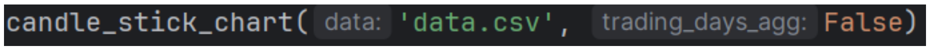
B.3

For this task, I created two functions, one for a candlestick chart, and one for a boxplot chart. I started with the candlestick chart. I had a lot of trouble getting started properly on this, as I had a lot of trouble with the Date column, but after I figured out what was wrong and understood how it was working, I got through the charts pretty quickly. So I spent most of the time just debugging and understanding a concept related to the load_data() function.

To start off, I added a string variable called "data" to the paraments, so that when called, the function will access the file specified. Later I added the Boolean trading_days_agg, but I will come back to that.

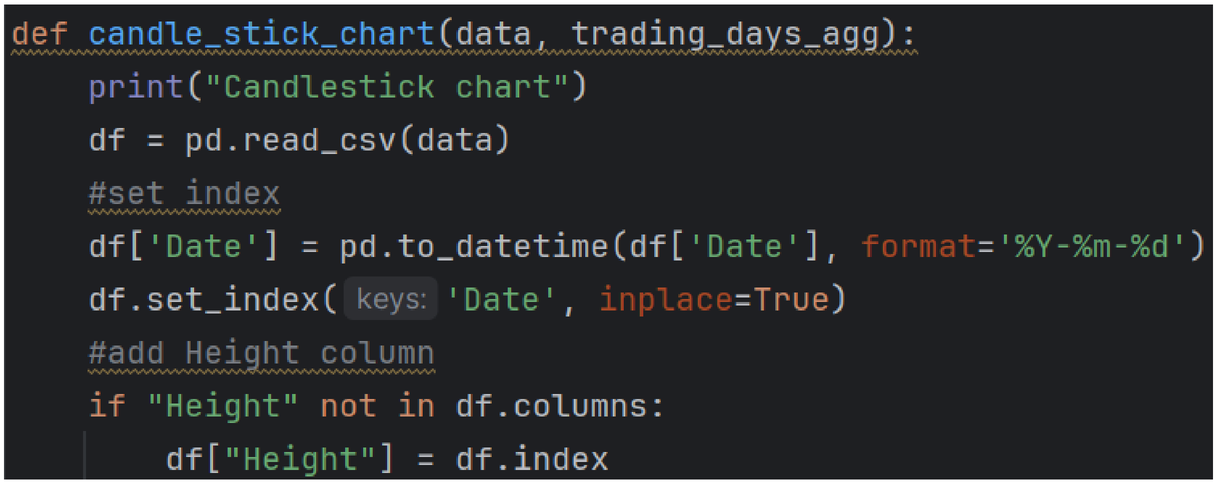
Candlestick Chart

Function call at end of file



```
candle_stick_chart( data: 'data.csv',  trading_days_agg: False)
```

Function

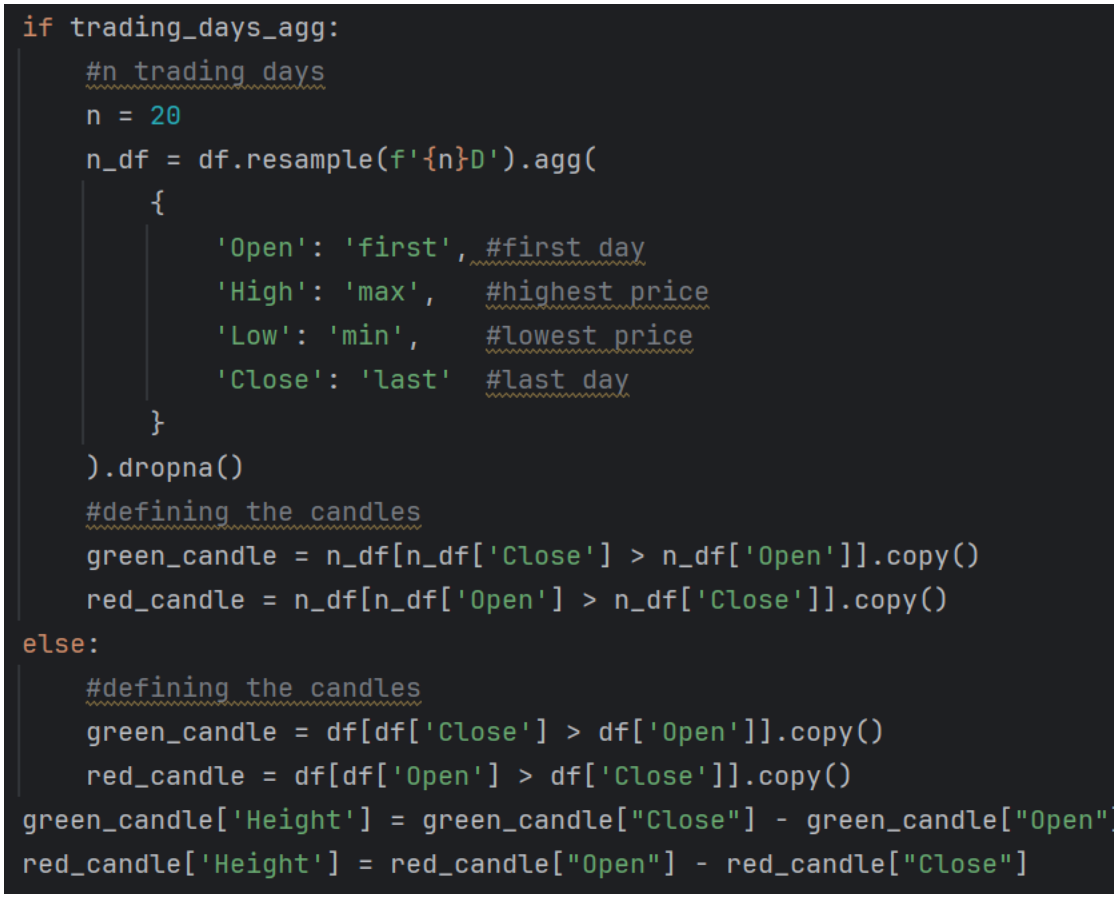


```
def candle_stick_chart(data, trading_days_agg):
    print("Candlestick chart")
    df = pd.read_csv(data)
    #set index
    df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d')
    df.set_index( keys: 'Date', inplace=True)
    #add Height column
    if "Height" not in df.columns:
        df["Height"] = df.index
```

