Final Project - Final Report

DeepTendies

Algorithmic Trading for the Masses using Deep Learning

ENEL 645 - Winter 2021
University of Calgary

Group 9:
Khaled Behairy
Stan Chen
Burak Gulseren
Lotfi Hasni
Mike Lasby

Introduction & Motivation

Stock markets play a vital role in global economic development, allowing companies to raise capital for business expansions and investors to profit on their stocks from their ownership. With the vast sums of wealth—$85 trillion in 2019 (Strader et al., 2017)—tied up in stocks, the ability to gain even a marginal advantage in one's investment strategies is highly significant. As such, there has always been considerable interest on the part of investment firms and individuals in leveraging data to inform their decision-making and improve returns. However, financial markets are notoriously unpredictable and highly difficult to forecast (Khan et al., 2020). According to proponents of the *efficient market theory*, this difficulty arises because stock market share prices represent all the relevant knowledge from the surrounding world (Pinheiro & Dras, 2017). Conversely, there are systemic proposals that demonstrate meticulous simulation and design of variables can lead to models that can quite effectively forecast future market prices and patterns in stock price movements (Mehtab et al., 2020a).

Where traditional quantitative analysis and statistical measures such as cumulative sum have proven insufficient, many have turned to machine learning (ML) algorithms and artificial intelligence (AI) to discern market patterns. According to a Statista survey, about 65% of respondents from financial organisations worldwide are either currently using or reviewing data science and ML strategies or are open to their implementation in the future (Statista, 2019). In 2012, it was reported that about 85 percent of transactions on American stock exchanges were carried out by algorithms (Nelson et al., 2017). Due to the tremendous quantities of data involved and their layered, complex interactions, deep learning solutions have been of particular interest lately.

The Long-Short-Term-Memory (LSTM) architecture was introduced in 1997 as an attempt to solve the exploding or vanishing gradient problem which afflicted other recurrent neural networks (Hochreiter & Schmidhuber, 1997). Although many variants have been created, featuring developments such as the introduction of peephole connections (Gers & Schmidhuber, 2000), bidirectionality (Graves & Schmidhuber, 2005), a gating unit activation function parameter (Doetsch et al., 2014), and the grid LSTM architecture (Kalchbrenner et al., 2015), the fundamental strengths of the vanilla LSTM model persist (Greff et al., 2017). With the cell state of each repeating module in the network's chain structure influenced by its input, output, and forget memory gate layers, the model is able to use a sequence's long-term history to drive future predictions (Olah, 2015). This is what makes LSTM networks useful for speech recognition, translation, and, most relevant to our needs, stock price prediction. There is a clear rising trend in this model's popularity (Li et al., 2019; Pinheiro & Dras, 2017) and so the motivation for this project is an interest in examining a stacked LSTM model's ability to use time series data to accurately predict stock prices in the short term. For comparison, we have also looked at the Gated Recurrent Unit (GRU) model, a newer, simpler, and more efficient offshoot of LSTM that has displayed better results in certain cases, such as with smaller datasets. This model has only two gates, update and reset, no memory unit and fewer parameters. To produce a baseline estimate for comparison, we also considered the ARIMA class of models, which use an equation with a combination of 3 parameters and a time series' own past values to forecast future results (this is discussed in greater detail later on).

While researching the history of quantitative trading and the present state of the stock market, our team has become increasingly interested in providing signals obtained from quantitative trading strategies to the general public. Since the advent of quantitative trading, large financial institutions and hedge funds have been the sole purveyors and consumers of quantitative trading strategies. The information age has

accelerated adoption of high-frequency, algorithmic trading strategies that are not available to the general public. Indeed, the Economist reported in 2019 that approximately “90% of equity-future trades and 80% of cash-equity trades are executed by algorithms without any human input” (The Economist, 2019). To ensure that the general public can remain competitive with Quantitative Hedge Funds, we intend to develop this project as an open source project available to anyone hoping to generate alpha using state of the art deep learning quantitative methods. The following outlines our methodology, results, discussion of findings, and future work to be completed.

Methodology

In the early stages of the project, we focused on data ingestion, feature engineering, and preprocessing. The second stage of the project entails establishing the workflow, acquiring exogenous data, and compiling metrics for reference. We have then invested more time and effort researching approaches and methods to further feature engineering and preprocessing, which is a critical step for long-running time series problems. Finally, we were able to use rolling window normalisation to reduce training time while increasing prediction performance.

Data Ingestion

We created wrapper functions that use FinnHub's Python API to obtain Open-High-Low-Close (OHLC) and exchange volume data for specific stocks. These functions allow us to conveniently question the API for every relevant US-based business over the last 25 years. The collected OHLC and exchange volume data has a regular resolution. Initially, we were looking at a number of other data sources as a base of information, but due to time constraints and data ingestion issues, we did not have enough time to execute them, so they have been moved to the future work section.

Feature Engineering

Hong and Yu (2009) showed that seasonality is significantly correlated to trade volume and volatility. Therefore, we have engineered a variety of temporal features such as day of week, day of year, financial quarter, and if a given day is at the end of the quarter. These features may be particularly important for those companies who experience strong seasonal trends, such as retail companies who report a large percentage of their annual earnings around the Christmas season.

We have also engineered a variety of traditional quantitative analysis metrics such as Weighted Moving Average (WMA) and Volume Weighted Average Price (VWAP). These metrics are calculated as follows:

$$WMA_n = \frac{\sum_{i=0}^{n} P_i * (n-i)}{n*(n+1)/2}$$

Where n is the lag period to calculate the average for and P_i is the average stock price for the i_{th} day, with i_0 being the current day. WMA is a common quantitative metric which may convey significant buy or sell signals. In particular, identification of crossover between WMAs with different lag periods. For example, the so-called “golden cross” occurs when a relatively short-term WMA crosses above a longer term WMA. This indicates a bullish breakout pattern and is usually considered as a strong buy signal (Chen, 2020).

$$P_{avg} = \frac{P_h + P_c + P_o}{3}$$

$$VWAP = \frac{\sum_{i=0}^{n} P_{i_{avg}} * V_i}{\sum_{i=0}^{n} V_i}$$

Where P_{avg}, P_h, P_c, P_o are the daily average, high, close, and open prices for a given stock, respectively, and V_i is the volume sold on a given day. VWAP is typically calculated on a daily basis using intraday tick data; however, since we are focused on daily OHLC data at this point, we will use a moving VWAP similar to WMA. Moving WVAPs are calculated using a sliding window of fixed lag size and can be calculated between any two time periods to provide insight into the relative price trend and trading momentum.

We also created several features to calculate the next high and low price of the stock over a variety of periods. For example, the high and low price in the next 5 days. These features can be generally described as follows:

$$NextHigh_n = max_{1 \le i \le n} (t_{i-high})$$

$$NextLow_n = min_{1 \le i \le n} (t_{i-low})$$

The $NextHigh_n$ and $NextLow_n$ features were used as targets for two (2) of the three (3) models trained for each stock ticker. The intent of these models is to provide more actionable insights in comparison with the model predicting tomorrow's closing price. For example, a potentially profitable entry and exit trading strategy employing these features is to place a buy order at the $NextLow_n$ and a sell order at the $NextHigh_n$. We decided to focus our efforts predicting the values of each of these features at n=5 so we could compare the generalisability of the model on different financial securities.

Preprocessing

Financial data is well suited to machine learning problems as the majority of the data is inherently numerical. We had two primary data types in our input data, continuous numerical data and categorical data. The categorical data is derived from calendar features such as day of the week, day of the year, and whether a given day is the end of a financial quarter. For these categorical features, we used a label encoding strategy rather than one hot encoding to avoid creating a high dimension, spare feature matrices.

Due to the unique normalization challenges that arise when considering the highly volatile and non-stationary nature of stock prices, we implemented a custom data generator class, `WindowNormTimeseriesGenerator`, which inherited from the Keras' `TimeSeriesGenerator` class. This class was responsible for creating the rolling window input data and corresponding target values. Based on the parameters passed to the constructor, the data generator applied a dynamic normalization scheme using a min/max or standard scaler to transform the input data based on each lag window rather than for the dataset as a whole. These scalers were also used to transform the target value. Each scaler was appended to a list of scalers corresponding to the window which they were applied to. This list of scalers could then be used to inverse transform the model

inferences back into the original dollar space. We used this methodology to normalize all numeric features in our feature matrix.

To prepare our $NextHigh_n$ and $NextLow_n$ features, we added an additional functionality to the `WindowNormTimeseriesGenerator` class to mask the last n values from the training dataset. This step was critical to prevent data leakage into the training set. This was accomplished by repeating the $n\text{-}1^{th}$ value in the last n values. For example, for predicting the high price in the next five (5) days, we masked the last five (5) values in the target column by repeating the value in the 94th position in the 95th to 100th rows. This prevented our model from training using data that would not be available to the model in production.

Once the features have been cleaned, normalized, and masked as required, we produced train and test data generators using a 70%/30% split. These generators were used to train and evaluate our model's results.

