Introduction/ Background

Stock price prediction uses data analysis and statistical models to predict future stock prices. Stock market forecasting assists traders and investors in making more educated choices on the purchase, sale, or holding of stocks. We will use the following dataset from Yahoo Finance with the following link: https://finance.yahoo.com. This website provides historical and real-time financial data for stocks, bonds, ETFs, and other financial instruments. Specifically, for our data set we will be using the historical data of one particular stock, Apple, from January 1st, 2018 to December 30, 2022, in intervals of 1 week to make our prediction. For subsequent work on our project we plan on potentially introducing new stocks into our testing. Our dataset was already practically cleaned, but we will discuss one method we used to further optimize our parameters shortly.

Problem Definition / Motivation

The motivation for this project largely remains the same, and it is that many people do not invest in the stock market because they fear losing money. That also causes them to lose money because interest rates make your money worth less yearly. We want to help those people by creating an easy way for them to understand the stock they are considering buying and how it has behaved over time. It's a part of the reason we want to have our data over a longer period of time categorized by weeks; so we can more clearly explain and predict the stock price over such an interval. Stock price prediction using machine learning is a rapidly expanding research field. Some of the researched topics are related to machine learning, such as text mining, machine vision, voice classification, etc.

Methods

We used many different supervised learning models past our midterm report, such as linear regression, random forest, Long Short-Term Memory (LSTM) , and Autoregressive Integrated Moving Average (ARIMA) to see which worked best towards our stock price prediction.

Linear Regression:

*The main library we used for our regression methods detailed below was sklearn

*Many of the graphs below can be found in the jupyter notebook labeled ML_project as well

Data Collection: In terms of our dataset itself, we initially had 6 parameters: Open, High, Low, Close, Adj Close, and Volume. We began by deleting "Adj Close" as we were simply planning to use Close for our prediction (Adj Close was just redundant). We then proceeded to expand our 5 parameters for a particular week to 15 parameters by looping through and finding the data for the prior three weeks as well. Our model plans on using the 5 params from each of the previous

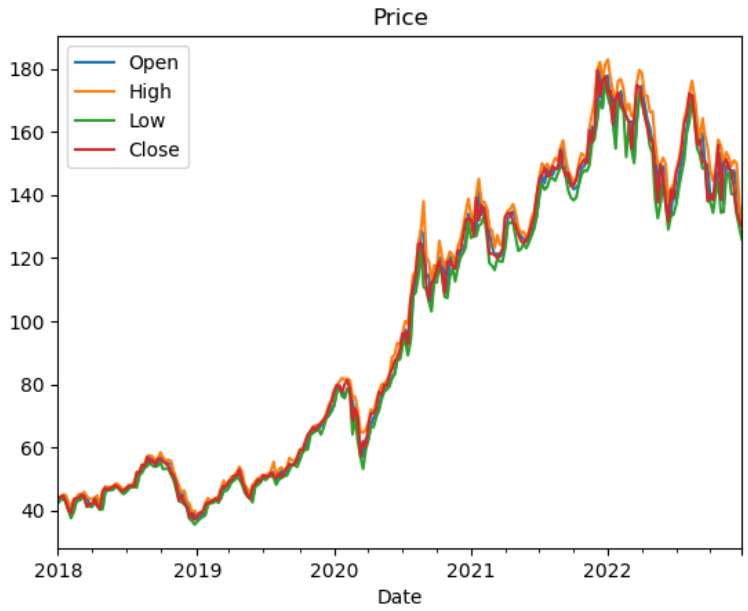
weeks to calculate the Close value for any particular given week. In other words, our data set now showed the Open, High, Low, Close, and Volume not only for the current week but also for the prior week, two weeks prior, and three weeks prior as shown below. We planned to use the current week parameters (close in particular) for testing, and the three prior weeks (with respect to each data point) for training. Of course, this would mean the first three weeks in our dataset would be incomplete (without a full set of 15 parameters) so we disregarded those weeks giving us a 258 x 20 matrix (258 data points, 20 features including the current week), with 80% of our data (206 data points) for training and 20% of our data for testing (52 data points) validation. We also display this data set categorized into two graphs, one being the features (open, high, low, close) vs date being displayed and the other showing the trading volume over time. It's hard to tell but the purpose of these visualizations is to see that there is some sort of trend we can hope to predict, and also show that there are margins for profit.

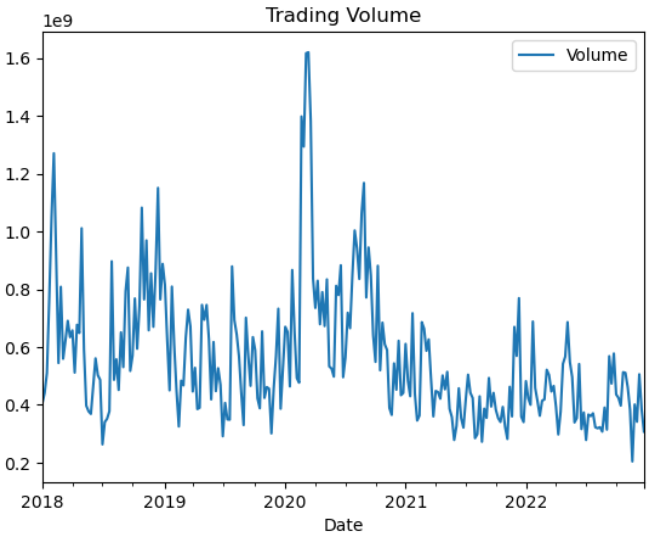
```
[ ] # Drops the first 3 rows, which have NaN
    data.dropna(inplace=True)
```

```
[ ] data
```

Date	Open	High	Low	Close	Volume	Open-1	High-1	Low-1	Close-1	Volume-1	Open-2	High-2	Low-2	Close-2	Volume-2	Open-3	High-3	Low-3	Close-3	Volume-3
2014-01-22	19.675358	19.903214	17.931070	18.089287	2832981200	19.768572	20.007143	19.282143	19.609644	1.376183e+09	19.243214	19.530714	18.924286	19.513929	1.553709e+09	19.845715	19.893929	19.057142	19.287144	1.356972e+09
2014-01-29	17.998215	18.195000	17.626785	18.171070	2424254000	19.675358	19.903214	17.931070	18.089287	2.832981e+09	19.768572	20.007143	19.282143	19.609644	1.376183e+09	19.243214	19.530714	18.924286	19.513929	1.553709e+09
2014-02-05	18.091429	19.205357	18.080357	19.141430	1584206400	17.998215	18.195000	17.626785	18.171070	2.424254e+09	19.675358	19.903214	17.931070	18.089287	2.832981e+09	19.768572	20.007143	19.282143	19.609644	1.376183e+09
2014-02-12	19.176786	19.685356	19.044287	19.499643	1148674800	18.091429	19.205357	18.080357	19.141430	1.584206e+09	17.998215	18.195000	17.626785	18.171070	2.424254e+09	19.675358	19.903214	17.931070	18.089287	2.832981e+09
2014-02-19	19.455357	19.531786	18.607143	18.645000	1419272400	19.176786	19.685356	19.044287	19.499643	1.148675e+09	18.091429	19.205357	18.080357	19.141430	1.584206e+09	17.998215	18.195000	17.626785	18.171070	2.424254e+09
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2019-11-27	66.394997	67.062500	64.072502	64.862503	320770800	66.385002	66.790001	65.099998	66.072502	4.821828e+08	65.282501	67.000000	65.267502	66.572502	4.549940e+08	64.192497	65.697502	63.842499	65.489998	4.100012e+08
2019-12-04	65.267502	67.750000	65.169998	67.120003	466144400	66.394997	67.062500	64.072502	64.862503	3.207708e+08	66.385002	66.790001	65.099998	66.072502	4.821828e+08	65.282501	67.000000	65.267502	66.572502	4.549940e+08
2019-12-11	67.202499	70.442497	66.830002	70.102501	591999200	65.267502	67.750000	65.169998	67.120003	4.661444e+08	66.394997	67.062500	64.072502	64.862503	3.207708e+08	66.385002	66.790001	65.099998	66.072502	4.821828e+08
2019-12-18	69.949997	71.222504	69.639999	71.067497	637426400	67.202499	70.442497	66.830002	70.102501	5.919992e+08	65.267502	67.750000	65.169998	67.120003	4.661444e+08	66.394997	67.062500	64.072502	64.862503	3.207708e+08
2019-12-25	71.205002	73.492500	71.175003	72.879997	383501600	69.949997	71.222504	69.639999	71.067497	6.374264e+08	67.202499	70.442497	66.830002	70.102501	5.919992e+08	65.267502	67.750000	65.169998	67.120003	4.661444e+08