First, I assigned the dataframe to be used with pd.read_csv(data), as the data has already been loaded and saved to a .csv file with the load_data() function.

I converted df['Date'] to datetime and then set the index to Date. At first I didn't think I needed to do this since I already did that in the load_data() function, but not having it was creating an error with the dates being displayed on the X axis of the chart figure.

Next, I added Height as a column as the candlesticks needed their own column for the calculations.

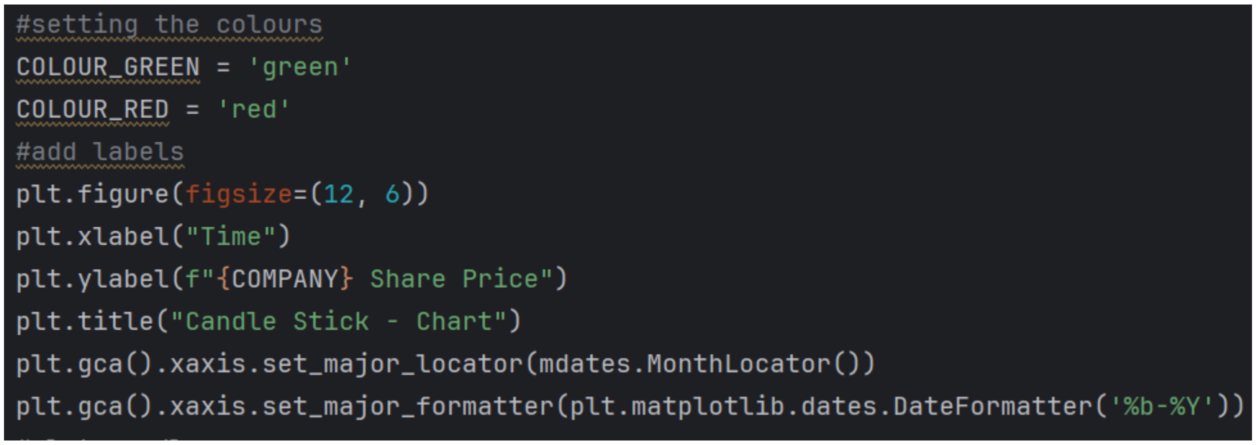


```
if trading_days_agg:
    #n trading days
    n = 20
    n_df = df.resample(f'{n}D').agg(
        {
            'Open': 'first', #first day
            'High': 'max',   #highest price
            'Low': 'min',    #lowest price
            'Close': 'last'  #last day
        }
    ).dropna()
    #defining the candles
    green_candle = n_df[n_df['Close'] > n_df['Open']].copy()
    red_candle = n_df[n_df['Open'] > n_df['Close']].copy()
else:
    #defining the candles
    green_candle = df[df['Close'] > df['Open']].copy()
    red_candle = df[df['Open'] > df['Close']].copy()
green_candle['Height'] = green_candle["Close"] - green_candle["Open"
red_candle['Height'] = red_candle["Open"] - red_candle["Close"]
```

I created a condition for the *trading_days_agg paramenter*. If True is parsed through the parameter, the data will resample and aggregate to display data according to *n* number of trading days (the D within the resample parameter represents days, if I . If False is parsed through, the candlestick chart wrote M, it would resample by *n* number of trading months. will display the data without aggregating.

The candles need to be defined by making a copy of the dataframe, where Close is greater than Open for green, and Open is greater than Close for red. If *trading_days_agg* is set to True, a new dataframe name (n_df) is used with the assigned aggregated data.

After the candles are first declared, they need to be altered to subtract Open or Close from each to show the difference during each trading timeframe. The green candles show the increase of prices, while the red candles show the decline in prices. These will be plotted with plt.bar. While plt.vlines will create the lines or "wicks" that go through them, that represents the overall height of the prices in trading days, with the top of the wick being the highest price reached, and the bottom of the wick being the lowest price reached.