Hyper parameter optimization

Our approach to hyperparameter tuning was largely ad hoc in nature. So, although we did explore both AutoML and Keras hypetune, hyperparameter selection was accomplished manually. This was mainly due to difficulties harmonizing existing techniques with our time series-based task. We experimented with a number of different parameters such as dropout, noise, batch size, patience, learning rate, and decay rate values, as well as with loss functions, numbers of cell units, statefulness, use of batch normalization, scalers, activation functions, and optimizers. Potential values were chosen based on known typical ranges. We chose a standard trailing window size of 100 days as a reasonable balance between our very short-term forecasting needs and the desire to use as much data as possible for interim predictions. Due to the structure of the problem, online learning (batch size of 1) was ultimately used in all cases. This caused weights to be updated after each sample, slowing training, but allowed us to make all of the necessary predictions for each sample. To maintain consistency, the hyperparameters were determined for each model using the AAPL stock data. The best hyperparameter values found for the models used are included in the Model Architectures section below.

ARIMA Model

To gauge the quality of our deep learning models, we implemented a Auto Regressive Integrated Moving Average (ARIMA) model to establish a baseline. An ARIMA may be described as follows:

ARIMA is a model class that extracts a set of normal temporal structures in time series data. It is a more complex version of the simplified Auto Regressive Moving Average that incorporates the idea of convergence. In research and forecasting, the ARIMA model, also known as the Box-Jenkins model or approach, is widely used (Adebiyi et al., 2014). It is generally considered as one of the most effective forecasting methods in social sciences, and therefore it is widely used for time series forecasting (Adebiyi et al., 2014).

Improving an ARIMA model's efficiency will indicate that our deep learning architectures are performing above the defined baseline. In order to find the best performing ARIMA parameter combination, we ran a basic grid search and took the smallest AIC producing parameter combination.

That gave us the ARIMA order of 1,1,1 and seasonal order of 1,1,0,12. We used this parameter combination to produce a baseline on predicting Apple's tomorrow closing price.

Machine Learning Systems Architecture (MLOps)

At our current stage of the project, we have developed several components of an envisioned machine learning systems architecture with automated deployment (MLOps). By Google's definition of MLOps definition, our work in this final project reflects level 0 MLOps.

MLOps level 0 is considered the basic level of maturity, or level 0. Every step is manual, including data analysis, data preparation, model training, and validation. This process is usually driven by experimental code that is written and executed in notebooks. For example, our current processes are:

- **Data extraction:** We pick and integrate appropriate data from different data sources for the ML project.
- **Data Preparation**: We performed feature engineering, then we separated the data into preparation, validation, and test sets. The data extraction and preparation functionality was wrapped into our `StockData` class. By using this architecture, we were able to ensure that the preprocessing pipeline would remain consistent amongst our various experiments. If we should decide to deploy this model, this class will be instrumental in preparing new data for the model to use for inference.
- **Model validation:** The model is validated to find the model and pipeline best suitable for our business logic and use case.
- **Model training pipeline:** Automated model training is an important component of an overall MLOps architecture. In this case, we used a wrapper class, `Trainer,` to manually train our models. This class was created to streamline the model training, production of predictions, and graphing of loss vs. epochs steps. As these steps must be repeated for each model developed, we created this class so that we can have objects representing each model that can store the necessary attributes which may be used at later steps for further evaluation.
- **Model evaluation and deployment**: Model evaluation was completed manually for this project. However, we did implement some helper classes and functions to easily obtain relevant result metrics and plots in the `ModelMetrics` and `ModelResultsPlots` classes. We hope to implement automated model evaluation and validation as future work.

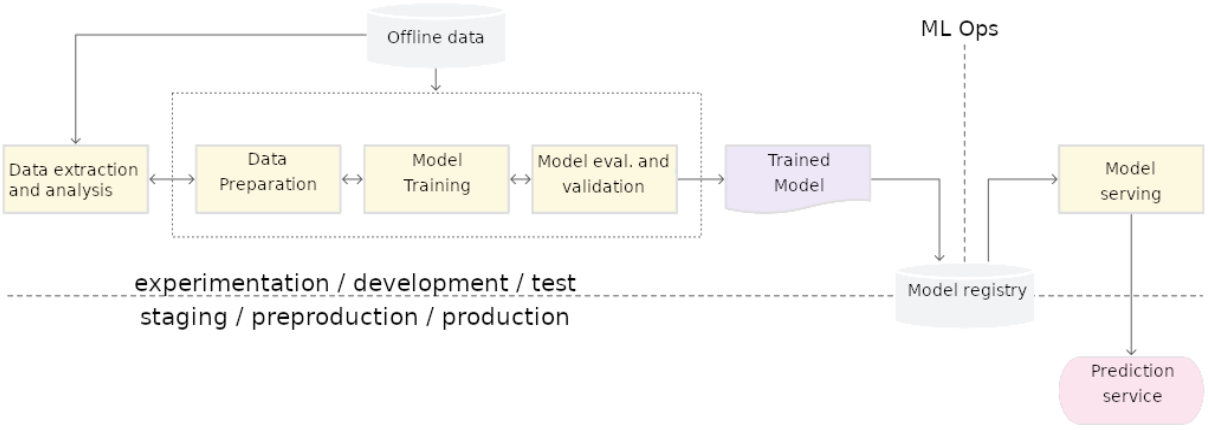


Fig 1.: Google's Definition of MLOps Level 0: Manual process

Hyperparameter Tuning

Model Architectures

Table 1.: Model Architectures used for Hyper Parameter Tuning

Model Name	Summary of Input data / Features	Summary of layers	Hyperparameters
LSTM-MM	• 100 day trailing window with standard scaler dynamic normalization • OHLC daily data • 20, 50, and 100 day weighted moving averages • 20, 50, and 100 day volume weighted average price • Trade volume data	• 20 Node LSTM with return sequences, return state, and stateful all equal to True • 20 Node LSTM with return sequences, return state, and stateful all equal to False • Single node dense layer	• Batch Size: 1 • Activation Function: Linear • Optimizer: Adam • Loss Function: MSE • Initial Learning Rate: 1E-2 • Patience: 5 • Learning Rate Decay: Halved every 10 epochs
LSTM-S	• 100 day trailing window with min/max dynamic normalization • OHLC daily data • 20, 50, and 100 day weighted moving averages • 20, 50, and 100 day volume weighted average price • Trade volume data	• 20 Node LSTM with return sequences, return state, and stateful all equal to True • 20 Node LSTM with return sequences, return state, and stateful all equal to False • Single node dense layer	• Batch Size: 1 • Activation Function: Linear • Optimizer: Adam • Loss Function: MSE • Initial Learning Rate: 1E-2 • Patience: 5 • Learning Rate Decay: Halved every 10 epochs
LSTM-100	• 100 day trailing window with min/max dynamic normalization • OHLC daily data • 20, 50, and 100 day weighted moving averages • 20, 50, and 100 day volume weighted average price • Trade volume data	• 100 Node LSTM with return sequences = False, return state = False,, and stateful = True • Single node dense layer	• Batch Size: 1 • Activation Function: Linear • Optimizer: Adam • Loss Function: MSE • Initial Learning Rate: 1E-2 • Patience: 5 • Learning Rate Decay: Halved every 10 epochs
AK-TS-SC	• OHLC daily data • Unable to use our custom time-series generator • Using Autokeras's built-in normalization instead of our custom rolling window normalization	• TimeseriesBlock, tunable, LSTM • RegressionHead, tunable • TimeseriesInput, lookback = 32	• Lookback: 100 • Batchsize: 32 • Predict_from = 1 • Predict_untill=1

LSTM-KB	• 100 day trailing window with min/max dynamic normalization • OHLC daily data • 20, 50, and 100 day weighted moving averages • 20, 50, and 100 day volume weighted average price • Trade volume data	• 10 Node LSTM with return sequences, return state, and stateful all equal to True • 10 Node LSTM with return sequences, return state and stateful all equal to False • Single node dense layer	• Batch Size: 1 • Activation Function: Linear • Loss Function: MSE • Initial Learning Rage: 1E-2 • Learning Rate Decay: Halved every 10 epochs • Optimizer: SGD • Momentum: 0
GRU-LH	• 100 day trailing window with min/max scaler • Time specific data (date, day of week/year, quarter end) • OHLC and volume data • 100, 50, and 20 weighted moving average and volume weighted average price	• Input with dynamic windowed normalization • Gaussian noise layer (0.1) • Two 20 Node Bidirectional GRU layers with return sequences and return state equal to True • One 20 Node Bidirectional GRU layer with return sequences equal to False • Single-node dense layer	• Batch Size: 1 • Activation Function: Linear • Optimizer: RMSprop • Loss: MSE • Initial LR: 1E-2 • Patience: 10 • Learning Rate Decay: Halved every 10 epochs
ARIMA Baseline	Single input of daily closing prices of 100 day trailing window	N/A	• Seasonal ARIMA parameter order order=(1, 1, 1) seasonal_order=(1, 1, 0, 12)

Hyperparameter tuning results

Table 2.: Hyperparameter tuning results

Model Name	Root Mean Squared Error ($)	Mean Absolute Error ($)
LSTM-MM	**1.470**	**0.845**
LSTM-S	1.531	0.911
LSTM-100	1.497	0.864
AK-TS-SC	350.79	236.89
GRU-LH	2.38	1.44
LSTM-KB	3.88	2.57
ARIMA - Apple	**1.16**	**0.59**

Based on the above, we selected the LSTM-MM Model architecture to predict the other stock tickers selected. As we can see above, the ARIMA model actually performed better than any of our deep learning models. This will be discussed in more detail below.

