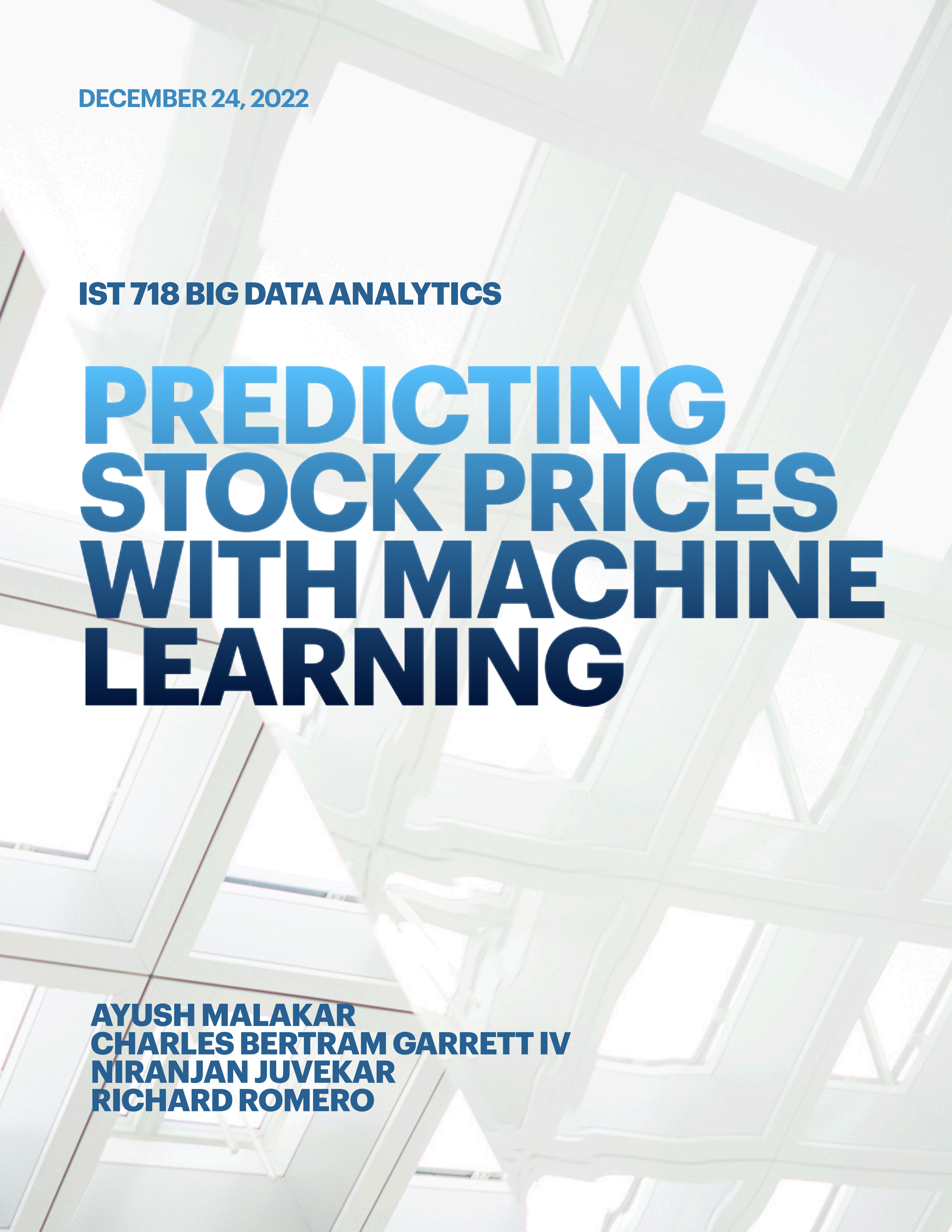


DECEMBER 24, 2022

IST 718 BIG DATA ANALYTICS

PREDICTING STOCK PRICES WITH MACHINE LEARNING

AYUSH MALAKAR
CHARLES BERTRAM GARRETT IV
NIRANJAN JUVEKAR
RICHARD ROMERO

Table Of Contents

ABSTRACT

This project is an attempt to predict the future stock prices looking at the historic values. Total 2700+ Ticker symbols are part of this dataset, some of the CSV files for the Tickers contain values that date back as far as 1970. It was important for the team to assure accurate models. Including the entire set of tickers is computationally heavy and very time consuming, therefore we selected only the 410 tickers that compose S&P 500.

