STOCK MARKET PREDICTION

COMPARISON OF MULTIPLE STRATEGIES FOR MAXIMIZED PERFORMANCE

THE CHALLENGE

- Due to the high number of rapidly changing forecast factors, there are no consistent patterns in stock market data that allow an analyst or investor to model stock prices over time with near-perfect accuracy
- Several different machine learning techniques and algorithms have been used.
- This project presents four different time series machine learning models created to predict future stock market price movements. My initial forecast was predicted using an LSTM model that was then compared to a liner regression model, decision tree model, and finally an XGBoost model. As a final step, the one day moving average was also determined using a standard averaging technique and the exponential moving average. The purpose of this was to compare the long-range prediction accuracy to single step accuracy.

THE MODELS

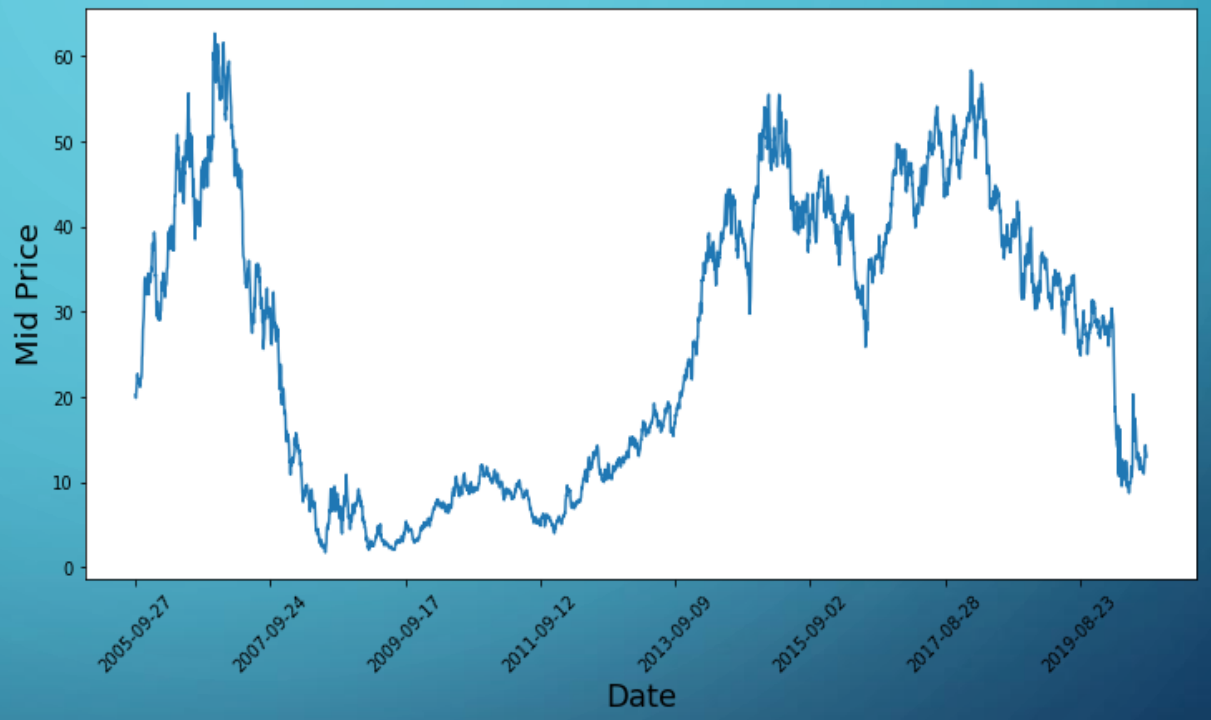
- LSTM Neural Network
- Liner Regression
- Decision Tree
- XGBoost
- Additional comparisons: Simple Moving Average, Exponential Moving Average

DATA ACQUISITION

Data imported from Alpha Vantage

Initial Summary:

```
Int64Index: 3748 entries, 0 to 3747
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    3748 non-null   object
 1   Low     3748 non-null   object
 2   High    3748 non-null   object
 3   Close   3748 non-null   object
 4   Open    3748 non-null   object
```