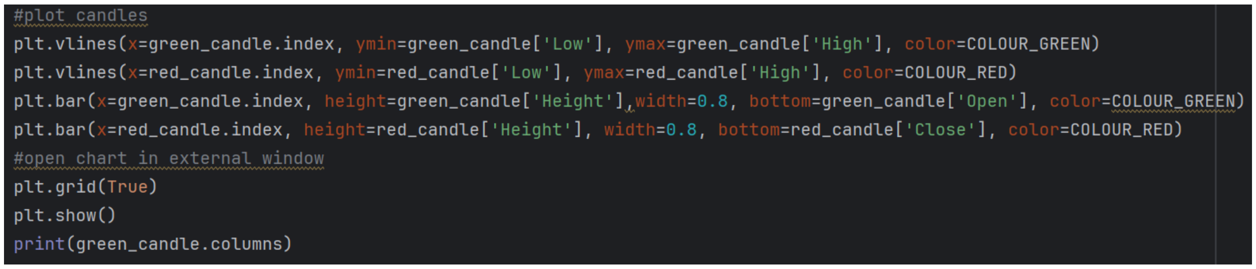


```
#setting the colours
COLOUR_GREEN = 'green'
COLOUR_RED = 'red'
#add labels
plt.figure(figsize=(12, 6))
plt.xlabel("Time")
plt.ylabel(f"{COMPANY} Share Price")
plt.title("Candle Stick - Chart")
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
plt.gca().xaxis.set_major_formatter(plt.matplotlib.dates.DateFormatter('%b-%Y'))
```

I created two constant variables for easy colour reference, to be used when plotting the chart. Then set the figure size, followed by labels. I noticed that plt.figure() should be coded before the labels, otherwise the labels will not show up on the figure.

Next, I wanted the dates to be displayed on the X axis by month, so that it’s not over-saturated with labels and smashing text together, so I used *plt.gca().xaxis.set_major_locator(mdates.MonthLocator())* to space out the time labels by month, *and plt.gca().xaxis.set_major_formatter(plt.matplotlib.dates.DateFormatter(‘%b-%Y’))* to format the labels to show a shorthand label for each month, a dash, and then the year (with a capital Y, as I learnt that capitals format 4 digits while lower case formats 2 digits).

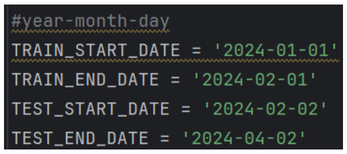
For plotting the candles, it was imperative that the index be set to ‘Date’, so that I could set *x=green/red_candle.index*, as *green/red_candle[‘Date’]* did not work for some reason, even though the man in supplied tutorial video did that without any issue.



```
#plot candles
plt.vlines(x=green_candle.index, ymin=green_candle['Low'], ymax=green_candle['High'], color=COLOUR_GREEN)
plt.vlines(x=red_candle.index, ymin=red_candle['Low'], ymax=red_candle['High'], color=COLOUR_RED)
plt.bar(x=green_candle.index, height=green_candle['Height'],width=0.8, bottom=green_candle['Open'], color=COLOUR_GREEN)
plt.bar(x=red_candle.index, height=red_candle['Height'], width=0.8, bottom=red_candle['Close'], color=COLOUR_RED)
#open chart in external window
plt.grid(True)
plt.show()
print(green_candle.columns)
```