310 rows × 20 columns





Dimensionality Reduction: Before we went ahead and used PCA for our dimensionality reduction, we first had to standardize our data. After scaling our dataset , we went ahead and preprocessed using PCA on our dataset with a target variance of 97 percent to determine the best number of parameters. We also attempted dimensionality reduction with SVD without any conclusive results. However, using PCA we saw that our optimal number of parameters was 3, so we reduced the dataset to 3 parameters.

Supervised Learning Method: Linear Regression: As mentioned earlier, before we began our supervised learning, we split our data set into 4 components x_train, x_test, y_train, y_test in a 80/20 split for the training and testing respectively. We then trained the model, instantiating a linear regression model (from sklearn), fitting the x_train with the y_train and using the x_train to predict our potential y's labeled as train_y_prediction.

Then, the training loss was calculated using mean_squared error (between our initially set aside y_train points and our recently calculated train_y_prediction points). We also found the mean absolute error between these two values.

Finally, we repeated the process to see how our predicted y values did against our testing data so instead of y_train (like in the previous step), we calculated our error metrics using y_test, the set of data that we initially set aside.

Improving Linear Regression:

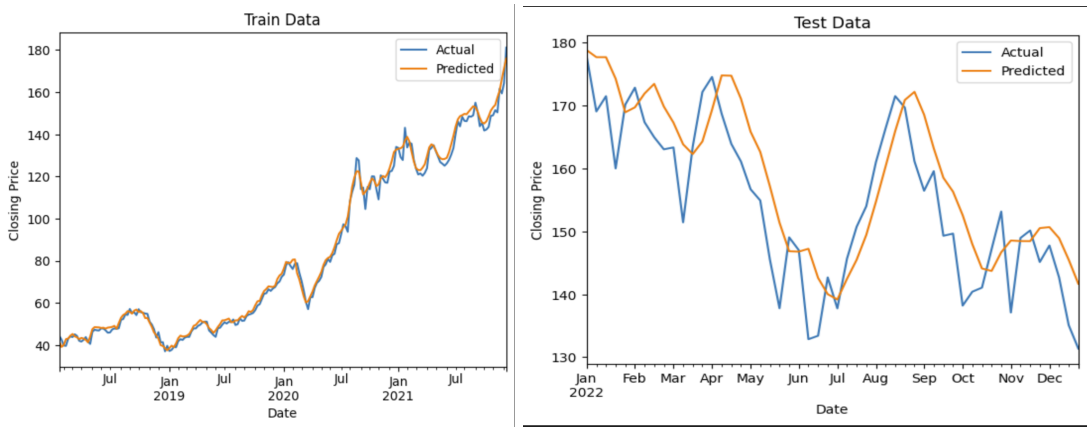
In terms of linear regression, the main purpose of our improvements involved fine tuning hyper-parameters and determining the optimal chosen number of features to keep when utilizing svd and pca feature reduction.

In our original testing, we used three weeks worth of past data to attempt to predict the weekly closing price of our chosen stock type.
The first major decision we made was to switch to predicting the weekly open price of a stock rather than the closing price, as predicting the closing price led to loss of a week's worth of data belonging to the current week being predicted.

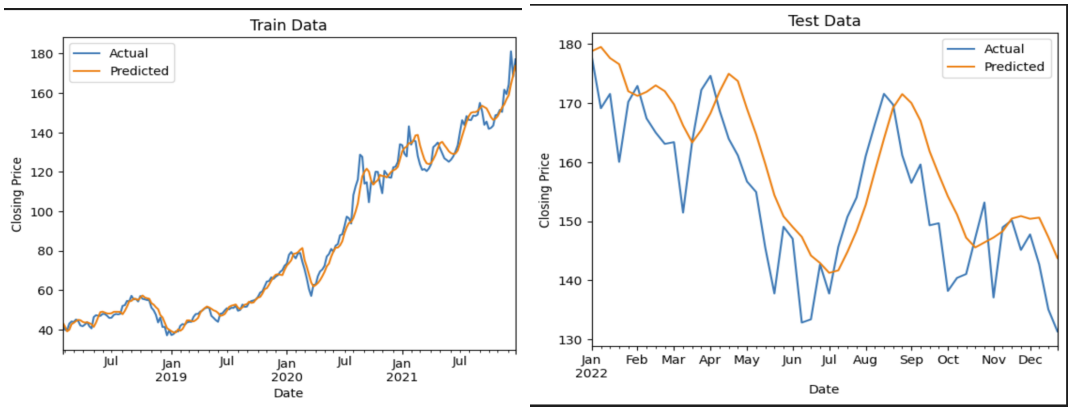
Hyper-parameter Tuning

Our edits to the hyper parameters revolved around using different amounts of past week data to predict the current week's open price. Originally, we used 3 weeks worth of past data in our predictions. In our testing we tried different hyperparameters of using 4 weeks and 5 weeks worth of past data in our predictions, increasing the number of features. In each test, pca and SVD were applied to feature reduce the set and RMSE was used as a metric to determine the optimal hyperparameter amount.

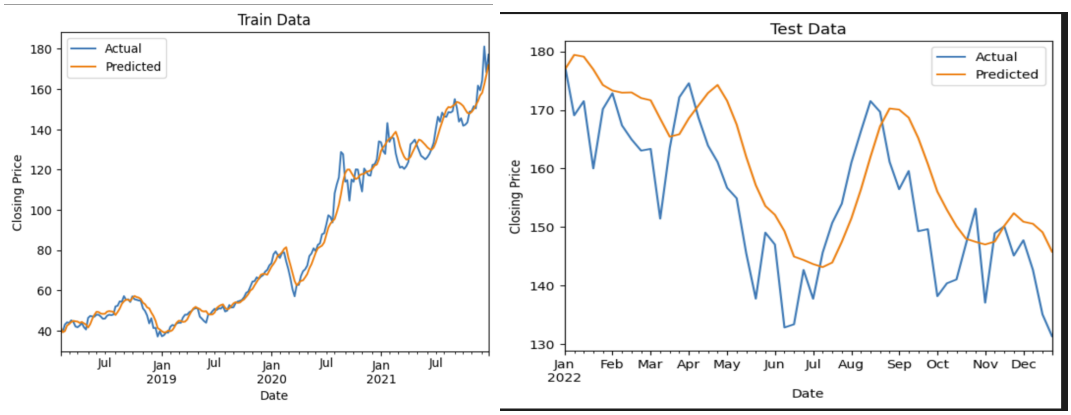
3 Weeks



4 Weeks



5 Weeks



RMSE 3 weeks : 59.266870018388246
RMSE 4 weeks: 78.63299118775484
RMSE 5 weeks: 99.71936775777125

Based on this data we determined that 3 weeks of past data was the optimal hyperparameter choice. 4 and 5 weeks appeared to skew the data to harshly as predictions would use information from weeks that would have little to no impact on the current week's stock prices.

The linear regression results for test data and train data appear to more closely correlate with the actual prices of 2 weeks prior instead of the current week when comparing predicted to actual prices. Shifting the predicted data 2 weeks over would reduce the RMSE to just over 20%. However we were unable to accomplish this shift as it would suggest that our stock prediction model would be using data from future weeks not yet available to it.