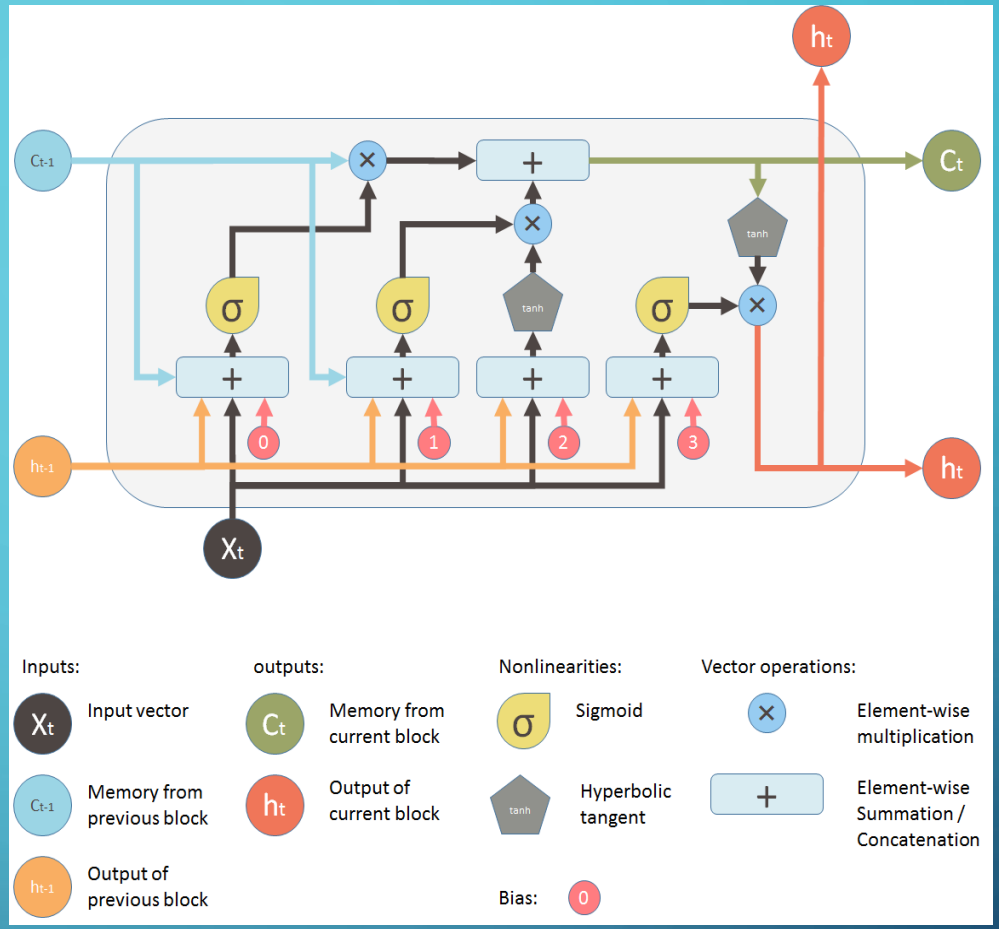
DATA PRE-PROCESSING

- The data was split into a training and testing set using 3230 as the dividing index
- normalized between 0 and 1
- Training data was normalized in batches of 800 data points to prevent the large values from overwhelming the smaller values
- Testing data was normalized in a single batch
- Data was smoothed using the Exponential Moving Average

LSTM NEURAL NETWORK

REVIEW OF MODEL ARCHITECTURE

- **The input gate:** The input gate adds information to the cell state
- **The forget gate:** The forget gate removes the information that is no longer required by the model
- **The output gate:** The output gate selects the information to be shown as output to the next step in the sequence



$$
\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}^{(xi)}\mathbf{x}_t + \mathbf{W}^{(hi)}\mathbf{h}_{t-1} + \mathbf{W}^{(ci)}\mathbf{c}_{t-1} + \mathbf{b}^{(i)}\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{W}^{(xf)}\mathbf{x}_t + \mathbf{W}^{(hf)}\mathbf{h}_{t-1} + \mathbf{W}^{(cf)}\mathbf{c}_{t-1} + \mathbf{b}^{(f)}\right) \\
\mathbf{c}_t &= \mathbf{f}_t \bullet \mathbf{c}_{t-1} + \mathbf{i}_t \bullet \tanh\left(\mathbf{W}^{(xc)}\mathbf{x}_t + \mathbf{W}^{(hc)}\mathbf{h}_{t-1} + \mathbf{b}^{(c)}\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}^{(xo)}\mathbf{x}_t + \mathbf{W}^{(ho)}\mathbf{h}_{t-1} + \mathbf{W}^{(co)}\mathbf{c}_t + \mathbf{b}^{(o)}\right) \\
\mathbf{h}_t &= \mathbf{o}_t \bullet \tanh\left(\mathbf{c}_t\right),
\end{aligned}
$$

https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714

DATA GENERATION

- Implemented a data generator function that split the data into N/b input batches obtained sequentially
- Each batch in the batch set had a set batch size (b)
- Each batch of input data had a corresponding output batch of data of the same size
- The corresponding output batch to each input batch was generated by randomly selecting one value from the original data set that was between x+b and x+b+5 values ahead

Input Data:
[x0,x10,x20,x30],[x1,x11,x21,x31],[x2,x12,x22,x32]

Corresponding Output Data:
[x1,x11,x21,x31],[x2,x12,x22,x32],[x3,x13,x23,x33]

LSTM SETUP

Hyperparameters:

- The dimensionality of the data: 1
- The number of continuous time steps considered for each optimization step: 50
- The number of data points in each batch (or time step): 500
- The number of LSTM layers: 3
- The number of nodes in each of the three layers of the LSTM cell: [200, 200, 150]
- The dropout percent: 20%

Loss and Optimization:

- Loss Metric: Mean Squared Error
- Optimization: Adam

LSTM TRAINING AND VALIDATION

30 Epochs Using the Following Procedure for each Epoch

1. For full sequence length of training data
   a. Unroll a set of batches
   b. Train the neural network with the unrolled batches

2. Calculate the average training loss

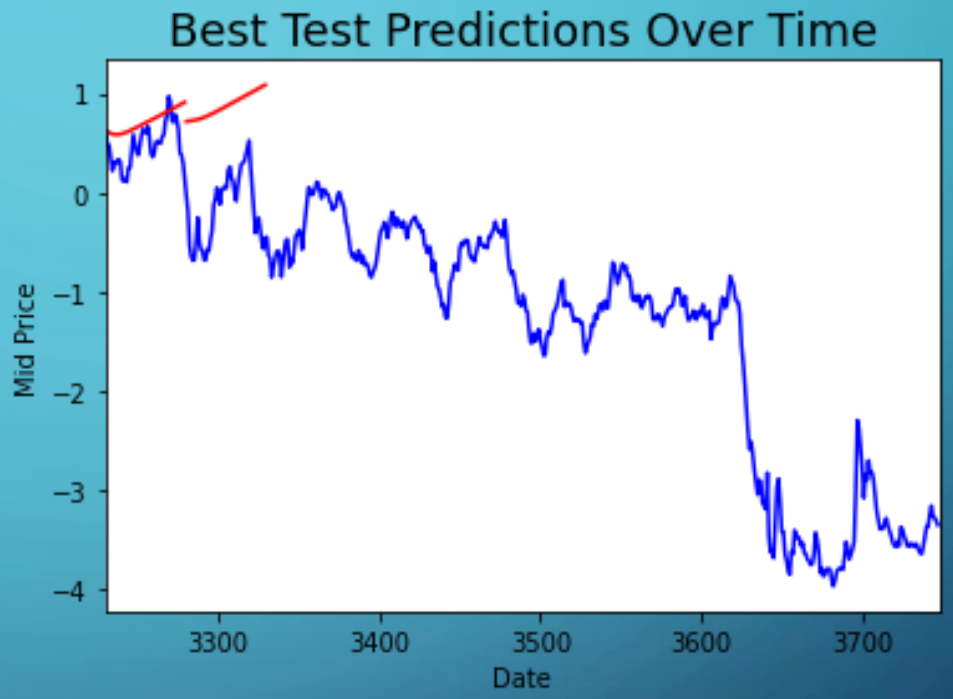
3. For each starting point in the test set
   a. Update the LSTM state by iterating through the previous data points found before the test point
   b. Make predictions for 50 steps continuously, using the previous prediction as the current input
   c. Calculate the MSE loss between the 50 points predicted and the true stock prices at those time stamps