As we should not rely upon just one technique, we have implemented multiple models and look at the predicted values from all of them. Our recommendation system uses results from ARIMA, LSTM and RNN models. RNN and LTSM are great models but require a lot of computational resources to run. One can get further accuracy with ARIMA models accounting for the seasonality.

The recommendation provided is a result of set of stocks that come up in all three model predictions to be under same category - Buy or Sell with greater rating. The recommendation engine follows simple rules:

For day trading, the recommendation engine looks at the maximum %difference between Low and High values and picks the top stocks out of those.

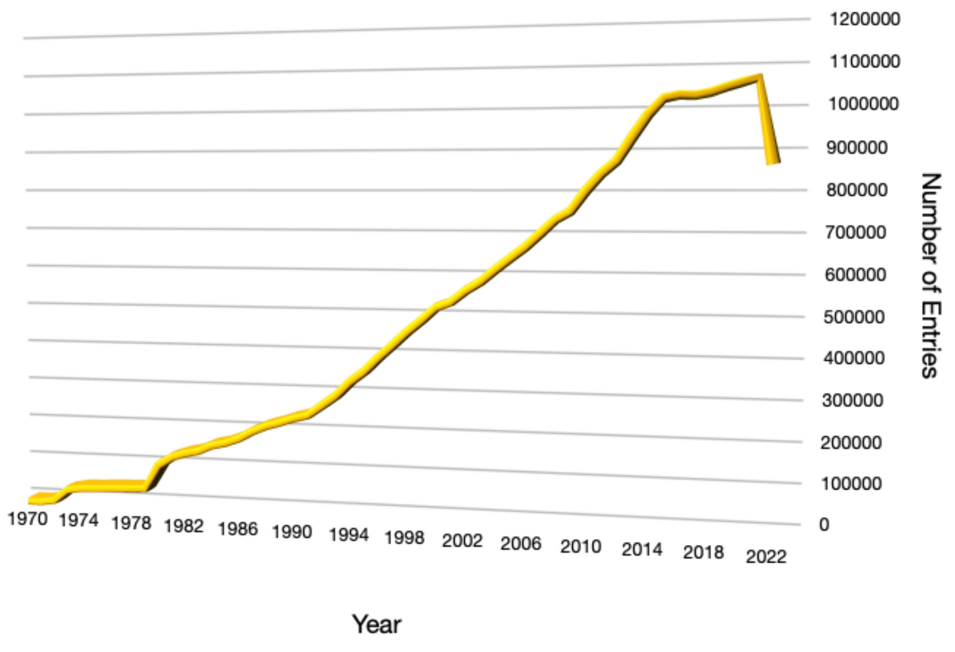
For Short-term investment, if all models agree on Open and Close price, if close price is greater than open price it's a "Buy", if close price is less than the open price, it's a sell.

INTRODUCTION

Individuals wanting to invest into stocks and want to buy or sell stocks in short term would be the stakeholders in this project.

Note : Stock investments are subject to market risks and depend upon several factors other than the historic values alone.

Data Set: We used a dataset from Kaggle (https://www.kaggle.com/datasets/paultimothymooney/stock-market-data). It holds very extensive data on a variety of different Tickers dating back to the 1970s. The dataset includes open, close, volume, high, low, and adjusted close values for each date. There are more than 3700 different ticker symbols in this dataset. For the scope of this project, we have decided to limit ourselves to the data available on 410 stocks that form the S&P 500 composite. Below is the number of data points we have for each of the years:



Data was downloaded in the middle of 2022, therefore there is a drop in the number of points in 2022.

Data Cleaning: The data as observed has gaps for some of the tickers i.e. for some dates in between the values are missing. We have filled these gaps by carrying forward the previous known value using ffill function.

Once above method is applied to get the gaps filled and a given ticker symbol is taken in for processing, dropping NA causes only the unknown values from the beginning to be dropped.

For comparisons, multiple such symbols can be picked into single data frame and dropping NA would give only the subset that has values across all the tickers..

Data Exploration: We used autocorrelation to do preliminary analysis of the stocks to find what affects the stock prices and whether there is a seasonality involved. This data tells us what previous values affect the future values of the stock the most and can be helpful in building better models.

Model generation: We generated ARIMA, LSTM and RNN models for each ticker symbol for “Open”, “Close”, “Low” and “High” data frames. Together that constitutes 410 tickers * 4 types of values * 3 types of models = 4920 models. Out of this, Just for ARIMA, the algorithm explores at least 9 models to pick the best possible model. Hence, model generation and selection is a computationally heavy and very time consuming process.