Long Short-Term Memory (LSTM)

LSTM is a type of recurrent neural network that can analyze time-series data and identify patterns to make predictions about future values. The model is capable of processing a large amount of data, making it suitable for analyzing stock market trends. By inputting historical stock data, including price, volume, the LSTM algorithm can learn patterns and correlations, and make predictions about future price movements.

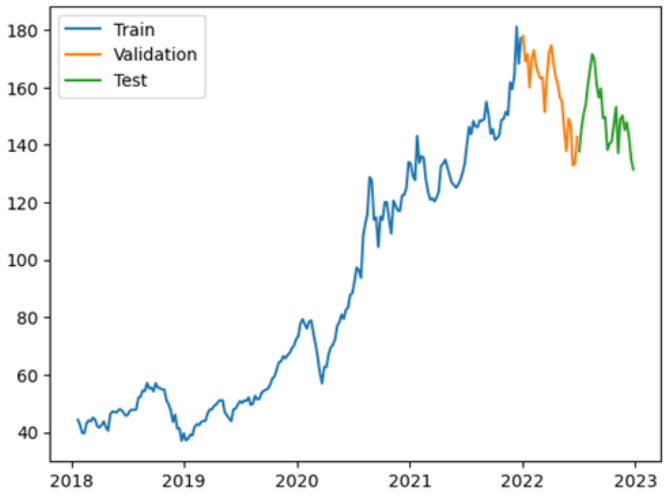
Similar to the Linear Regression, we used data of the previous 3 weeks to predict the opening stock price of a particular week. The data collection and dimensionality reduction procedure used is the same.

We used a 4-layer model with one LSTM layer followed by 3 dense layers. It has 20k parameters.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_13 (LSTM)              (None, 64)                16896

 dense_39 (Dense)            (None, 32)                2080

 dense_40 (Dense)            (None, 32)                1056

 dense_41 (Dense)            (None, 1)                 33

=================================================================
Total params: 20,065
Trainable params: 20,065
Non-trainable params: 0
```

We used MSE loss, Adam optimizer and Mean Absolute Error as metric to evaluate the performance during training.

We used a 80/10/10 split. 80% data for training, 10% for validation and 10% for testing. This is depicted in the visualization below.



We trained the model using model.fit function in tensorflow. We then passed training data, validation data and test data to find the predicted ys and calculate the respective error metrics.

```
Train - Mean Squared Error =  10.349279784636026
Train - Mean Absolute Error =  2.326456088464237

Validation - Mean Squared Error =  50.166763232949265
Validation - Mean Absolute Error =  6.12920907827524
```

```
Test - Mean Squared Error =  40.297798856116756
Test - Mean Absolute Error =  5.539185157189002
```

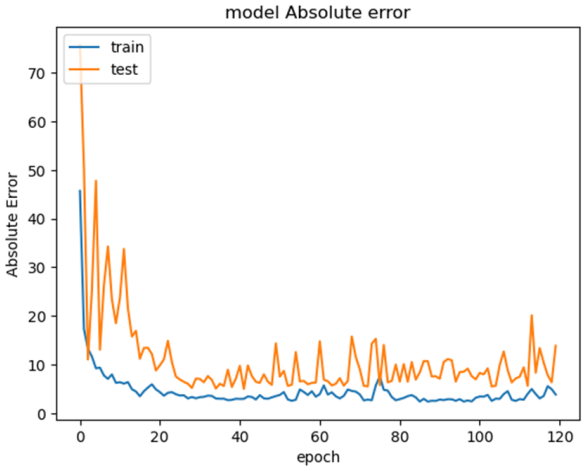
We also changed the learning rate and compared performances. Fixed epochs to 120.

LR	MSE	MAE
0.001	47.42	5.89
0.005	40.29	5.53
0.01	48.52	5.94
0.05	60.62	6.43
0.1	83.47	7.08

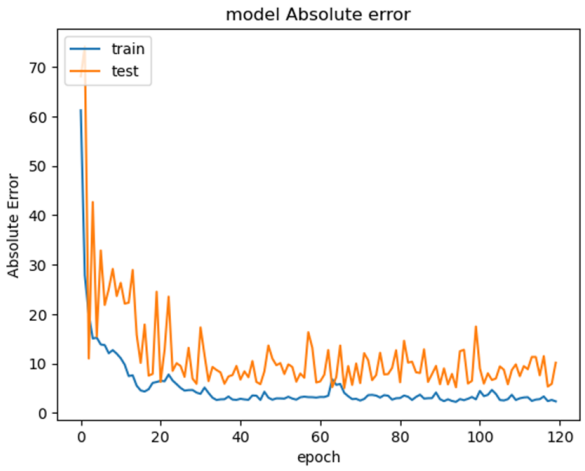
We can see the LR = 0.005 gives best performance.

We observed that as LR increases, the model starts oscillating and doesn't converge.

LR = 0.1



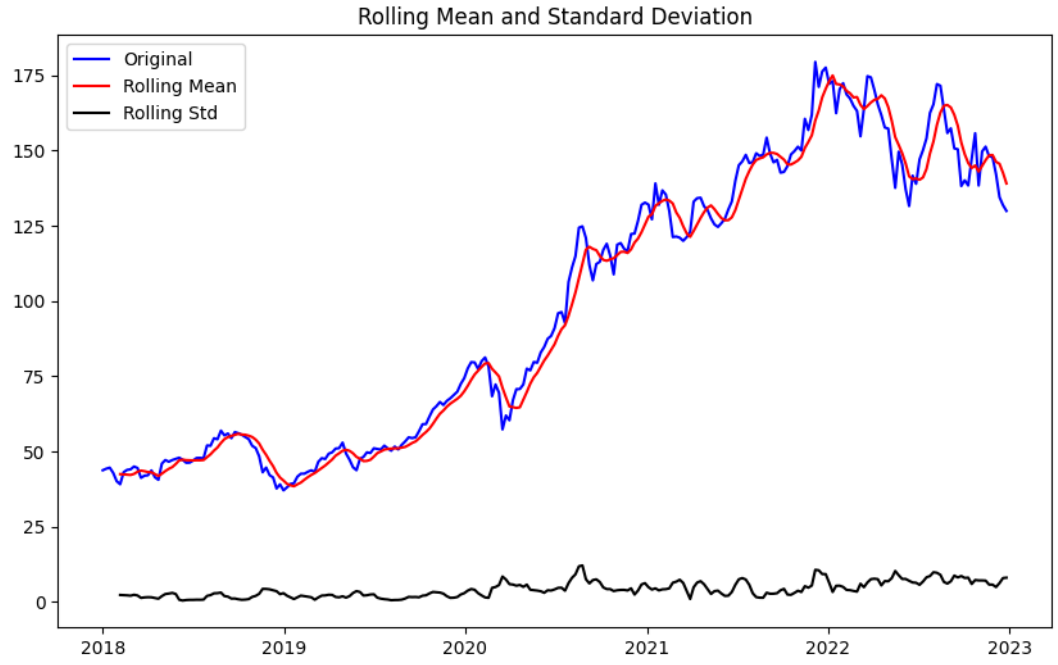
LR = 0.05



Autoregressive Integrated Moving Average (ARIMA)

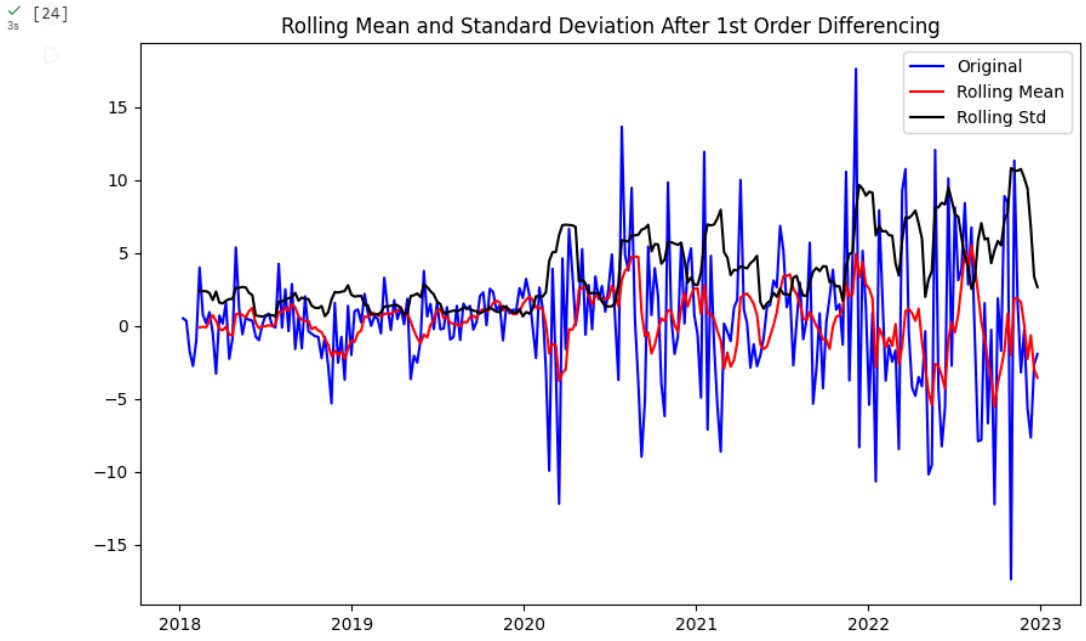
Data Collection: In terms of our dataset itself, we initially had 6 parameters: Open, High, Low, Close, Adj Close, and Volume. We began by deleting "Adj Close" as we were simply planning to use Close for our prediction (Adj Close was just redundant). Since we were using time series analysis (with ARIMA) we didn't have to do any other feature engineering as the dataset was already clean at this point. We used 80% of our data (206 data points) for training and 20% of our data for testing (52 data points) validation. We also display this data set's closing price over time as well as its autocorrelation (with respect to lag) over time. These visualizations showed us that prediction would be hard as there is no long-lasting trend or correlation that we can easily use to our advantage

Dickey-Fuller Test: Ideally, when using a time series approach to modeling like ARIMA it would be nice if the series itself was stationary (the mean and standard deviation of the series is constant). A way to test this is the Dickey-Fuller test which tests the null hypothesis that a unit root is present. In other words, if p is greater than 0, we know our time series is not stationary and otherwise (p = 0), we can reject the null hypothesis and say our time series is stationary. The results of our Dickey-Fuller test are shown below.