INITIAL RESULTS

Best Epoch = 4

Loss = 0.394196

Mean Squared Error = 0.14829



LINEAR REGRESSION

BASIC REGRESSION MODEL

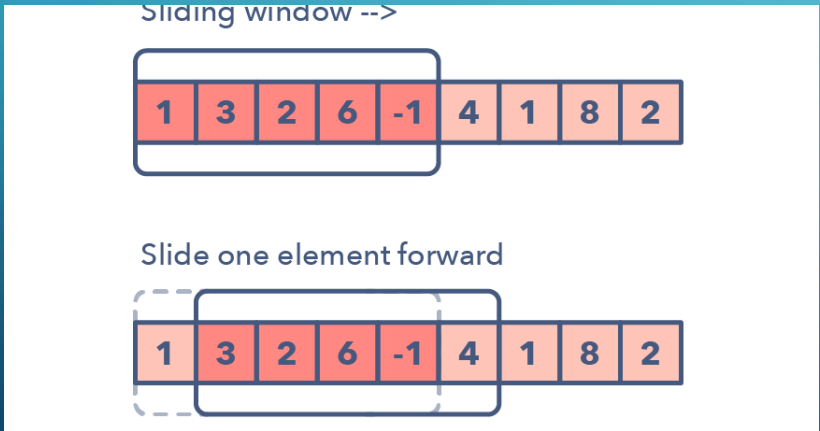
The equation for linear regression can be written as:

$$Y = w0x0 + w1x1+w2x2+\ldots\ldots\ldots+wnxn+b$$

Where, x_1, x_2,….x_n represent the independent variables while the coefficients w_1, w_2, …. w_n represent the weights.

DATA PREPROCESSING

The Sliding Window Method



In time series prediction, the time series are typically expanded into three or higher-dimensional space by restructuring the series into a data frame to look like a supervised learning problem. This can be done by using previous time steps as input variables and assigning future time step as the output variable.

** The data processing in this step was reproduced for the Linear Regression, Decision Tree Regression, and XGBoost Models