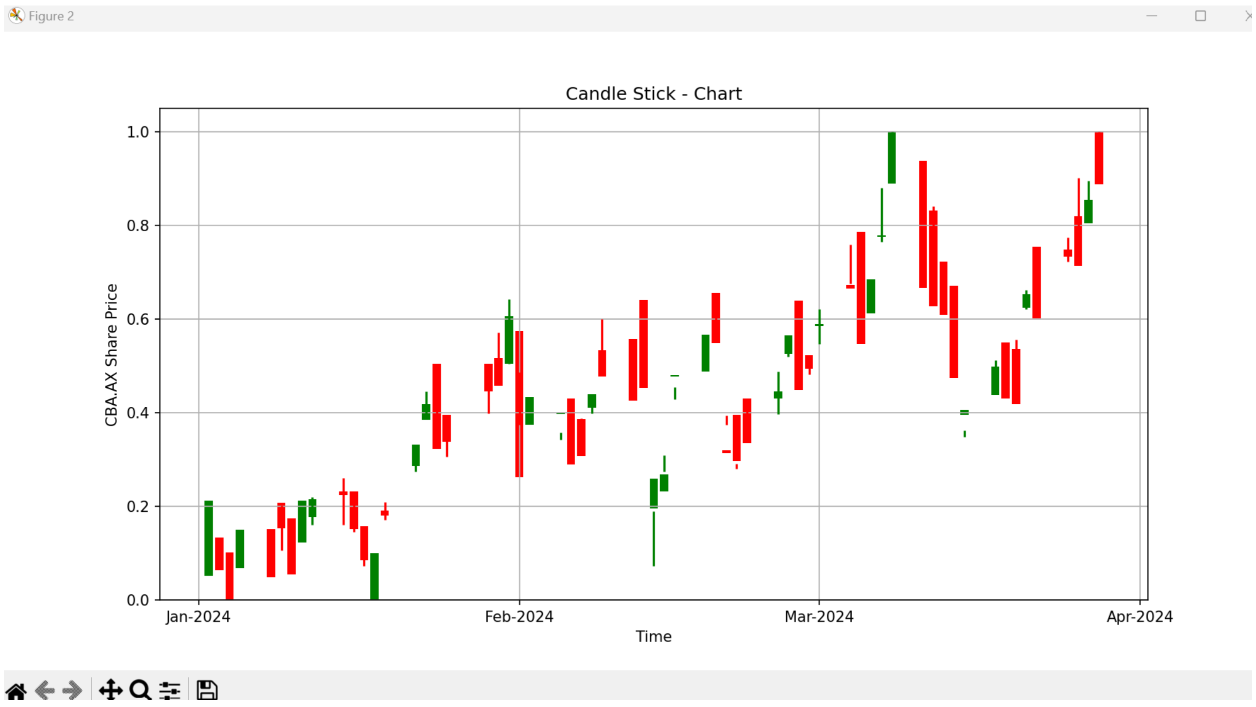
Results

For this task, I set the dates to a shorter timeframe to make debugging faster.



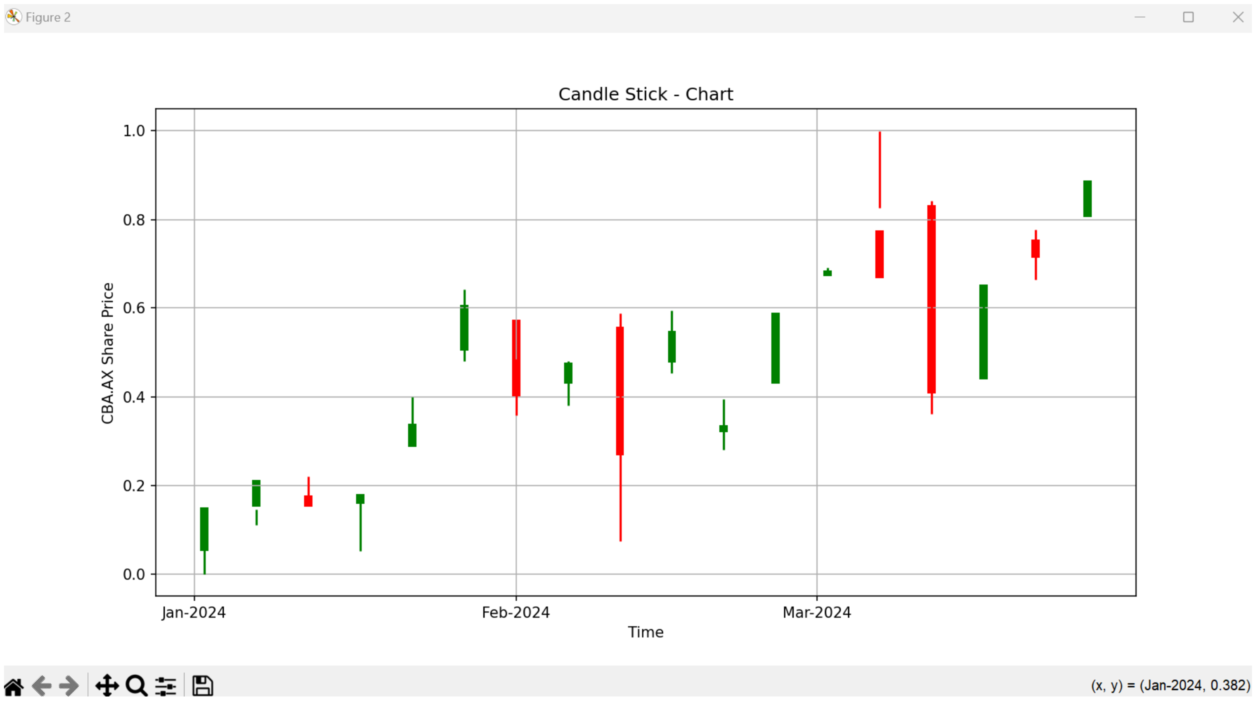
```
#year-month-day
TRAIN_START_DATE = '2024-01-01'
TRAIN_END_DATE = '2024-02-01'
TEST_START_DATE = '2024-02-02'
TEST_END_DATE = '2024-04-02'
```