```
Results of dickey fuller test
Test Statistics                -1.064082
p-value                         0.729181
No. of lags used                0.000000
Number of observations used   260.000000
critical value (1%)            -3.455754
critical value (5%)            -2.872721
critical value (10%)           -2.572728
dtype: float64
Reject Null Hypothesis given P value IF trend is not stationary from visualization above
```

As seen from the p-value and the clear trend of the mean , as expected of stock prices we can verify that our time series is not stationary. So, we proceeded to use differencing on our time series to make it stationary: the results are shown below in terms of its visualization and now passing the Dickey-Fuller Test.

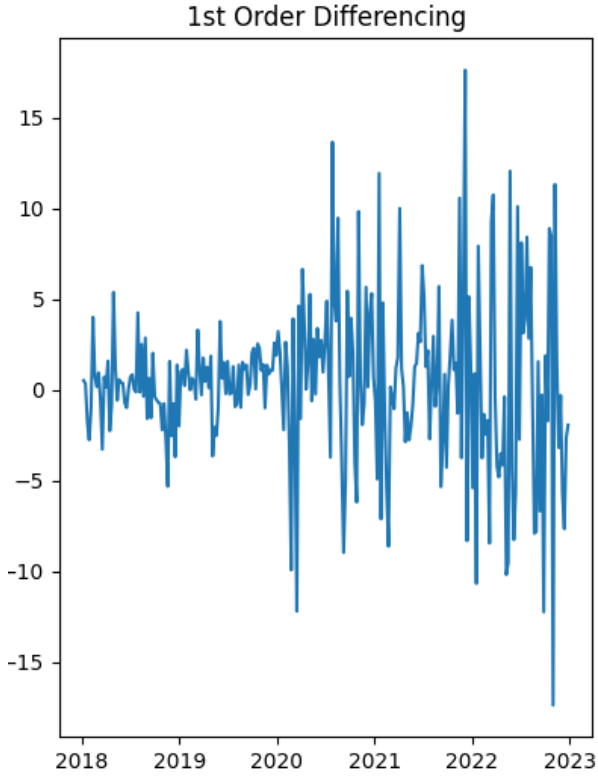


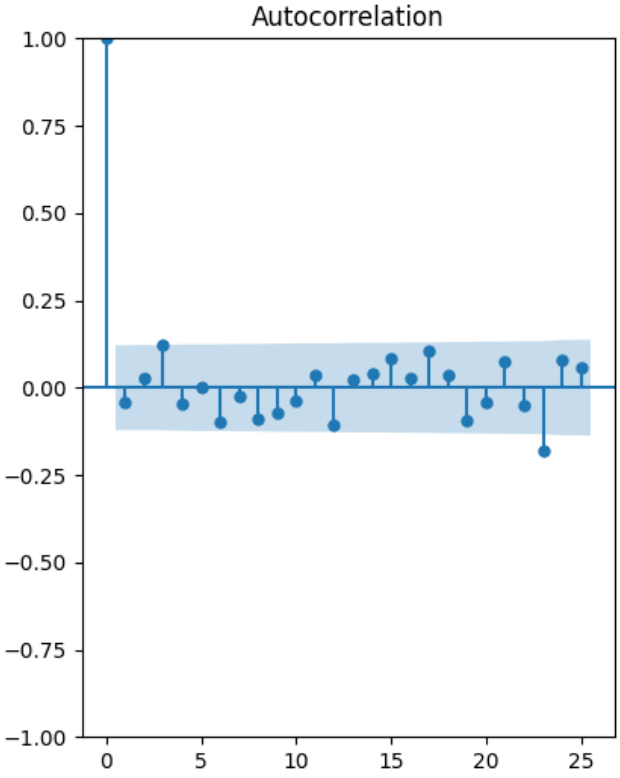