REGRESSION MODEL SET UP

Training Split: 80%

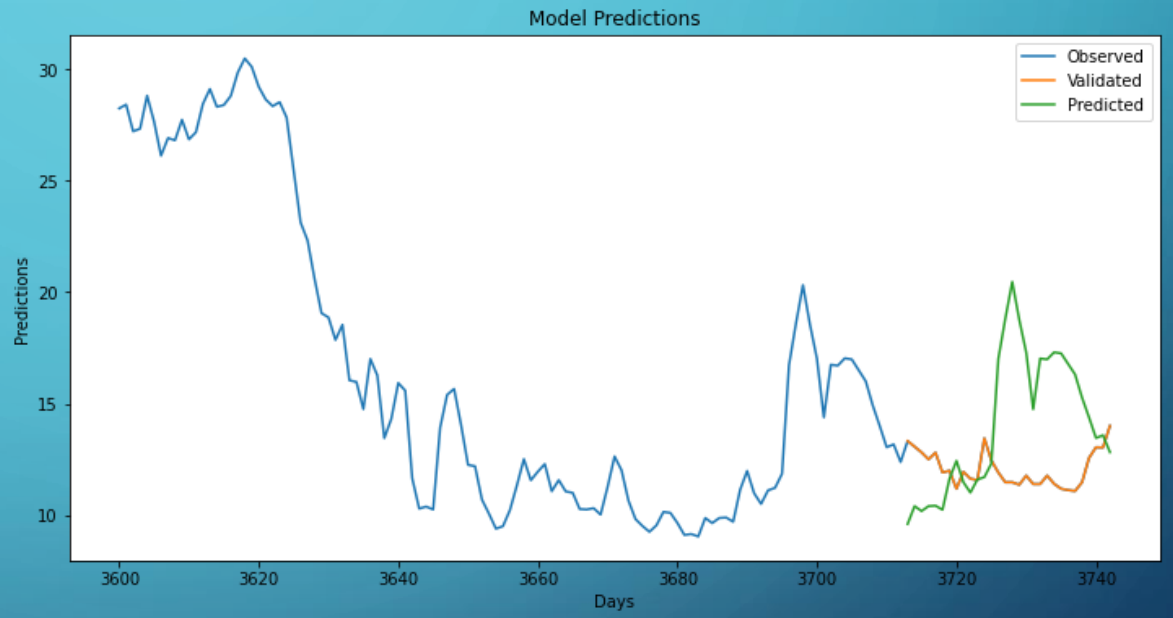
Validation Split: 20%

Loss Function: Mean Squared Error

LINEAR REGRESSION RESULTS

Validation Mean Squared Error: 21.2527

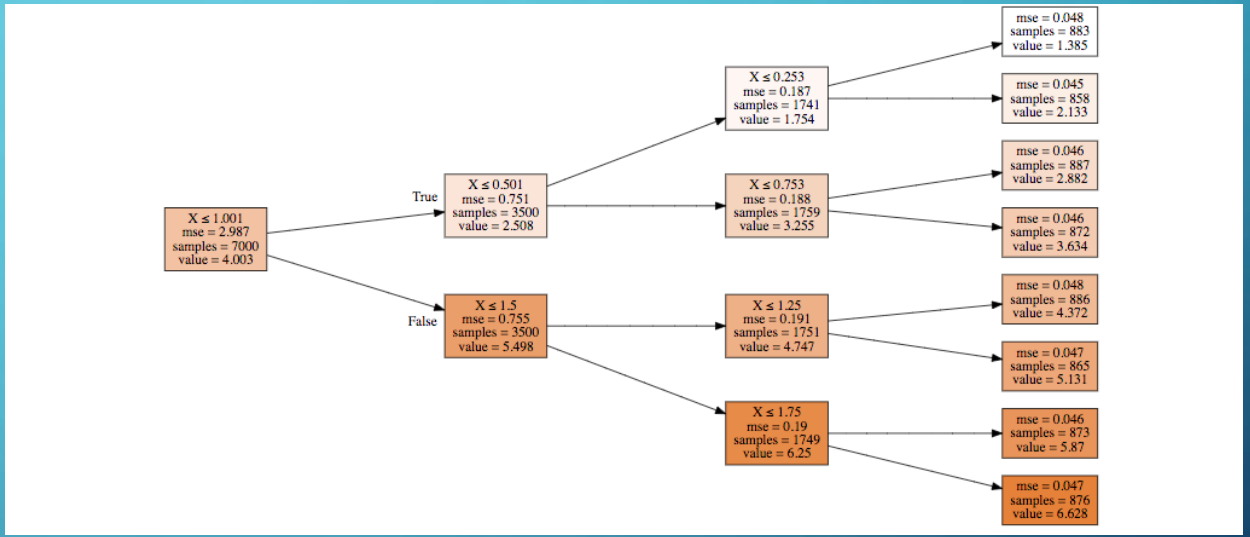
Prediction Mean Squared Error: 0.1673



DECISION TREE REGRESSION

BASIC DECISION TREE REGRESSION MODEL

1. Initially, all values in the training set are grouped into one large node.
2. The algorithm selects data splits that minimize the sum of the squared deviations from the mean in the two separate partitions
3. The same splitting rule is applied to each of the new branching execution
4. This process continues until each node reaches a user-specified minimum node size and becomes a terminal node



DECISION TREE MODEL SETUP

Training Split: 80%

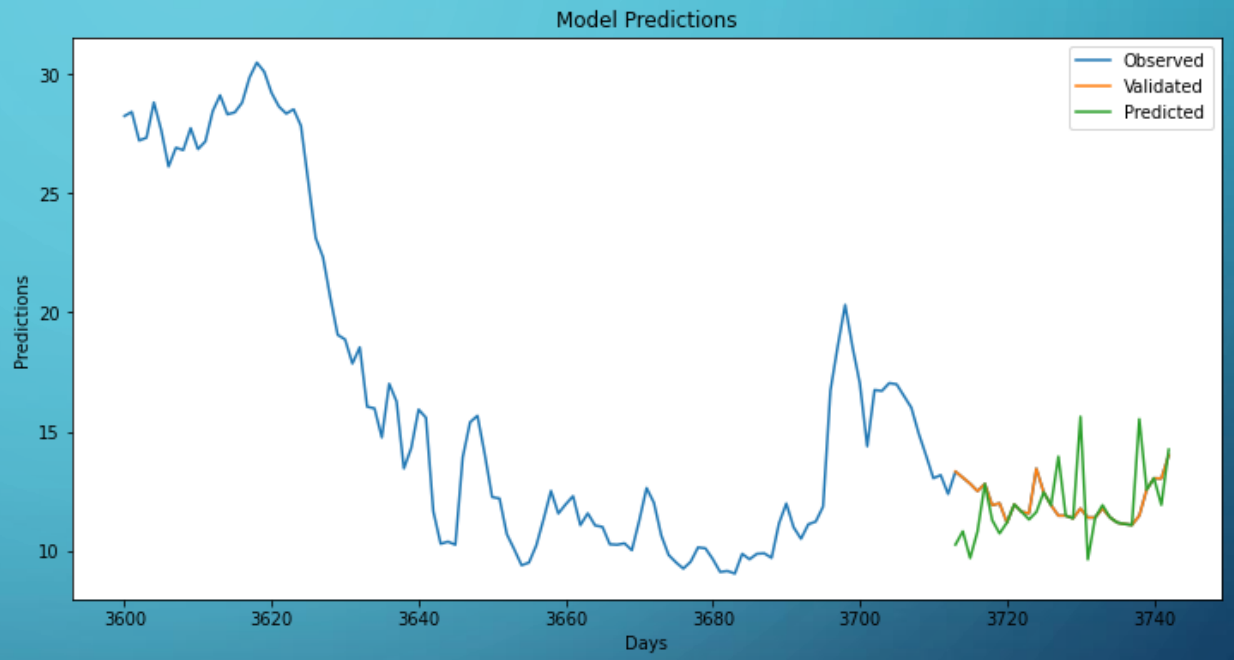
Validation Split: 20%

Loss Function: Mean Squared Error

DECISION TREE REGRESSION RESULTS

Validation Mean Squared Error: 32.099

Prediction Mean Squared Error: 13.140



XG BOOST REGRESSION

XG BOOST MODEL DESIGN OVERVIEW

Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction

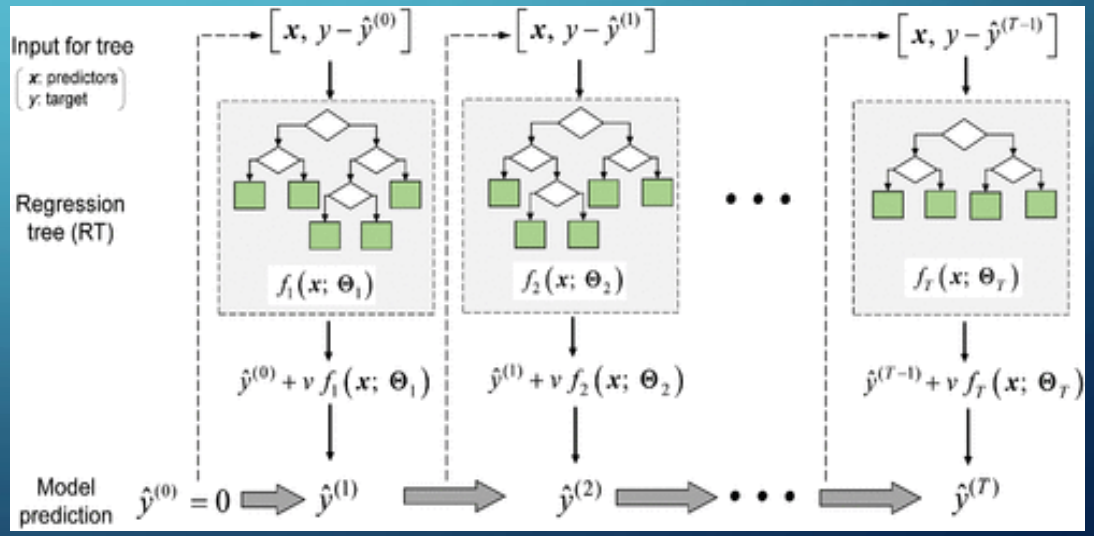
The objective of the XGBoost model is given as:

$$Obj = L + \Omega$$

Where,

L is the loss function which controls the predictive power

Ω is regularization component which controls generalizability and overfitting



3 XG BOOST MODELS CREATED

- Basic model with default hyperparameters
- 4 Fold Cross Validated
- 5 Fold Cross Validated with Randomized Hyperparameter Search

BASIC XG BOOST MODEL

- Default Hyperparameters
- 5 Boosting Rounds
- Results:
- Mean Squared Error: 33.910

XG BOOST WITH 4 FOLD CROSS VALIDATION

- Max tree depth: 4
- Number of validation folds: 4
- Number of boosting rounds: 5

Results:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	22.043914	0.163485	22.047628	0.547003
1	15.761451	0.111533	15.784873	0.417605
2	11.464096	0.074700	11.508514	0.327262
3	8.591789	0.047476	8.661346	0.269104
4	**6.740023**	0.033503	**6.841255**	0.224286

XG BOOST MODEL WITH 4 FOLD CROSS VALIDATION AND RANDOMIZED HYPERPARAMETER SEARCH

Searched HyperParameter Space

Samples per tree: [0.2, 0.3, 0.4, 0.5, 0.6, 0.7]

Learning Rate: [0.04, 0.06, 0.08, 0.1, …, 0.98, 1.0]

Maximum Tree Depth: [3,4,5,6,7,8,9,10]

Number of Estimators: [50, 100, 150, 200]

Best Model Found

Samples per tree: 0.2

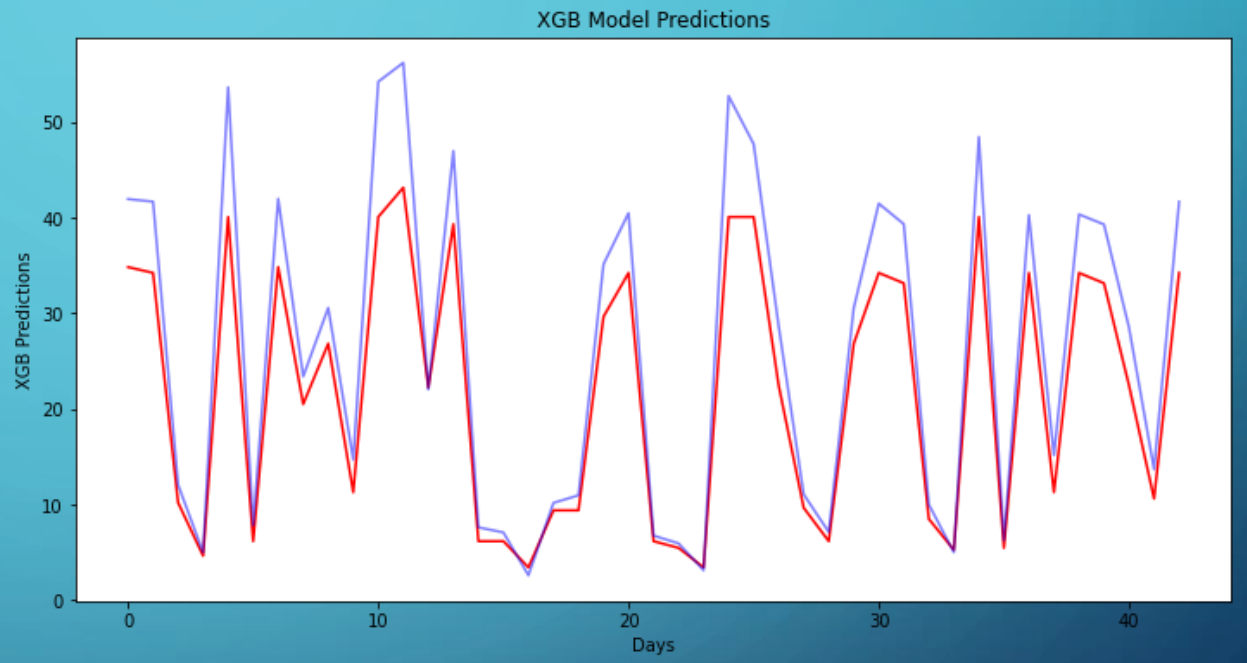
Learning Rate: 0.32

Maximum Tree Depth: 3

Number of Estimators: 100

RESULTS WITH BEST HYPERPARAMETER COMBINATION

MSE: 33.706



ONE DAY AHEAD PREDICTION MODELING

SHORT TERM VS LONG TERM PREDICTION

Despite the challenges of predicting stock market price movement for time periods far into the future, predicting day to day stock market movement (one step ahead), has been much more successful. Out of curious comparison, I created 2 models to predict single day market movements and compared the performance to the future prediction models

2 MOVING AVERAGE MODELS

Simple Moving Average

Exponential Moving Average

The major difference between the exponential moving average (EMA) and the simple moving average is how the data values are weighted. The EMA assigns larger weights to recent prices, while the SMA assigns equal weights to all values. Since the EMA places a higher weighting on recent data than on older data, it is more reactive to the latest price changes than SMAs.