INFRASTRUCTURE

Before we proceed into our analysis, we would like to give some information about the techniques to enforce discipline and to speed up the process of model generation and prediction that we employed for this project.

1. THREADING & QUEUE CONTROL

Advanced processors in our laptops can handle multiple independent threads. Firing each process under a separate thread utilizes full potential and bandwidth of the microprocessor.

Using this technique, running ARIMA model on only one frame (say, just the “Open” data frame for all the tickers) takes 5 hours. So, running under threads, on an average a model for a given ticker takes around 44 seconds.

Experiments found that there are errors if we try to generate models using more than 6 threads on Intel i9 processor, so we have to limit number of max_threads to 6 for processes using this hardware configuration. Apple M1 laptops are capable of handling 8 threads more efficiently. Increasing the number of threads greater than 8 may cause the efficiency to drop. A queue of appropriate size was implemented that would keep a track of how many models are currently being generated under individual threads. Once a given model generation is done, processing of the data can also be done as the new models are fired in the other loop.

2. CODE MAINTENANCE & DEBUGGING

In order to facilitate multiple people to do the development simultaneously and still conform to the standards of the project code quality, we did following two things:

a. SET UP A REPOSITORY

This ensured that the development is at one place and visible to all the members. Our code resides at: https://bitbucket.org/nirusj/predictingstockprice/

b. USE A STANDARD TEMPLATE

We created a standard template in Jupyter Notebook that has separate and clear places (cells) to import packages, do user settings, writing functions and running the threads.

This ensured that the code with minimum number of changes to the user settings would work on the other user's system. Also, it reduces the learning time for the other user.

c. MESSAGE LOGGING

There are two logs that are implemented that help with tracking and debugging.

- Phase log : This documents the important milestone start and end points for the runs
- Details log : This documents user settings and the details of the steps executed in each cell