```
Results of dickey fuller test
Test Statistics                -8.014366e+00
p-value                         2.158667e-12
No. of lags used                2.000000e+00
Number of observations used     2.570000e+02
critical value (1%)            -3.456054e+00
critical value (5%)            -2.872853e+00
critical value (10%)           -2.572799e+00
dtype: float64
```

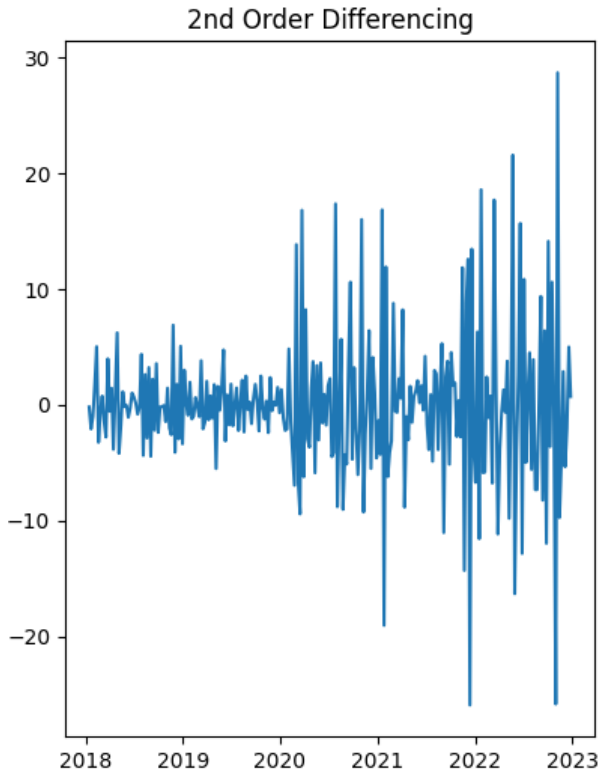
(We pass it because our p-value is < .05)
How did we know only to do 1st order differencing and not 2nd order (we can but it would be unnecessary and lead to overfitting), that is because of how we found “d” explained below.

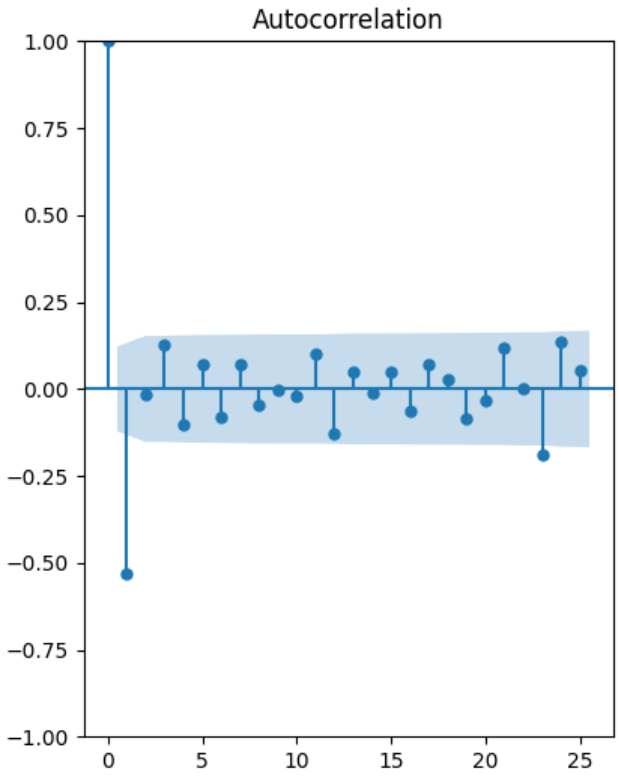
Finding p, d, and q: ARIMA’s parameters are p, d , and q which correspond to the number of autoregressive terms, the nonseasonal differences, and the number of lagged forecast errors in the prediction respectively. We are going to find these parameters using 2 methods, one with visualizations (involving pacf and acf) and the auto_arima model.

- Visualization Method:
    - Finding d: d can only be optimal until a value of 2, so we can use that too our advantage. So we took both the first and second order differencing visualizations with autocorrelation to see which would be the optimal parameter. As we can see from below, as well as the (mini) Dickey-Fuller Test, that 1st order differencing gives us our desired p-value since the autocorrelation plot for 1st order is not positive for several consecutive lags.


1st Order Differencing
15
10
5
0
−5
−10
−15
2018
2019
2020
2021
2022
2023


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25

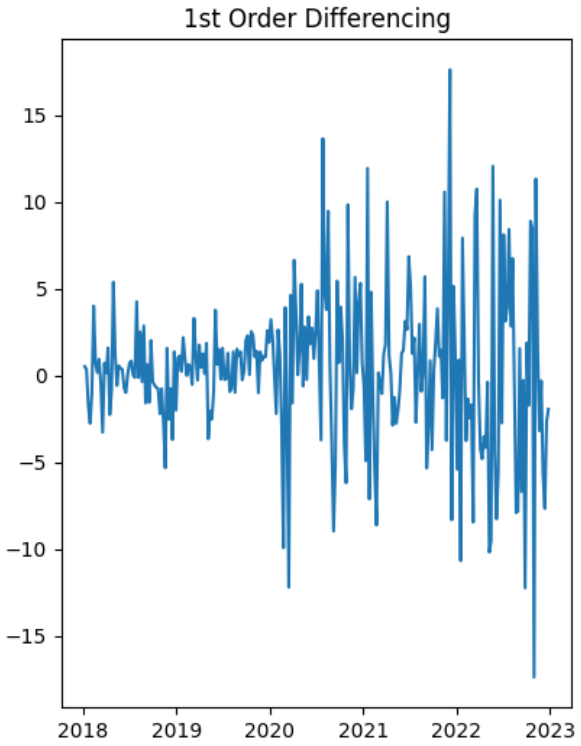

2nd Order Differencing
30
20
10
0
−10
−20
2018
2019
2020
2021
2022
2023

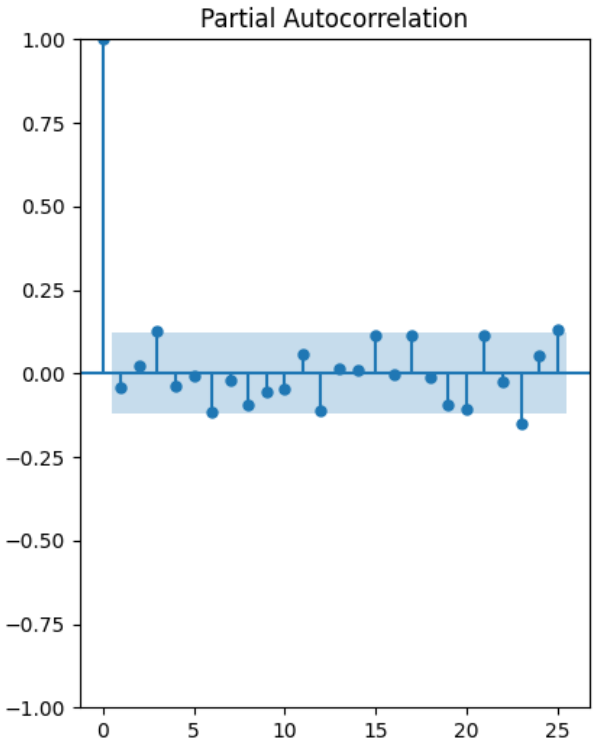

Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25

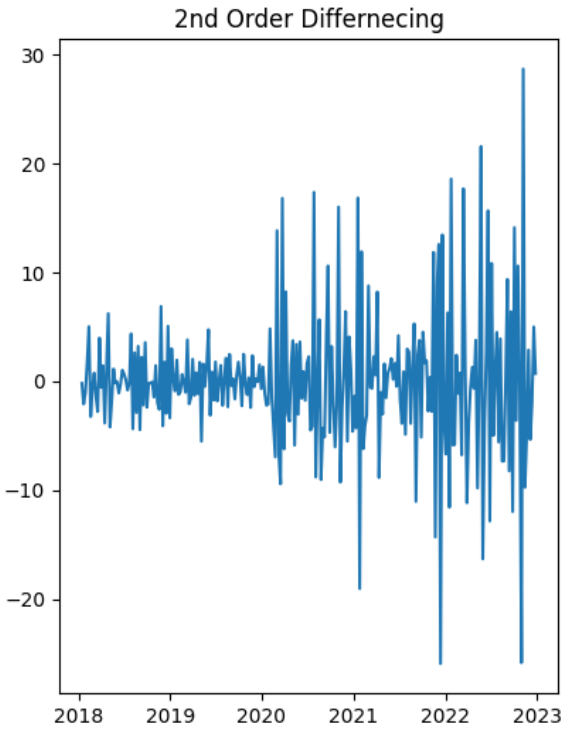
```
#Mini Augmented Dickey-Fuller test - verify that d should be 1 when p < 0.5
res = adfuller(actualClosePrice.dropna())
print('pvalue:' ,res[1])

res = adfuller(actualClosePrice.diff().dropna())
print('pvalue:' ,res[1])

