COS30018- Option B- Task 3: Data processing 2

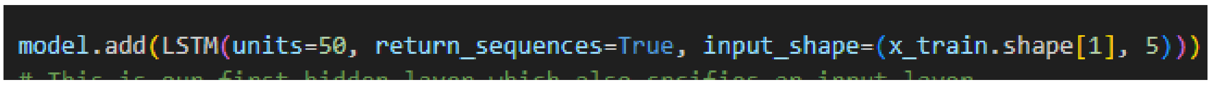
Cody Cronin-Sporys

103610020

Preparation:

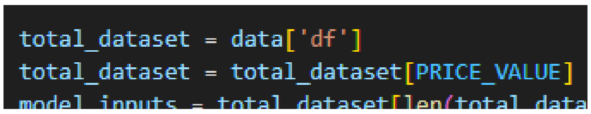
For the first part of this task we want to create a candlestick to represent the stock market financial data. A candlestick is used to show the data of different feature columns (open, close, high, low) in a meaningful and intuitive way. The thick body of the candlestick shows the difference between close and open price, while the thin lines show the variance between high and low prices. When the close price is higher than the open price, that represents an increase in price and is represented by a green candlestick.

Firstly, I needed to fix an issue that had occurred due to the change in code from last weeks task. The model was not being built due to the input shape being incorrect. This was because the x_train array now stores 5 values per line instead of 1, and was fixed in the code here:

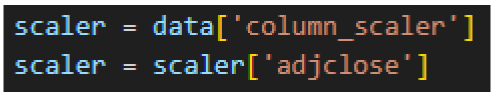


```
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 5)))
# This is our first hidden layer which also specifies an input layer
```

Next the test dataset was not being created correctly due to a key issue when concatenating the test and training data. I tried changing the total dataset to fix this but it created another issue with the scaler not having a transform function because it is a dict. This was fixed by making adjusting the scaler to use the adjclose feature column.



```
total_dataset = data['df']
total_dataset = total_dataset[PRICE_VALUE]
model_inputs = total_dataset[len(total_data
```



```
scaler = data['column_scaler']
scaler = scaler['adjclose']
```

This then caused another issue. Due to the fix with the x_train data and the model shape, there was now a mismatch between test data and model expected shape. The model expected 5 feature columns but the current test data process only uses close. To fix this I removed total_dataset[PRICE_VALUE] line from the code. This caused other issues so I looked through the p1 code and realised that most of the code in v0.1 was unnecessary, so I commented it out and replaced it with the x_test and y_test datasets which fixed all the problems after some scaling and inverse scaling, the code now runs fine and I can begin working on the candlestick.

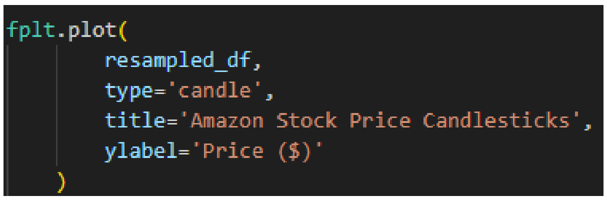
Candlestick:

To get candlesticks working we first need to install new libraries and update the requirements text file with the following:

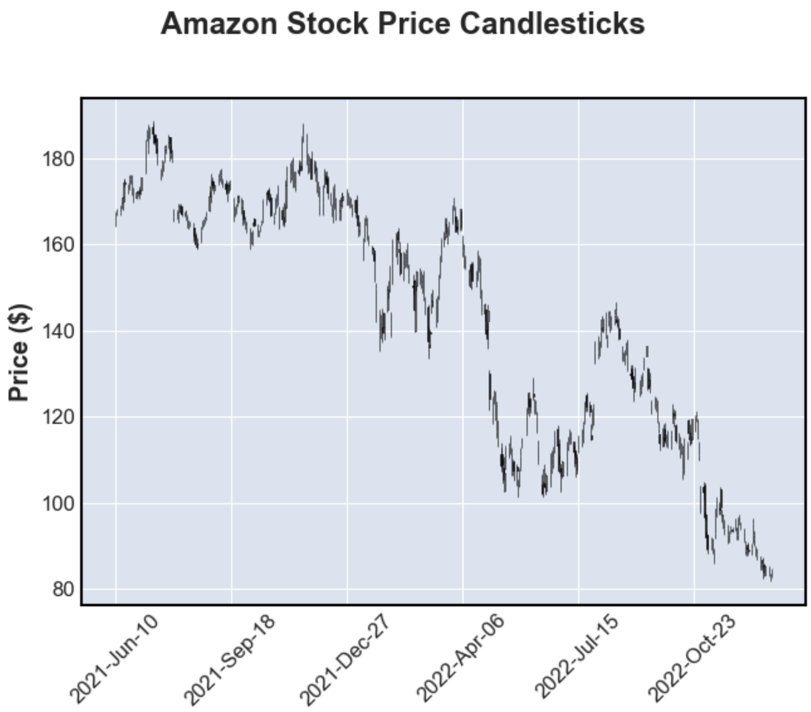
TA-lib, mplfinace

However trying to install talib failed, After some research it seems that talib needs to be installed twice, first through c then through python. For now I'll move on since I can still make the required candlestick without the talib library.

