Predicting Security Prices

STATS 202 Final

Kaggle team name: QQ Labs

Quentin Hsu
qhsu@stanford.edu

Julie Lederer
ledererj@stanford.edu

1 Summary

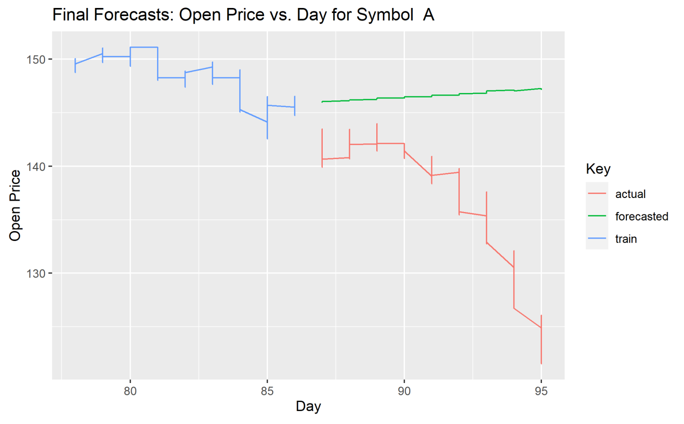
- **Final Submission** - We submitted forecasts from a **direct forecasting model** that uses lasso to generate 5 different models for 1 hour, 1 day, 3 days, 6 days, and 9 days into the future. This model achieved an average **error score of 48.8 (+2.7% vs baseline) for days 0-3 and 2966.6 (+15.6% vs baseline)** in the final predictions. The errors were higher primarily due to the external events that caused the stock market to tank in the final evaluation period.
- **Cross-Validation Performance** - During our own cross validation, our direct forecasting model **performed 29% (first 4 days) and 50% (last 5 days) better** than the naive baseline approach of carrying forward the last observation. The baseline approach actually performed surprisingly well compared to many of the more advanced models we tried.
  - We believe the simplicity of our final model (i.e., most of the forecasts look like straight lines) is due to the fact that the underlying stock prices themselves seem steady during the training period. More advanced models that try to fit sudden spikes tend to overfit the training data and result in higher error during cross validation whenever they happen to guess something wrong. A more simple model averages out errors from spikes and is better at informing us of trends. This aligns with the efficient market hypothesis, which explains why ETFs tend to outperform traders in the long run.
- **Features** - We included day and hour level features for the past 14 days as lagging features. We also derived several features from the given prices and times such as the range of prices, standard deviation of prices, and day of the week. The highest contributing features we used were the open price of the previous hour, the standard deviation in open prices during the previous hour, and the range in prices during the previous hour.
- **Classical Time Series Models** - We also tested classical time series models like moving average and ARIMA. These models' performance was similar to the baseline model's performance, and they had higher errors on the test set than the direct forecasting lasso model. Surprisingly, the moving average model actually performed the best out of the models we tried against the actual Kaggle forecast data.
- Our code can be found on this colab notebook.

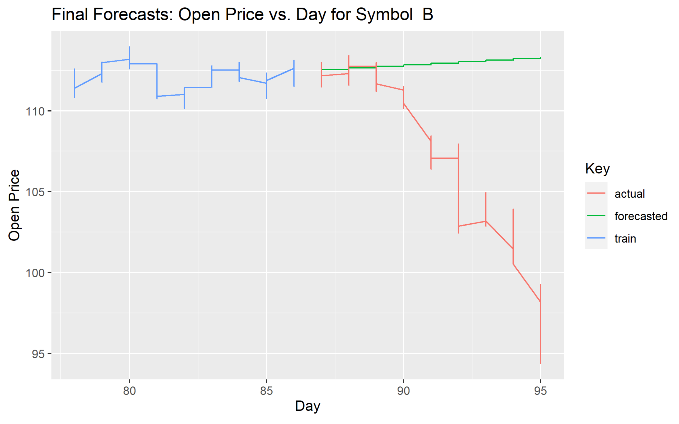
The table below shows errors calculated according to the formula on page 2 of the assignment handout. The left-most columns show the errors on the held-out portion of the training set (days 78-86) and the right-most columns show the errors on the actual data (days 87-95). We show results for four models: the lasso model that we used for our Kaggle submission and three simpler approaches: last observation carried forward (LOCF), moving average, and ARIMA.

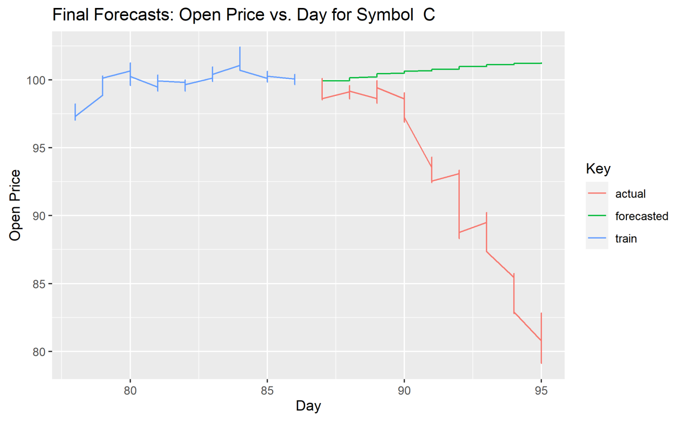
	Test set (days 78-86)		Actual (days 87-95)	
Model	First 4 days	Last 5 days	First 4 days	Last 5 days
Lasso	139	167	49	2967
LOCF	195	335	48	2566
Moving Average	175	312	39	2480
ARIMA	213	287	83	2953

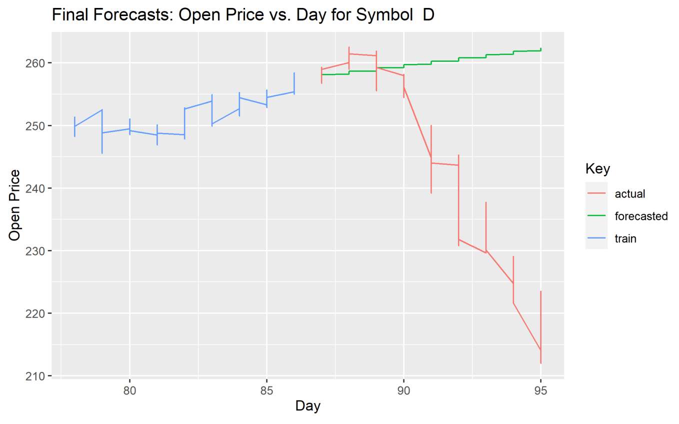
2 Forecasts

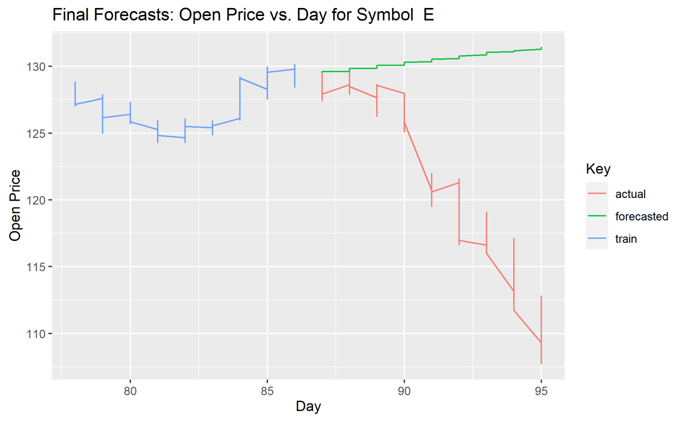
Plots of our submitted forecasts by stock ticker are shown below, along with the actual prices. The training data, which isn't shown in full in the plots below, covers October 11, 2019 (day 0 in our data set) through February 15, 2020 (day 86). The forecast period covers February 16, 2020 (day 87) through February 29, 2020 (day 95). Our forecasts do not anticipate the steep dropoff in prices that occurred between February 20 and April 7, 2020 [1].

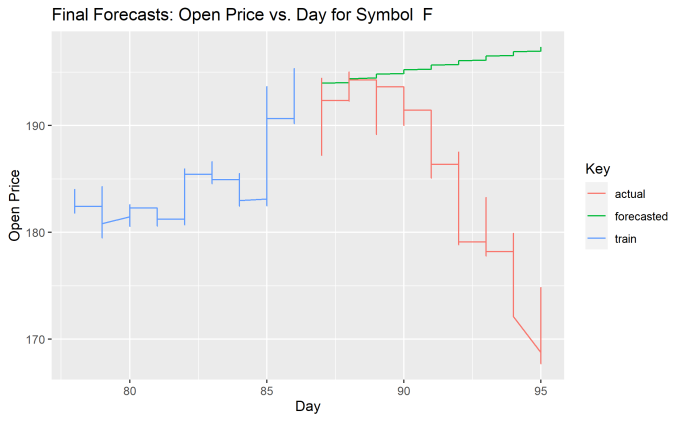


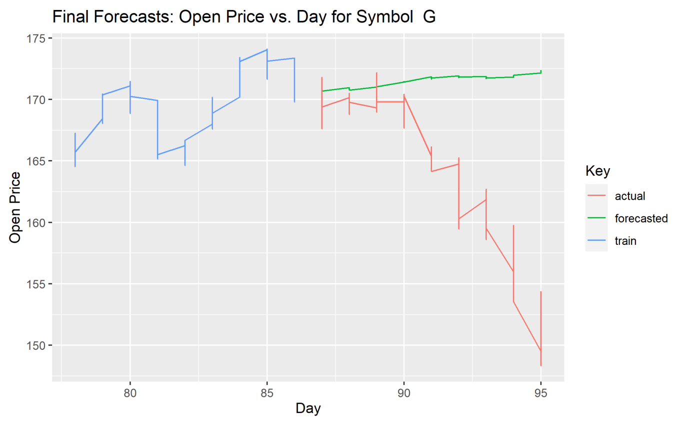


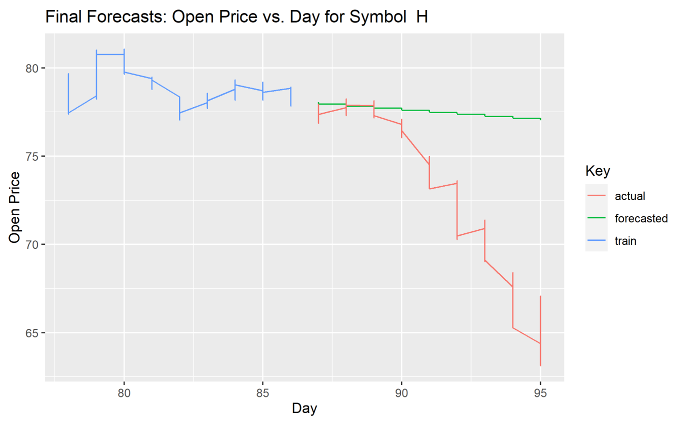


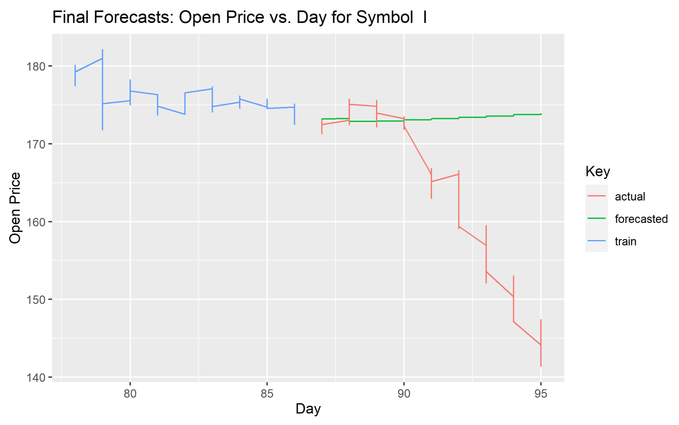


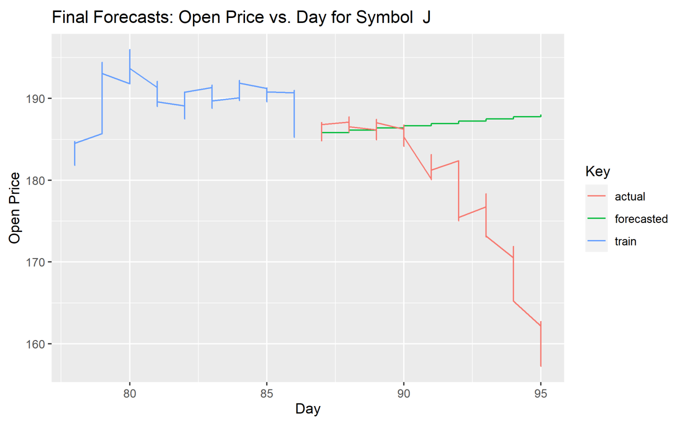












3 Approach

3.1 Goal

- Our goal for this project is to build a **forecasting model for open prices that achieves a low error rate** as defined by the formula on page 2 of the assignment handout.
- A broader goal that may be more relevant to the task is to select a model that helps us **make directionally correct investing decisions with the given data**. This would be a more relaxed goal of making predictions that have positive returns on investment within the next 9 days.

3.2 Forecasting Approach

We explored two main categories of forecasting:

1. **Direct Forecasting** (described in Section 7): This approach splits the data and uses lagged features to create a dataset that can be used to predict a specific horizon into the future. With this structure, we can use our regular prediction models like lasso and randomForest to make specific forecasts. This allows us to use more features and include domain knowledge about the stock market.
    - We explore Direct Forecasting with different models we learned from class to leverage additional features we came up with and likely create better forecasts.
    - In addition to selecting the best modeling method, we also split up our forecasts into different prediction horizons and combine the forecasts to yield an ensemble prediction.
2. **Classical Time Series** (described in Sections 6, 8 and 13): The classical time series approaches seek to leverage autocorrelation in the Open variable in order to make forecasts into the future. Additional features

and domain knowledge cannot be applied to the basic versions of these models. We used several classical time series methods - carry forward last observation, moving average, and ARIMA - to serve as baselines.

4 Data Processing

4.1 Partial Days - Custom Time Index

TLDR: We created a custom time index to help map stock trading hours into one continuous time series.

The stock market only trades during certain hours of the day. This creates problems with our time series data since we have gaps throughout the day and many packages and analyses assume that we have a continuous timeframe. We solve this by creating a custom time index mapping that lets us connect the data together for analyses and forecasting, then transform back to the original time scale for final submission.

We need to map a normal day back to the number of time periods we have for the stock data. From a high level, we take the number of 5-sec intervals in the stock day and set that as a cycle of one stock day. The first 5-sec at 6:00:00 on day 0 will be mapped to index 0; the second 5-sec segment at 6:00:05 on day 0 will be mapped to index 1; the last 5-sec segment of that day at 12:55:55 on day 0 will be mapped to the total number of 5-sec segments for a day, which is 5040.

- 1 min = 12 segments (60 seconds/5 sec)
- 1 hr = 720 segments (60 min/hour)
- 1 day = 5040 segments (7 hrs/day)
- days = (original_row_count)/5040
- hours = (original_row_count % 5040) / 720 + 6
- min = (original_row_count % 5040 % 720) / 12
- seconds = (original_row_count % 5040 % 720 % 12) * 5

This custom index allows us to easily use lag values as if we had a whole day. Note that because of this custom index, the absolute dates on the x axis for many of our graphs will not be correct, they will just be relative to all the data that we have. (We assume the 1st segment represents the 1st second of UNIXTIME and the 2nd segment represents the 2nd second). The final submission and comparisons against real data will be scaled back to the correct timestamps.