res = adfuller(actualClosePrice.diff().diff().dropna())
print('pvalue:' ,res[1])
```

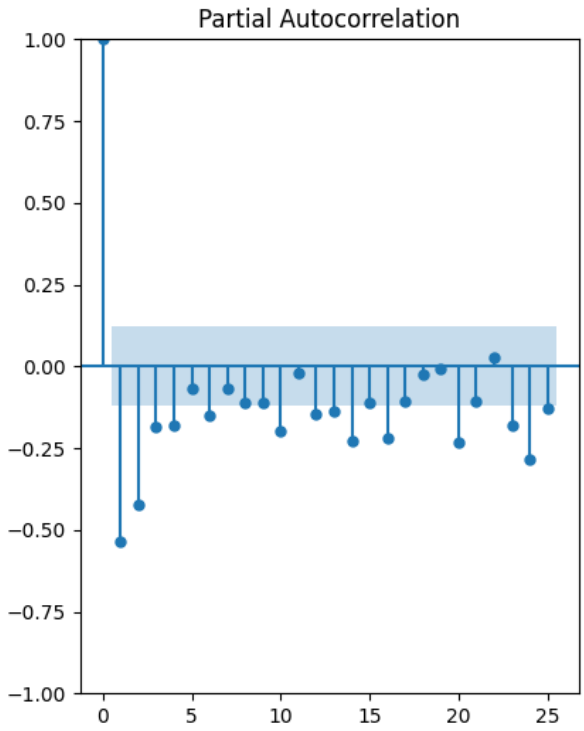
```
pvalue: 0.7291809039267443
pvalue: 2.1586674302151083e-12
pvalue: 6.386345447942726e-13
```

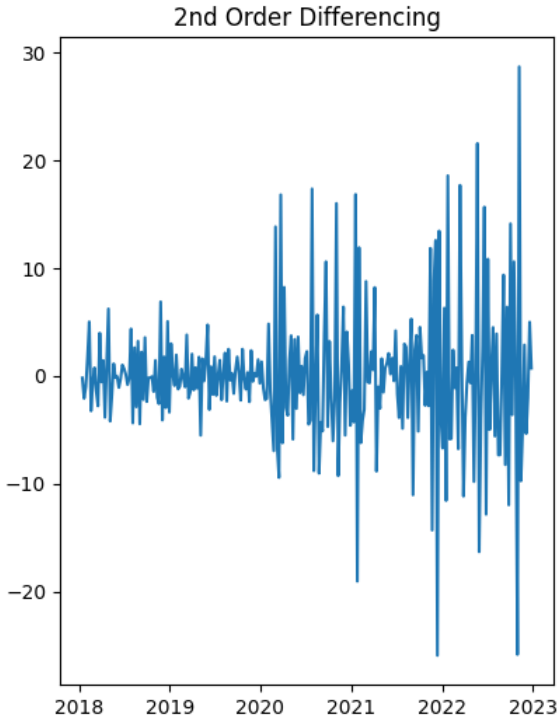
- Finding p: For p, we used the partial autocorrelation function plot to see a correlation between the time series and lag. In short, we can see here that a parameter of 0 would be best suited for our model, since the 0th lag is the most significant (most correlation)

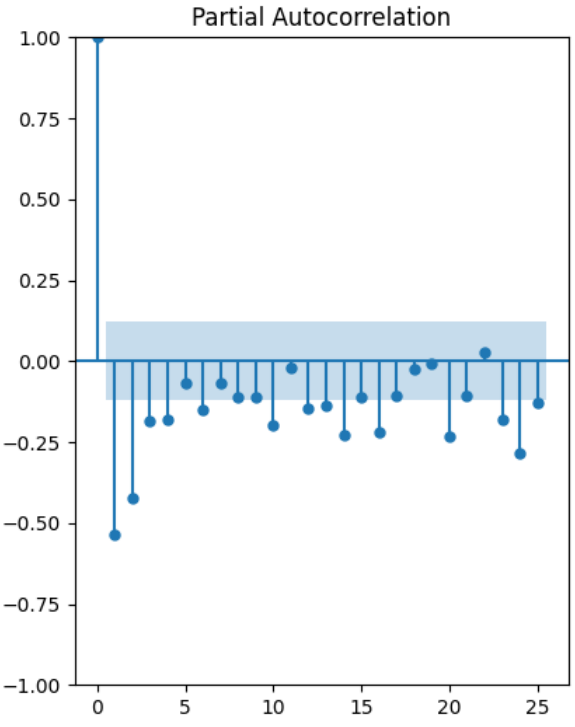












- Finding q: For q we used the same method as p except using just the autocorrelation function plot. Looking at how many lags cross our threshold, we

can see that after the 0th lag would hold enough weight (because of its contribution) so we can set q to 0.

- autoArima: Lastly, to verify our above results from the visualization as well as to see what optimal parameters autoArima gives. As seen below, autoArima also gives us the same model parameter results