Results

Below is the summary of each model's results predicting tomorrow's closing price, the Next 5 Day High, and Next 5 Day Low of a variety of stocks. We considered the root mean squared error (RMSE) and mean absolute error (MAE) to analyze our results as both of these metrics retain the original units of the predicted quantity, dollars.

Table 3.: Best Model's Prediction Results on Different Stocks:

Stock Name	Closing	Next 5 Day High	Next 5 Day Low
AAPL (Apple)	RMSE: 1.97 MAE: 1.18	RMSE: 2.63 MAE: 1.59	RMSE: 3.36 MAE: 1.96
MSFT (Microsoft)	RMSE: 3.12 MAE: 1.95	RMSE: 4.01 MAE: 2.53	RMSE: 4.27 MAE: 2.78
BA (Boeing)	RMSE: 10.08 MAE: 5.88	RMSE: 14.04 MAE: 9.05	RMSE: 14.51 MAE: 8.98
TSLA (Tesla)	RMSE: 29.74 MAE: 13.93	RMSE: 38.85 MAE: 21.57	RMSE: 42.04 MAE: 21.73
KO (Coca-Cola)	**RMSE: 0.783 MAE: 0.536**	**RMSE: 2.145 MAE: 0.907**	**RMSE: 2.195 MAE: 0.852**

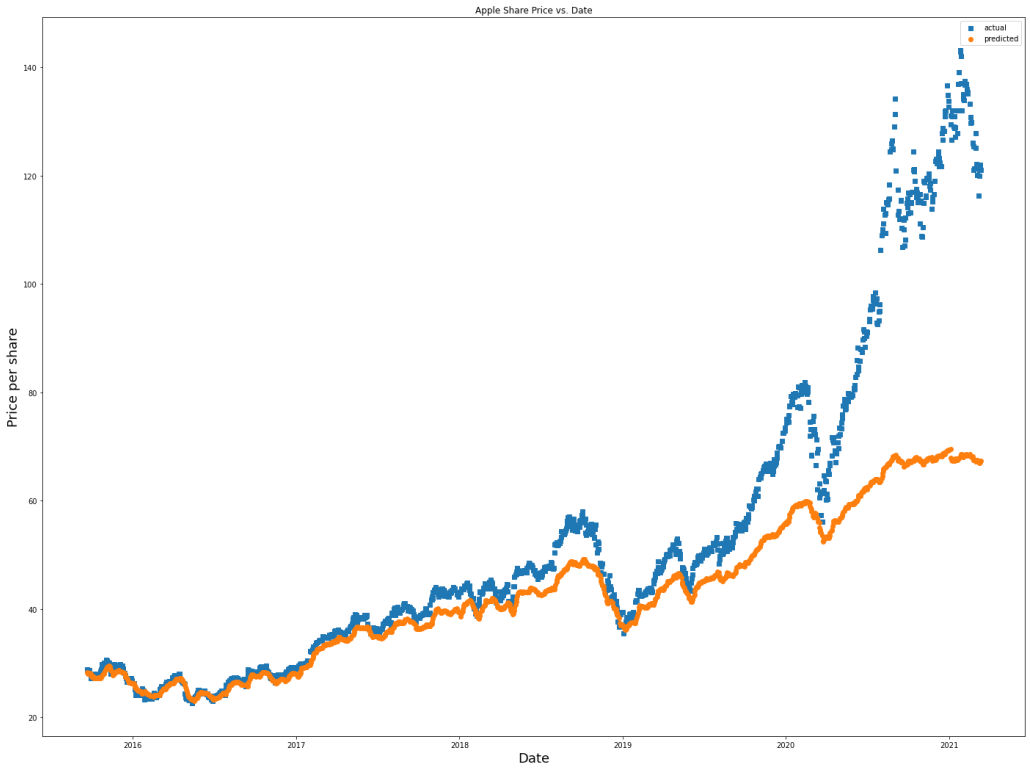


Fig 2. - Tomorrow's Closing Stock Price vs. Time for Apple from 2016 to 2021 from Midterm

The midterm results predicting tomorrow's closing price are repeated above to facilitate ease of comparison with our final plots below.

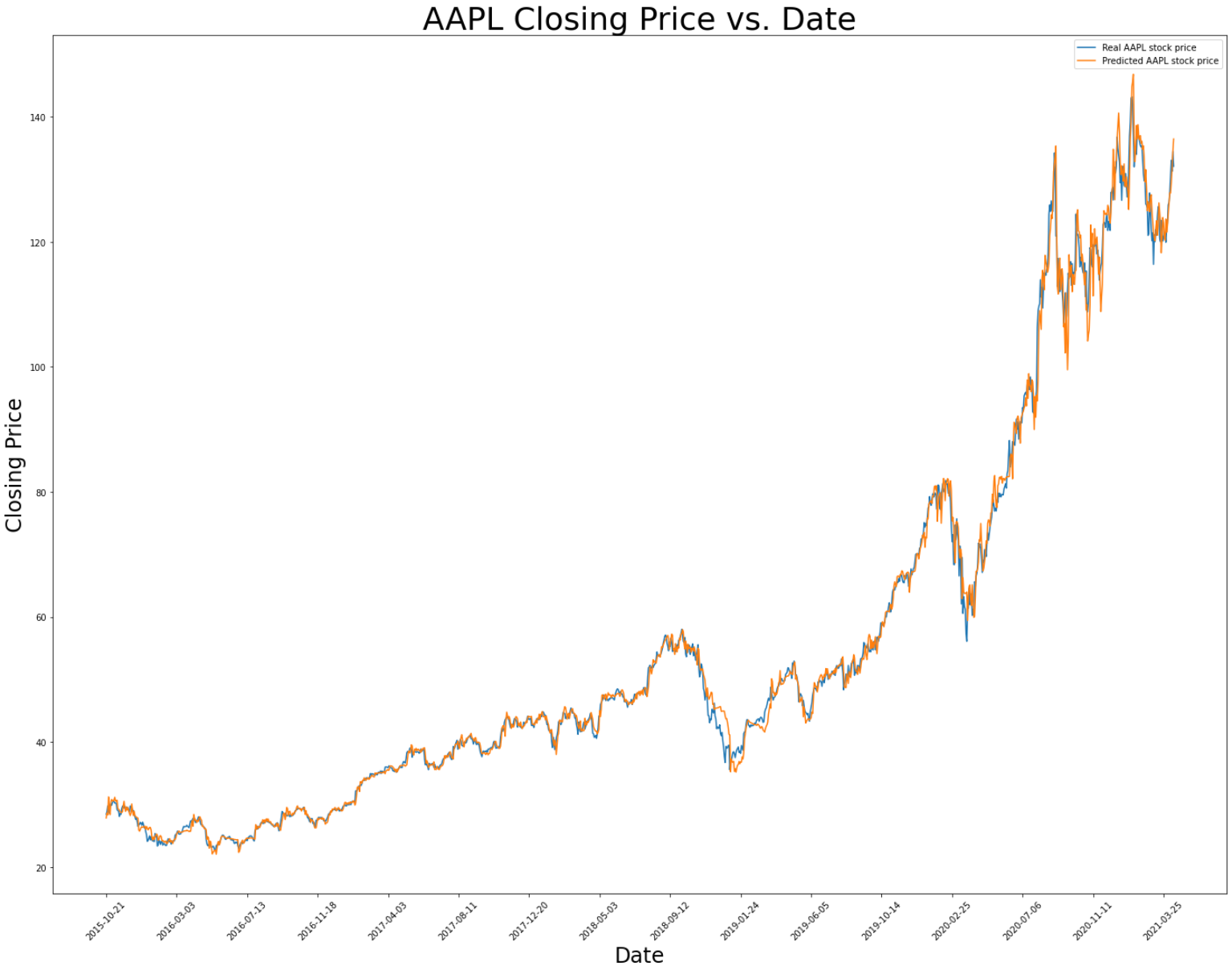


Fig 3. - Tomorrow's Closing Stock Price vs. Time for Apple from 2016 to 2021 from Final Results

Fig 4. - Next 5 Day High Stock Price vs. Time for Apple from 2016 to 2021 from Final Results