4.2 Missing Data

TLDR: We filled missing data with the last observation carried forward. For the missing data on day 0, we carried backward the first observation after the missing data. Missing data should not have a significant effect on our models.

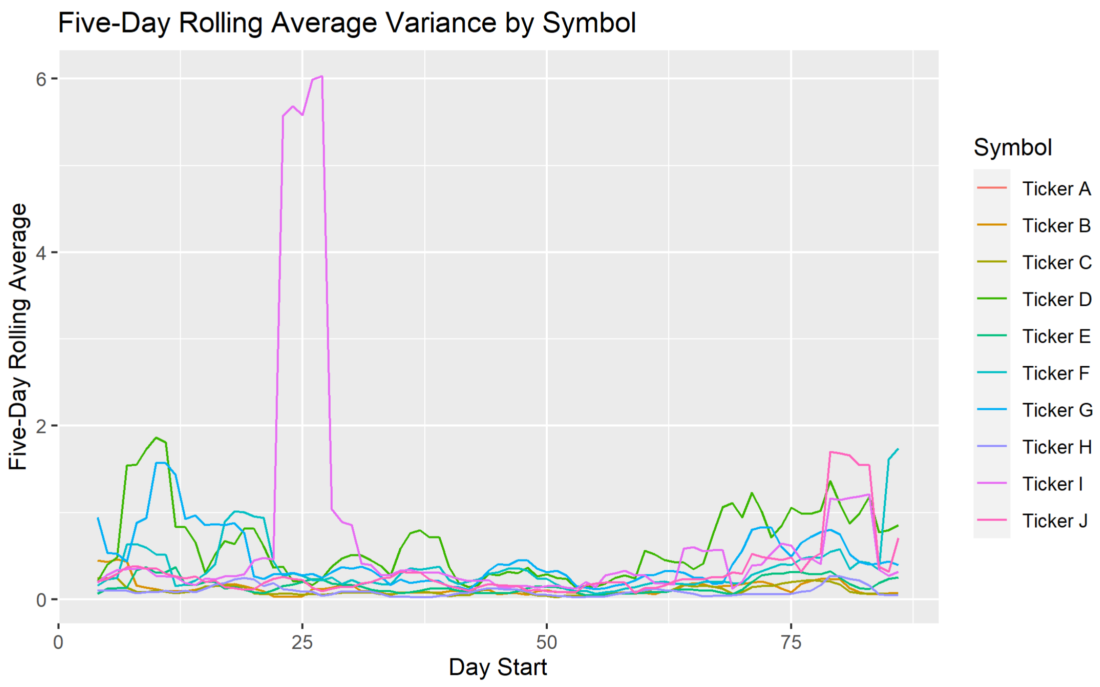
Nine of the ten stock tickers have missing data.

- Tickers A, B, E, F, G, and J have some missing data in the first half hour of trading (i.e., between 6 and 6:30am). These may be situations in which the exchange instituted an opening delay [2]. It could also simply reflect the fact that there were no trades on these tickers early in the trading day.
- Certain tickers have one-hour blocks of missing data. For example:
  - Tickers D and H have one hour of missing data from 6 to 7am on Day 0.
  - Ticker D has one hour of missing data from 9 to 10am on Day 31.
  - Tickers F, I, and J have one hour of missing data from 8 to 9am on Day 23. Ticker I has an additional hour of missing data from 10 to 11am on Day 42.

  Given that these blocks are exactly one hour long, these situations probably represent trading halts instituted by the exchange "in anticipation of a news announcement, to correct an order imbalance, as a result of a technical glitch, or due to regulatory concerns" [2].

The graph below shows the rolling average variance, in five-day blocks, of open prices for the different tickers. The significant jump for ticker I around days 22 through 25 may shed some light on the missing data for Day 23. The range in open prices for days 22 through 25 is significantly higher for ticker I than for the other stocks ($16.48 for ticker I versus an average of $4.20 for the nine other tickers). We surmise that the exchange may have halted trading due to high price volatility.

In addition, this graph suggests that volatility is generally higher near the beginning and end of the training period, which could pose challenges in forecasting.



Since missing data tended to be earlier in the training set (far before the 14 day lag we were using to make predictions after day 86), we do not believe these missing values will have a significant impact on our model performance. Based on our hypotheses of trading halts and no trades, we will take a simple approach of filling in the missing data using the last observation carried forward when possible. For the stocks that have missing data for the first time period of day 0 (tickers A, D, F, and H), we used the first observation available and carried it backward. The forecastML package we used for our final Kaggle predictions can handle missing data by leaving the entries as NA's, but we filled the data for consistency purposes and for the classical time series approaches.

4.3 Large Dataset & Memory Limits - Operating at the Hourly Level

TLDR: We built our direct forecasting models at the hourly level, made predictions at the hourly level, and assigned all 5-second segments in that hour to have the same prediction as that hour. This does not significantly increase our error compared to a 5-second model and enables us to include more features and iterate more on our models due to limited compute.

At the 5-second granularity, we have a large dataset that requires high compute. This compute cost limits the number of features and lagged features we can include if we wanted to model at the 5-second granularity.

As we see in Section 5.2, the variance of open prices at different hours is not that high. Even for the first hour that has high variance compared to other hours, the variance still only maxes around 0.45, meaning the standard deviation of open prices within an hour is less than 75 cents. Given the low variance, we do not expect a large decrease in error rate if we perform our modeling at a higher granularity.

The hourly granularity provides a **good balance of features to use and tractable compute times**. Thus, we built our direct forecasting models at the hourly level, made predictions at the hourly level, and assigned all 5-second segments in that hour to have the same prediction as that hour.

To validate that the error increase isn't significant, we ran our simplest baseline model of carry forward last observation at the hourly level as well as at the 5-second granularity level. When calculating our error score, we saw a minimal increase from the hourly error rate when we applied them to the seconds level data, meaning the hourly predictions still yield good results.

Model	Test set (days 78-86)	
	First 4 days	Last 5 days
CFLO Hourly Level	170	300
CFLO Seconds Level	195	335

Remember that our subgoal is to ensure we can create a model that provides directionally good predictions. Elevating the granularity of the predictions here still gives us a model that has directionally good predictions.

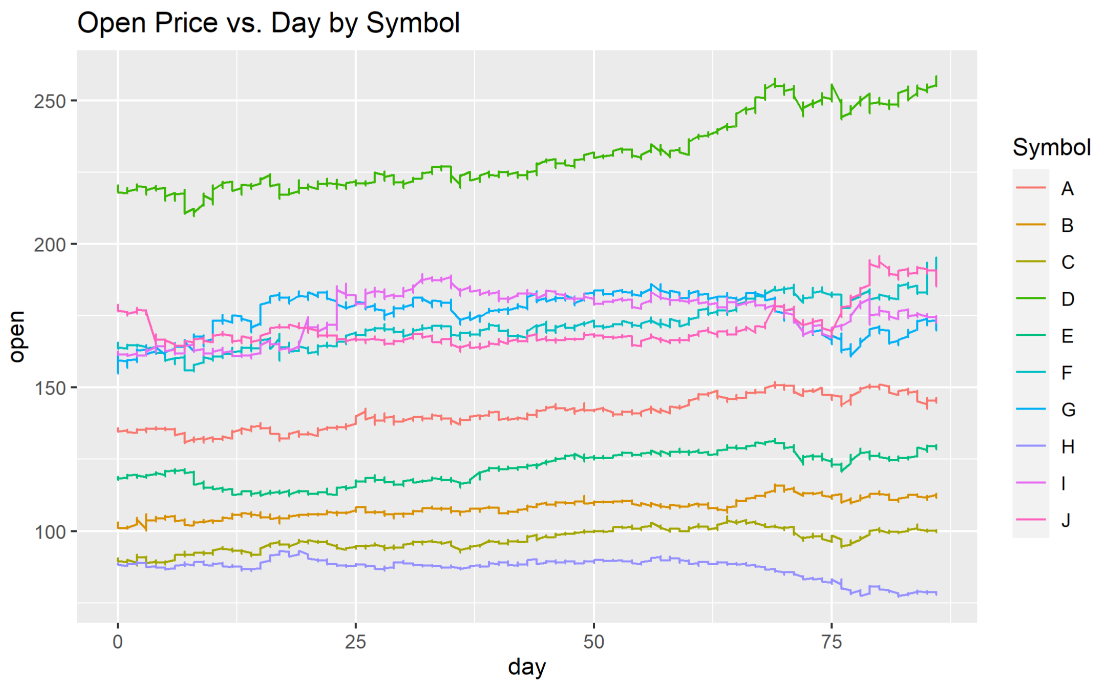
4.4 Log Transformation for Classical Time Series Models

For the classical time series methods, we model the natural log of the open prices instead of the raw prices. This amounts to modeling returns intead of prices. We make this transformation because stock prices tend to be non-stationary, and the models we use require stationarity [3]. The variance of the price changes over time and is proportional to the magnitude of the price, while the return tends to stay around zero over time with a much lower variance. Logging the prices may not make the time series stationary (in fact, it seems that it does not), but it still might be better than modeling raw prices.

5 Exploratory Data Analysis

5.1 Basic Features of the Data

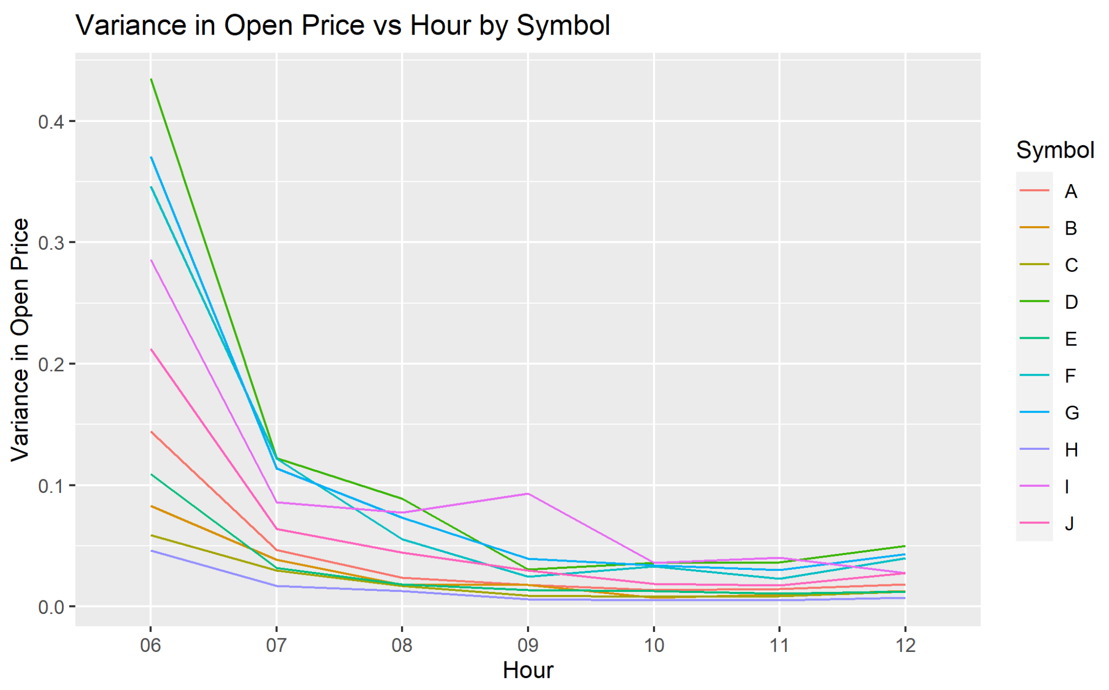
The graph below shows open prices over time by stock ticker in the training data set. The price levels differ by ticker.



5.2 Variance of open price

TLDR: We added standard deviation of open price during the past hour and past day as a feature in the direct forecasting models.

Given the missing data in early morning hours, we wondered if the variance in open prices by ticker differed by hour of the day. We found this to be true: **Variance is higher in the early morning**, as shown in the graph below. We added this as a feature in the hopes that our model will add extra variance around the prices in the early morning.



5.3 Stationarity

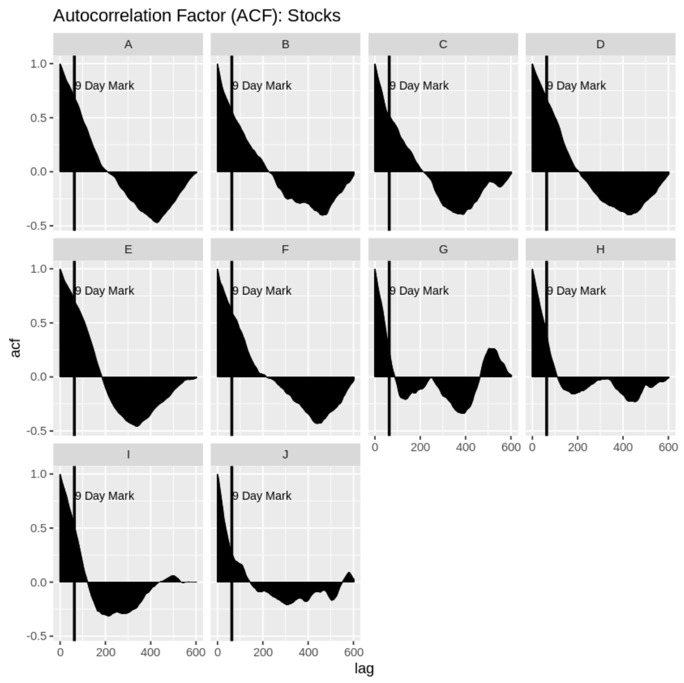
TLDR: The stock prices are not stationary. For the direct forecasting approaches, we make our time series stationary by taking the 1-day difference between the prices so that we predict the change in price rather than the actual price.

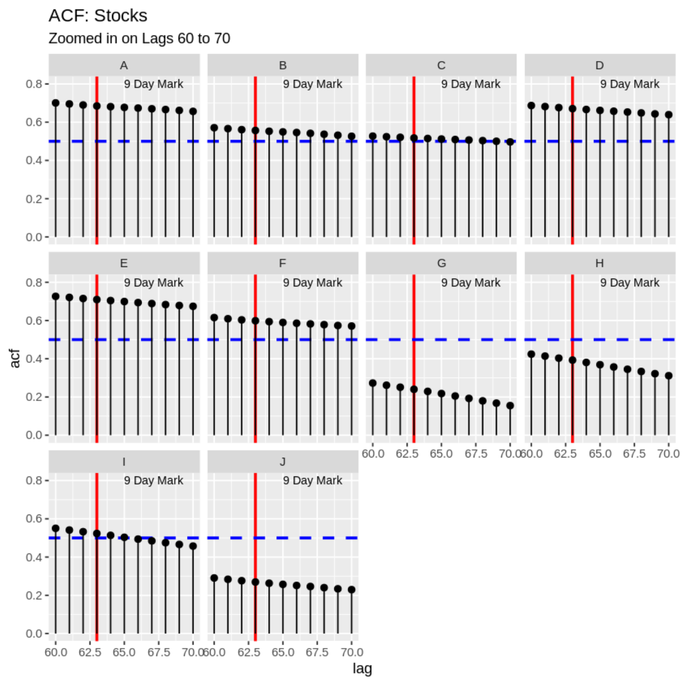
Stationary data is generally easier to predict than non-stationary data. Moreover, several of the models we employ assume that the time series data is stationary. From the open price vs. day plot in Section 5.1, we can see an increasing trend for several tickers and predict that the data is not stationary. We performed several tests to confirm. Since we run the moving average regression model (which requires stationarity) on logged prices, we also use logged prices in the tests described below in Sections 5.3.2, 5.3.3, and 5.3.4.