```
model_autoARIMA = auto_arima(train_data, start_p=0, start_q=0,
                      test='adf',       # use adftest to find optimal 'd'
                      max_p=3, max_q=3, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)
print(model_autoARIMA.summary())
model_autoARIMA.plot_diagnostics(figsize=(15,8))
plt.show()
```

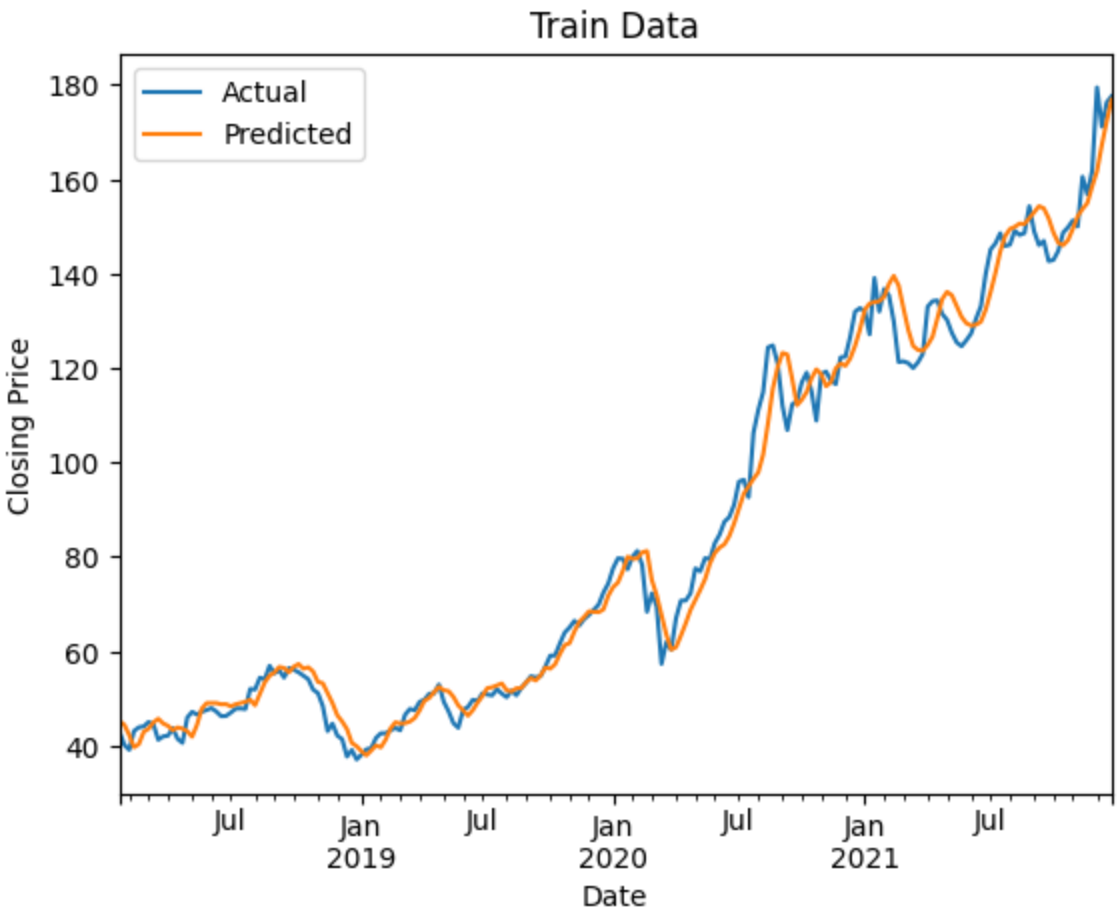
```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-713.492, Time=0.12 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-711.580, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-711.569, Time=0.19 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-709.932, Time=0.05 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-709.574, Time=0.13 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.557 seconds
```

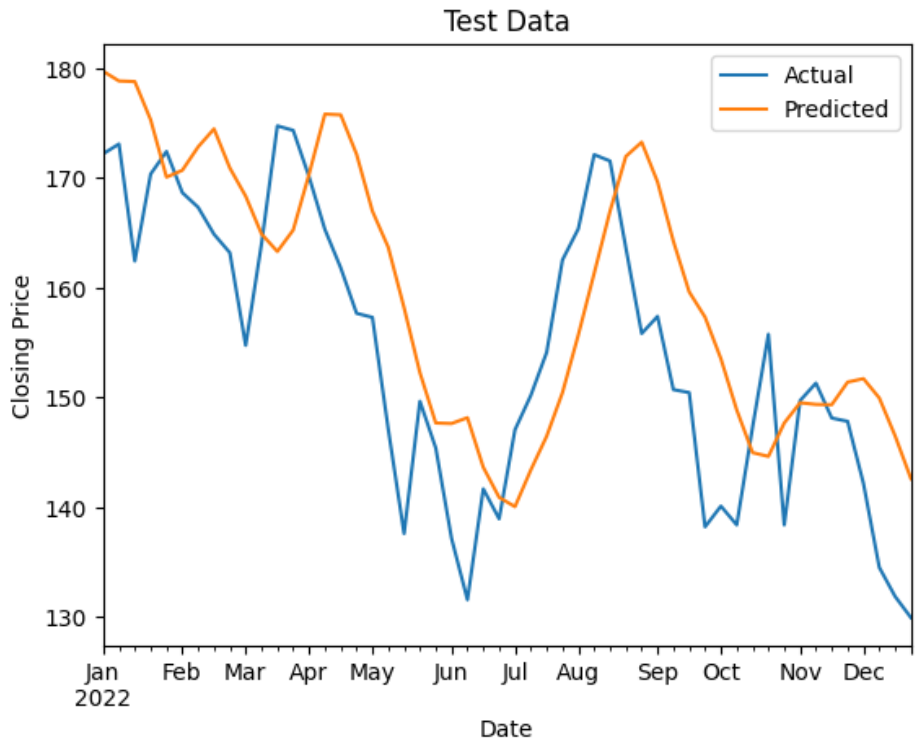
ARIMA: Lastly, we finally implemented the ARIMA model using the parameters we found above (0,1, 0) and fitted it, leading us to use the forecast function to create our y predicted.

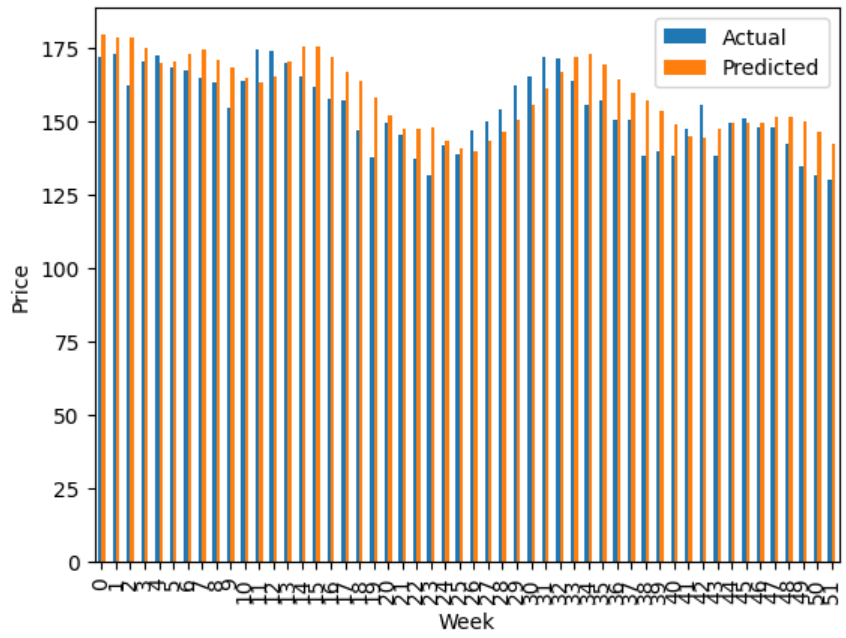
Results and Discussion:

Linear Regression:

As we expected, our linear regression model/approach worked relatively well with our training data but not as consistently with our testing data. This is due to the nature of stocks themselves, their unpredictability gives linear regression a difficult time in actually performing well with testing data. These results are pictured below, with the training data vs our predicted data, and our testing data vs our predicted data.


Train Data
Actual
Predicted
Closing Price
180
160
140
120
100
80
60
40
Jul
Jan
2019
Jul
Jan
2020
Jul
Jan
2021
Jul
Date





Random Forests

Long Short-Term Memory (LSTM)

LSTM works really well. It correctly follows the movement of the stock and predicts the price with less error as compared to linear regression.

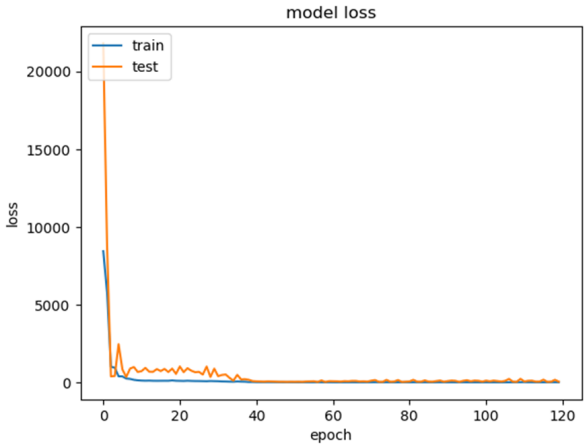
It works well because it has feedback connections and is particularly effective in capturing long-term dependencies and patterns in sequential data. Stock prediction involves analyzing time-series data, which is a type of sequential data.

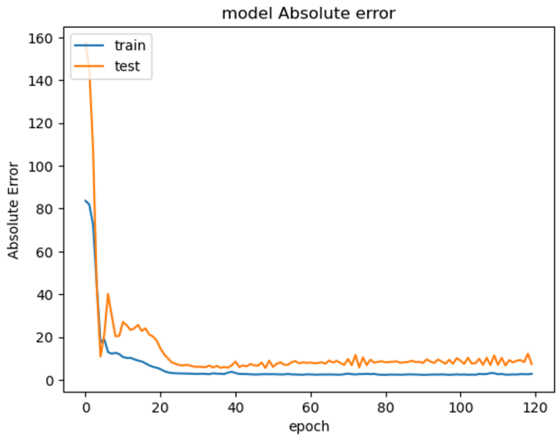
In addition, LSTM can learn to selectively forget or remember certain information based on its relevance to the current prediction task. This ability is especially important in stock prediction, where there may be many factors that contribute to the stock's price, but only a few are relevant at a given time.

These results are pictured below.

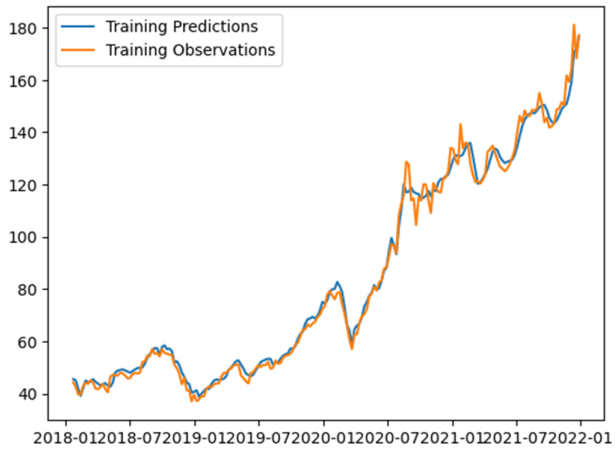
For Epochs = 120, LR = 0.005

Loss plot and MAE value

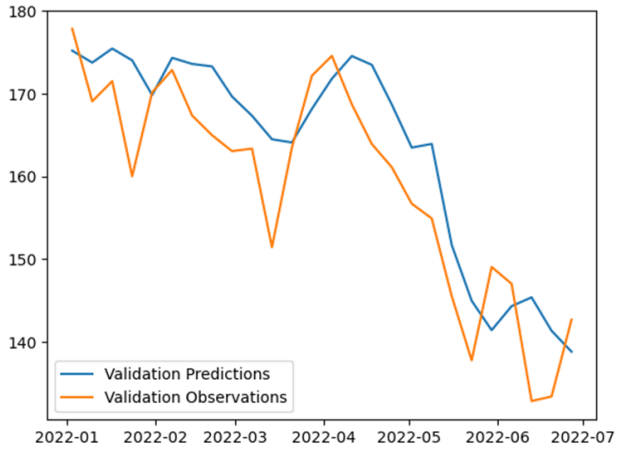




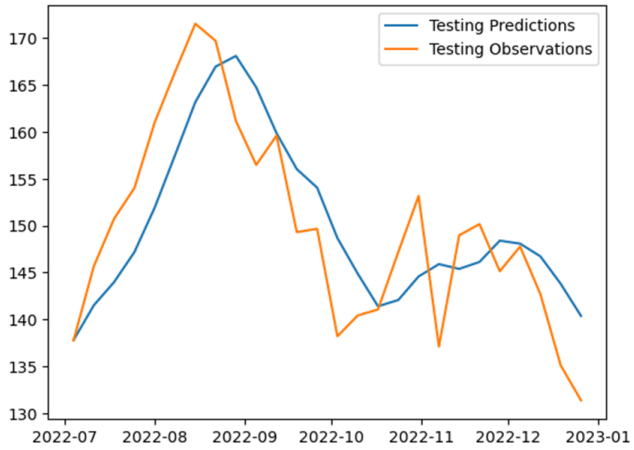
Training Data



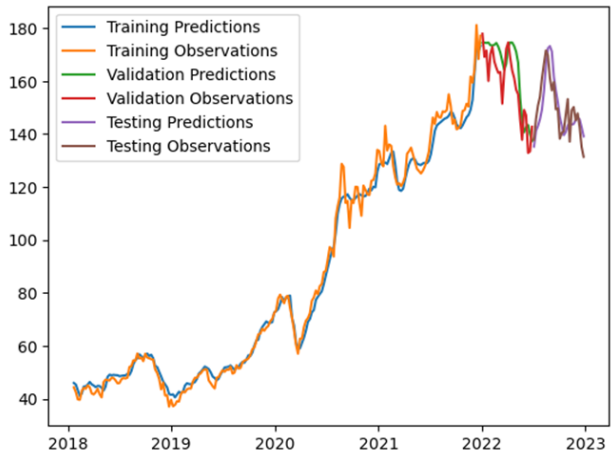
Validation Data



Testing Data



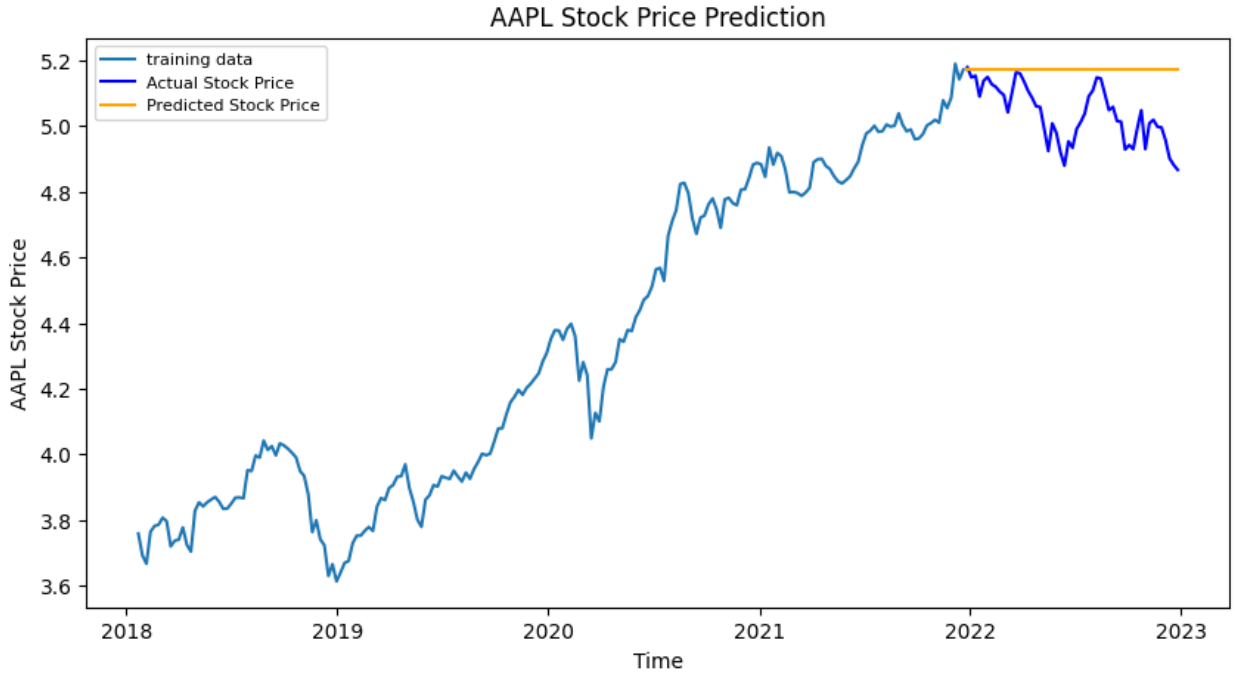
All data together -