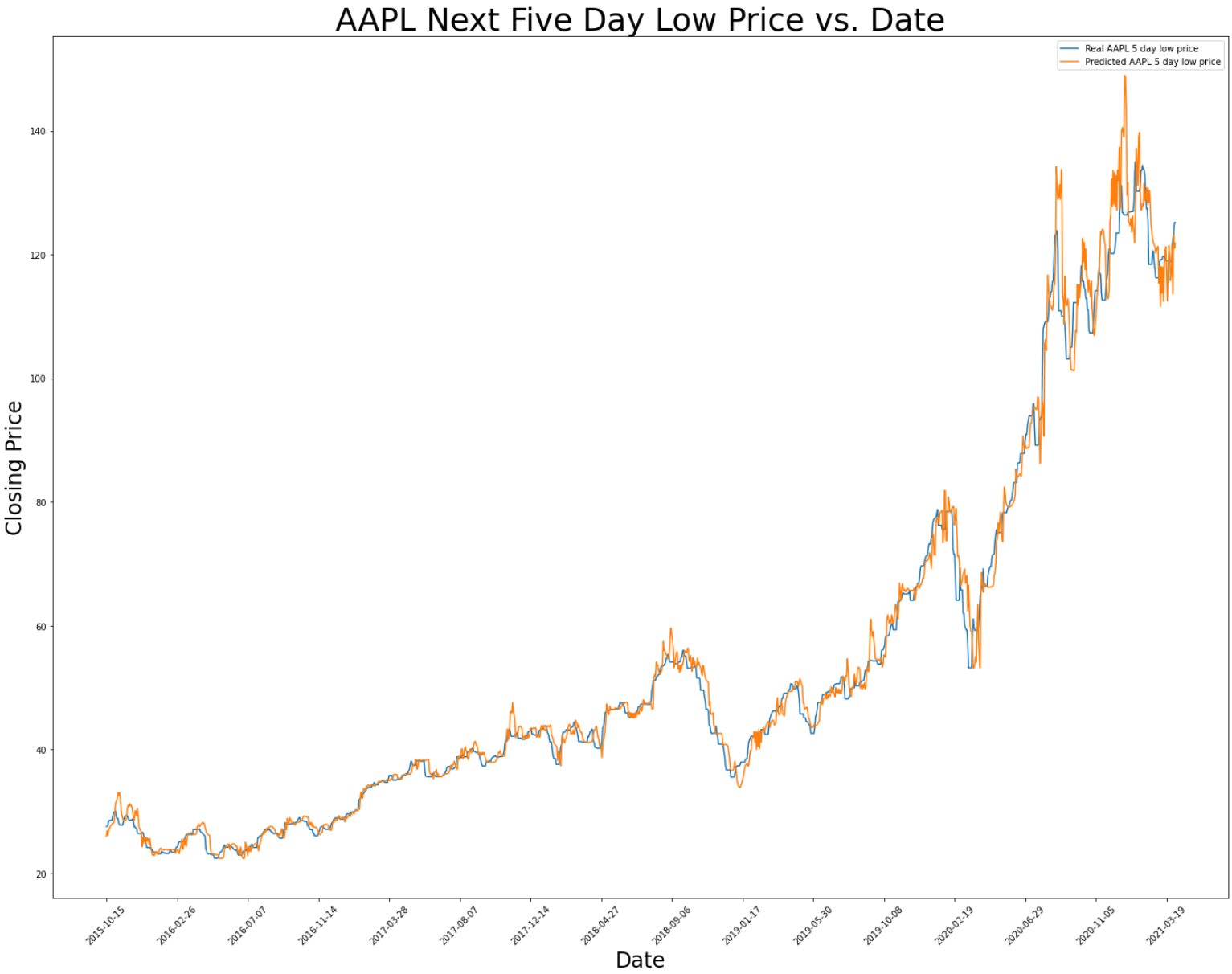


Fig 5. - Next 5 Day Low Stock Price vs. Time for Apple from 2016 to 2021 from Final Results

Please see Appendix A for additional results.

Discussion

Our best performing model was on the Coca-Cola stock; however, the Apple results are presented above to compare with our midterm results. Regardless, when we re-ran the ARIMA model on the Coca-Cola we obtained a RMSE and MAE of $0.504 and $0.276, respectively, outperforming our best LSTM model.

Unfortunately, we were unable to develop a deep learning model that outperformed an ARIMA model predicting tomorrow's closing prices. There may be a number of reasons for this, for example, the calendar features we are using may actually be adding noise to the model pipeline rather than improving the overall quality of inference. However, the space of all possible model architectures and hyperparameter combinations is large, so we are confident with more experimentation we will be able to outperform the ARIMA model.

One of the key challenges we encountered which prohibited the amount of experimentation was the time required to train the models. The typical LSTM-MM model took approximately three (3) hours to train for 100 epochs on a Google Collab TPU using Pro accounts. Therefore, for the 15 final models trained we required 45 total hours of compute. The primary reason that the models took so long to train is our use of *online learning*, using a batch size of one (1). This meant that the LSTM network would update the weights and bias terms within the network after each training sample. Online training has some specific benefits for time series sequence modelling as it is capable of adapting to drifting data where the mean is evolving through time. However, online learning is slower than batch learning, so we were unable to run as many experiments as we may have liked.

Another drawback of online learning is it may lead to instability in the backpropagation process, causing the model's weights to vary widely from sample to sample. We may have been directly affected by this instability while experimenting, as we were unable to reproduce our best hyperparameter tuning model's results in our final notebooks despite using the same architecture and hyperparameters. We can also see some instability in the NextHigh_n and NextLow_n predictions, particularly towards the end of the test period for AAPL, where the stock reached all times high that were several times the average of the stock price during the training period.

In Table 3, we can see that the selected model architecture performed reasonably well on the Apple stock data; however, much worse on the other stocks except for Coca-Cola. This suggests that each stock may need to have its own hyperparameter tuning and model selection process completed to obtain reasonable overall results. It is our opinion that AutoKeras and similar AutoML solutions may be the best approach to rapidly and frequently tune models for each stock of interest with relatively little human intervention. However, time series modelling in these libraries remains in early development. As such, some of our group members intend to become active contributors in the AutoKeras project to spur further development in this area.

Regardless of the above, we still view this project as a success. In comparing our midterm to our final results, it is clear that we greatly improved on our initial attempts. This project was the first time our group members had ever worked with financial securities data, so the learning curve was quite large as this is a very well studied and advanced area of research.

As noted in the introduction, we intend to continue development on this project as an open source, or at least, a crowd sourced project. While it may be counterintuitive to open source a project intended to generate alpha, it is our hope that we may be able to provide investment strategies to the public that level the playing field when competing with quantitative hedge funds. Should we stumble upon a truly innovative and effective approach, we may need to revise our stance in this regard.

Future Work

Much of the future work in this project would entail expanding the features used in order to improve predictive performance and generalizability. As a further feature engineering task, we would like to look at the differences between WMAs with different window periods to explicitly identify “crosses”. Exogenous data is data that is not directly tied to stock signals or indicators. This includes overall economic indicators, market indices, Federal Reserve System announcements, search engine trends, etc. If properly engineered and implemented by data pipelines, this data may be another valuable source of knowledge for our models to learn.

We would also like to use news releases for sentiment analysis. To effectively employ trading strategies based on stock news releases, it is important to be able to analyze the possible effects on stock performance quickly as news travels rapidly online. Utilizing Natural Language Processing, we can scrape news releases and headlines for a specified stock or set of stocks, analyze it and evaluate whether it is positive, neutral, or negative news. Additionally, we can look at other features such as popularity of a certain stock based on the number of news headlines and predicting the polarity of news releases. Due to the unpredictability of stock prices, exemplified by the “pump and dump” phenomena, we would like to further investigate the effects of introducing artificial noise to prevent overfitting to fluctuations. We have experimented with Gaussian noise layers in our models, but they appeared to have little effect so far. It is our hope that a more sophisticated look at noise injection would be beneficial.

One way we would like to expand on our existing work is by utilizing AutoKeras’ nascent time series forecasting in order to identify the optimal model to predict stock performance. We would also like to further explore creating grid search functionalities and using AutoML.

Most importantly, we would like to expand the DeepTendies Open Source Project Team. In general, there are different strategies to be profitable with the stock market. With more developers contributing to the project, we can achieve more models that can tackle multiple types of trading styles. This can include day trading, swing trading, trend following, reinforcing learning, and long term investments.

Conclusion

We have greatly improved our original results in a relatively short amount of time and we are looking forward to continuing development of the DeepTendies project. We would like to thank Dr. Souza for his mentorship throughout this project and encouragement in tackling the more challenging parts of the data preprocessing involved in this project.

Sincerely,

The DeepTendies Team

Bibliography

Adebiyi, A. A., Adewumi, A. O., & Ayo, C. K. (2014, March 5). Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction. *Journal of Applied Mathematics, 2014*(Article ID 614342), 7. Open Access. 10.1155/2014/614342

Balaji, A. J., Ram, D.S. H., & Nair, B. B. (2018). Applicability of Deep Learning Models for Stock Price Forecasting An Empirical Study on BANKEX Data. *Procedia Computer Science, 143*, 947-953.

Chen, J. (2020, October 7). *Crossover*. Investopedia. Retrieved March 13, 2021, from https://www.investopedia.com/terms/c/crossover.asp

Das, S., Behera, R. K., Kumar, M., & Rath, S. K. (2018). Real-Time Sentiment Analysis of Twitter Streaming data for Stock Prediction. *Procedia Computer Science*, *132*, 956-964. https://doi.org/10.1016/j.procs.2018.05.111

Doetsch, P., Kozielski, M., & Ney, H. (2014). Fast and robust training of recurrent neural networks for offline handwriting recognition. *14th International Conference on Frontiers in Handwriting Recognition*.

The Economist. (2019, October 5). The stockmarket is now run by computers, algorithms and passive managers. *The Economist*. https://www.economist.com/briefing/2019/10/05/the-stockmarket-is-now-run-by-computers-algorithms-and-passive-managers

Gers, F. A., & Schmidhuber, J. (2000). Recurrent nets that time and count. *Neural Networks, 3*, 189-194.