5.3.1 Autocorrelation

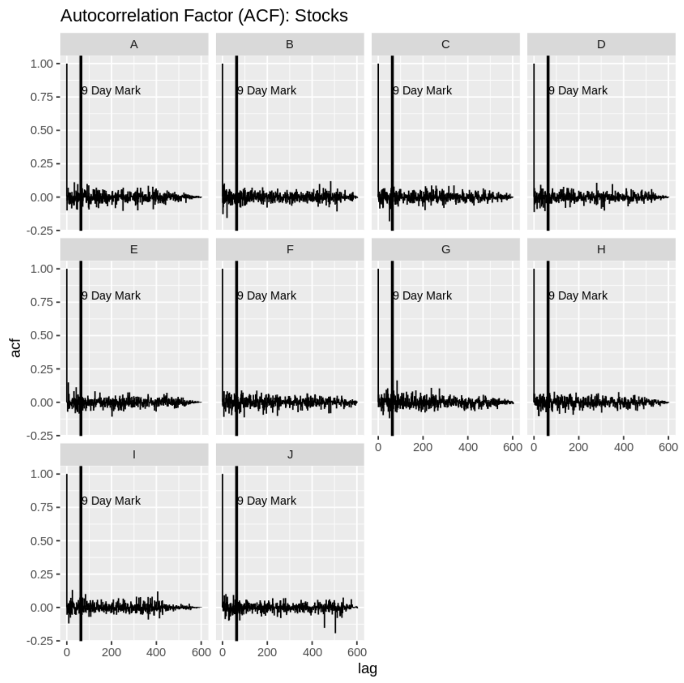
We can check to see if there is autocorrelation in the open prices as a way to see if our data is stationary. We are predicting 9 days into the future, so we want to see if there is high autocorrelation at the 9 day mark.

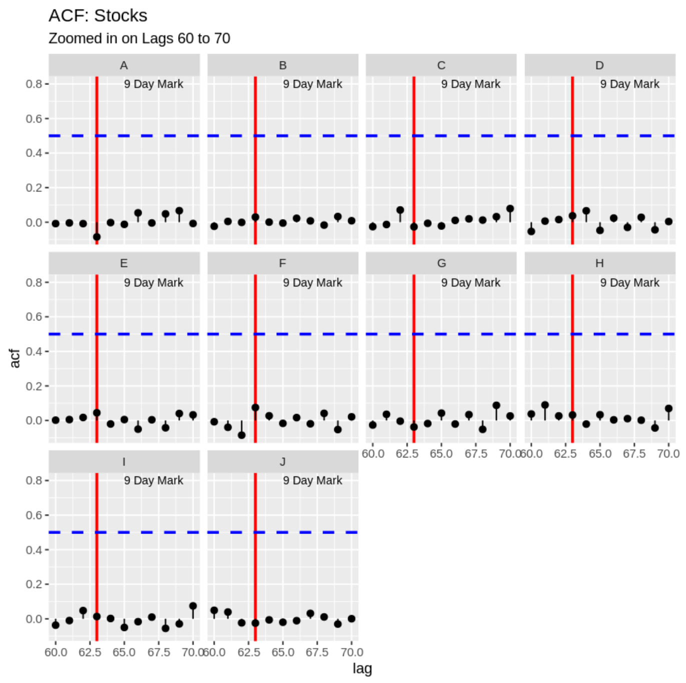
There does seem to be high autocorrelation in the data, with ACF values > 0.5. This would be good for a neural network model like LSTM, but makes it more difficult for our direct forecasting approach.





We can get rid of this autocorrelation by taking the difference between the open price and the previous open price.





5.3.2 Ljung-Box test

The Ljung-Box test studies whether there is evidence for non-zero correlations within a time series [4]. The null hypothesis is that there is an absence of serial correlation. Using hourly data and a lag length of 25, we found that the p-values for each ticker are very low, suggesting that we can reject the null hypothesis of no serial correlation.

5.3.3 Augmented Dickey–Fuller (ADF) t-statistic test

The null hypothesis of the ADF test is that the time series has a unit root. (A time series with a unit root is non-stationary.) Tickers C through J have high p-values on this test, suggesting that we fail to reject the null hypothesis of a unit root for these stocks' logged prices [4], [5].

5.3.4 Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test

The KPSS test examines whether a time series is trend-stationary (i.e., stationary around a deterministic root) [6]. All of the stock tickers have low p-values on this test, suggesting that we can reject the null hypothesis of trend stationarity [7].

5.4 Ticker Correlation

TLDR: Some of the stocks are highly correlated with each other. We will try training our direct forecasting models in batches of correlated stocks to see if that improves performance.

The graph in Section 5.1 suggests that some tickers (such as A, B, and D) display similar changes in open prices over time, sloping upward and plateauing around Day 70. Other tickers, such as stock I, have more idiosyncratic price movements. The table below formalizes this by showing the correlation between tickers in average daily open prices.

Correlation between stock tickers of average daily open prices

	A	B	C	D	E	F	G	H	I	J
A	1.000	0.922	0.760	0.928	0.846	0.928	0.048	-0.586	0.430	0.539
B	0.922	1.000	0.768	0.909	0.783	0.905	0.144	-0.555	0.414	0.508
C	0.760	0.768	1.000	0.716	0.790	0.716	0.565	-0.144	0.521	0.288
D	0.928	0.909	0.716	1.000	0.793	0.967	-0.073	-0.706	0.205	0.687
E	0.846	0.783	0.790	0.793	1.000	0.771	0.101	-0.353	0.379	0.377
F	0.928	0.905	0.716	0.967	0.771	1.000	-0.008	-0.698	0.357	0.659
G	0.048	0.144	0.565	-0.073	0.101	-0.008	1.000	0.589	0.569	-0.411
H	-0.586	-0.555	-0.144	-0.706	-0.353	-0.698	0.589	1.000	0.039	-0.813
I	0.430	0.414	0.521	0.205	0.379	0.357	0.569	0.039	1.000	-0.138
J	0.539	0.508	0.288	0.687	0.377	0.659	-0.411	-0.813	-0.138	1.000

For our direct forecasting approach, we try to leverage this correlation by training correlated stocks together. However, as the CDO crisis showed, using past correlations to estimate future correlations can be risky [8].

5.5 Increasing Variability for Latest Dates

TLDR: We make sure to include the latest dates as a specific evaluation period and give it more weight since we will be using the features from the latest dates to make our final Kaggle forecasts.

As mentioned in Section 4.2, the last few days in the training set have much more volatility compared to the previous days. This indicates that we should expect higher errors during these times. We also know that recent volatility could indicate continued volatility in the stock market, so we want to make sure we are looking at our model performance during these latest time frames as one of our cross validation folds.

6 Baseline Model

As a naive baseline model, we calculated errors for carry forward last observation (CFLO). This model simply takes the last value of the time series in training and uses that as the forecast for future predictions. We calculate the performance of this model with each of our cross validation windows and reference this baseline throughout the report.

7 Direct Forecasting

We leverage an R package called forecastML to help set up the framework for our direct forecasting. This package provides us with a way to fill missing data, lag our predictors, create multiple horizons and models for each horizon, make forecasts, and combine forecasts. We built a generating function using the forecastML package to help us test different model configurations and explore how different settings affect performance.

We evaluate our models by **creating 3 sets of train test data**. We picked **days 25, 50, and 77 as our validation cutoffs**, meaning we trained with data before those days and predicted values 9 days after those days. This gave us a good representation of how the model would perform at different time periods since the time series changes quite significantly near the later dates.

We evaluated the errors for cv (average of the 3 cross vaildation windows we picked) as well as for latest (only the last window of day 77) since, as mentioned in Section 5.5, the latest period has high impact on our actual forecasts.

We implemented the same error function as provided in the project directions and consider forecasting errors for days 0-3 separately from days 4-9.

We considered the following configurations:

- Stationary Transform
- Lag Window
- Prediction Horizons
- Correlated Symbols
- Features to Include

Finding the absolute minimum of all these configurations would require too many simulations. Instead, we set some reasonable sounding default values and tested each configuration one at a time, holding the other factors constant. We updated the defaults as we found better options. This greedy approach will give us a local minimum that is likely sufficient for our needs. We also compare these to our baseline of carry forward last observation.

The initial defaults we started with were 14-day lag window, 5 prediction horizons (1 hour, 1 day, 3 day, 6 day, 9 day), all symbols trained together, and aggregated features only (average, sd, range).

The final list of errors that we tested can be found in this google sheet.

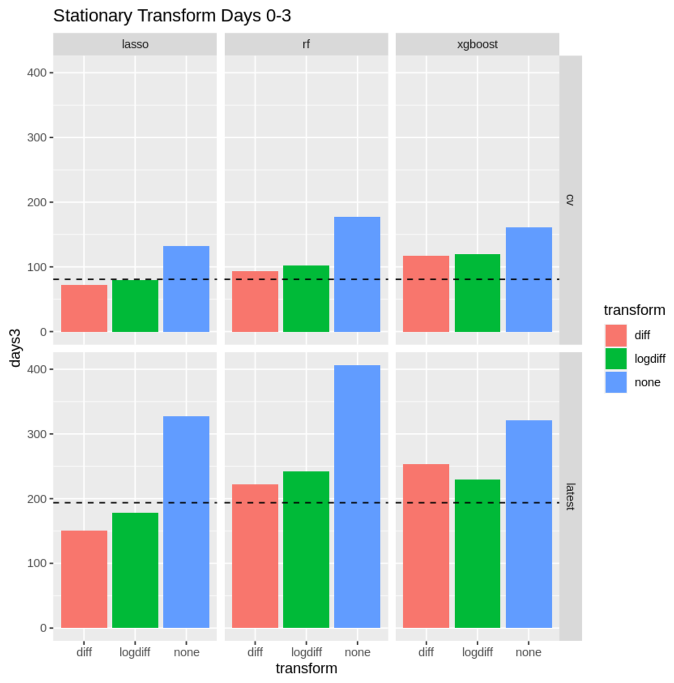
7.1 Stationary Transform

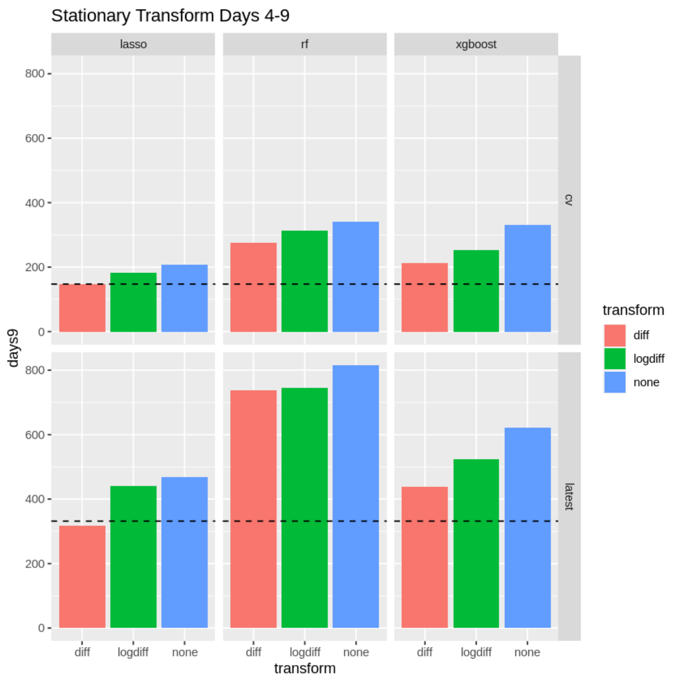
TLDR: We set our model to use diff transform.

As mentioned in the stationarity analysis, the time series as given is not stationary and could be more difficult to predict. Taking the difference or the change between each time period could help make the time series stationary. Taking the log of the values before the difference could help even out the variance of the changes.

We tested 3 configurations of stationary transformations:

1. none - no transform
2. diff - transform to be the difference between current price and previous price in 5-sec interval
3. logdiff - log the prices, then take the 1 segment difference between current price and previous price





From the results, we can see that transforming the variables does make our models perform better. Just taking the diff without the log seems to perform much better in all cases. We know from the exploratory analysis that the variance of the time series changes, especially near the latest values at the end. Therefore, stabilizing the variance with logs may have reduced an important signal, and thus performs worse than just doing diff transform.

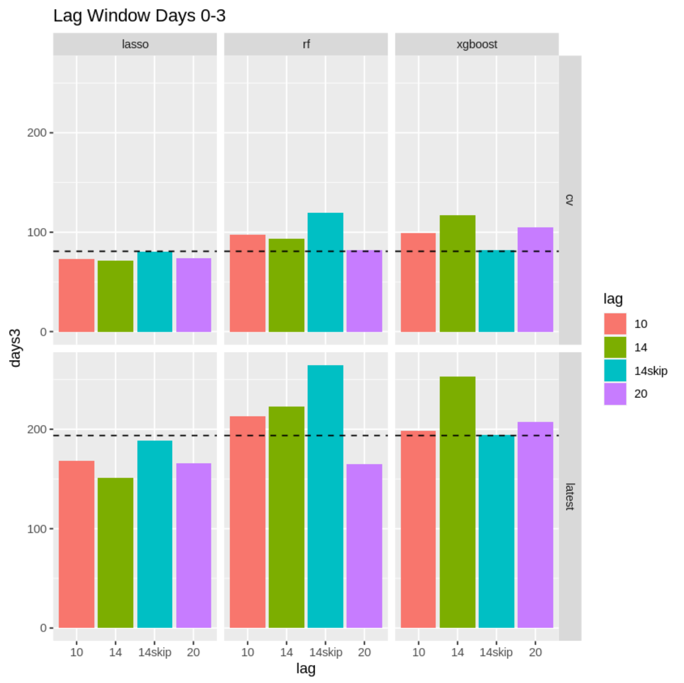
7.2 Lag Window

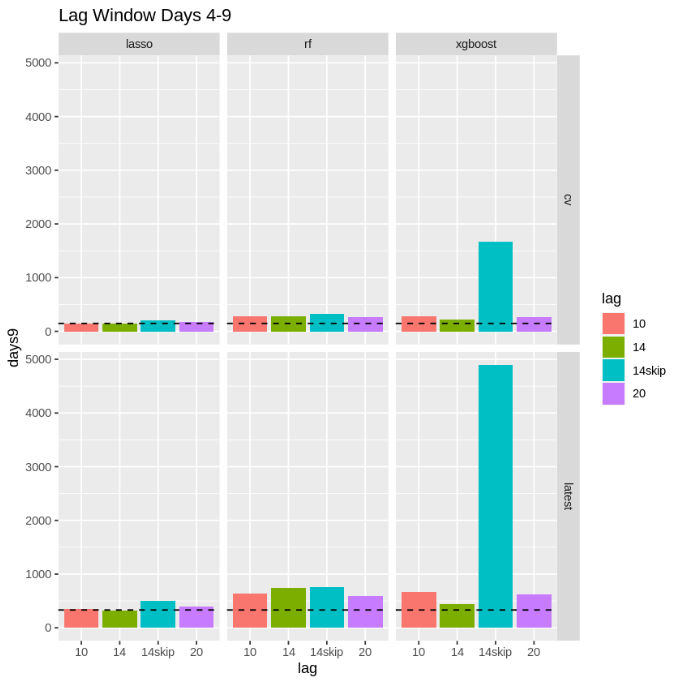
TLDR: We set our model to use a 14 day lag window for its lag predictors.