With trading_days_agg set to False

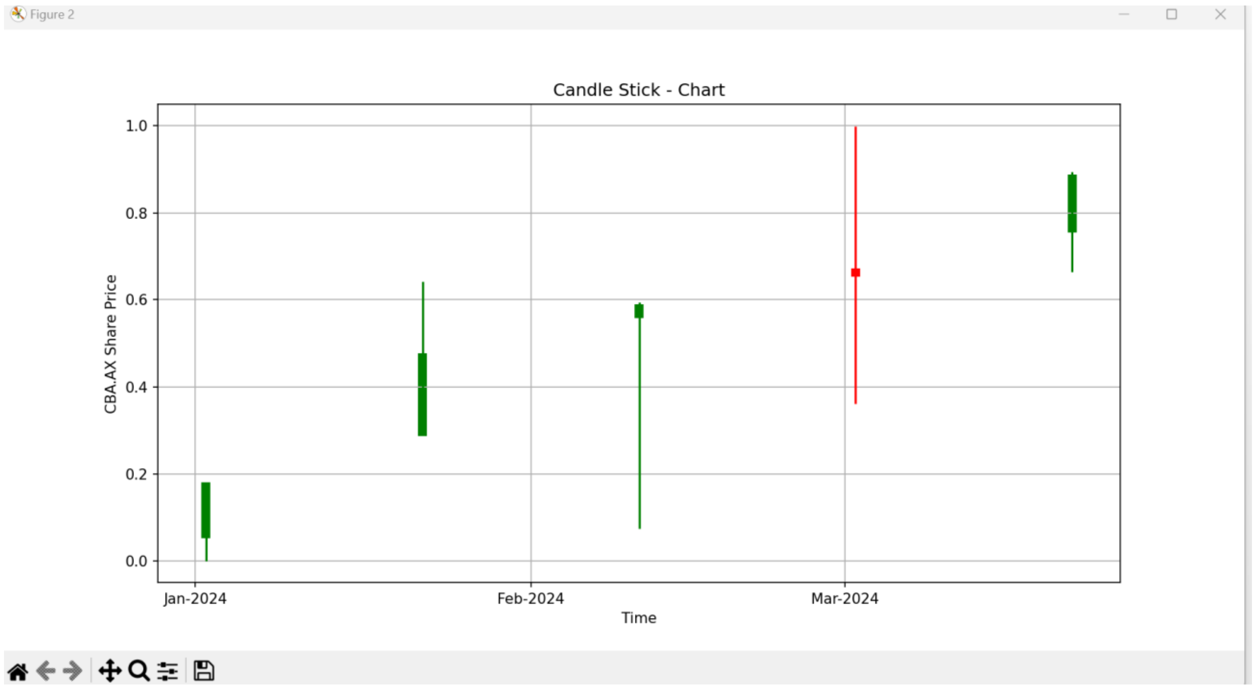


With it set to True, and n is 5

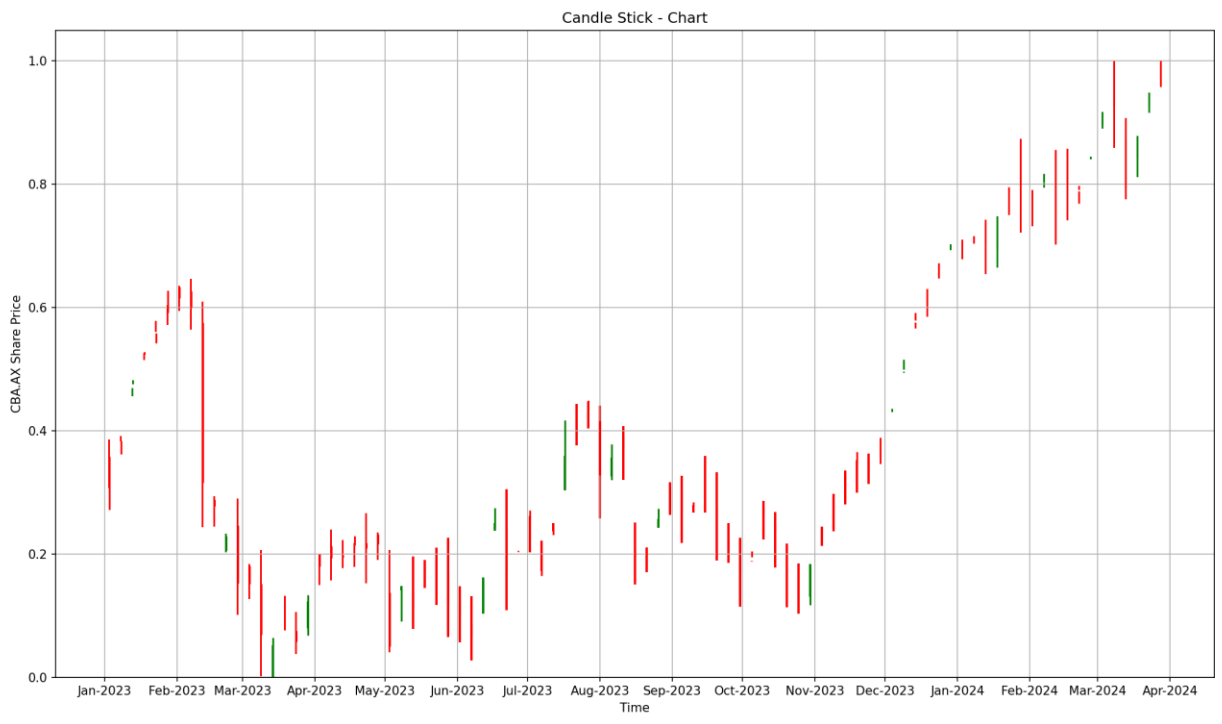
When the data is aggregated to represent n trading days, the candlesticks should get longer, with fewer appearing as the days (or n) increases.