Google. (2020, 11 16). *MLOps level 0: Manual process*. Google Cloud. https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning#devops_versus_mlops

Gopalakrishnan, E. A., Menon, V. K., & Kp, S. (2018). NSE Stock Market Prediction Using Deep-Learning Models. *Procedia Computer Science, 132*, 1351-1362. 10.1016/j.procs.2018.05.050

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks, 18*(5-6), 602-610.

Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems, 28*(10), 2222-2232.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*(8), 1735-1780.

Hong, H., & Yu, J. (2009, November). Gone fishin': Seasonality in trading activity and asset prices. *Journal of Financial Markets, 12*(4), 672-702.

Kalchbrenner, N., Danihelka, I., & Graves, A. (2015). Grid long short-term memory. *CoRR, abs/1507.01526*. http://arxiv.org/abs/1507.01526

Khan, W., Ghazanfar, M. a., Azam, M. A., & Karami, A. (2020). Stock market prediction using machine learning classifiers and social media, news. *Journal of Ambient Intelligence and Humanized Computing*, *11*(3).

Li, X., Li, Y., Yang, H., Yang, L., & Liu, X.-Y. (2019). DP-LSTM: Differential Privacy-inspired LSTM for Stock Prediction Using Financial News. *ArXiv*. https://arxiv.org/abs/1912.10806

Lubitz, M. (2017). *Who drives the market ? Sentiment analysis of financial news posted on Reddit and Financial Times*. Faculty of Engineering, University of Freiburg. https://ad-publications.cs.uni-freiburg.de/theses/Bachelor_Michael_Lubitz_2018.pdf

Mehtab, S., & Sen, J. (2020). *Stock Price Prediction Using Convolutional Neural Networks on a Multivariate Timeseries*. ArXiv. https://arxiv.org/abs/2001.09769

Mehtab, S., Sen, J., & Dutta, A. (2020, October). Stock Price Prediction Using Machine Learning and LSTM-Based Deep Learning Models. *Second International Symposium on Machine Learning and Metaheuristics Algorithms*.

Mittal, A., & Goel, A. (2011). *Stock Prediction Using Twitter Sentiment Analysis*. Stanford CS229. http://cs229.stanford.edu/proj2011/GoelMittal-StockMarketPredictionUsingTwitterSentimentAnalysis.pdf

Nelson, D. M.Q., de Oliveira, R. A., & Pereira, A. C.M. (2017). Stock market's price movement prediction with LSTM neural networks. *2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK*, 1419-1426. 10.1109/IJCNN.2017.7966019

Olah, C. (2015). *Understanding LSTM Networks*. colah's blog. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2019, December 18). Deep Adaptive Input Normalization for Time Series Forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(9), 3760-3765. 10.1109/TNNLS.2019.2944933

Pinheiro, L. d. S., & Dras, M. (2017). Stock Market Prediction with Deep Learning: A Character-based Neural Language Model for Event-based Trading. *Proceedings of the Australasian Language Technology Association Workshop 2017*, 6-15. https://www.aclweb.org/anthology/U17-1001

Statista. (2019, September 30). *Deployment status of data science and machine learning in organizations worldwide as of 2019, by function*. Statista Estimates. Retrieved 2 15, 2021, from http://www.statista.com/statistics/1053561/data-science-machine-learning-deployment-by-function/

Strader, T. J., Rozycki, J. J., Root, T. H., & Huang, Y.-H. (. (2017). Machine Learning Stock Market Prediction Studies: Review and Research Directions. *Journal of International Technology and Information Management*, *28*(4), 63-83.

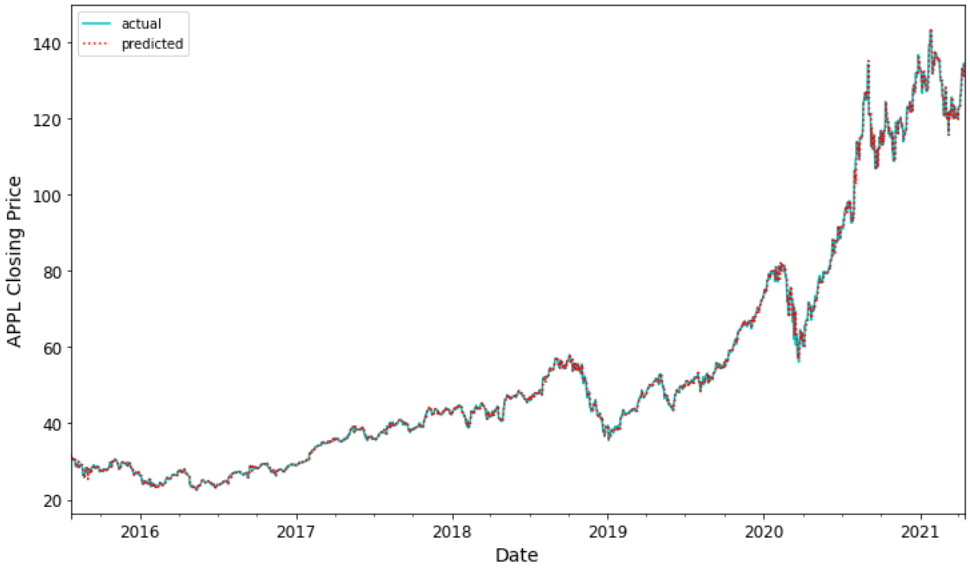
Team Work Distribution:

Team Member	Grade Adjustment	Work Completed
Khaled Behairy	**3 - No Grade Reduction**	Research, LSTM Hyperparameter Tuning, Future work section Code: trainer.py, hyperparameter tuning, KO stock
Stan Chen	**3 - No Grade Reduction**	Research, Introduction, Data Ingestion / Preprocessing functions/ Future work Code: stock_data.py, stonks.py, package architecture, DJIA sample code, AutoKeras experimentation
Burak Gulseren	**3 - No Grade Reduction**	Final Report: Research, Introduction, ARIMA Model Section Code: sample_arima_prediction.ipynb, ARIMA testing on various stocks, MSFT stock
Lotfi Hasni	**3 - No Grade Reduction**	Final Report: Contributed to research, discussion of models and hyperparameters, future work Code: model metrics and results plotting, GRU testing, BA stock
Mike Lasby	**3 - No Grade Reduction**	Final Report: Research, selected portions of introduction, feature engineering, preprocessing, discussion, conclusion. Code: stock_data.py, window_norm_timerseries_generator.py, example notebooks, final notebook template, AAPL stock, bug squashing