Adjusting the lag window lets us include more or fewer predictors for each prediction. A longer lag window includes more predictors, which may help the model, but may also produce more noise and also make training become computationally expensive/infeasible.

We need to forecast 9 days into the future, so at a minimum, we need a 9 day lag window. We test the following 3 configurations:

1. 10 days - this lets us at least have 1 day of data for predictors
2. 14 days - this is the default in our stationary transform test, so we just reuse the diff model
3. 20 days
4. 14 days skip - this is 14 day lookback, but only including 1 data point per day (every 7 segments)





From the results, 14 days seems like a good balance for the lookback. 14 day skip doesn't seem to work too well, since we are only taking the exact hour of every day. Prices are likely to be more similar to prices closer to the time period of prediction.

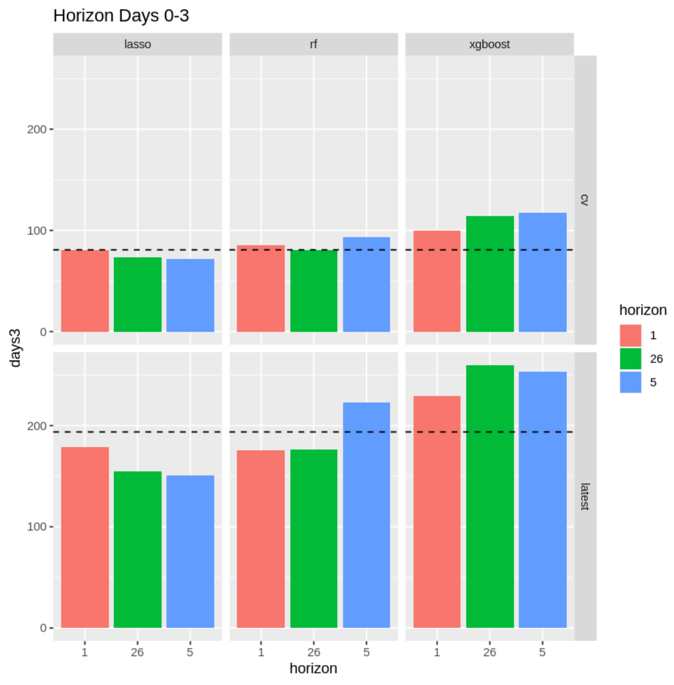
7.3 Prediction Horizon

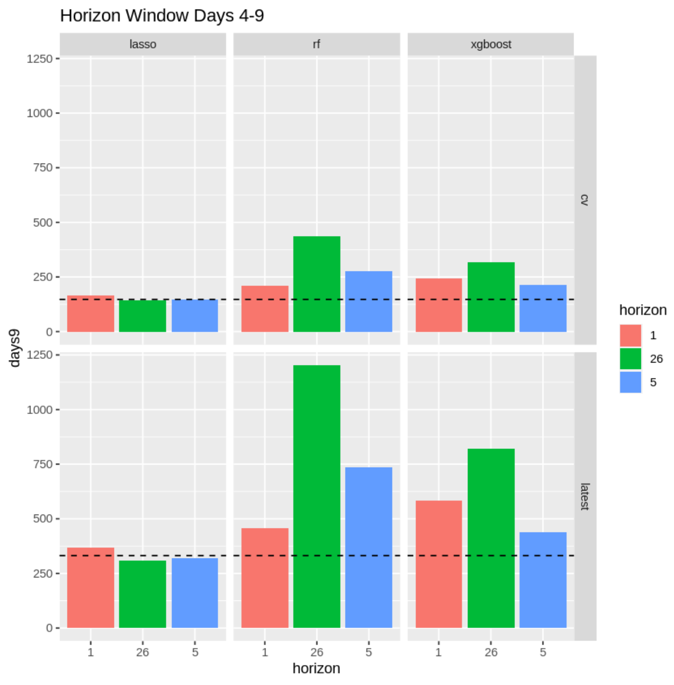
TLDR: We set 5 horizons, training 5 separate models before combining them.

forecastML lets us set different horizons and train a different model per horizon. For example, we can train a 1 day model to forecast 1 day, a 3 day model to forecast 3 days, etc. Afterwards, forecastML lets us combine these models into 1 forecast. We use the naive approach of using the shorter horizon models for the shorter dates (e.g., day 1 uses a 1 day horizon model, days 2-3 use a 3 day horizon model, etc).

We checked a few different horizon combinations to see what worked well:

1. 1 - 1 horizon, predict whole batch of hours 1 to 63 all at once
2. 5 - 1 hour, 1 day, 3 days, 6 days, 9 days
3. 26 - 1 model for each hour in the first 3 days (21 models), then 1 model per day afterwards until day 9 (5 models)





1 horizon seems to work well for randomforest and xgboost. These tree models likely are better at classifying everything in one go.

The 26 and 5 horizons worked well for lasso. 26 took an extremely long time to train and didn't seem too far in performance from 5 horizon. 5 horizon seemed like a good balance to stay with. Having multiple horizons helps more since the short term predictions can use more closer features. The lag feature of forecastML will automatically remove the lag days for features within the prediction window. For example, if we are predicting 9 days into the future with a 14 day lag window, we can only predict with features 10-14 days before any of the forecast days even if we technically have closer values for some of the early days. Therefore, having more horizons and models specific for the closer days lets us include more close features in the earlier predictions and improve the performance there.

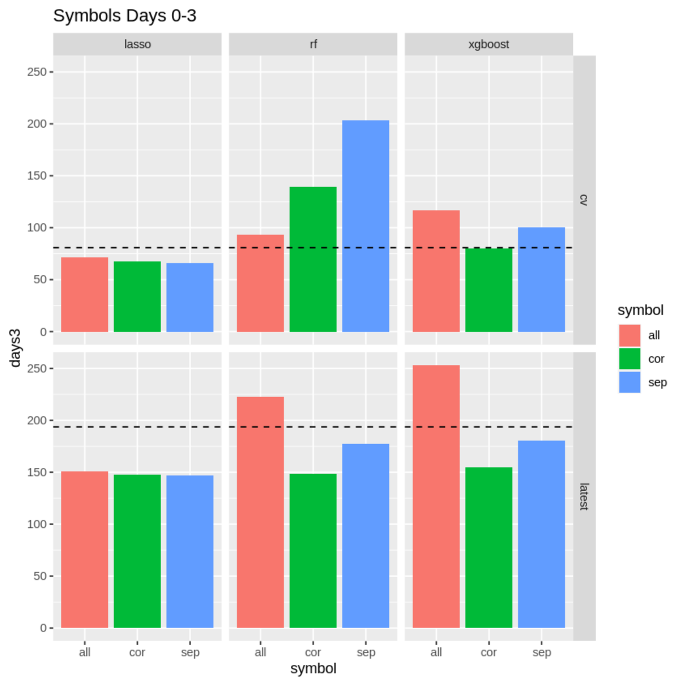
7.4 Correlated Symbols

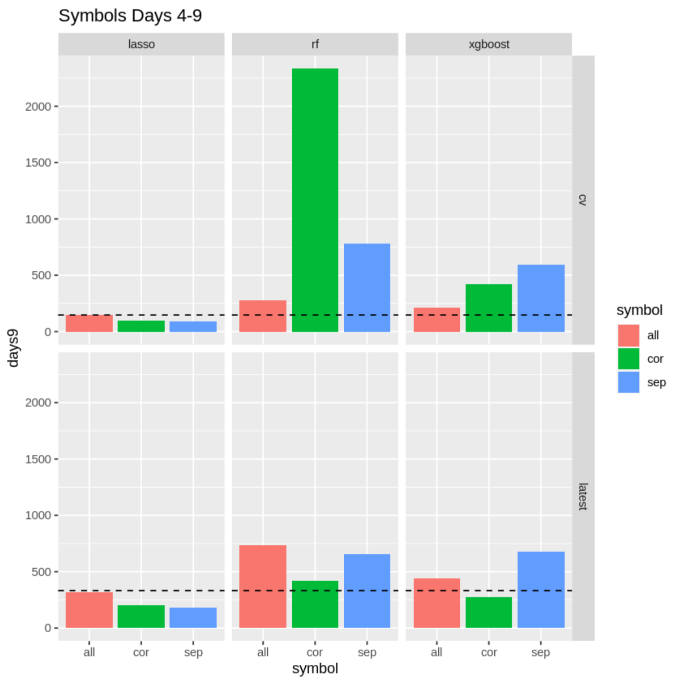
TLDR: We train our models on each symbol separately. Training in batches of correlated symbols seems to increase the error rate.

We saw in Section 5.4 that several symbols were highly correlated with each other. We wanted to try to leverage this feature by training our models with correlated symbols together. By doing so, the model may pick up general shifts.

We train 3 different batches:

1. all - all the symbols trained in one model together
2. correlated - symbols trained in groups of highest correlations: A, B, C, D, E, F, G, I, H, J
3. separated - each symbol trained by itself





We found that training each symbol separately yielded the lowest errors for lasso. It's likely that the correlation was just introducing more noise during the training when we try to use the symbol as an extra variable.

For randomForest and xgboost, correlated symbols seemed to perform pretty well except for a weird spike in cross validated errors for randomForest on days 4-9.

Another way would have been to add the features of the other correlated symbols together. This required more compute and we did not have the time to explore this option more. Also, we saw that training the correlated symbols separately seemed to perform the best.

7.5 Features

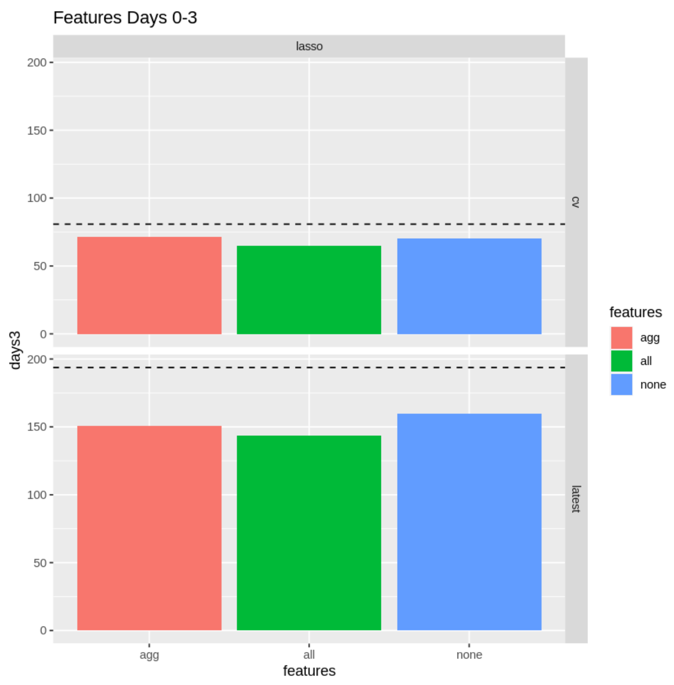
We included the following features for our model.

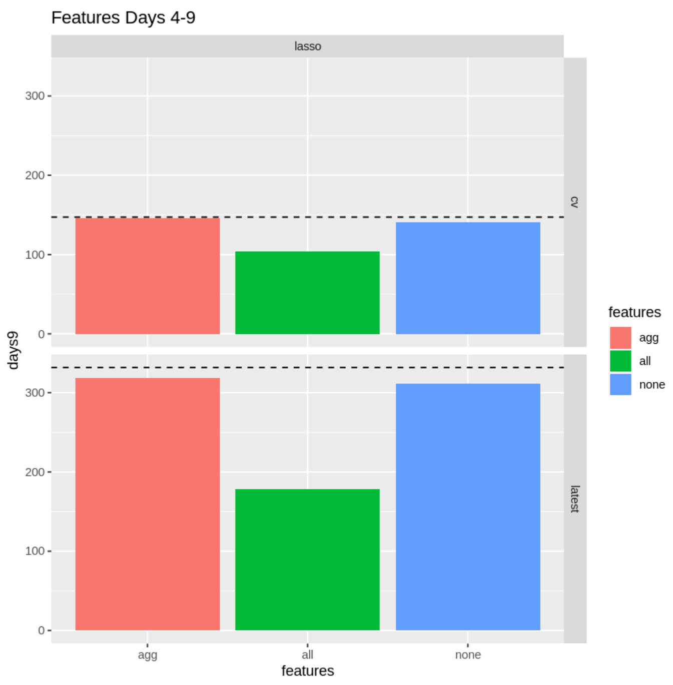
- day
- hours
- day_of_week - day modulus 5 to create a factor of 5 days representing each day of the week. This may be messed up if there is a holiday.
- open_day - open price of previous day
- close_day - close price of previous day
- high_day - high price of previous day
- low_day - low price of previous day
- average_day - average of open prices for all 5-sec segments of the previous day
- sd_day - standard deviation of open prices for all 5-sec segments of the previous day
- range_day - high price - low price of previous day
- open_hours - open price of previous hour (not including current hour)
- close_hours - close price of previous hour
- high_hours - high price of previous hour
- open_hours - low price of previous hour
- average_hours - average of open prices for all 5-sec segments of the previous hour

- sd_hours - standard deviation of open prices for all 5-sec segments of the previous hour
- range_hours - high price - low price of previous hour

One of the configurations we explored was how many features to include.

1. all - include all the features we added
2. aggregated - include just the aggregation features like average, range, and sd. Ignore the close, high, and lows.
3. none - don't include any other price features





Including all features seemed to perform the best except for the 3 day prediction in the last window when no features performed the best.

Since we basically had 50 different models combined into one model for the final submission, it is difficult to evaluate what were the important features. To get a directional sense, we investigated the lasso model trained with all symbols at once for the 3 day horizon and the 9 day horizon.

The 3 day horizon model mostly used open_hour_diff, sd_hours, and range_hours.

Feature	Coefficient	Feature	Coefficient
(Intercept)	0.0280301570	sd_hours.y_lag_78	-0.0038404091
open_hours_diff_lag_25	0.0006718077	range_hours.y_lag_29	-0.0015108383
open_hours_diff_lag_84	0.0313278486	range_hours.y_lag_41	0.0181645529
open_hours_diff_lag_90	0.0014023805	range_hours.y_lag_48	0.0039968268
sd_hours.y_lag_27	0.0166955080	range_hours.y_lag_63	-0.0008062212
sd_hours.y_lag_41	0.0004034906	high_day_lag_29	-0.0125833602

The 9 day horizon model kept many of the lagging open_hour features, range features, and sd features. It also kept the day of the week feature.