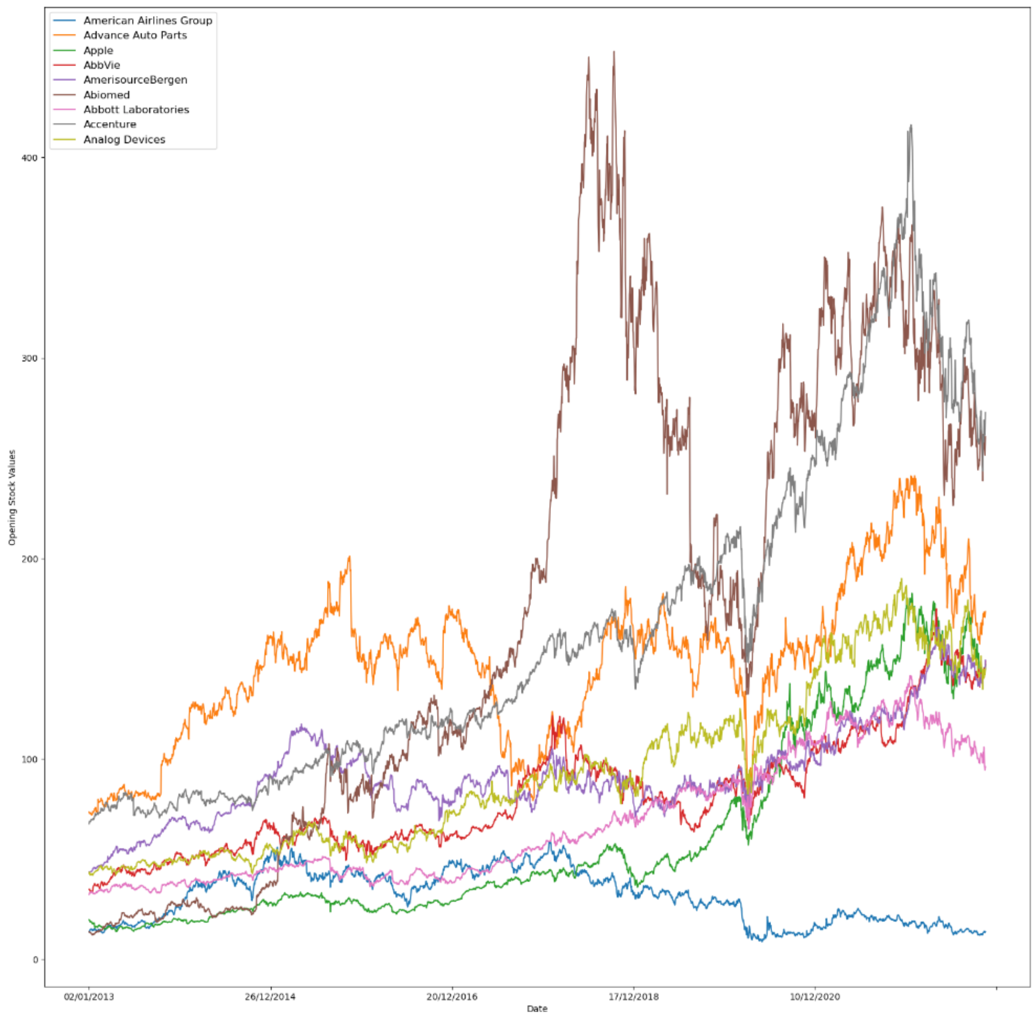
d. DEBUGGING METHODOLOGY

Debugging a threaded process is particularly challenging, so, we published a standard methodology in order to probe the function for debugging purposes.

All the above methodologies combined together resulted in tons of time savings that enabled us to finish the project.

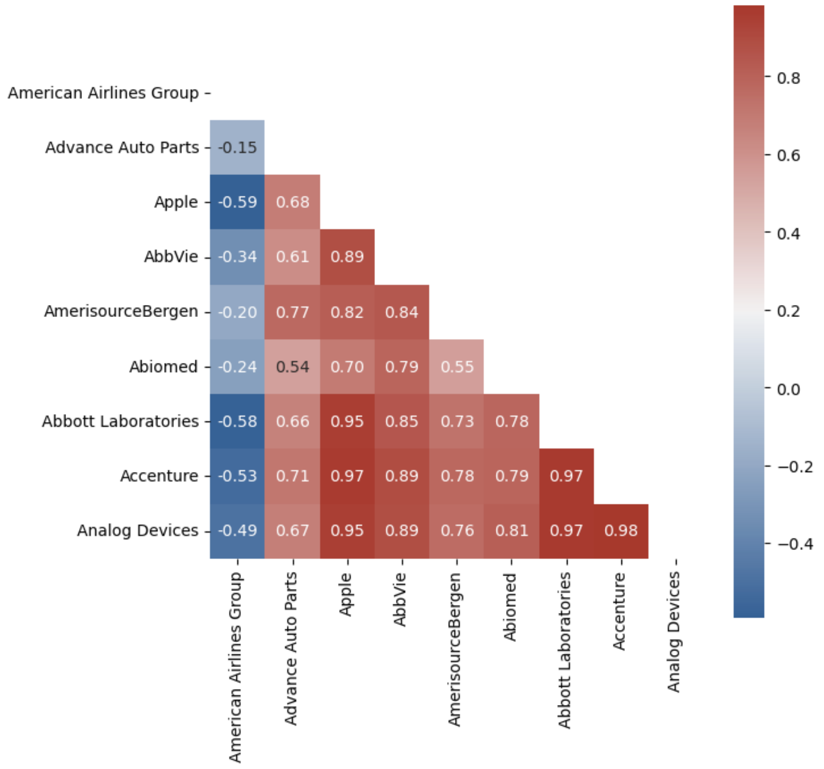
DATA EXPLORATION & AUTOCORRELATION

Below is a subsample of a few stocks looks like. We see there are occasionally some gaps in the dataset and thus we had to remove null values when running any models. For our correlations, we only ran values across stocks for where they each had a value for that given date.