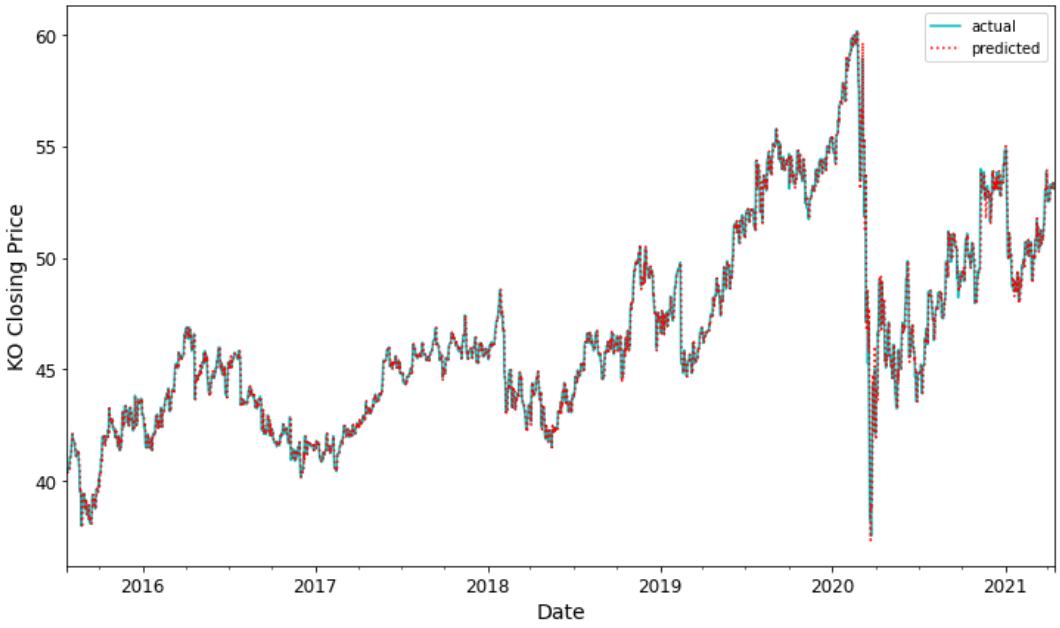
APPENDIX A - Additional Results

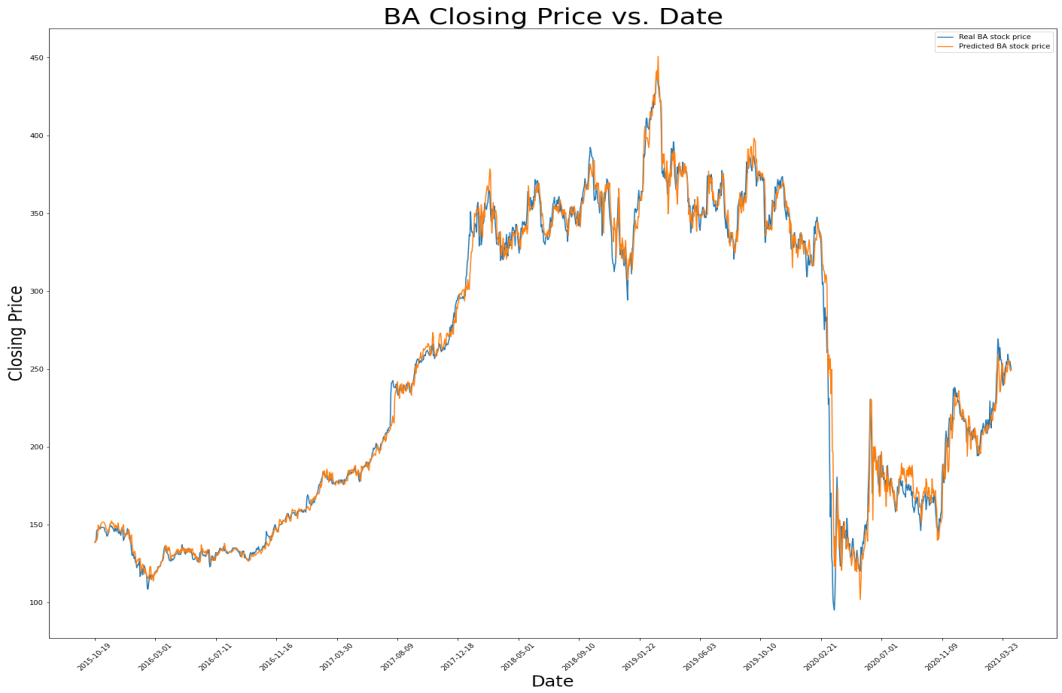
GRU-LH Plot Using AAPL Data

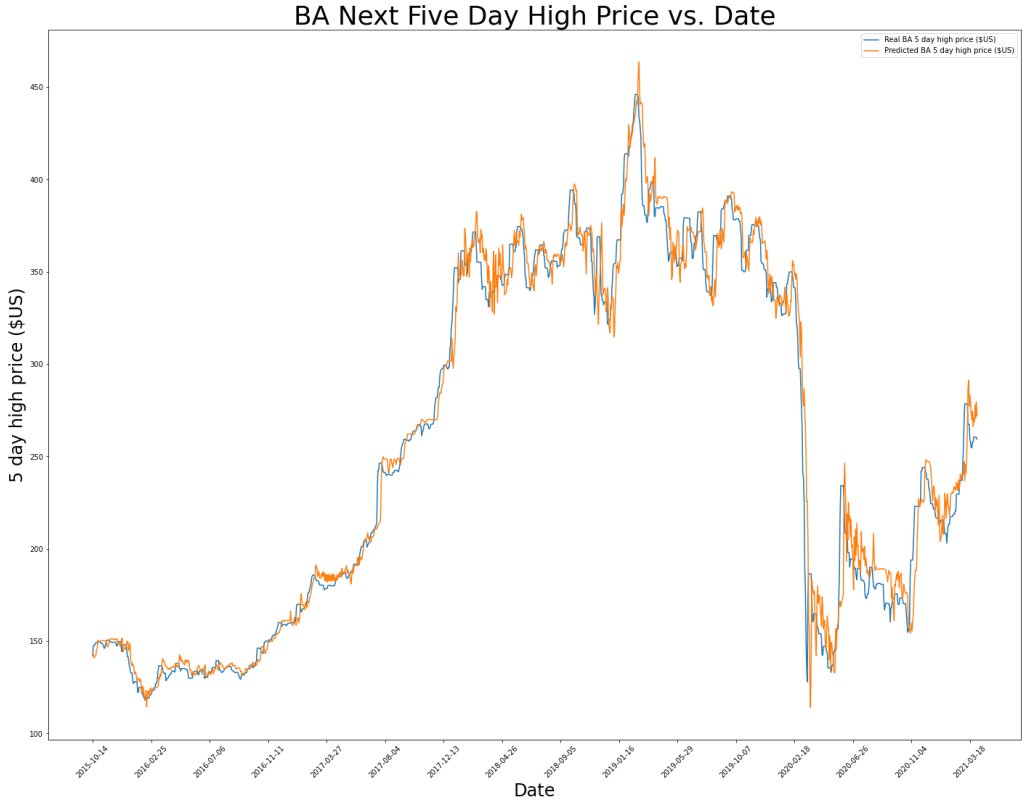
ARIMA Plot Using AAPL Data

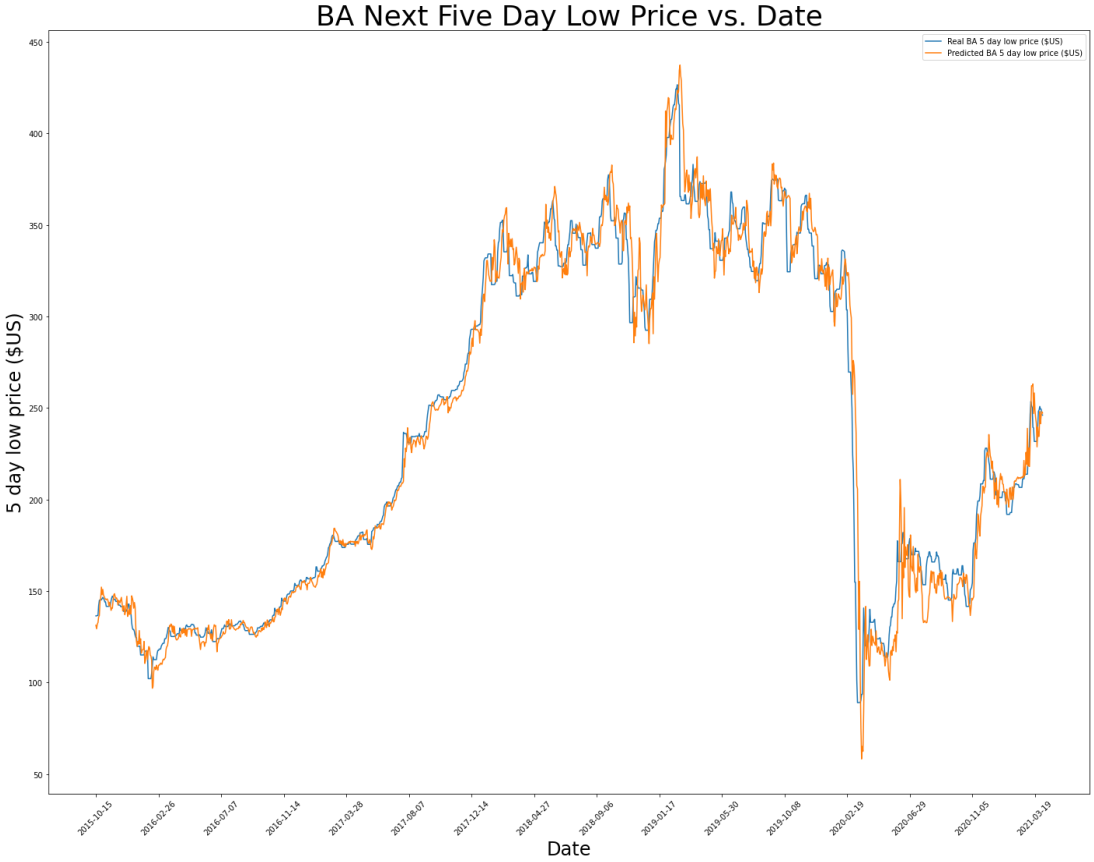


ARIMA Plot Using KO Data

