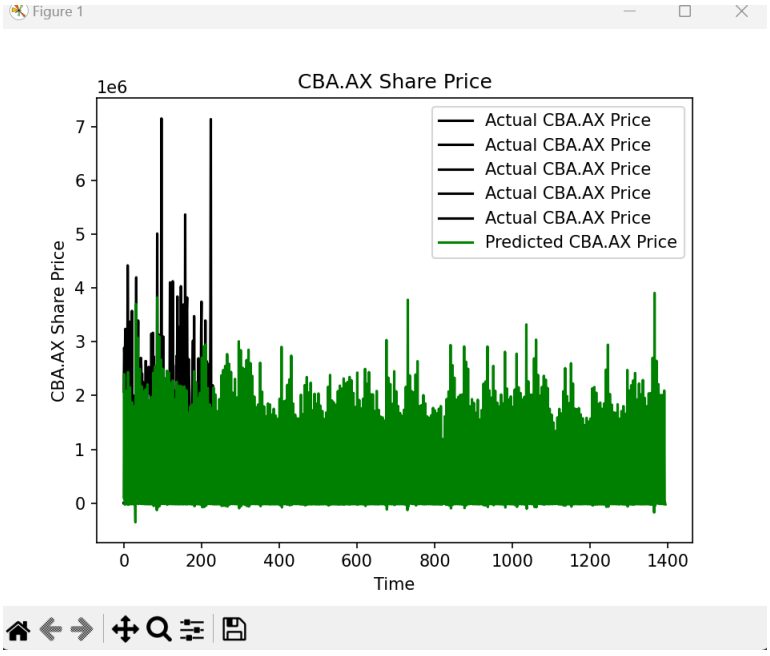
B.2

In order to create a function with the required features, I had a look at the P.1 code, and also went over that YouTube tutorial again. I noticed the function in P.1 takes a ticker value, which, on research, is a unique identifier for a company on the stock exchange, consisting of 1-5 letters.

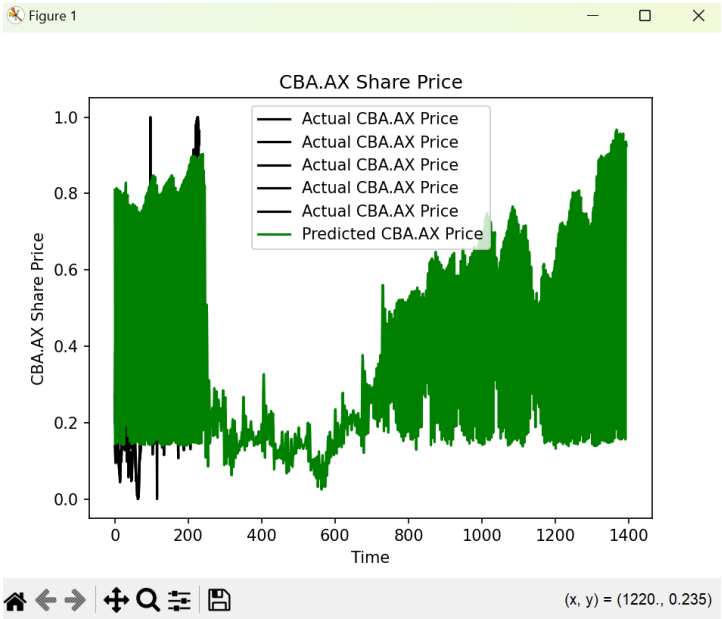
I also found that a scaler is a value between 0-10. The scaler is used to compress values higher than that to between 0-10. The ticker was not necessary for this task, but the scaling was, as well as handling NaN values, splitting the data, and saving the data.

I'm really sorry about the wait for this task, but I was confused about the results for my program, and I was not doing well (ASD related, plus had the flu). I thought there was an issue with the scaling, but I'm not exactly sure what the issue is now. Looking over the task document, I believe I have achieved the requirements of the task, regardless. If there is an issue I will resolve it in the next task.