Feature	Coefficient	Feature	Coefficient
(Intercept)	1.456642e-02	range_hours.y_lag_69	4.608209e-03
open_hours_diff_lag_70	6.383332e-03	range_hours.y_lag_76	2.172364e-03
open_hours_diff_lag_73	-1.060476e-02	range_hours.y_lag_81	-3.260835e-03
open_hours_diff_lag_77	-1.217987e-02	range_hours.y_lag_83	2.146889e-02
open_hours_diff_lag_79	-3.357909e-03	range_hours.y_lag_92	-5.267391e-03
open_hours_diff_lag_84	4.587403e-02	sd_day_lag_64	-3.501414e-03
open_hours_diff_lag_88	-3.947626e-03	range_day_lag_78	-7.221302e-03
open_hours_diff_lag_90	1.551609e-02	open_day_lag_71	3.874009e-05
open_hours_diff_lag_91	1.472980e-03	close_day_lag_98	-1.525196e-03
sd_hours.y_lag_70	-4.874027e-03	high_day_lag_64	-1.498921e-02
sd_hours.y_lag_78	-1.069495e-02	high_hours.y_lag_70	-4.883284e-06
sd_hours.y_lag_90	9.025280e-03	day_of_week	2.510267e-03
range_hours.y_lag_63	-6.419512e-03		

8 Classical Time Series Models

We considered three types of classical time series forecasts. One is described below, and two are described in the appendix (Section 13). While we did not use these forecasts for the Kaggle submission, they served as useful baselines.

In order to compare the performance of these models, we split the data set into three parts: training (first 68 days), development (next 9 days), and test (final 9 days). We used the development set to tune hyperparameters as needed, then calculated the error of each model on the test set.

8.1 Moving Average Model

As shown in Section 1, this model had the lowest error on the forecast data set of the models we considered. (In fact, it had the lowest error during period 1 of any of our classmates' submissions to the Kaggle competition.)

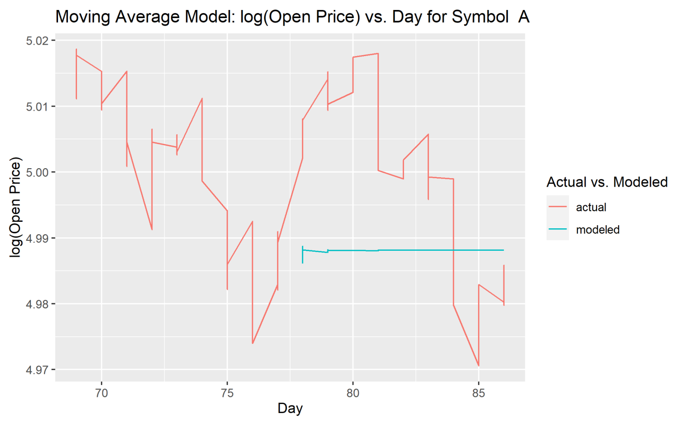
8.1.1 Building the Model

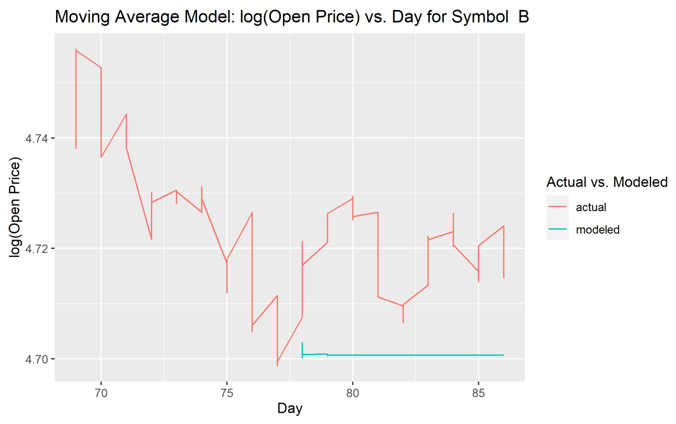
This very simple model uses moving averages by ticker to make predictions. In particular, the next open price of a stock is estimated to be the average of x past prices, where x is a parameter we choose.

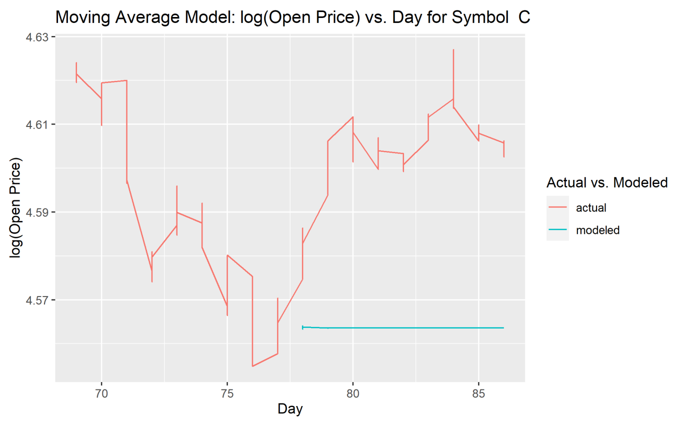
We used the training and development sets to test various averages, from a one-day average to a ten-day average. For each of these candidate averages, we fit the model using the training data and calculated the resulting error on the development set data. While the situation varied by ticker, we found that, overall, a shorter-term average predicts better for early days in the development set, while longer-term averages predict better for later days. Therefore, we decided to use a one-day average to predict the first three days of open prices and a three-day average to predict the last six days.

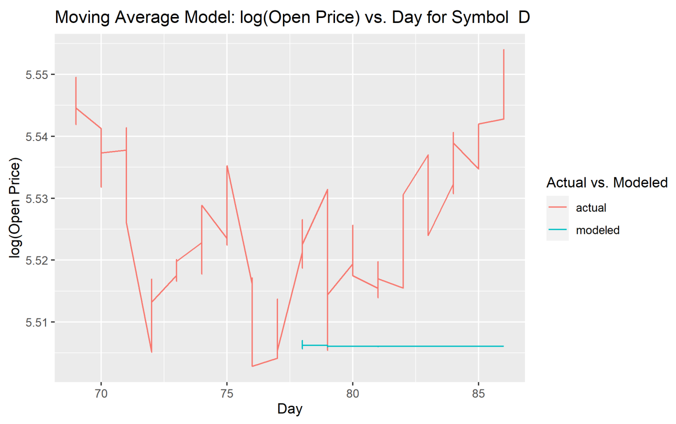
Finally, we used both the training and development data to calculate the error under the final model for the test set. For the first four days in the test set, the error is **175**, and for the last five days, the error is **312**. (While we modeled the log of the open prices instead of the prices themselves, these errors are in terms of raw prices.)

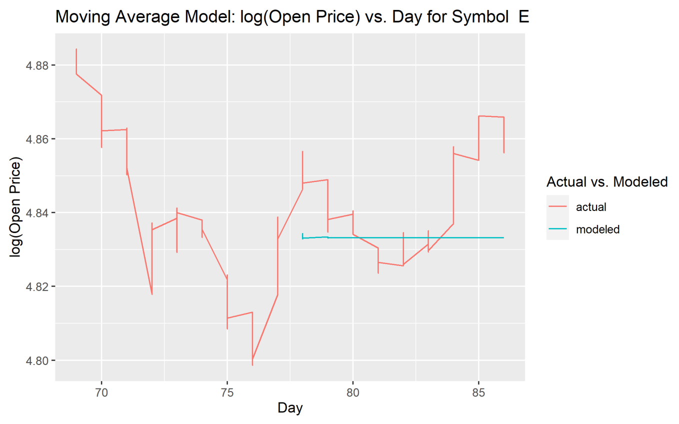
Plots of the predicted and actual open prices are shown below.

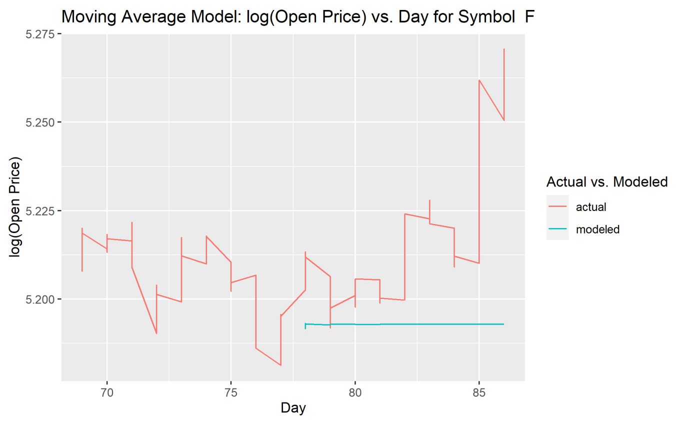

Moving Average Model: log(Open Price) vs. Day for Symbol A
log(Open Price)
Day
Actual vs. Modeled
actual
modeled

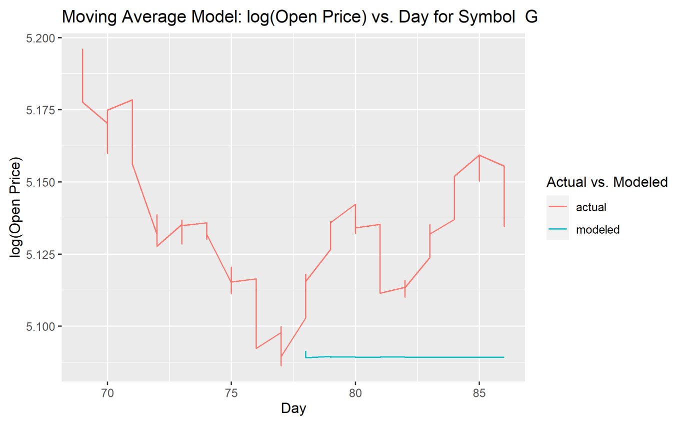

Moving Average Model: log(Open Price) vs. Day for Symbol B
log(Open Price)
Day
Actual vs. Modeled
actual
modeled

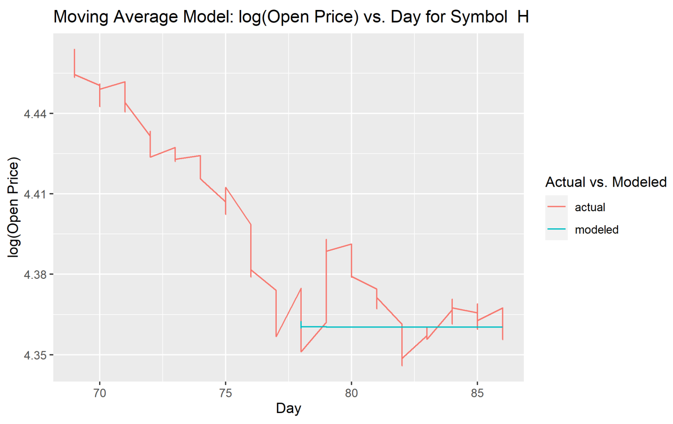

Moving Average Model: log(Open Price) vs. Day for Symbol C
log(Open Price)
Day
Actual vs. Modeled
actual
modeled

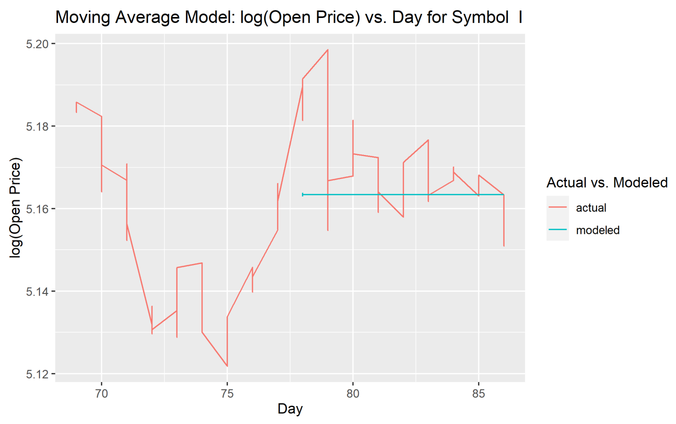

Moving Average Model: log(Open Price) vs. Day for Symbol D
log(Open Price)
Day
Actual vs. Modeled
actual
modeled

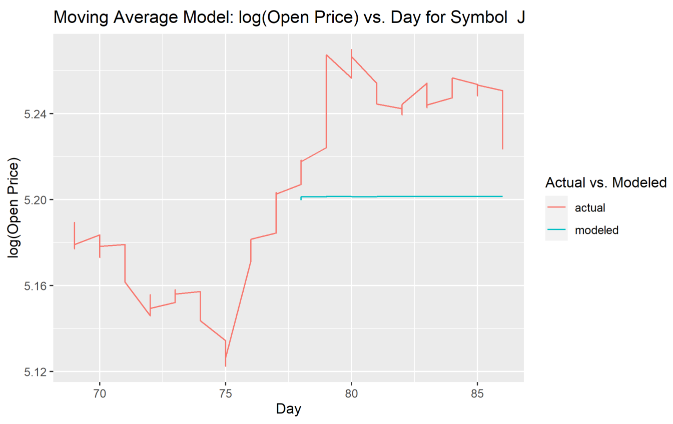

Moving Average Model: log(Open Price) vs. Day for Symbol E
log(Open Price)
Day
Actual vs. Modeled
actual
modeled


Moving Average Model: log(Open Price) vs. Day for Symbol F
log(Open Price)
Day
Actual vs. Modeled
actual
modeled





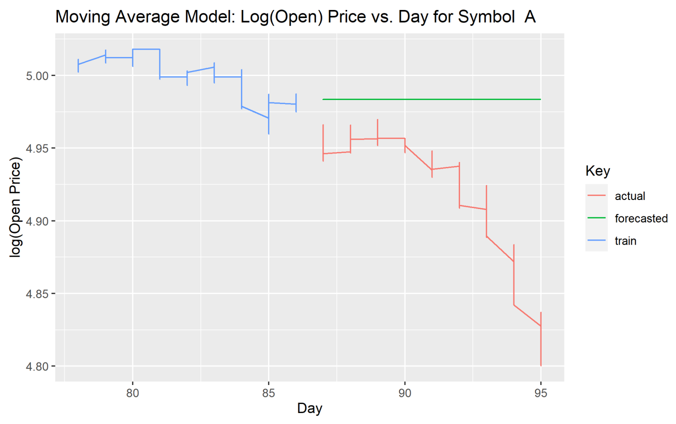


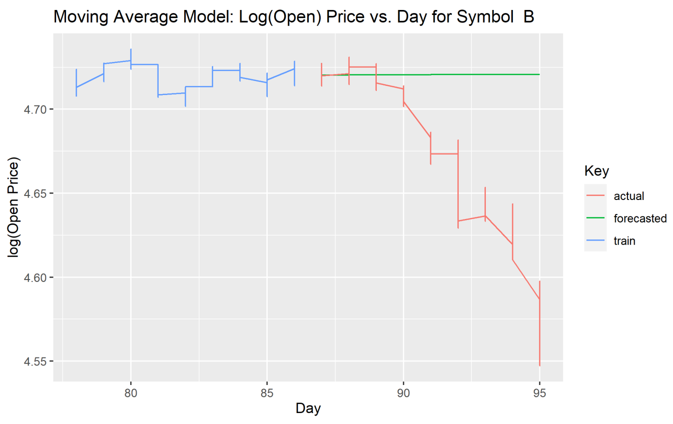


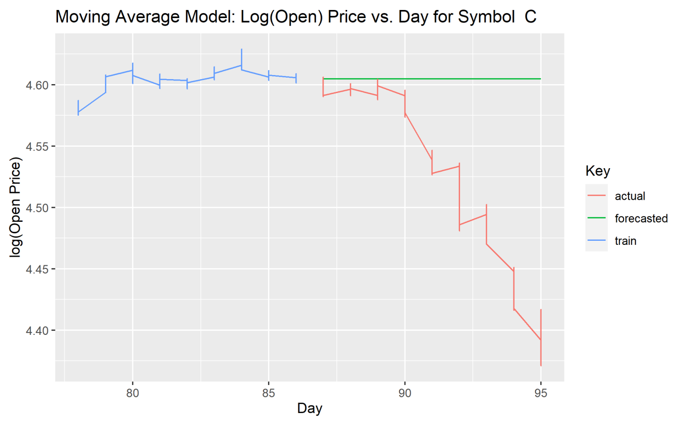
The predictions for each stock quickly settle down to a constant. This reflects the fact that, as we move farther into the future, more and more of the elements of the moving average represent averages themselves.