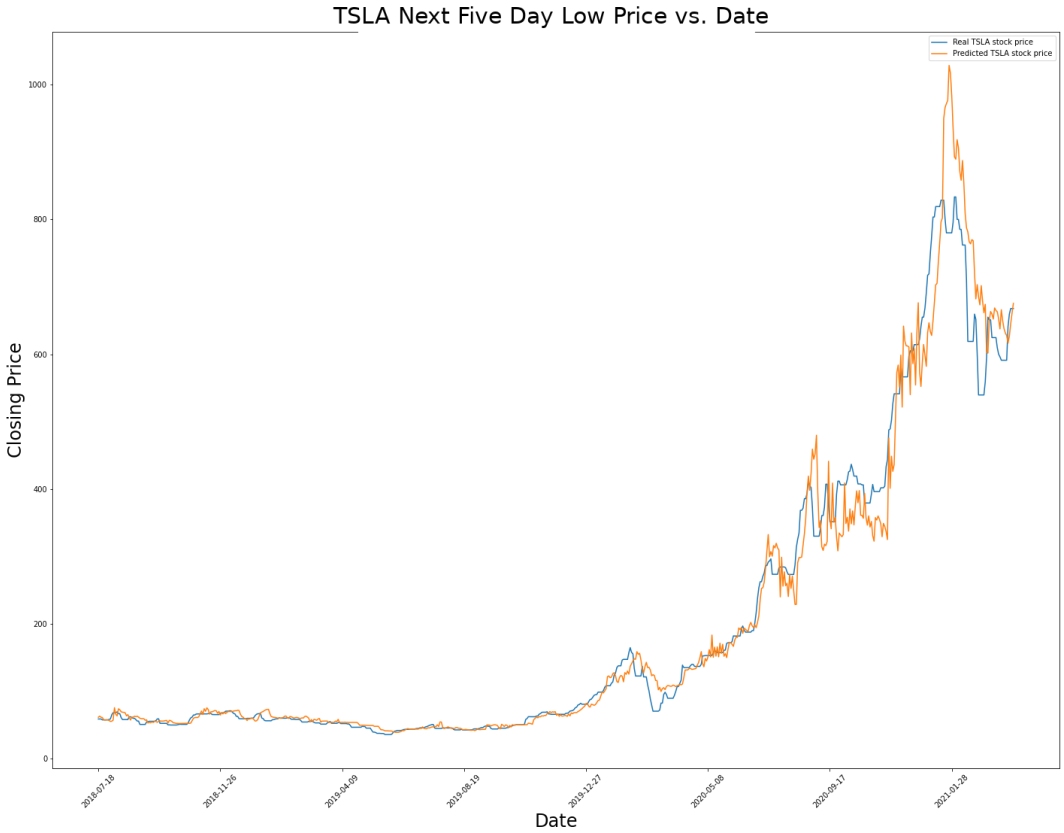


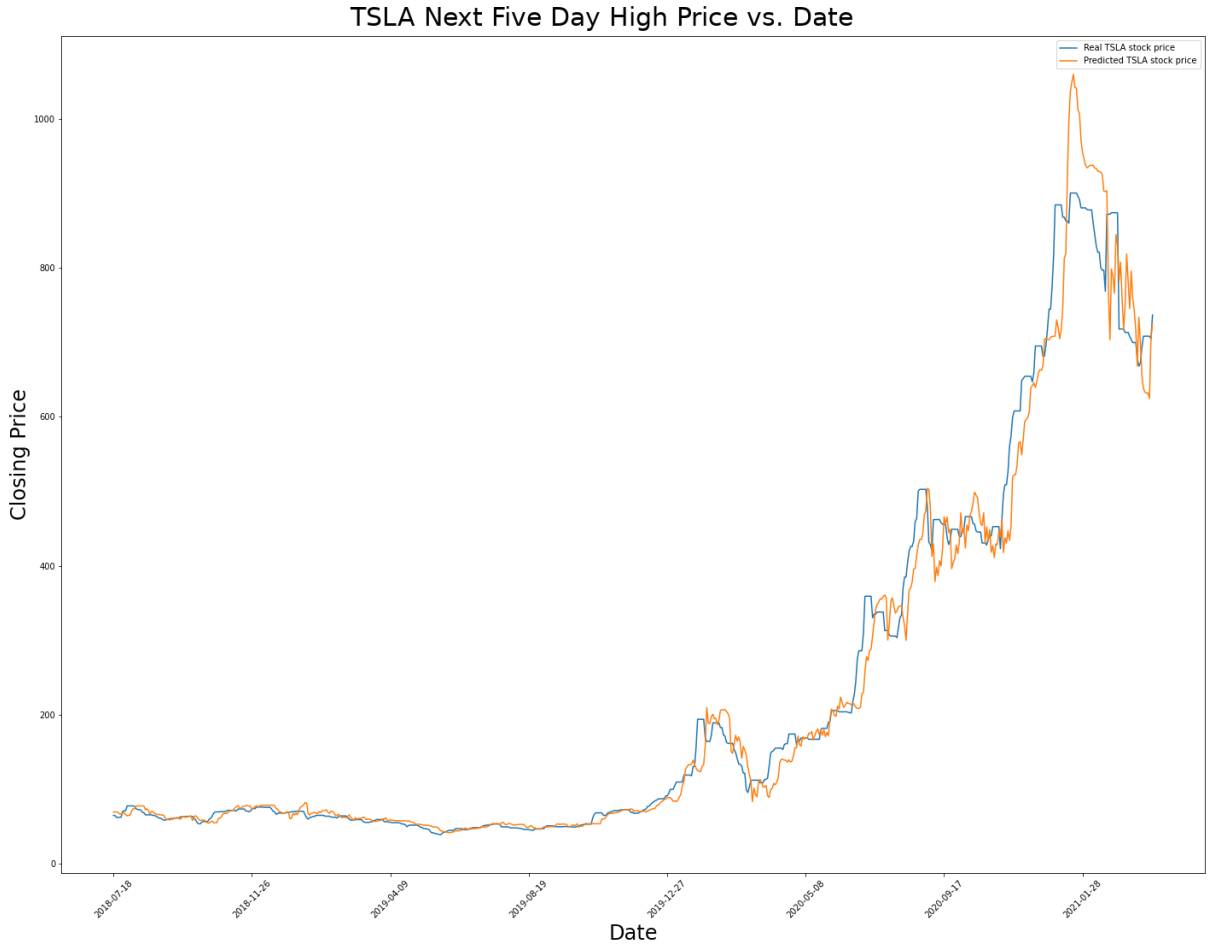


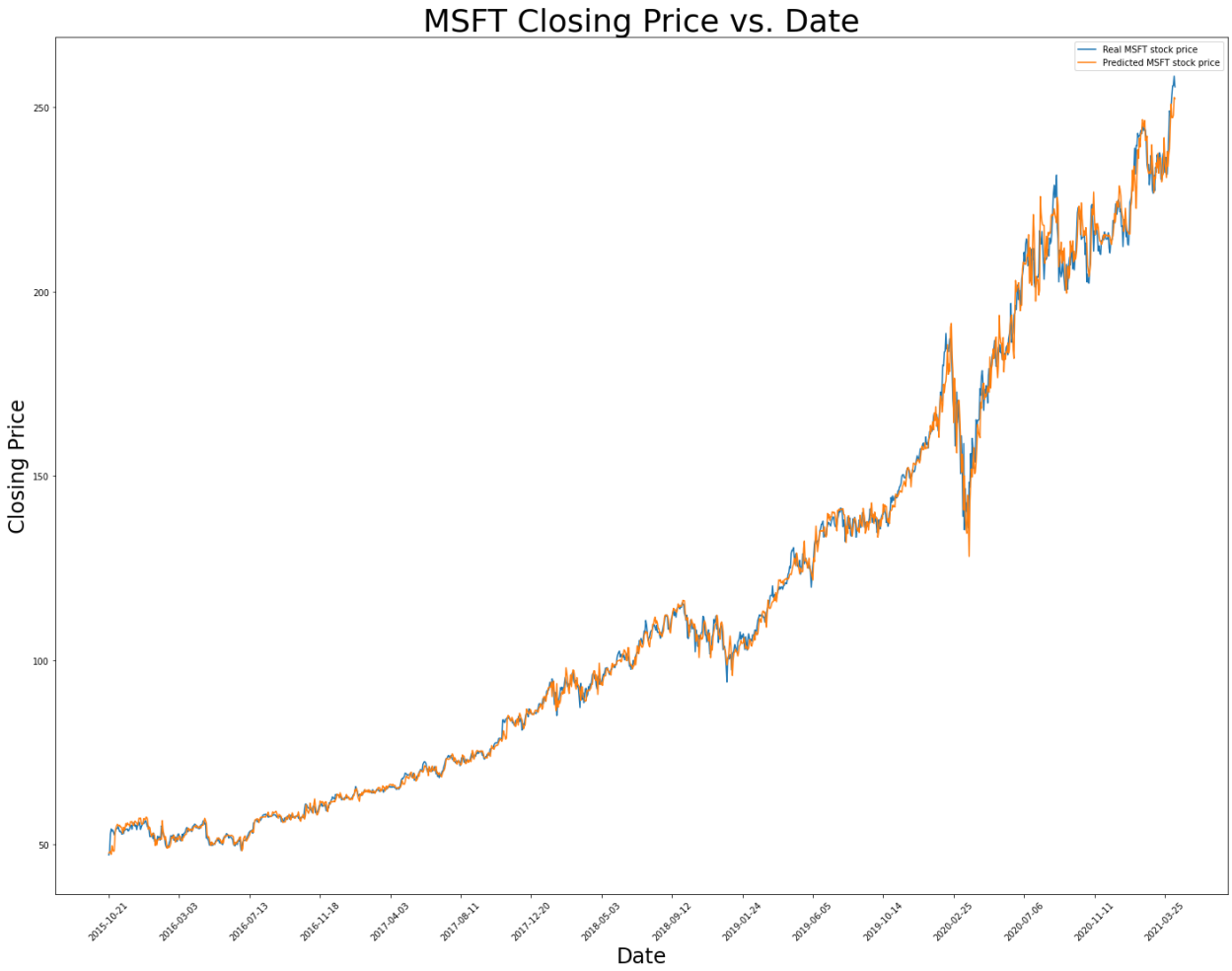

BA Next Five Day High Price vs. Date
Real BA 5 day high price ($US)
Predicted BA 5 day high price ($US)
5 day high price ($US)
Date