SIMPLE MOVING AVERAGE

The simple moving average was calculated with a window size of 50 days and resulted in an MSE error of 0.0205

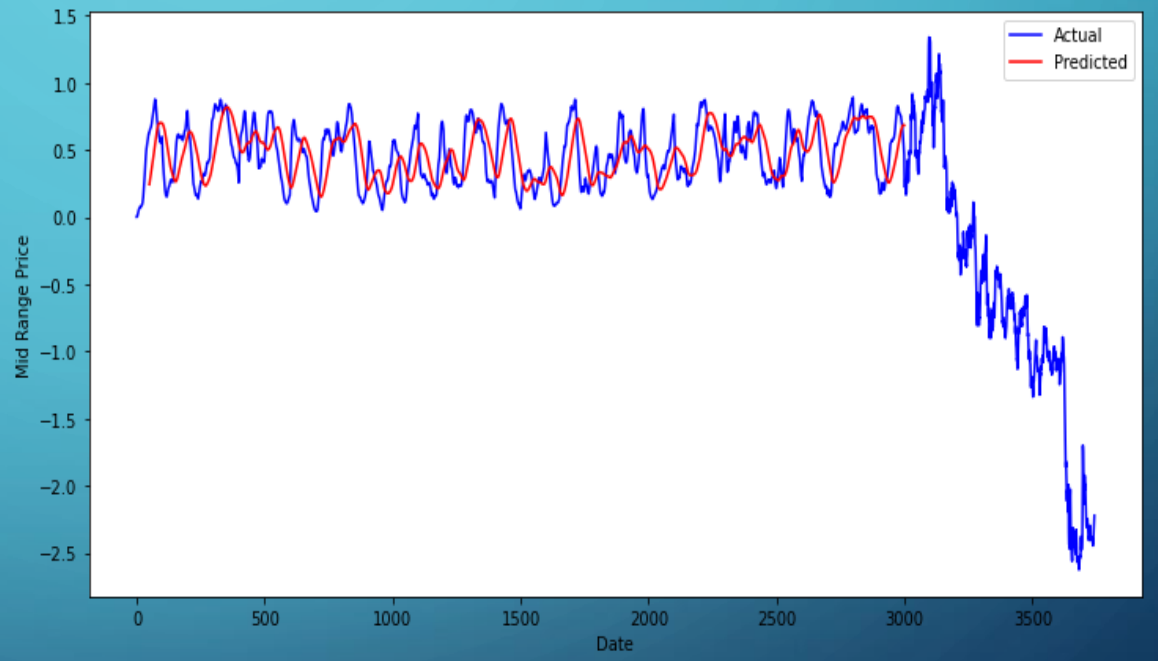
The simple moving average (SMA) is calculated by adding recent prices and then dividing by the number of time periods in the calculation average.

$SMA = (A_1 + A_2 + ... + A_n)/n$

where:

A_n = the price of an asset at period n

N = the total number of periods

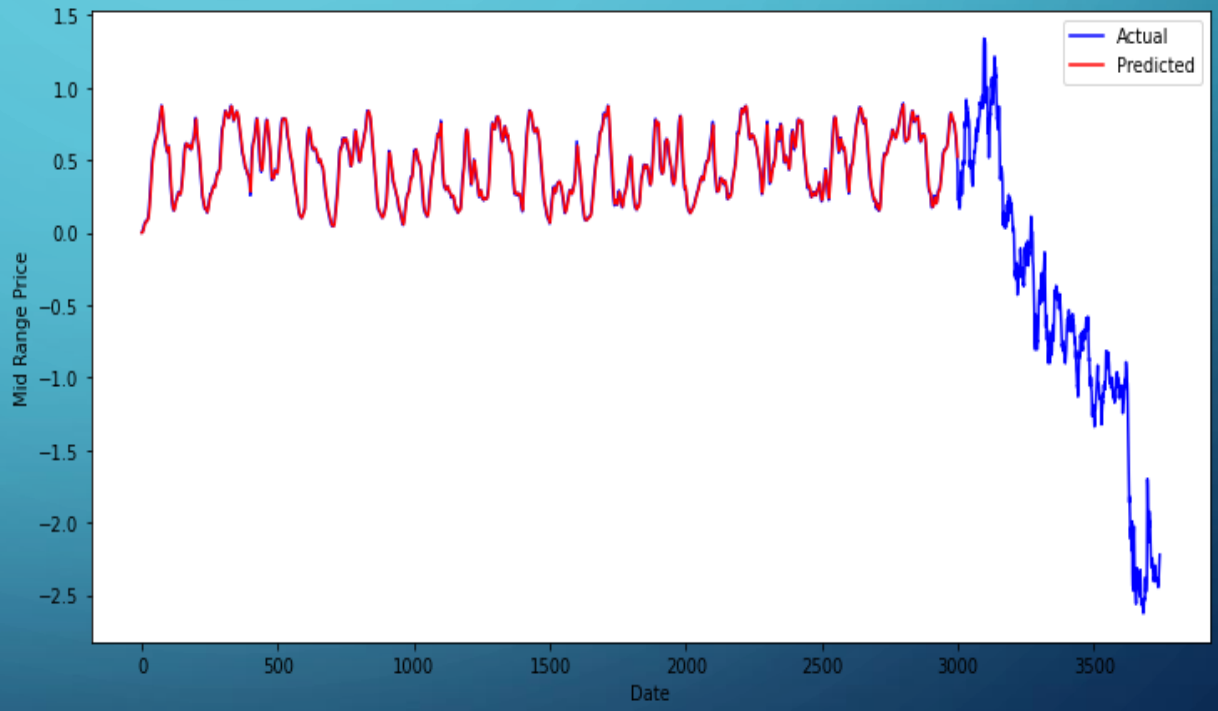


EXPONENTIAL MOVING AVERAGE

The exponential moving average (EMA) is a moving average that places a greater weight on the most recent data points. An exponentially weighted moving average reacts more significantly to recent price changes than a simple moving average

$$EMAToday = (ValueToday * (Smoothing/1+Days)) + EMAYesterday * (1-(Smoothing/1+Days))$$

The exponential moving average was also calculated with a window size of 50 days and a decay rate of 0.5. The resulting MSE error was 0.00051



INITIAL CONCLUSIONS AND NEXT STEPS

The progress so far demonstrates the models with the best predictive power to be the Linear Regression model and the LSTM model. The success of tree-based models has been poor, however this may be a matter of more thoughtful hyperparameter tuning. The next steps in this project will be improvement of each model's accuracy through hyperparameter tuning. I will also test the generalizability of all models by testing them on multiple tickers to expose them to new data.