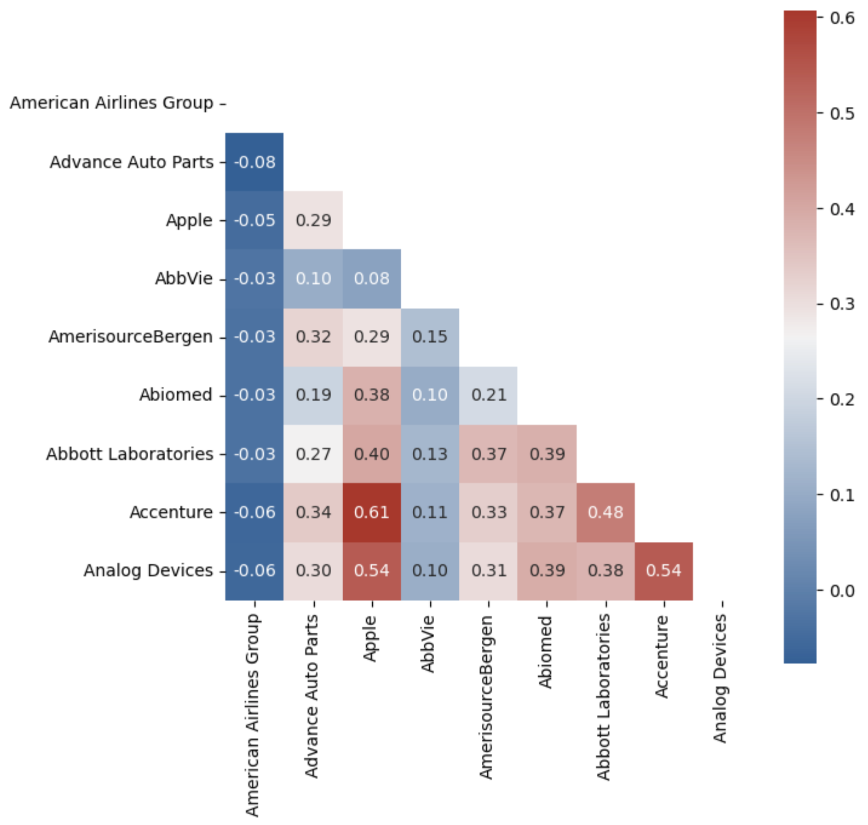
Exploring the data, we found that there exist some seasonality trends and some autocorrelation. We can see that the broader market that stocks are a part of can be subject to trends and events such as the holidays, start of war, climate catastrophe or news of inflation. Neither do they affect all stocks nor equally

CORRELATION - BEFORE REMOVING TRENDS



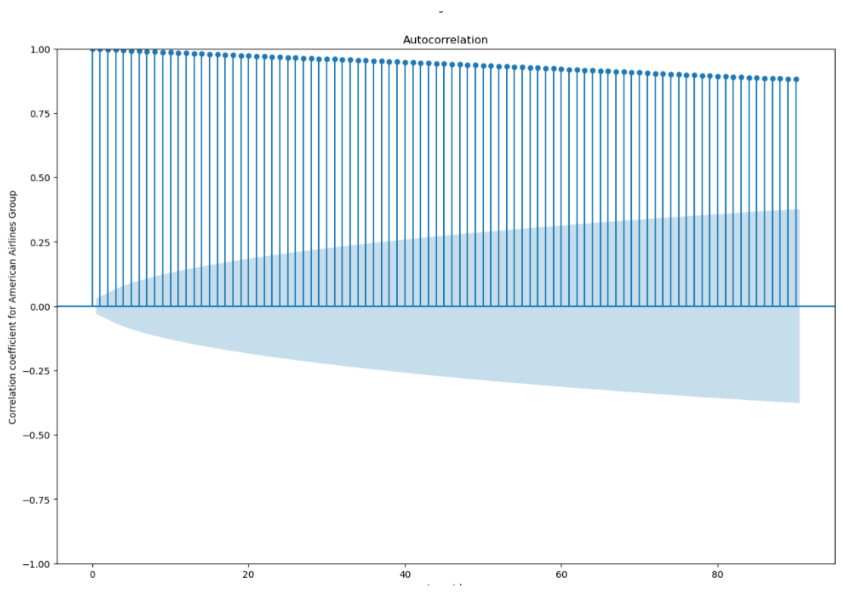
Here we see that there is a high correlation among all the stock values, except American Airlines Group. The reason they have a high correlation is that their values all have been increasing. However, once we remove the trends and seasonalities, we see a different picture.