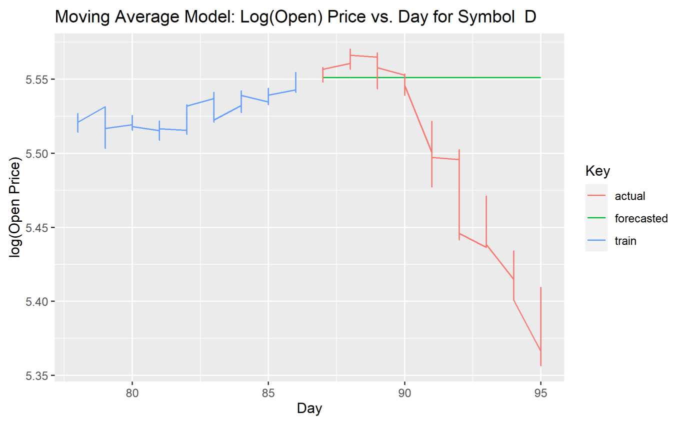
8.1.2 Making Forecasts

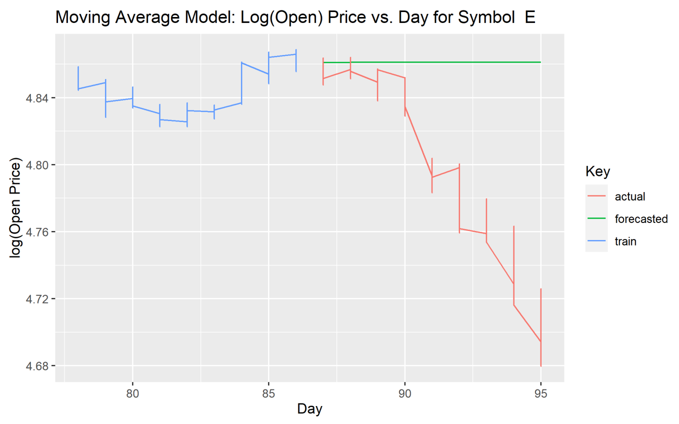
For the forecast period (days 87 through 95), we used a one-day average to predict open prices for the first three days of the period and a three-day average to make predictions for the last six days of the period. Plots of the forecasts are shown below, along with the actual prices during the forecast period.

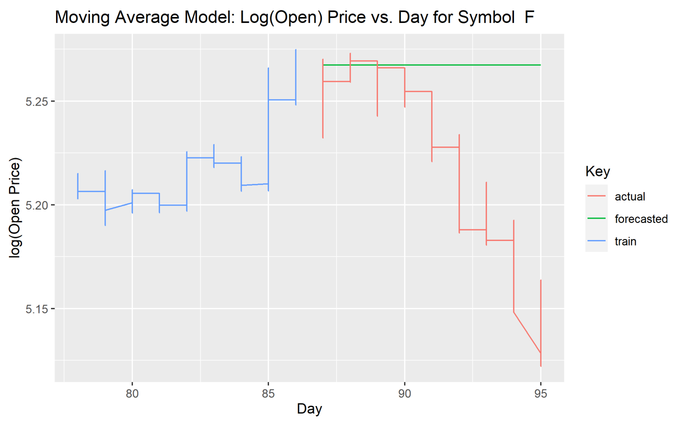


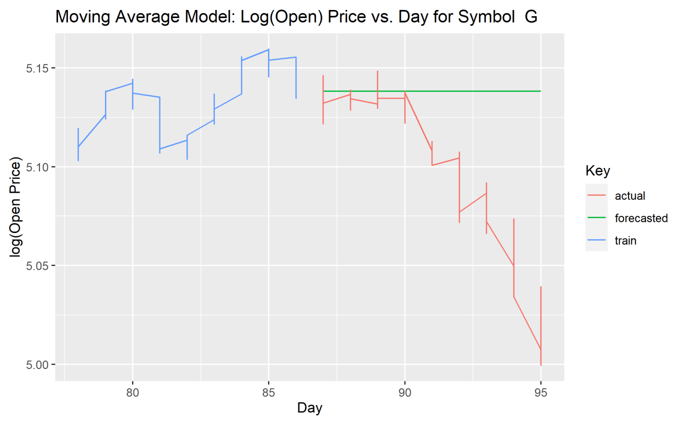


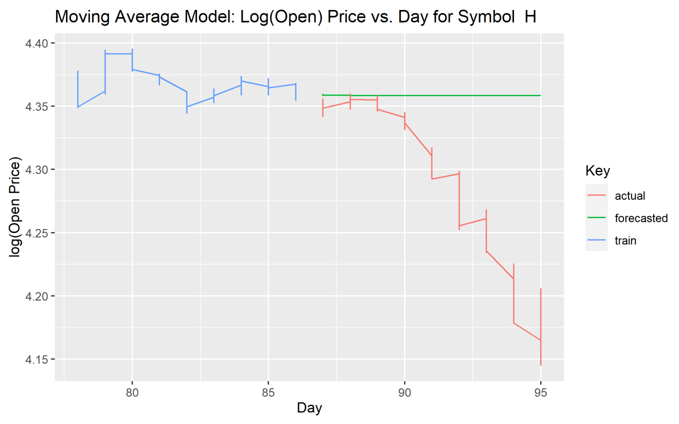

Moving Average Model: Log(Open) Price vs. Day for Symbol C
log(Open Price)
4.60
4.55
4.50
4.45
4.40
80
85
90
95
Day
Key
actual
forecasted
train

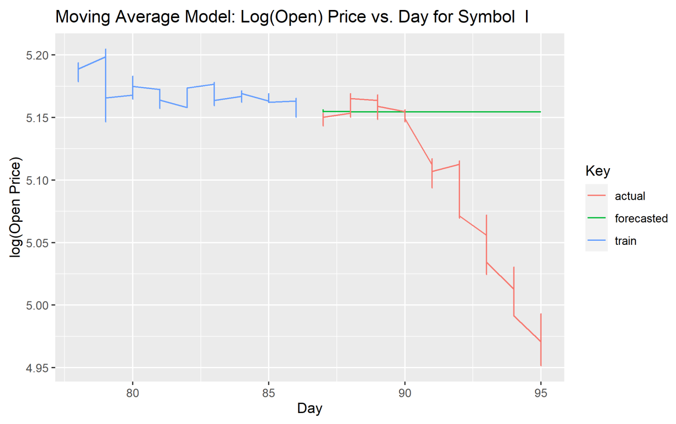

Moving Average Model: Log(Open) Price vs. Day for Symbol D
log(Open Price)
5.55
5.50
5.45
5.40
5.35
80
85
90
95
Day
Key
actual
forecasted
train

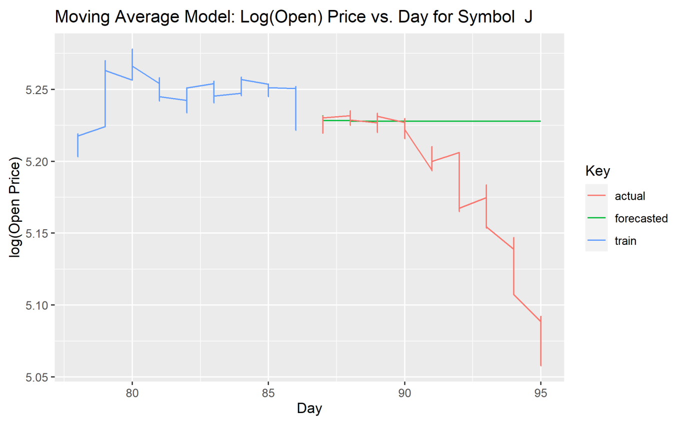

Moving Average Model: Log(Open) Price vs. Day for Symbol E
log(Open Price)
4.84
4.80
4.76
4.72
4.68
80
85
90
95
Day
Key
actual
forecasted
train


Moving Average Model: Log(Open) Price vs. Day for Symbol F
log(Open Price)
5.25
5.20
5.15
80
85
90
95
Day
Key
actual
forecasted
train


Moving Average Model: Log(Open) Price vs. Day for Symbol G
log(Open Price)
5.15
5.10
5.05
5.00
80
85
90
95
Day
Key
actual
forecasted
train


Moving Average Model: Log(Open) Price vs. Day for Symbol H
log(Open Price)
4.40
4.35
4.30
4.25
4.20
4.15
80
85
90
95
Day
Key
actual
forecasted
train





9 Kaggle Results

Our model performed slightly worse than the baseline CFLO in the final Kaggle results. From the information provided, we knew that the test data used for evaluation of the competition results was during the time period of Feb 18, 2020 - Feb 28, 2020. This timeframe is the start of COVID cases in the US and contains the peak in prices right before the huge crash from COVID lockdowns. This was an external, uncontrolled factor that should not have been predictable from historical data. We can see this is the case as all the submissions had extremely high error rates for Period 2, when the crash started happening.

Our data relied on a 14 day lookback, so the earliest relevant data point that our model used was day 72. We saw that there was increased volatility near the end of the training data and that in general, there were lots of upward trending stocks. We can see that our lasso model provided upward trending forecasts to match the trends it was seeing for many of the stocks. Downward trending stocks like stock H were also paired with a slightly downwards trending forecast. Given the information we trained our lasso model on, the forecasts felt reasonable.

If this was a real life scenario, we would have rerun our model when we started seeing the decline in stock prices across the board. If we were to do something like forecast days 7-9 using all the known data including days 0-6, the lasso model would have caught on to the change and predicted a lower declining slope. It likely would still miss the even more drastic drop that occurred in March when the lockdowns started to solidify, but we would have been directionally correct.

10 Challenges

We faced several challenges as we were working on this assignment.

- We were limited in the features we could use, since we did not have access to any features for the forecast interval.
- Since our task is to forecast nine days into the future, we relied heavily on the last nine days of the training data set in order to fit our models. This time period, however, differs from the rest of the data set in several important ways. For one thing, the variance is higher for the days at the end of the data set than for the days in the middle.
- The size of the data set meant that we ran into memory issues. The run time needed to select tuning parameters for the various models sometimes exceeded two hours. This was especially problematic for Julie, who's never met a `for` loop she didn't like. (Quentin kept recommending vectorizing, so she'll have to look into that.)

Moreover, as mentioned in Section 1, the data may not be amenable to a forecasting task. It is notoriously hard to predict stock prices, which is why "chartists" (who get paid to spend all day on the problem) have not been able to make outsized investment returns. The efficient market hypothesis says that the current price of a stock contains all information about that stock.

11 Additional Ideas and Future Work

We had many different ideas to explore for this project, but were limited by the time and our knowledge/comfort with building models. Here are some additional things we would have liked to try out.

- We saw LSTM pop up often as a powerful method for time series forecasts. We did try to explore LSTM, but the keras package would not initialize properly and we saw that there was likely a bug that was never fixed with the package. Therefore, we stopped pursuing the LSTM route since we were coding in R.
- We only explored 3 models with forecastML: lasso, randomForest, and xgBoost. There are likely other more suitable models for the task, but we lack the experience to know what would be a good model to use.
- We likely could've done more with correlation, such as including the features of correlated symbols together. However, this wouldn't have helped in this case since the whole stock market crashed.

12 Contributions of Each Team Member

Quentin built the direct forecasting models and ran simulations to pick out the best one, which we used for our Kaggle submission (described in Section 7). He also set up the Google colab with R kernel support so that we could share code.

Julie built the classical time series models (described in Sections 8 and 13), performed data analysis, and set up the Overleaf file for the report.

13 Appendix: Two Other Classical Time Series Approaches

13.1 Moving Average Regression Model

The moving average regression model assumes that the open price is linearly related to the weighted moving average of current and previous error terms; the weights are parameters to be estimated [9]. The model assumes that the time series for each ticker is "locally stationary with a slowly varying mean" [10]. Based on the data analysis (see Section 5.3), it seems that our data may *not* be stationary, even after we log the prices.

Accordingly, when we attempted to use R's `arima()` function with `order=c(0,0,1)` to fit a moving average model for each ticker, we got error messages. The default optimization routine did not work. We changed the optimization routine to the Nelder-Mead algorithm and were able to obtain output. But some of the predictions did not look reasonable; for example, the predictions of log(open prices) for stock I were near zero. These issues may be because the data is not stationary and hence a model of order (0,0,1) is not appropriate [11], [12]. Since the underlying assumptions may not be satisfied, we did not pursue this model further.

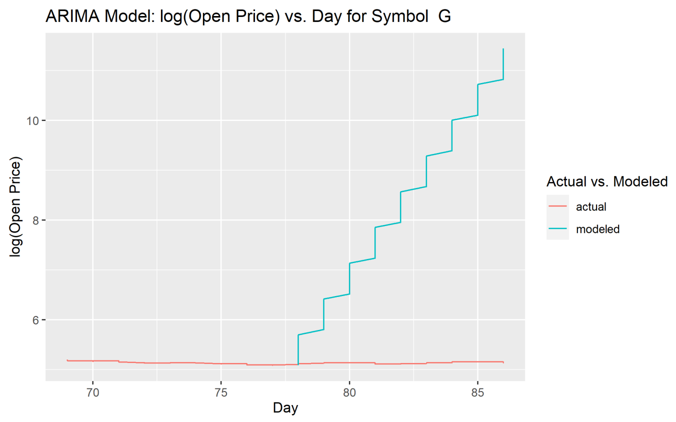
13.2 ARIMA Model

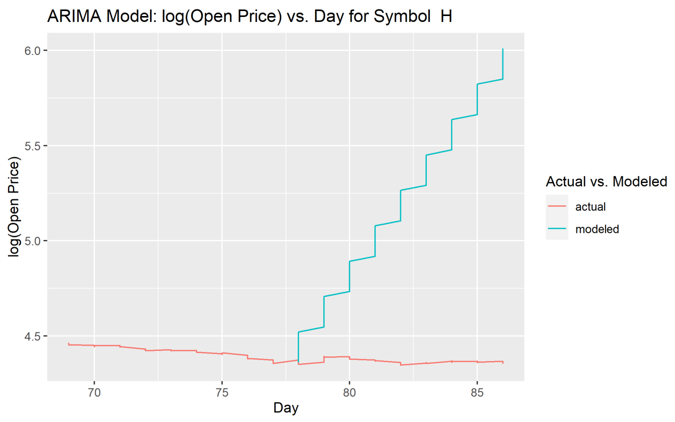
13.2.1 Building the Model

As noted above, one concern with using the moving average regression model is that our data may not be stationary, even though we're working with logged prices instead of raw prices. We partially addressed this issue by fitting ARIMA models to each ticker. The ARIMA model has the same assumptions as the moving average regression model, but the "I" [Integrated] process in "ARIMA" can be used to turn a non-stationary time series into a stationary one, as long as the non-stationarity meets certain criteria [13].