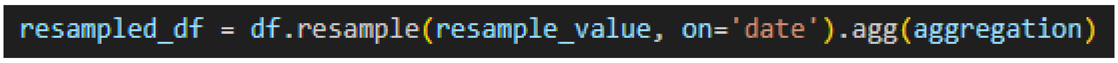
To create the candlestick we use the mplfinace plot function to create the candlesticks. Using the dataframe with all the feature columns (Open, High, Low, Close) as the first parameter as well as setting the type to candle, we can create a basic candlestick graph:



```
fplt.plot(
        resampled_df,
        type='candle',
        title='Amazon Stock Price Candlesticks',
        ylabel='Price ($)'
    )
```

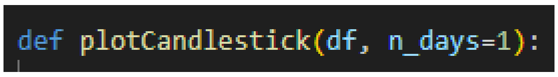


The immediate problem with this graph is that there are too many data points to even see the candlesticks. The solution is going to be implementing a parameter for the candlestick function that allows n days of data per candlestick. To do this we will resample the data frame using pandas resample function.

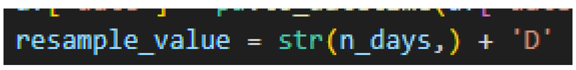


```
resampled_df = df.resample(resample_value, on='date').agg(aggregation)
```

The first parameter is the amount we're resampling by and is a combination of the letter "D" for days (instead of "W" for weeks for example) and the value n_days, which is a parameter of the boxplot function.



```
def plotCandlestick(df, n_days=1):
```

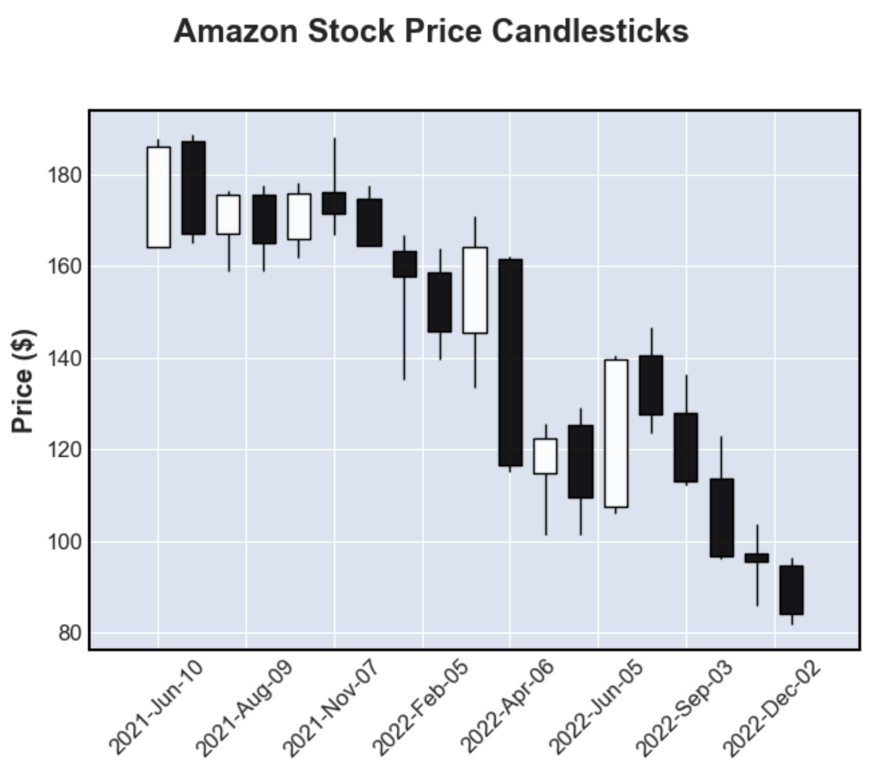


```
resample_value = str(n_days,) + 'D'
```

The on parameter is set to 'data', which specifies the column to use for resampling, this column must be in datetime format so we convert it before resampling.

```python
df['date'] = pd.to_datetime(df['date'])
```

Finally the .agg is short for aggregation and is the method we use for resampling the data. The aggregation variable is a dict, which contains a rule for each column on how they will be aggregated. For example, if we resample 5 days into 1, open will take the first of those 5 days for the result, while high will take the largest value of the 5 days. If we now call the boxplot function with 30 as a value for n_days (so each candlestick contains 30 days of data) we get this graph:



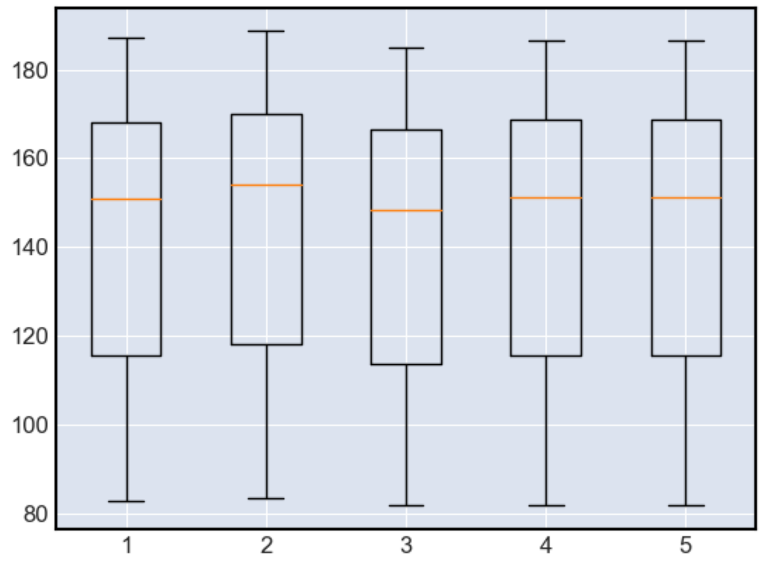
Boxplot:

To create the boxplot we will use the matplotlib boxplot function. To use this we first need to create a new data array since the boxplot function won't accept the current dataframe. This data array will include all the feature columns from the dataframe but not the date.

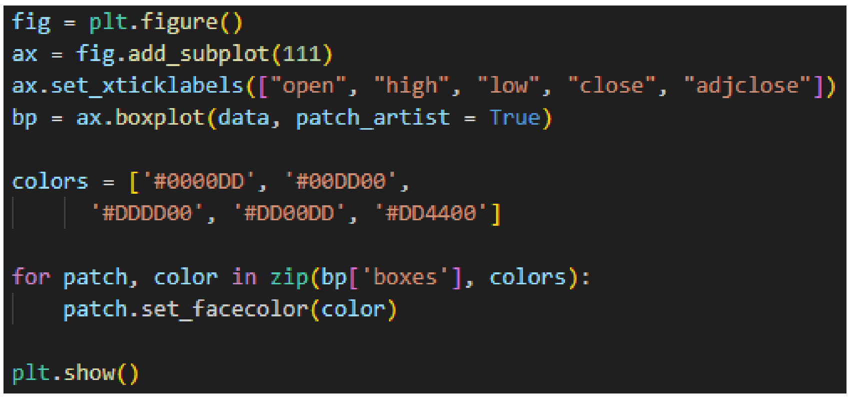
```python
# There should be one boxplot per column, so a boxplot for each of open,
data = [df['open'], df['high'], df['low'], df['close'], df['adjclose']]
```

We then pass this data value as a parameter for the boxplot function and use the show function to display the graph. To explain the boxplot graph, each segment of the boxplot makes up a quarter of the dataset, the very middle is the median, the 2 ends of the central box are the lower and upper quartiles, while the very ends of the whiskers are the minimum and maximum values (expect for outliers)

```python
plt.boxplot(data)
plt.show()
```



Now that we have the basic boxplot function working, we can add some extra lines of code to label the axis with the column names and recolour the different boxplots.

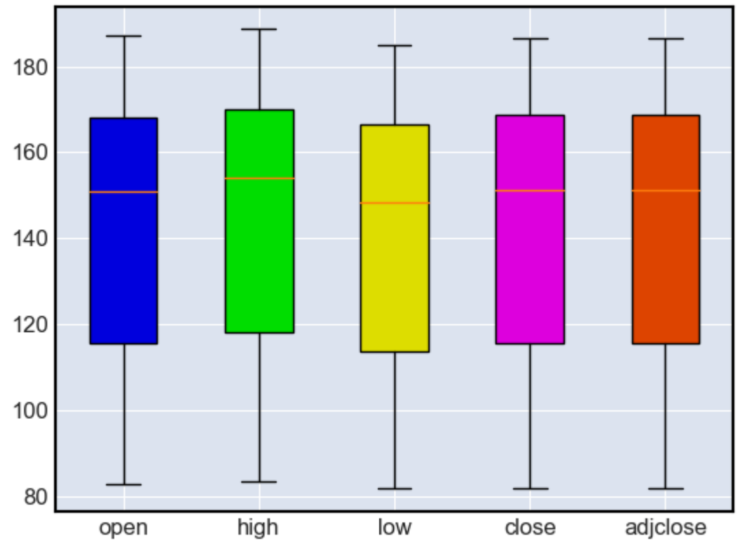


```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_xticklabels(["open", "high", "low", "close", "adjclose"])
bp = ax.boxplot(data, patch_artist = True)

colors = ['#0000DD', '#00DD00',
        '#DDDD00', '#DD00DD', '#DD4400']

for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)

plt.show()
```


open
high
low
close
adjclose