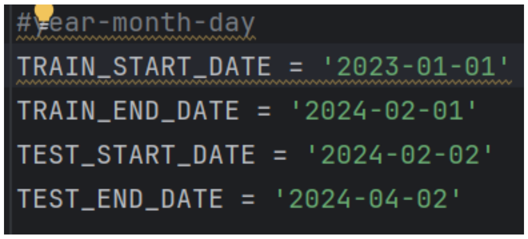


<u>With n set to 20</u>



<u>After reducing the TRAIN_START_DATE by a year (just for demonstrating)</u>

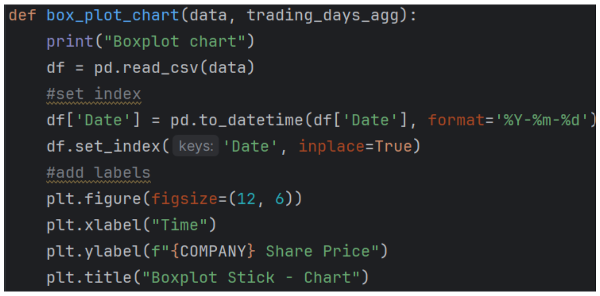




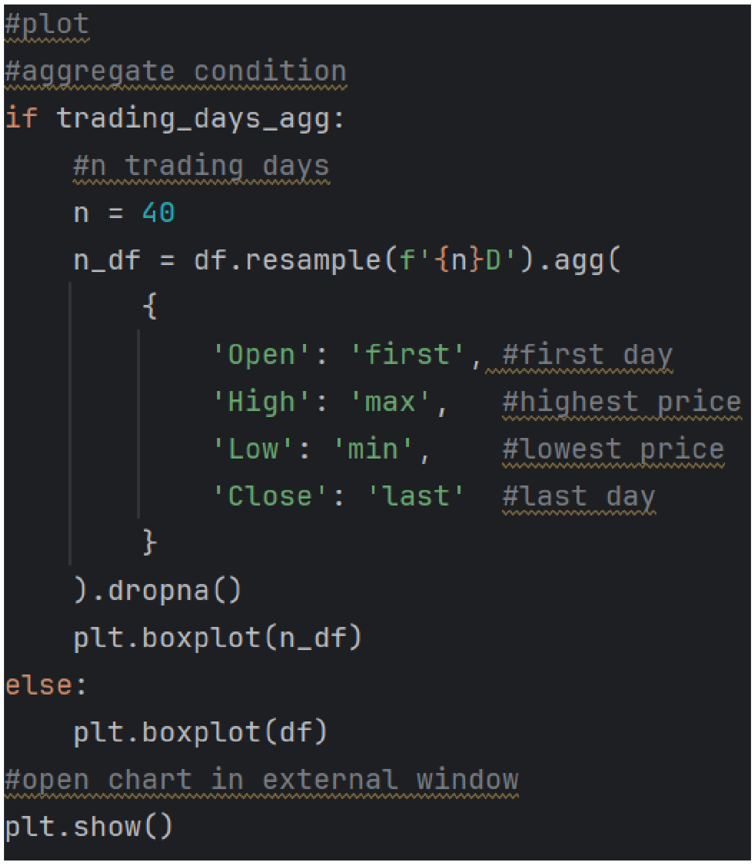
```
#year-month-day
TRAIN_START_DATE = '2023-01-01'
TRAIN_END_DATE = '2024-02-01'
TEST_START_DATE = '2024-02-02'
TEST_END_DATE = '2024-04-02'
```

Boxplot Chart

The boxplot chart was a lot simpler to work with, and mainly just involved parsing the dataframe into the plt.boxplot() function