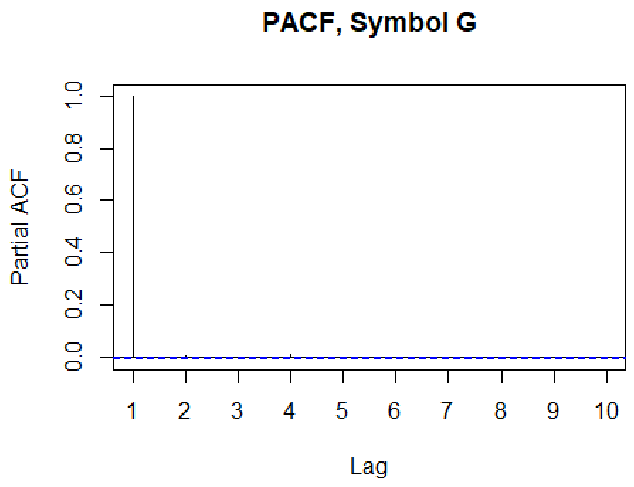
We used R's `auto.arima()` function to select the optimal p, d, and q parameters, then used the `predict()` function to forecast open prices for the test set. The `auto.arima()` function selected a value of $d = 1$ for eight of the ten tickers and $d = 2$ for one ticker; the d parameter represents the number of differencing transformations needed to make the time series stationary [14], [13].

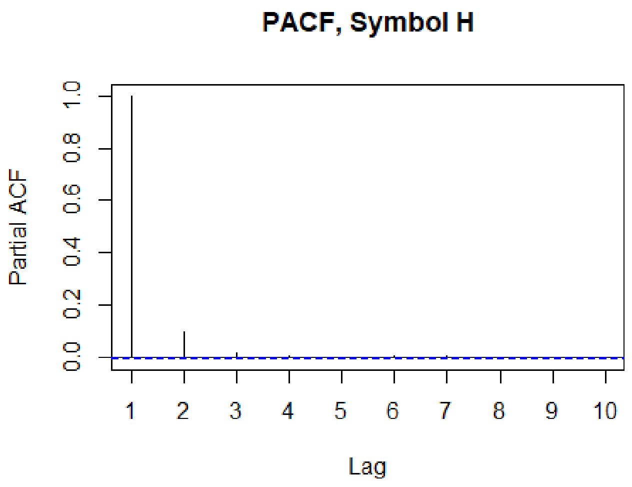
For two tickers, G and H, the parameters selected by `auto.arima()` ($p = 5, d = 2$, and $q = 0$) did not appear to yield reasonable predictions. It seems that this may be because of the order of the auto regression (AR) process for these stocks, which is selected to be $p = 5$ in both case; this seems high, as the next highest value of p chosen by `auto.arima()` is 3, for stock B. The plots below show the actual versus predicted prices in the test set for these two stocks.





These tickers may not be amenable to ARIMA modeling, regardless of the parameters selected, but it might be possible to find parameters that work better than those selected by `auto.arima()`. The plots below show the PACF plots. For stock G, the lags cut off at lag 1, suggesting that we might be able to use $p = 1$ in the ARIMA model [10]. For stock H, we might be able to use $p = 2$ or $p = 3$. At any rate, it seems that we might be able to reduce the value of p from 5 for both tickers.





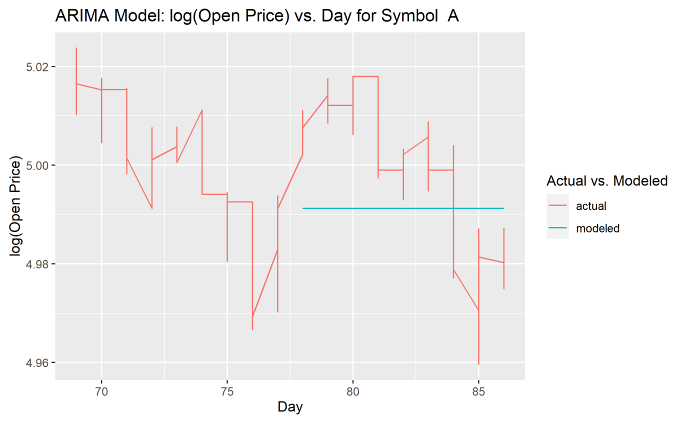
We used the training set to test all values of p, d, and q between 0 and 2 and calculated the resulting error on the development set. For ticker G, parameters $p = 0, d = 0$, and $q = 2$ give the lowest error on the development set. But the parameters $p = 0, d = 0$, and $q = 0$ give a similar error. For reasons of parsimony, we decided to go with $p = 0, d = 0$, and $q = 0$ for this ticker. This is a pure white noise model [15].

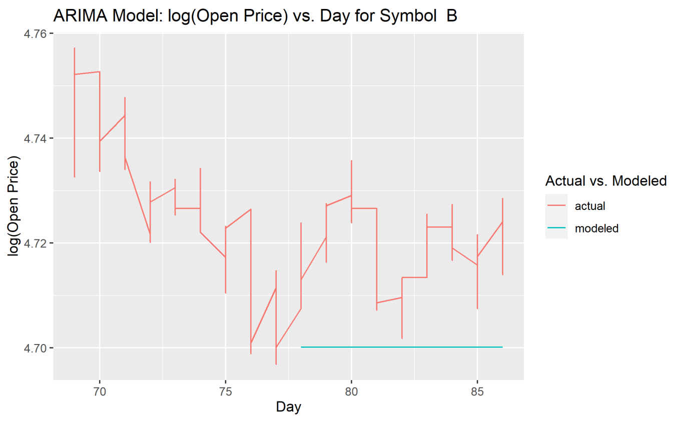
For ticker H, the lowest squared error on the development set is achieved with $p = 0, d = 2$, and $q = 2$. But these parameters still yielded predictions with a stairstepping pattern that did not look reasonable. After reviewing the squared errors and the plots for several combinations of parameters, we decided to go with $p = 0, d = 1$, and $q = 0$.

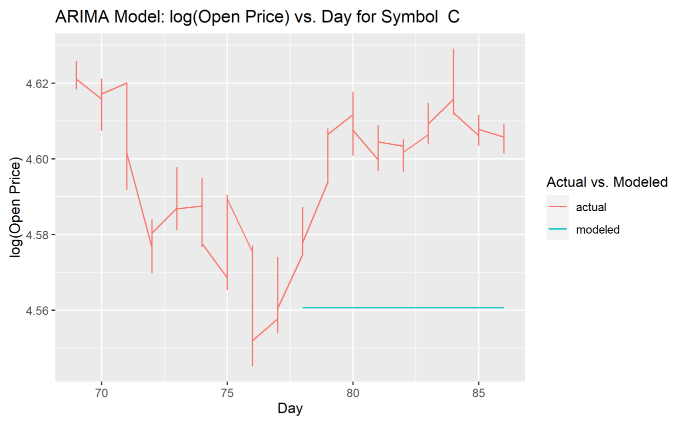
With these model changes made, we calculated the error on the test set with the ARIMA models. The revised parameters for G and H significantly reduced the error. For the first four days in the test set, the error is **213**, and for the last five days, the error is **287**. (While we modeled the log of the open prices instead of the prices themselves, these errors are in terms of raw prices.)

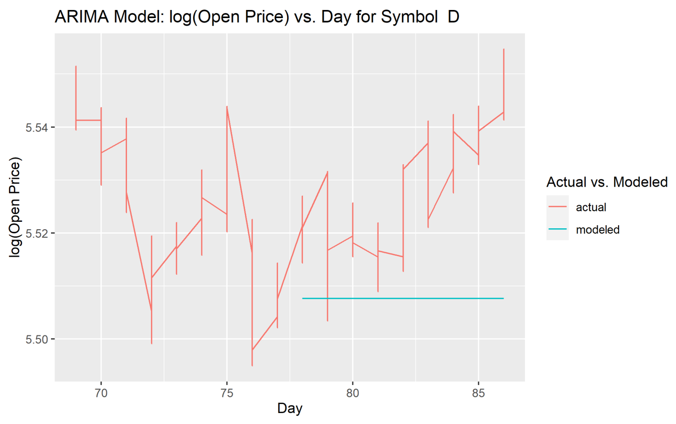
Even though the ARIMA model is more sophisticated than the simple moving average model described in Section 8.1 and contains optimal parameters selected by R's `auto.arima()` function, the total error on the ARIMA models is actually slightly higher than the total error on the simple moving average model. This suggests that it's hard to use past open prices to predict future stock prices.

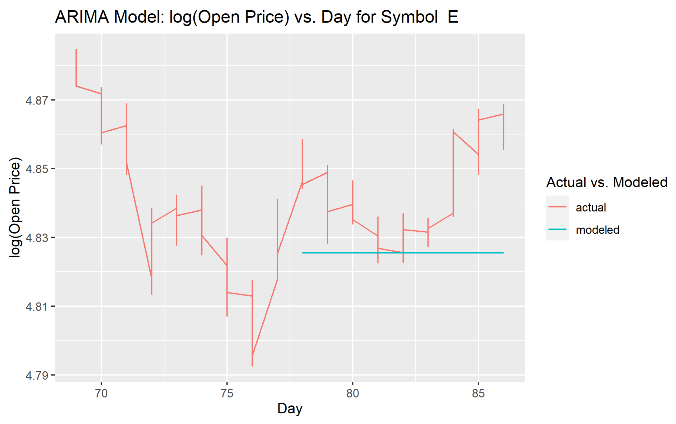
Plots of the predicted and actual open prices on the test set are shown below.

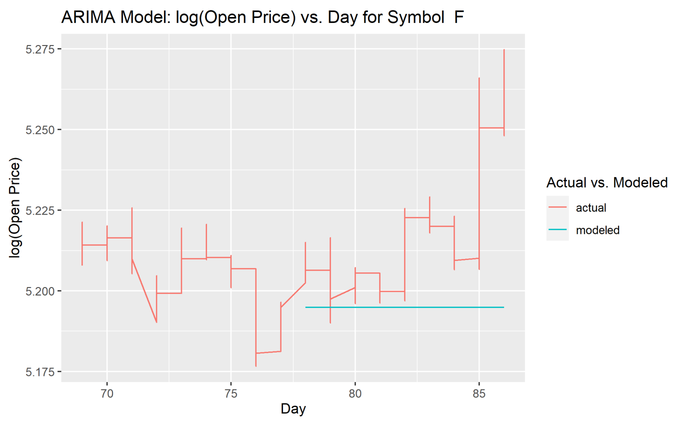


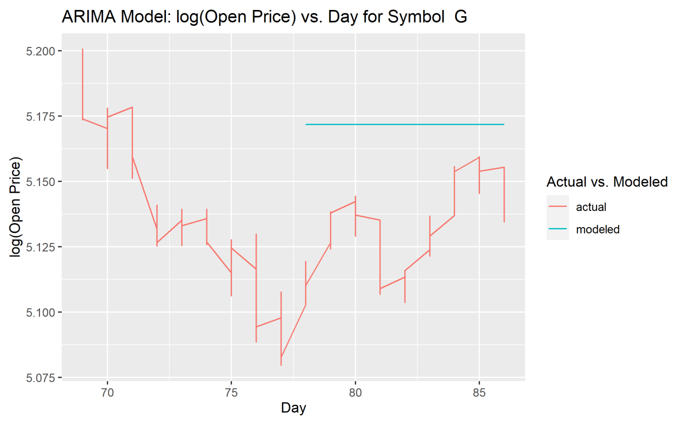


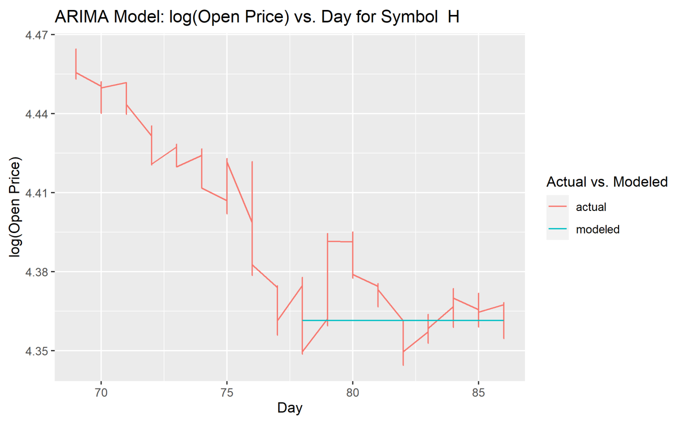

ARIMA Model: log(Open Price) vs. Day for Symbol C
log(Open Price)
4.62
4.60
4.58
4.56
70
75
80
85
Day
Actual vs. Modeled
actual
modeled

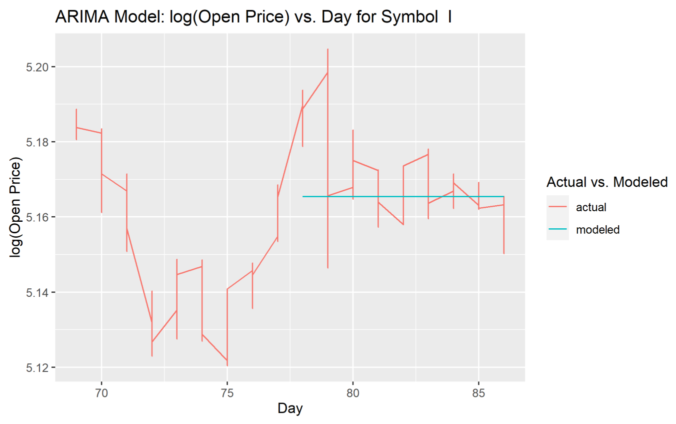

ARIMA Model: log(Open Price) vs. Day for Symbol D
log(Open Price)
5.54
5.52
5.50
70
75
80
85
Day
Actual vs. Modeled
actual
modeled

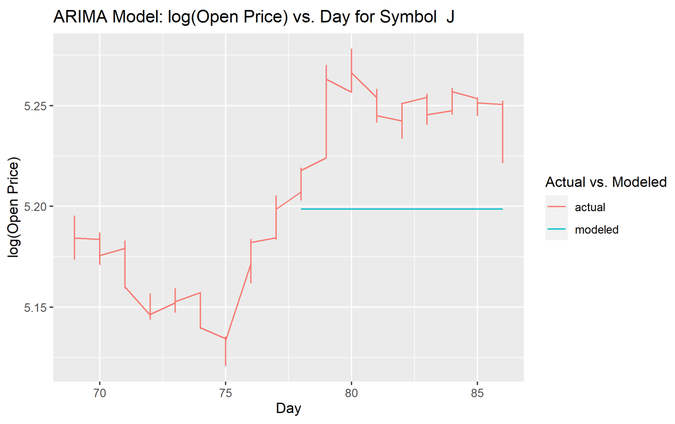

ARIMA Model: log(Open Price) vs. Day for Symbol E
log(Open Price)
4.87
4.85
4.83
4.81
4.79
70
75
80
85
Day
Actual vs. Modeled
actual
modeled


ARIMA Model: log(Open Price) vs. Day for Symbol F
log(Open Price)
5.275
5.250
5.225
5.200
5.175
70
75
80
85
Day
Actual vs. Modeled
actual
modeled


ARIMA Model: log(Open Price) vs. Day for Symbol G
log(Open Price)
5.200
5.175
5.150
5.125
5.100
5.075
70
75
80
85
Day
Actual vs. Modeled
actual
modeled


ARIMA Model: log(Open Price) vs. Day for Symbol H
log(Open Price)
4.47
4.44
4.41
4.38
4.35
70
75
80
85
Day
Actual vs. Modeled
actual
modeled