CORRELATION - AFTER REMOVING TRENDS

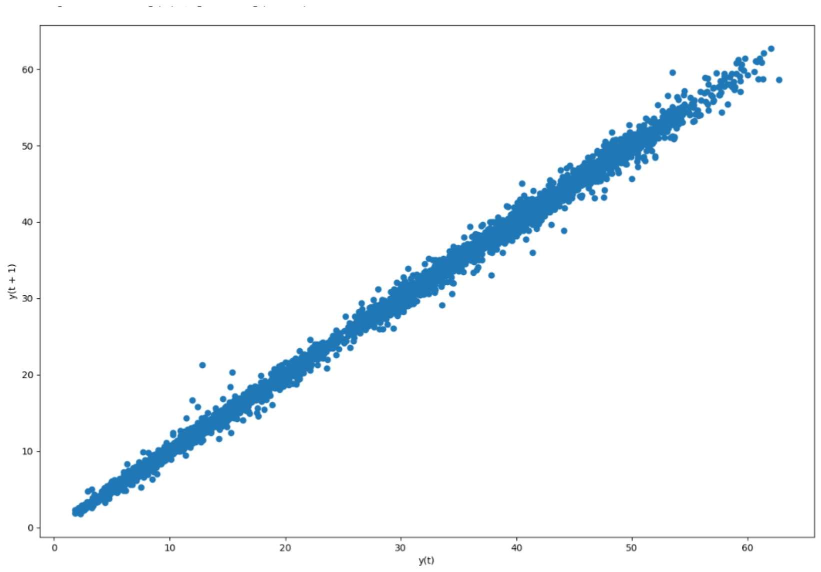


Once we remove the trends and normalize the data we now see different results. The datas are not in a high correlation as they were above. This is similar in almost every stock values.

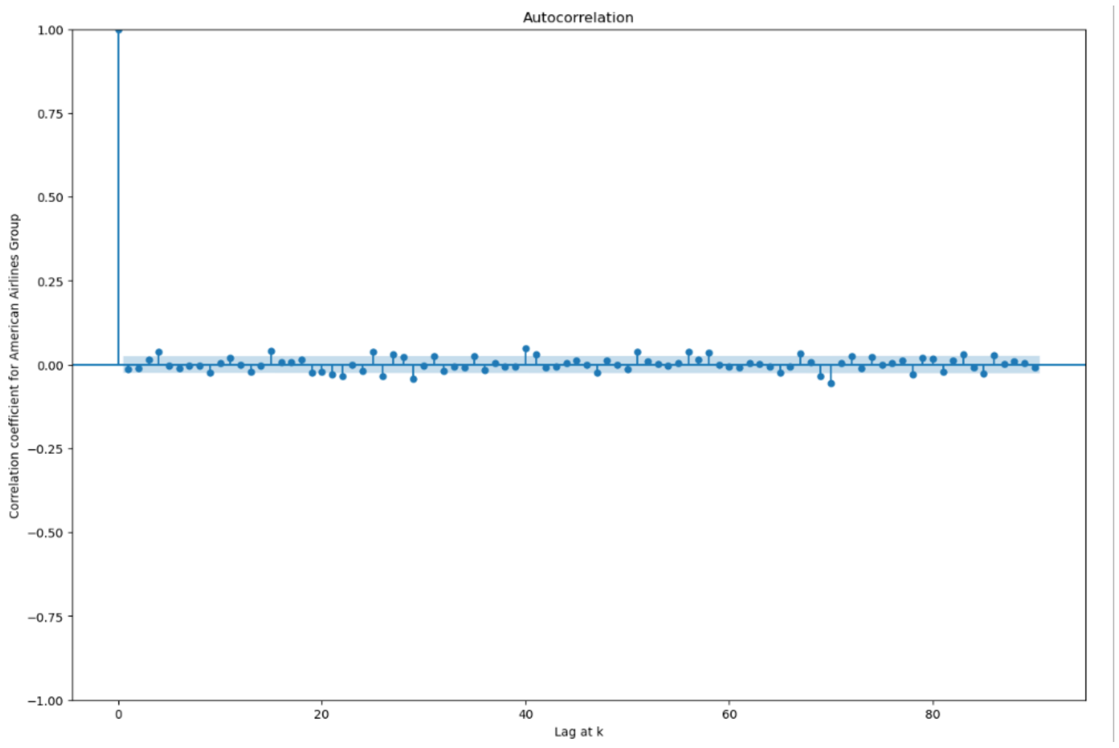
AUTOCORRELATION



With the initial autocorrelation without normalizing, we can see that the data does not reflect too much of the seasonalities or trends. However, it does depict that all the data have a very strong autocorrelation among each other. This can also be observed from the lagged plot



After we normalize the value, we can actually see how the previous data affected its next value



Based on the chart above, we can see how each value influences the next value at lag K. We can see that the 2nd value had a positive influence on the 3rd and the 3rd had a positive influence on the 4th however, the 4th had a negative influence. This pattern might not be the same across all the stocks but one the trends and seasonalities have been removed we get a clear understanding of how the values are actually affected.

ARIMA

ARIMA is one of the most widely used time series forecasting methods. This method tries to use the autocorrelation in the data.

There are 3 main parameters that need to be tuned in order to get this model running correctly. P, q and d.

