 'D')
```

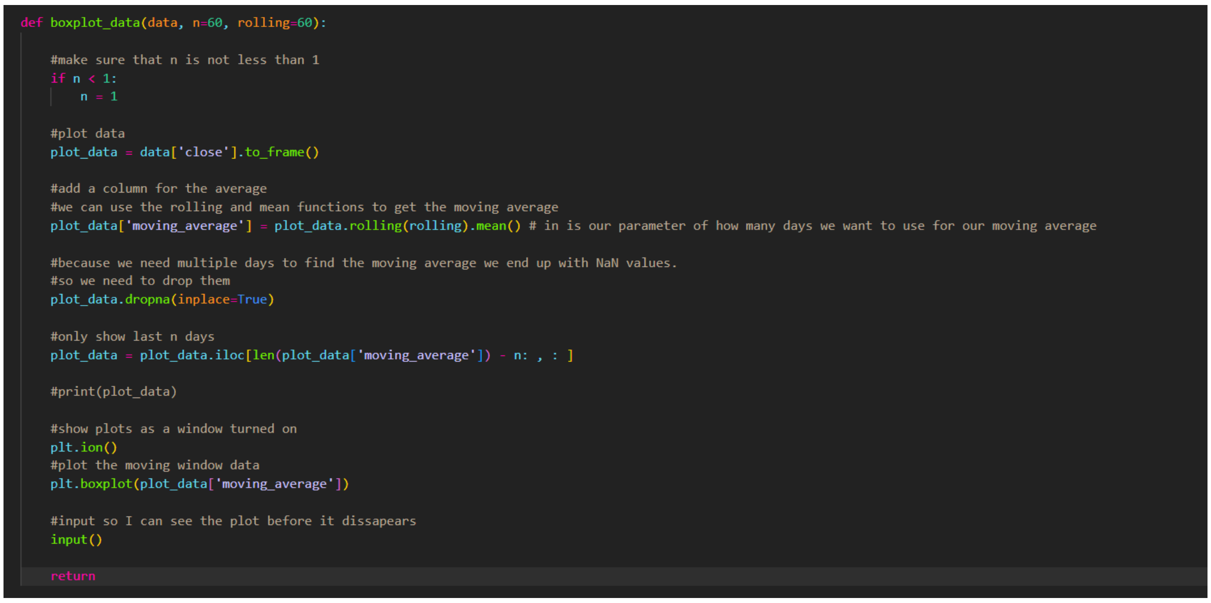
Pandas lets us change the frequency of the data so that each row specifies a set amount of time.



This is what the data looks like when set so that every candle represents every 60 days. It's much easier to visualise the characteristics of the candlestick graph this way.

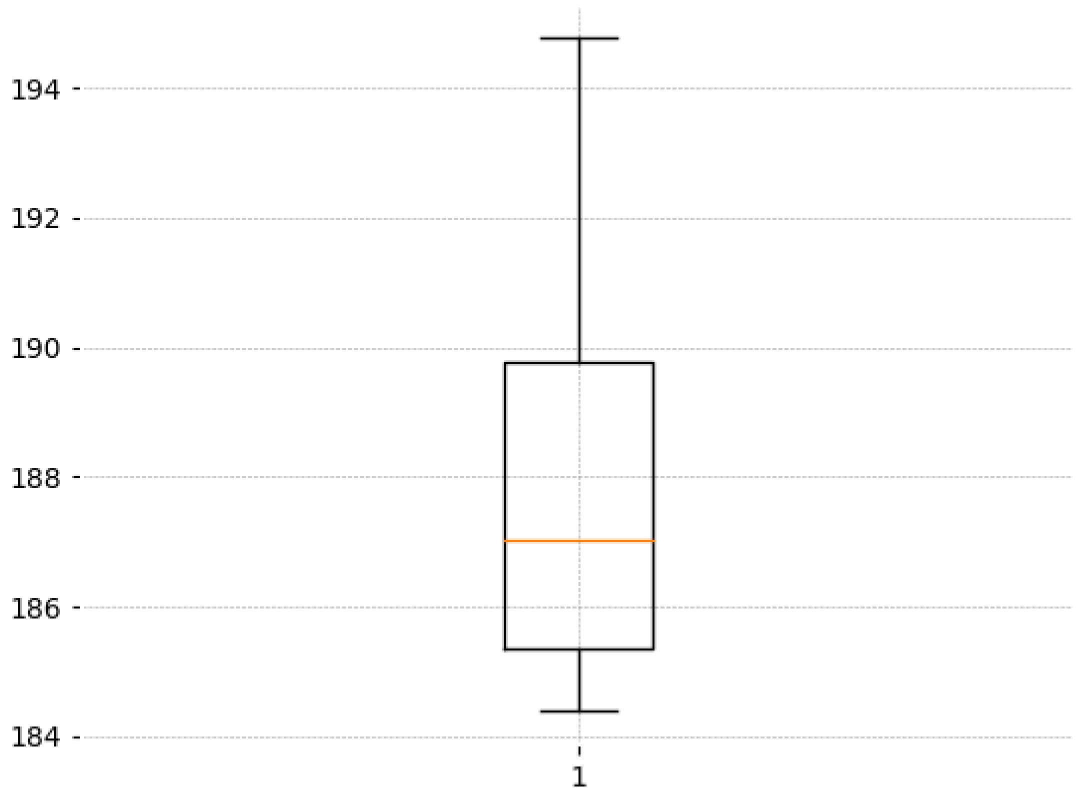
Box plot function

The other function Creates a box plot of the moving average over the last n amount of days.



```python
def boxplot_data(data, n=60, rolling=60):

    #make sure that n is not less than 1
    if n < 1:
        n = 1

    #plot data
    plot_data = data['close'].to_frame()

    #add a column for the average
    #we can use the rolling and mean functions to get the moving average
    plot_data['moving_average'] = plot_data.rolling(rolling).mean() # in is our parameter of how many days we want to use for our moving average

    #because we need multiple days to find the moving average we end up with NaN values.
    #so we need to drop them
    plot_data.dropna(inplace=True)

    #only show last n days
    plot_data = plot_data.iloc[len(plot_data['moving_average']) - n: , : ]

    #print(plot_data)

    #show plots as a window turned on
    plt.ion()
    #plot the moving window data
    plt.boxplot(plot_data['moving_average'])

    #input so I can see the plot before it dissapears
    input()

    return
```

For this I had to use the normal version of matplotlib. Using the rolling and mean functions I could find the average closing price for the last number of specified days. Then cut the data down aftwards and plotted the graph.



This function was a little challenging to design as I wasn’t sure how the data should be represented in this format. But I think this is a good way to show the average over the a period of time.