```
Test - Mean Squared Error =  40.297798856116756
Test - Mean Absolute Error =  5.539185157189002
```

Autoregressive Integrated Moving Average (ARIMA)

Using ARIMA, our error metrics are shown below as well as the predicted visualizations. Clearly, modeling something like stocks with time series is not an easy task and it's clear our model needs more tuning. We are unsure as to why the error is suspiciously low, but we can say we have a very basic implementation of modeling ARIMA and what went behind it. Anyone looking at this visualization, especially newcomers may not feel the most encouraged or confident to invest in stocks. However, these results in light of everything still provide some sort of basis for what can happen next in terms of the trend line.



```
# report performance
mse = mean_squared_error(test_data, ypredicted_series)
print('MSE: '+str(mse))
mae = mean_absolute_error(test_data, ypredicted_series)
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(test_data, ypredicted_series))
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(ypredicted_series - test_data)/np.abs(test_data))
print('MAPE: '+str(mape))
```

```
MSE: 0.025542336785024124
MAE: 0.13504280824107348
RMSE: 0.15981970086639546
MAPE: 0.027106830147315294
```

Conclusion

Thus, we successfully completed the goal that we set out to achieve. We were able to make a model that predicted stock prices with some error.
Based on our observation, LSTM neural network works the best, it follows the price trends and predicts the price with lowest error.
However, it is important to note that our model has limitations. It only uses historical stock price data to predict future prices. It doesn't take into account other factors like weather, natural calamities and politics that affect the stock prices as well. Future work could include mathematically modeling these factors and using these along with historical prices to train the model.

Updated Contribution Table

Isaac	Worked with David to optimize linear regression and relevant parts of the report and video slides
David	Worked with Isaac to optimize linear regression and helped Hari verify ARIMA results and relevant parts of the report and video slides

Manit	Worked primarily on LSTM and relevant parts of the report and video slides
Hari	Worked primarily on ARIMA and relevant parts of the report and video slides
Luis	Worked primarily on report and slides