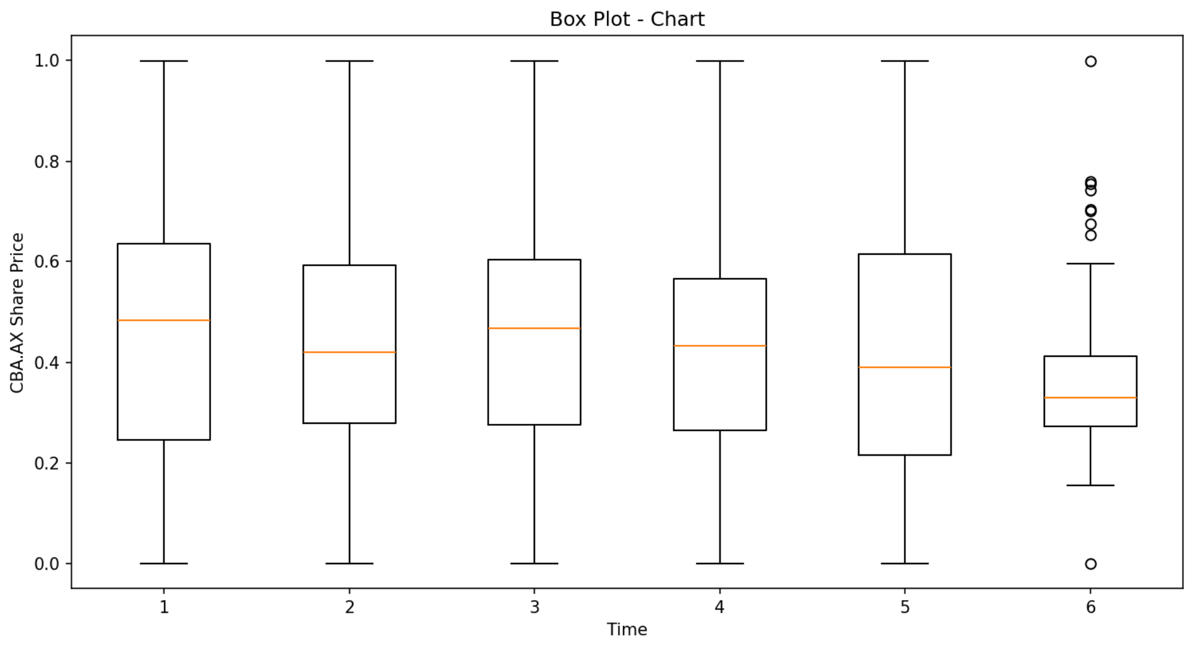
```
def box_plot_chart(data, trading_days_agg):
    print("Boxplot chart")
    df = pd.read_csv(data)
    #set index
    df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d')
    df.set_index( keys: 'Date', inplace=True)
    #add labels
    plt.figure(figsize=(12, 6))
    plt.xlabel("Time")
    plt.ylabel(f"{COMPANY} Share Price")
    plt.title("Boxplot Stick - Chart")
```



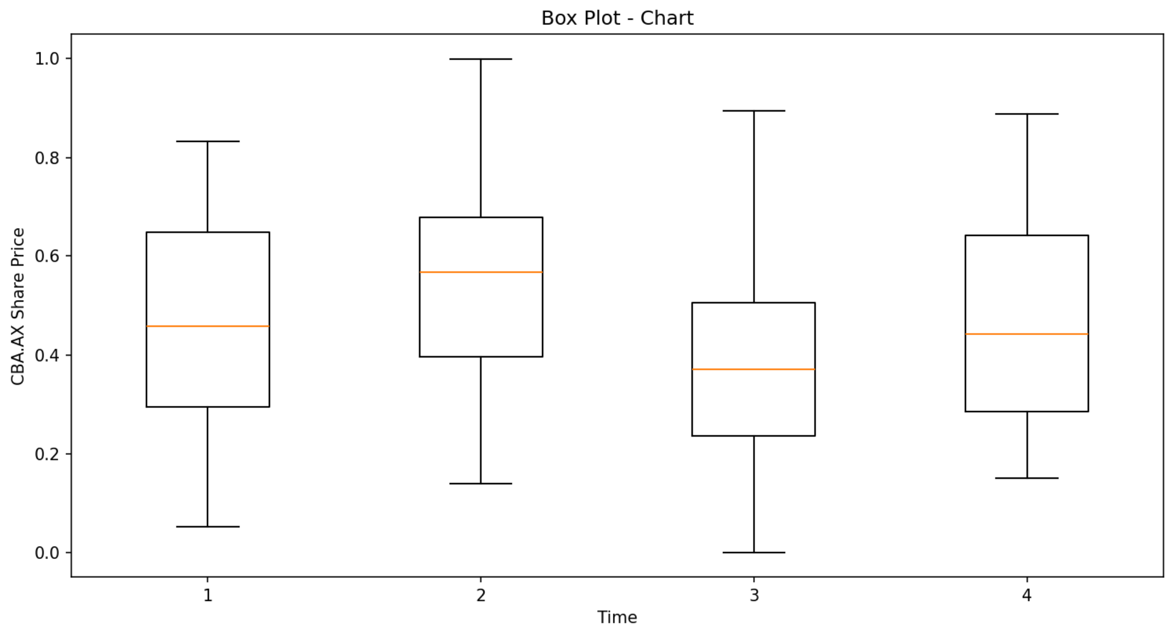
```
#plot
#aggregate condition
if trading_days_agg:
    #n trading days
    n = 40
    n_df = df.resample(f'{n}D').agg(
        {
            'Open': 'first', #first day
            'High': 'max',   #highest price
            'Low': 'min',    #lowest price
            'Close': 'last'  #last day
        }
    ).dropna()
    plt.boxplot(n_df)
else:
    plt.boxplot(df)
#open chart in external window
plt.show()
```