> p is the order of the AR term
> q is the order of the MA term
> d is the order of differencing that is required to make the time series stationary

On top of that the seasonality term if obtained, the model performs better. Partial Autocorrelation on 1st order differencing would give order of the AR term and Autocorrelation plot would show the order of MA term. An implementation of this may be more cumbersome and error prone, therefore, in the interest of time, pmdarima implementation was used that would select best values of p, q and d.

It is observed that the confidence interval keeps growing as we go further into the future therefore, day trading and short term investing are predicted best with any of the techniques of forecasting.

LSTM

LSTM stands for Long Short-Term Memory and is useful for running on stock data because it stores past information to predict the future. We used an 80/20 train/test split. Previous 80% of samples were used to help predict the most recent 20% of each Ticker symbols, then apply that forward to generate future 30 business days of data.

Initially with the LSTM model, we tried using a model that was not pre-built and adding layers to it. This model was still using Sequential from the Keras package and using the adam optimizer. The layers we added on top were LSTM layers as well as dense layers. This model has some potential, but part of the issue we ran into this was just how long it took to run. In the same time we were able to run all 25 epochs of the final LSTM model we ended up using, we would just be finishing up the second epoch of this model. With the large number of models we needed to run to complete this project, it was just not feasible, though it could work if this project ever progressed to the point where we wanted to have a user input a symbol and the model would run on the spot.

Given more epochs, this model probably had the potential to predict on the test better than the final model we used. Because of this, we decided to slightly alter the specifications of the model by adding an activation function. In this case, the relu activation function was used and only one dense layer was added, instead of the 4 total layers of the initial model. This allowed us to complete each epoch in about 4 seconds and finish each total model call in about two minutes.

For the final model model, we used a pre-trained sequential model within the TensorFlow package, using the adam optimizer and using MSE to limit the loss. This model performed well with some stocks, but had some overfitting on some of the others, predicting a few stocks to rise or fall precipitously. When running the same model type across all 410 tickers, this can be prone to happening.

RNN

Recurrent Neural Networks (RNN) is a type of neural network that is used more specifically for time series and sequential data. RNN's hidden layer utilizes historical input data such as historical stock prices to be able to streamline time series data. For example, the feedback layer causes the hidden layer and output layer to be dependent upon each other whereas a feedforward neural network considers each layer independent from each other.

In this model, we split the data 80/20 and had 10 epochs since RNN models are complex in nature and take a significant amount of computing power to run. Utilizing the Tensorflow package which had a lot of keras package models such as SimpleRNN allowed us to be able to set the layers based on the shape of the array.

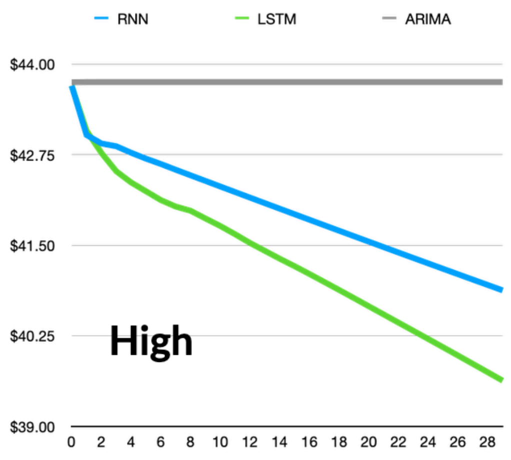
PUTTING IT TOGETHER

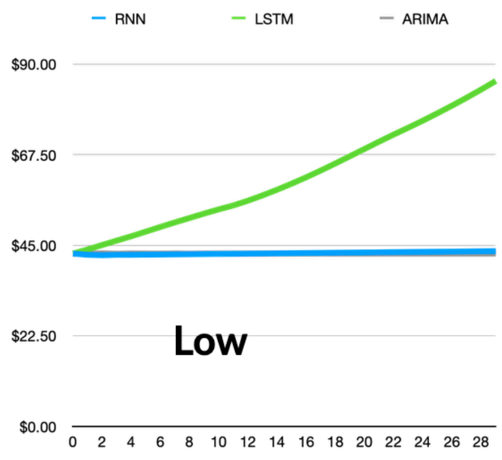
The LSTM model projection shows that one should buy AMAT (Applied Materials), TJX (TJX Companies Inc), and CDNS (Cadence Design Systems Inc) for the highest raw profits and should buy CNWT (Cistera Networks), NTRR (Neutra Corp), and NMHLY (NMC Health) for the highest percentage profits. The customer should avoid buying BRK-A (Berkshire Hathaway), AZO (Autozone), IDXX (IDEXX Laboratories Inc.) as they are expected to see the largest gross losses, while INTH (Innotech Corp), CPICQ (CPI Corp), NOXL (Noxel Corp) have the highest predicted percentage losses.