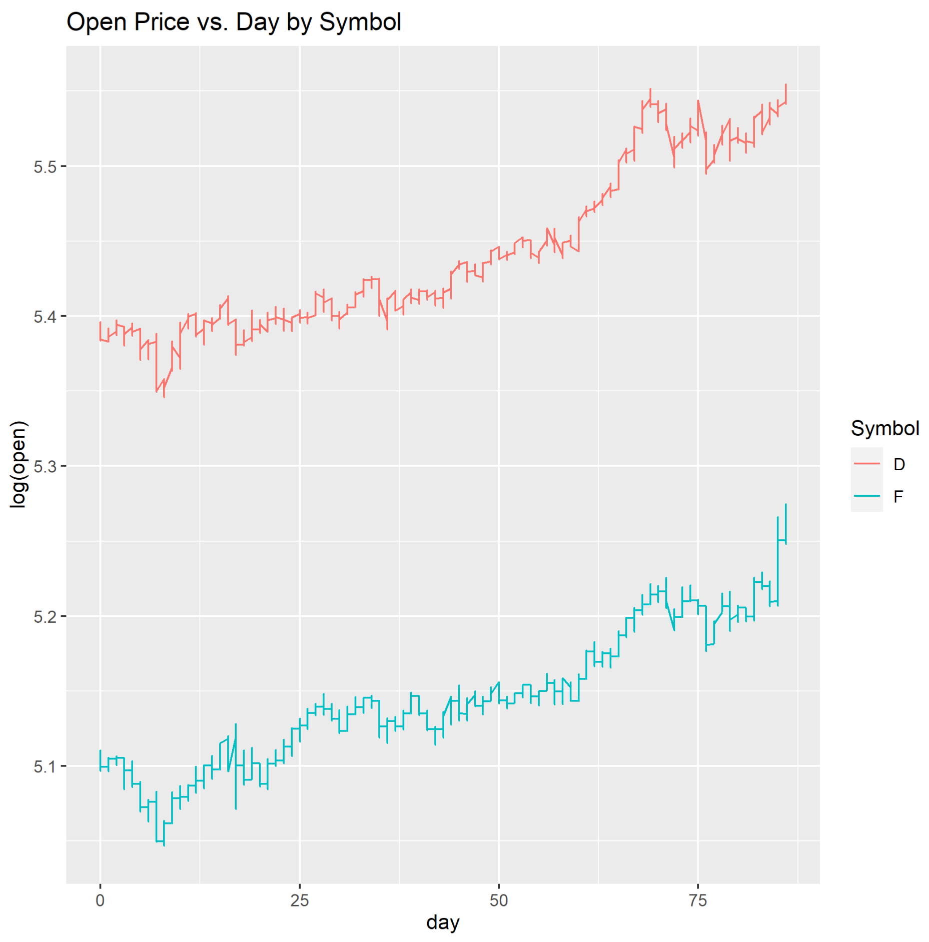




13.2.2 Making Forecasts

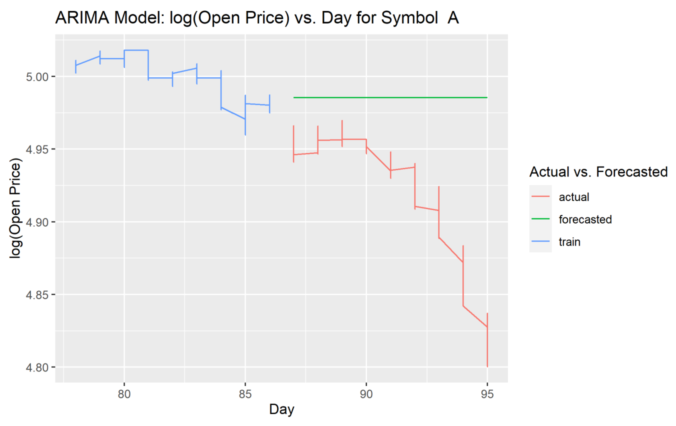
For all tickers except G and H, we used R's `auto.arima()` function on the full data set (training + development + test) to select parameters, then used the resulting model to forecast prices for nine days into the future. For tickers G and H, we used the parameters we selected by reviewing the performance on the development set.

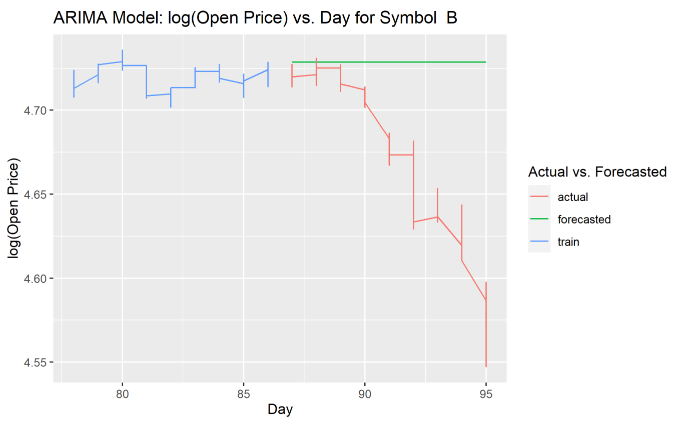
For two tickers, D and F, the `auto.arima()` function recommended using a model with drift. This doesn't seem unreasonable based on the graph of D and F's open prices over time, shown below.

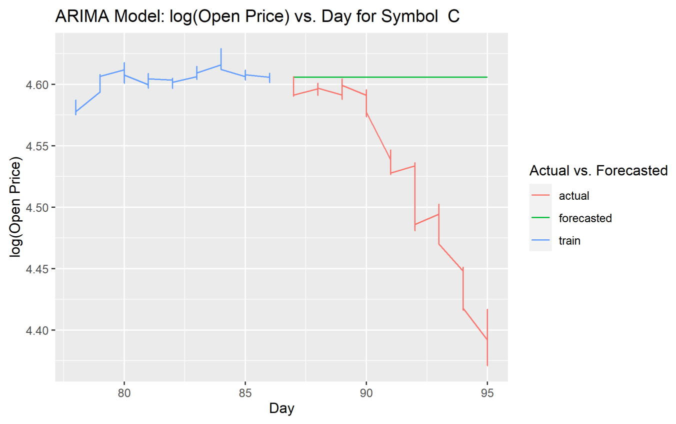


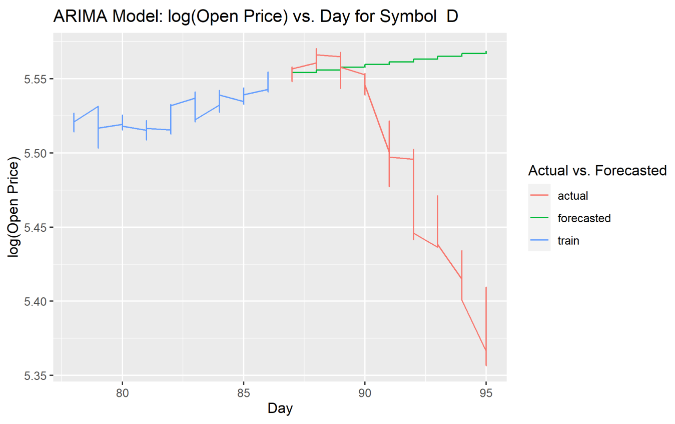
For these two stocks, we used the `Arima()` function in the `forecast` library to make predictions that include drift.

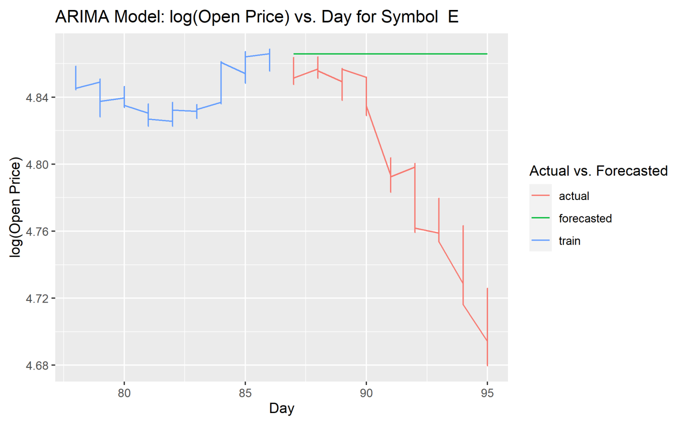
Plots of the forecasts are shown below, along with the actual prices.

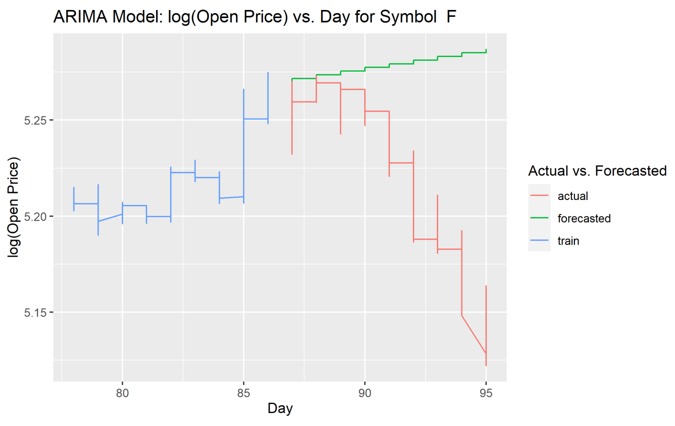


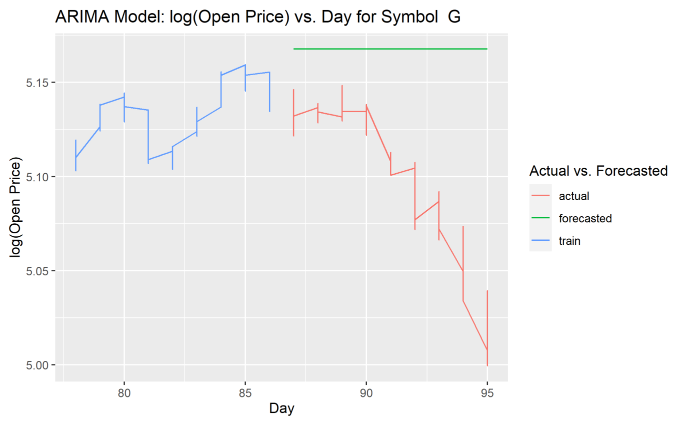


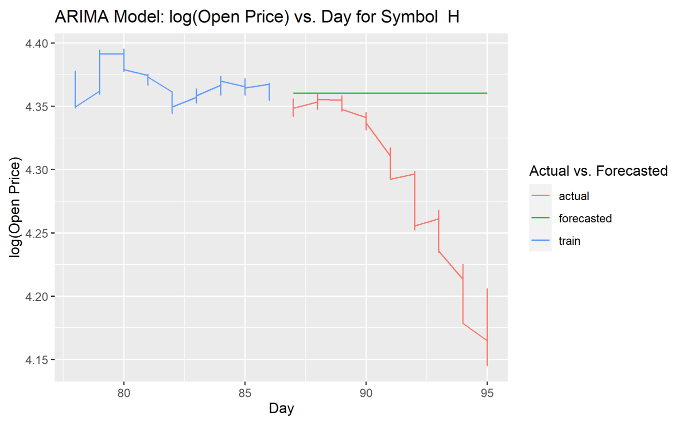


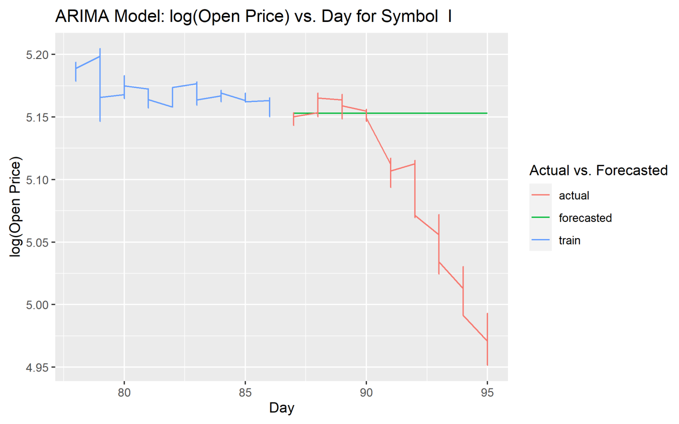


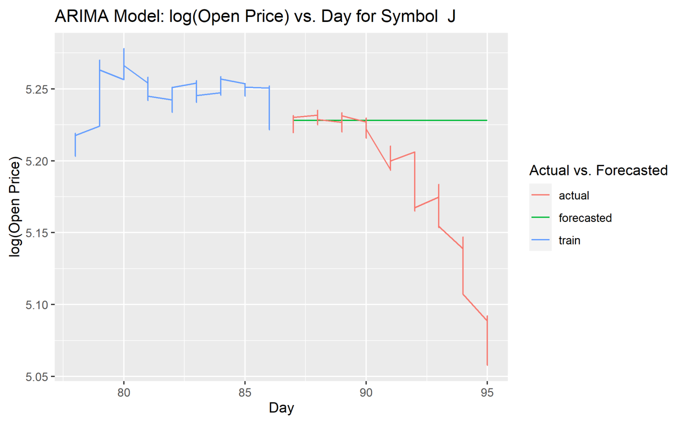












References

[1] 2020 stock market crash. 2021-07-13. wikipedia. https://en.wikipedia.org/wiki/2020_stock _market_crash.

[2] James Chen. Trading halt. *Investopedia*. 2021-06-13. https://www.investopedia.com/terms/t/tradinghalt.asp.

[3] mlofton. Can arima models be used to models stock prices? 2020-04-19. StackExchange. https://stats.stackexchange.com/questions/461424/can-arima-models-be-used-to-models-stock-prices.

[4] Kyle T. Rich. Stationarity testing. RPubs by RStudio. https://rpubs.com/richkt/269797.

[5] Unit root. 2021-08-14. Wikipedia. https://en.wikipedia.org/wiki/Unit_root.

[6] Kpss test. 2020-12-27. Wikipedia. https://en.wikipedia.org/wiki/KPSS_test.

[7] Sven S. R: Box.test vs adf.test vs kpss.test. 2016-05-26. StackExchange. https://stats.stackexchange.com/questions/214789/r-box-test-vs-adf-test-vs-kpss-test.

[8] Financial Crisis Inquiry Commission. The financial crisis inquiry report: Final report of the national commission on the causes of the financial and economic crisis in the united states. 2011-02-25. https://www.govinfo.gov/app/details/GPO-FCIC.

[9] Moving average. 2021-08-10. Wikipedia. https://en.wikipedia.org/wiki/Moving_average.

[10] Robert Nau. Statistical forecasting: notes on regression and time series analysis. 2020. https://people.duke.edu/ rnau/411home.htm.

[11] Prof J C Nash. optim fails when using arima. 2012-02-12. R-help – Main R Mailing List. https://stat.ethz.ch/pipermail/r-help/2015-February/425777.html.

[12] Vaidotas Zemlys. optim error in arima. 2006-09-28. R-help – Main R Mailing List. https://stat.ethz.ch/pipermail/r-help/2004-September/058074.html.

[13] SWIM S. Should my time series be stationary to use arima model. StackExchange. 2019-03-08. https://stats.stackexchange.com/questions/394796/should-my-time-series-be-stationary-to-use-arima-model.

[14] Sangarshanan. Time series forecasting — arima models. Towards Data Science. 2018-10-03. https://towardsdatascience.com/time-series-forecasting-arima-models-7f221e9eee06.

[15] Chris Haug. How to interpret arima(0,0,0). StackExchange. 2016-09-23. https://stats.stackexchange.com/questions/236566/how-to-interpret-arima0-0-0.