SIMPLE MODEL IMPROVEMENT THROUGH HYPERPARAMETER TUNING

RIDGE REGRESSION MODEL

By adding a bias term to the regression coefficient estimates, ridge regression reduces the standard errors. The ridge regression algorithm works by imposing a penalty on the size of the coefficients according to the following equation:

$$\min_{w} ||Xw - y||_2^2 + \alpha ||w||_2^2$$

$\alpha \geq 0$ controls the amount of coefficient shrinkage. The larger the value of α, the greater the amount of shrinkage

RIDGE REGRESSION MODEL SET UP

Training Split: 80%

Validation Split: 20%

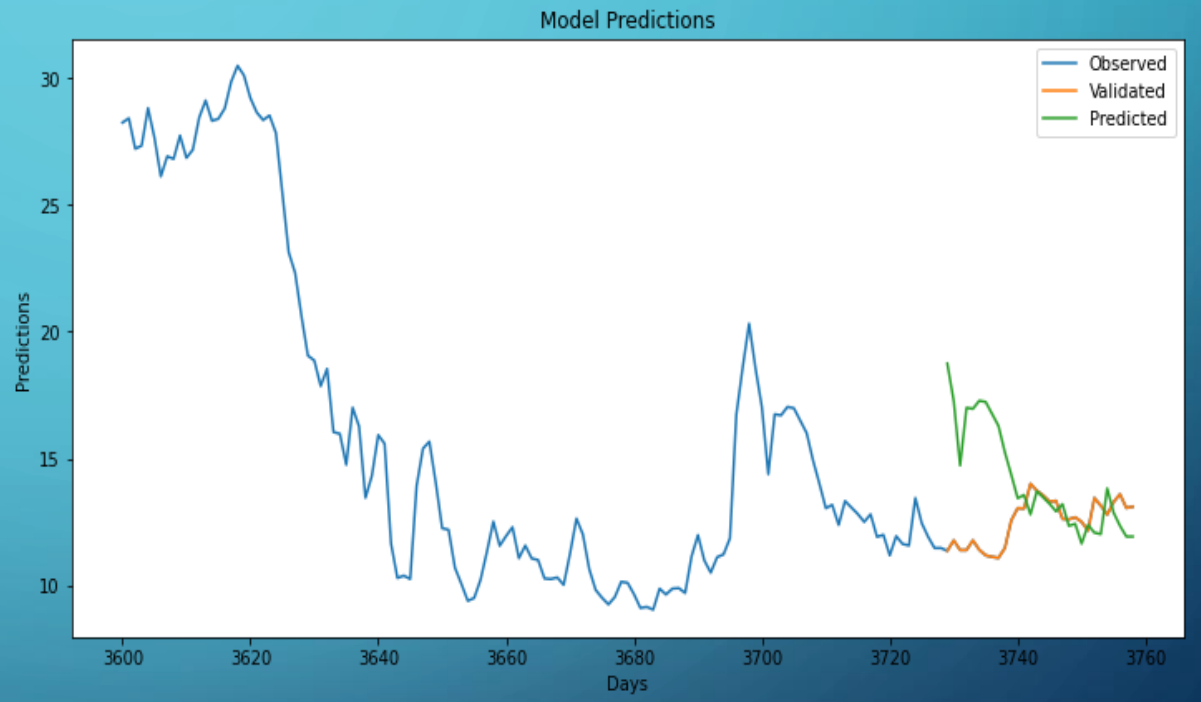
Loss Function: Mean Squared Error

Alpha: 0.5

RIDGE REGRESSION RESULTS

Validation Mean Squared Error: 20.665

Prediction Mean Squared Error: 0.140



LASSO REGRESSION MODEL

Like ridge regression, the lasso regression algorithm adds a penalty term to the loss function. Unlike ridge regression, the penalty term for lasso regression is the absolute sum of the coefficients multiplied by alpha. The objective function to minimize is:

$$\min_{w} \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha ||w||_1$$

where α is a constant and $||w||_1$ is the $\ell 1$-norm of the coefficient vector.