<u>This is the initial result before scaling the columns (and not date splitting)</u>



And after scaling the columns




```
'column_scaler': {'Adj Close': MinMaxScaler(), 'Volume': MinMaxScaler(), 'Open': MinMaxScaler(), 'High': MinMaxScaler(), 'Low': MinMaxScaler()}}
```

As you can see in the terminal output, MinMaxScaler has been applied to the columns. The issue that I couldn't understand is the chaotic nature of the green on the graph. There is a lot of shading in of green, so I assumed there was something wrong, but if there is, I'm yet to figure out how to fix it. I stored the scalers to a dataframe and returned it at the end of the function.

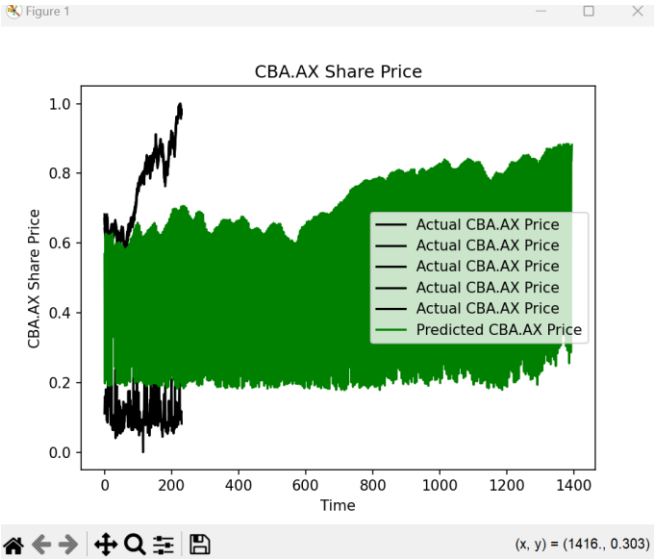
For this task, it was my understanding that I needed to make a function that loads data for the Yahoo yfinance dataframe, with parameters that let me pass the start and end dates for the dataframe. So, I put the initialisation of the dataframe inside the function, and passed the date parameters into the dataframe declaration, so that when the function is called elsewhere, different dates can be specified for the data loading.

For the NaN issue, I used the dropna() function. ffill() and bfill() were an option but the P1 code used dropna() so I went with what they did.

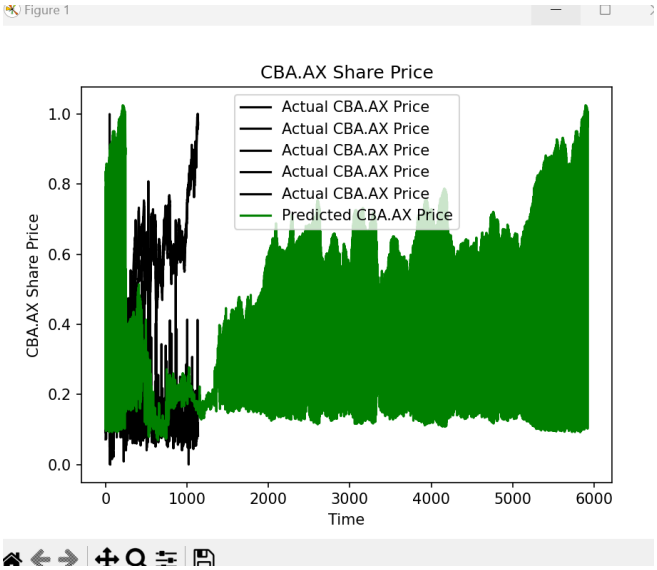
For splitting the data, I followed the general layout from the P1 code but changed what they labelled as "split_by_date" as it didn't look like it was splitting by date to me, but samples rather.

At the end of the function, I saved the dataframe to a csv file using *df.to_csv('data.csv', index=False)*. On the first run, this would create the data.csv file, so I went back to the top and created an if condition to set the dataframe to *df.read_csv('data.csv')*, if it exists, else, df would be set to the usual *yf.download(COMPANY, start=tr_start_date, end=te_end_date)*.

Splitting by date



Splitting randomly



When it came to splitting the data randomly, I used the train_test_split function, but there was an error (I believe regarding n_samples=0), so I implemented some extra code that was in P1 to see if that was related, and it fixed the issue. It seems it had something to do with time series prediction.