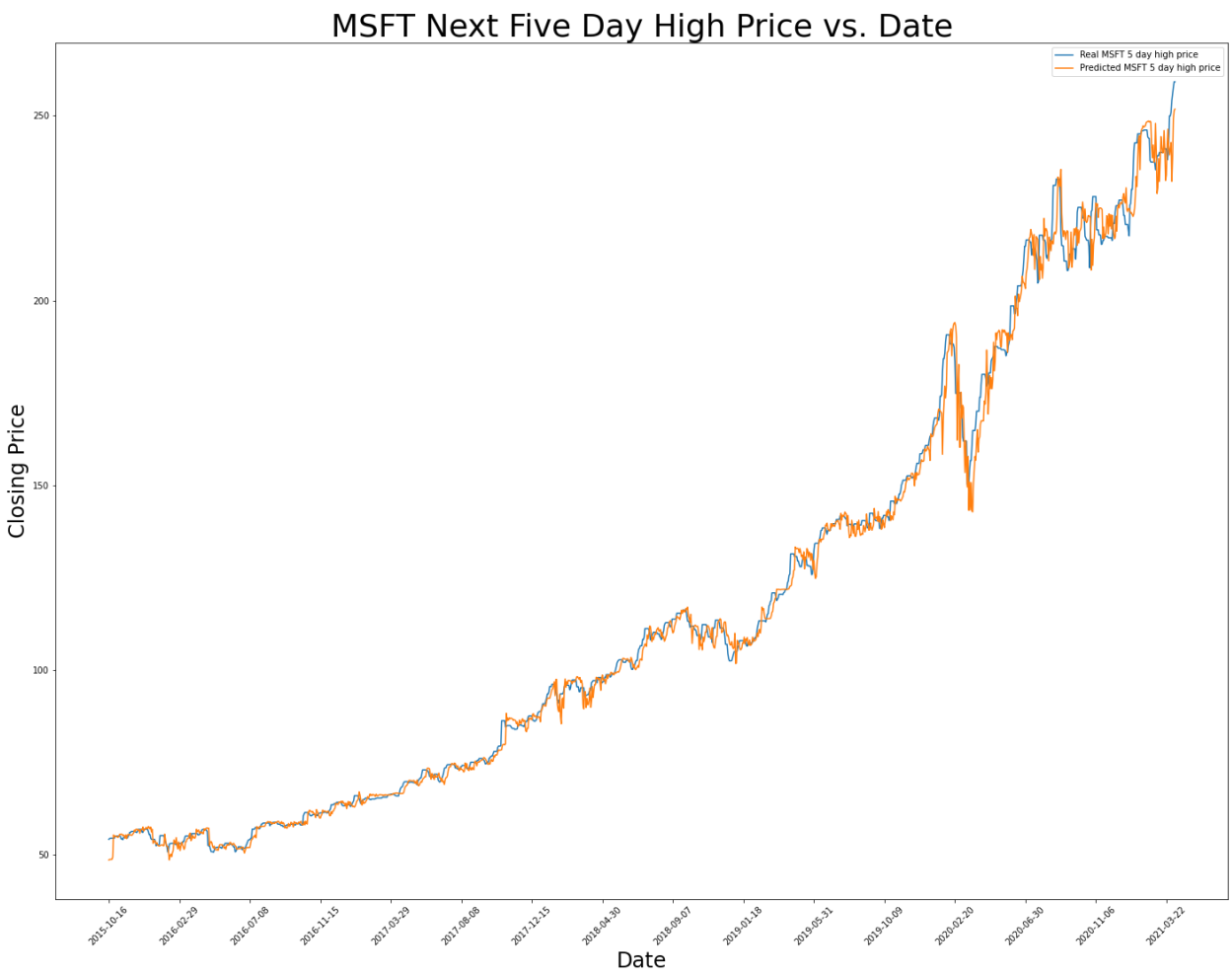

BA Next Five Day Low Price vs. Date
Real BA 5 day low price ($US)
Predicted BA 5 day low price ($US)
5 day low price ($US)
Date

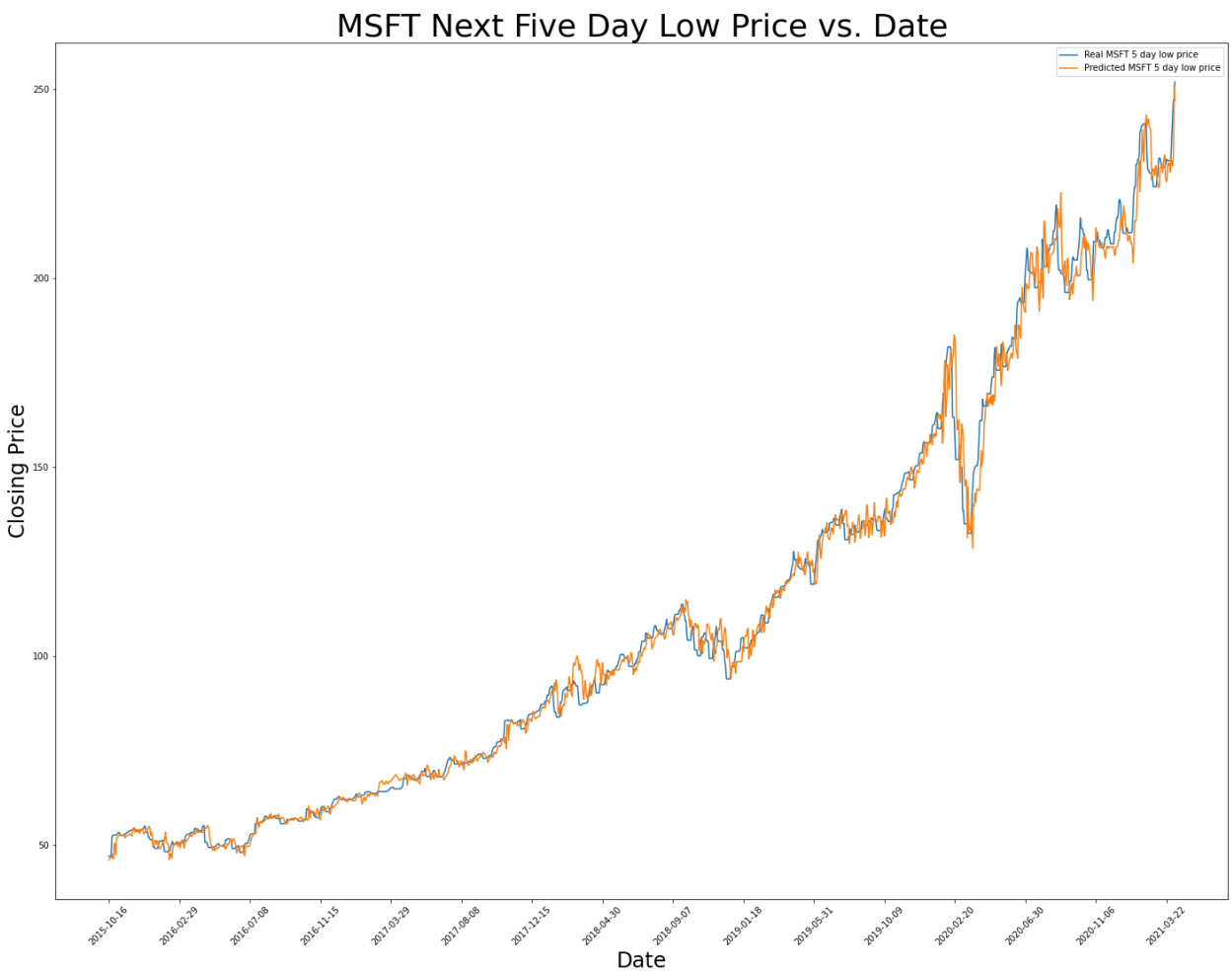

TSLA Closing Price vs. Date
Real TSLA stock price
Predicted TSLA stock price
Closing Price
Date
0
200
400
600
800
1000
2018-07-23
2018-11-29
2019-04-12
2019-08-22
2020-01-02
2020-05-13
2020-09-22
2021-02-02


TSLA Next Five Day Low Price vs. Date
Real TSLA stock price
Predicted TSLA stock price
Closing Price
Date
0
200
400
600
800
1000
2018-07-18
2018-11-26
2019-04-09
2019-08-19
2019-12-27
2020-05-08
2020-09-17
2021-01-28


TSLA Next Five Day High Price vs. Date
Real TSLA stock price
Predicted TSLA stock price
Closing Price
0
200
400
600
800
1000
2018-07-18
2018-11-26
2019-04-09
2019-08-19
2019-12-27
2020-05-08
2020-09-17
2021-01-28
Date


MSFT Closing Price vs. Date
Real MSFT stock price
Predicted MSFT stock price
Closing Price
Date
250
200
150
100
50
2015-10-21
2016-03-03
2016-07-13
2016-11-18
2017-04-03
2017-08-11
2017-12-20
2018-05-03
2018-09-12
2019-01-24
2019-06-05
2019-10-14
2020-02-25
2020-07-06
2020-11-11
2021-03-25


MSFT Next Five Day High Price vs. Date
Real MSFT 5 day high price
Predicted MSFT 5 day high price
Closing Price
Date
250
200
150
100
50
2015-10-16
2016-02-29
2016-07-08
2016-11-15
2017-03-29
2017-08-08
2017-12-15
2018-04-30
2018-09-07
2019-01-18
2019-05-31
2019-10-09
2020-02-20
2020-06-30
2020-11-06
2021-03-22


MSFT Next Five Day Low Price vs. Date
Real MSFT 5 day low price
Predicted MSFT 5 day low price
Closing Price
Date
250
200
150
100
50
2015-10-16
2016-02-29
2016-07-08
2016-11-15
2017-03-29
2017-08-08
2017-12-15
2018-04-30
2018-09-07
2019-01-18
2019-05-31
2019-10-09
2020-02-20
2020-06-30
2020-11-06
2021-03-22