LASSO REGRESSION MODEL SET UP

Training Split: 80%

Validation Split: 20%

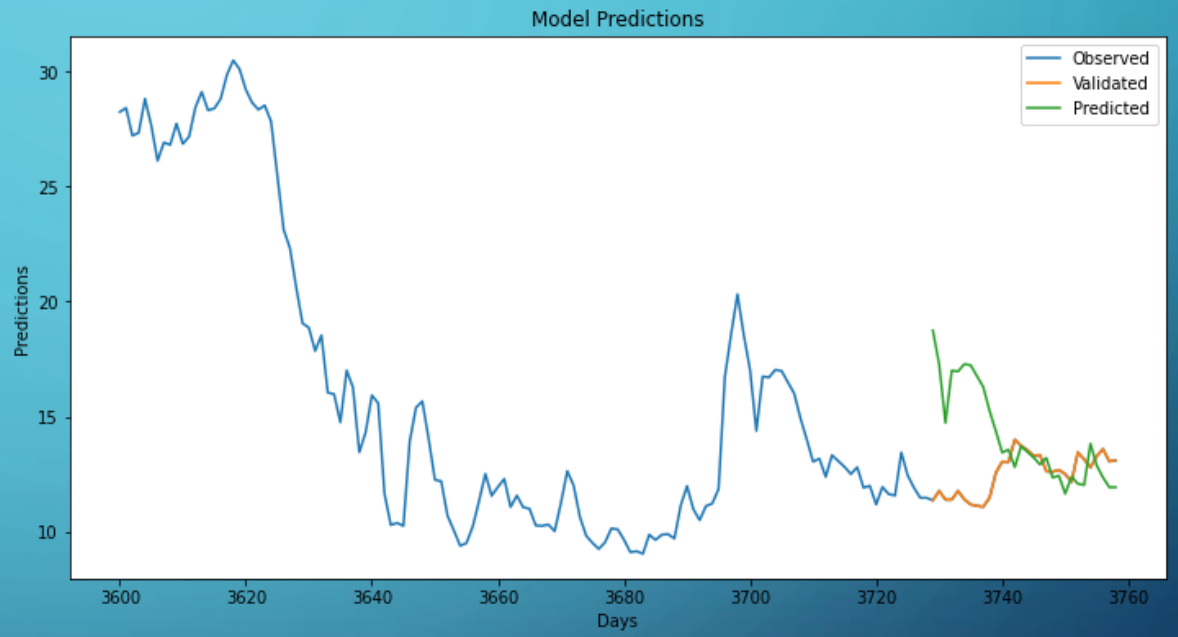
Loss Function: Mean Squared Error

Alpha: 0.1

LASSO REGRESSION RESULTS

Validation Mean Squared Error: 20.668

Prediction Mean Squared Error: 0.144



DECISION TREE REGRESSION WITH RANDOMIZED HYPERPARAMETER SEARCH - LOGIC

- Reduce Overfitting
- Efficiently Attempt Several Hyperparameter Combinations Over a Large Hyperparameter Space
- Use Optimized Hyperparameter Combination to Learn More About the Data

DECISION TREE REGRESSION WITH RANDOMIZED HYPERPARAMETER SEARCH – MODEL SET UP

Training Split: 80%

Validation Split: 20%

Loss Function: Mean Squared Error

10 Hyperparameter Combinations

5 Fold Cross Validation Each Hyperparameter Combination

DECISION TREE REGRESSION WITH RANDOMIZED HYPERPARAMETER SEARCH

Searched HyperParameter Space

Maximum Tree Depth: [2,3,4,5,6,7,8,9,10]
Maximum Features for Best Split: [auto , sqrt]
Quality of Split Criterion : [mse, mae]
Node Split Strategy: [best, random]
Min Samples per Leaf: [5,6,7,…..,19,20]

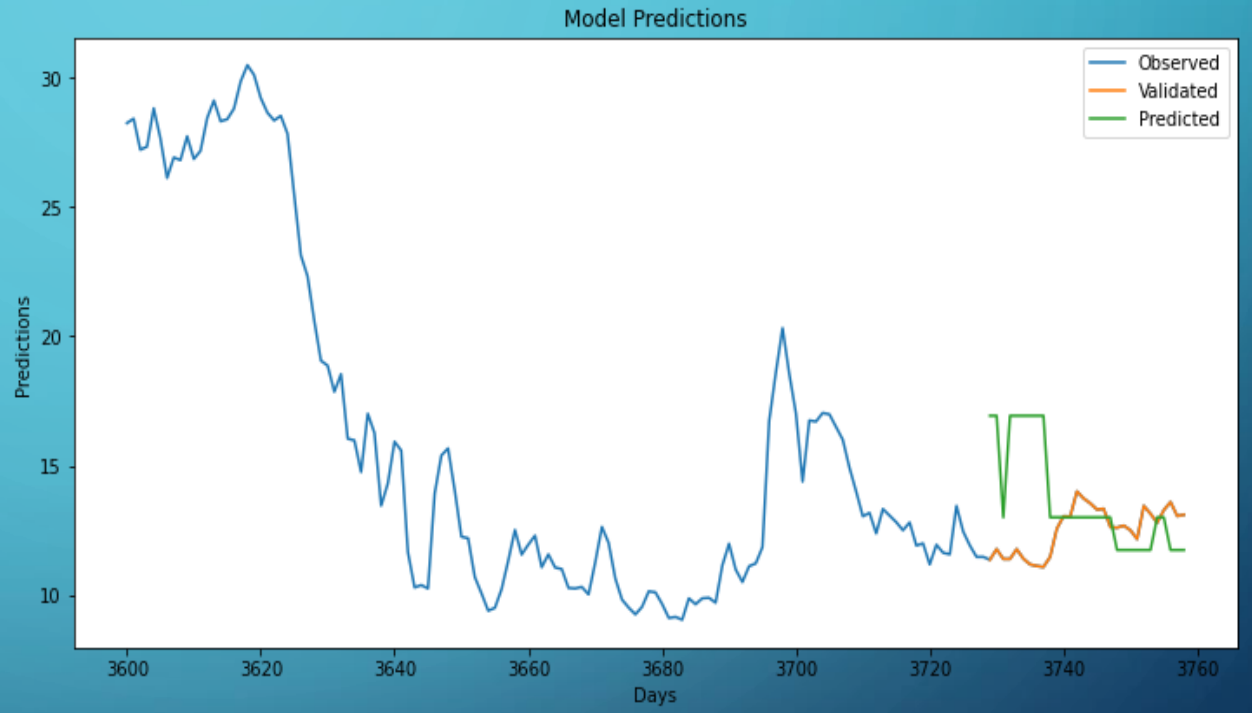
Best Model Found

Maximum Tree Depth: 6
Maximum Features for Best Split: sqrt
Quality of Split Criterion : mean squared error
Node Split Strategy: best
Min Samples per Leaf: 5

RESULTS WITH BEST MODEL

Validation Mean Squared Error: 2.915

Prediction Mean Squared Error: 0.413



COMPILED RESULTS

Model	Mean Squared Error
One Day Ahead, Exponential Moving Average	0.0005
One Day Ahead, Simple Moving Average	0.021
Ordinary Least Squared Linear Regression	0.140
Ridge Regression	0.140
Lasso Regression	0.144
LSTM Neural Network	0.150
Decision Tree with Hyperparameter Search	0.413
Decision Tree	7.684
XGBoost with Hyperparameter Search	33.706
XGBoost, default	33.910
XGBoost with Cross Validation	46.803

INITIAL CONCLUSION

- Ordinary Least Squares Regression provides the best results for future predictions
- One day ahead predictions are the most reliable but are limited in use for some purposes that require longer range insights
- Models with a tendency towards overfitting are less effective for stock market prediction even after robust hyperparameter tuning

NEXT AND FINAL STEPS

Next I will look at how the entire series of models performs on alternate tickers to check model performance on entirely different stock market trends