Results

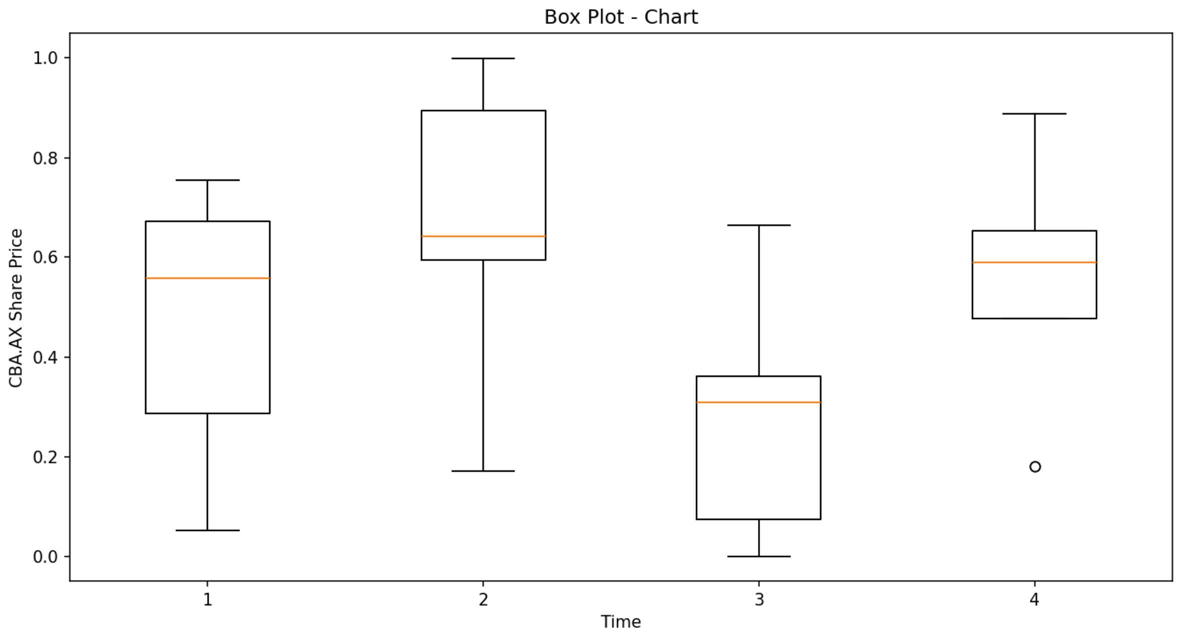
<u>With trading_days_agg set to False</u>



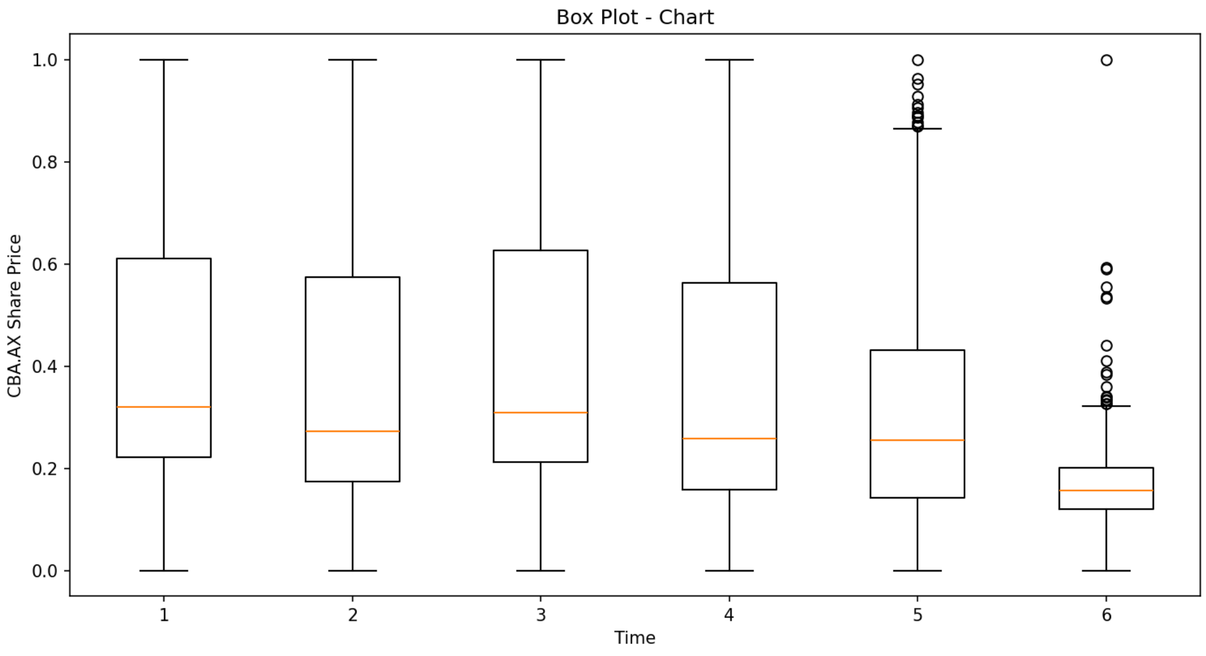
<u>With it set to True, with n being 5</u>



<u>With n set to 20</u>



<u>Example with the TRAIN_START_DATE set to 2023 (for demonstration)</u>



Useful resources:

https://www.earthdatascience.org/courses/use-data-open-source-python/use-time-series-data-in-python/date-time-types-in-pandas-python/resample-time-series-data-pandas-python/

https://www.geeksforgeeks.org/box-plot-in-python-using-matplotlib/

https://wellbeingatschool.org.nz/information-sheet/understanding-and-interpreting-box-plots

https://www.coinbase.com/en-au/learn/tips-and-tutorials/how-to-read-candlestick-charts

---

File path: *project/venv/scripts/stock_prediction3.py*