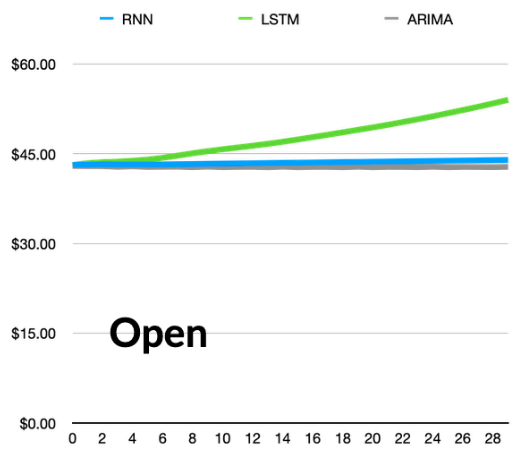
Based on the RNN model, one should buy NTRR, TROW, and GPN for the highest gross percentage profit which means it is relatively safe as some of these stocks are now dropping to a cheaper price and is projected to grow even bigger as it rebounds from Covid 19. Customers should avoid buying INTU, MDT, and IDXX because it is essentially at its peak and is expected to drop.

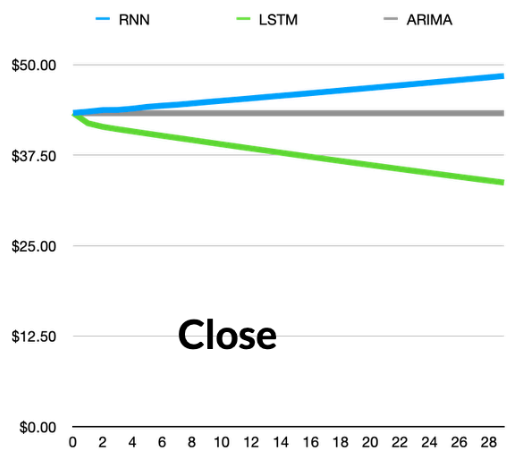
But looking at the above, the two models - LSTM and RNN - have predicted completely different set of stocks to be bought or sold. Is one model more accurate than the other? Well, this is model to model difference and it’s very hard to pick one over the other.

Following graphs compare the results of the three models for Open, Close, High and Low for CSCO.



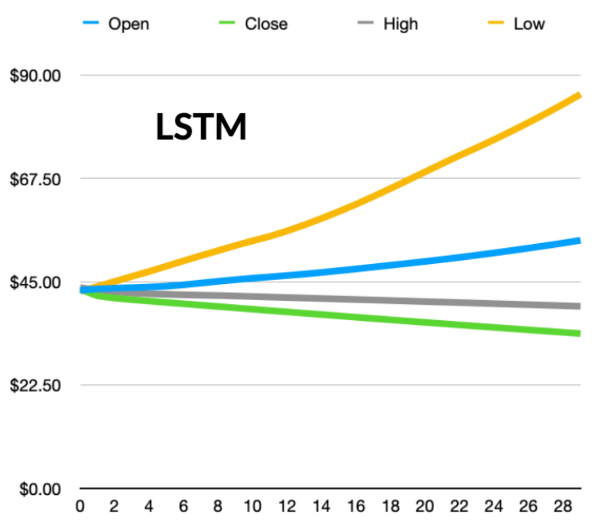


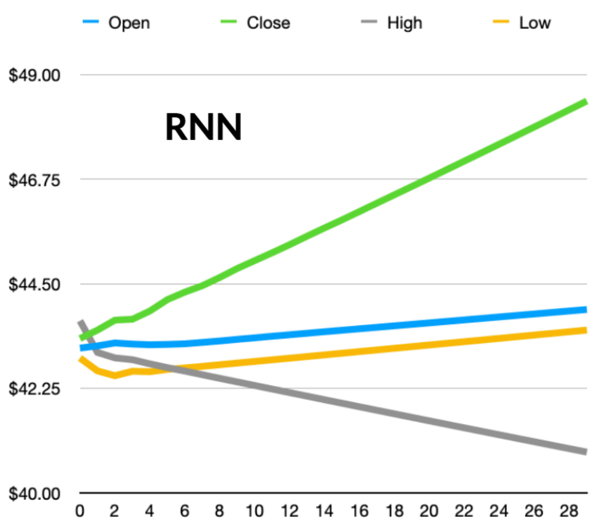


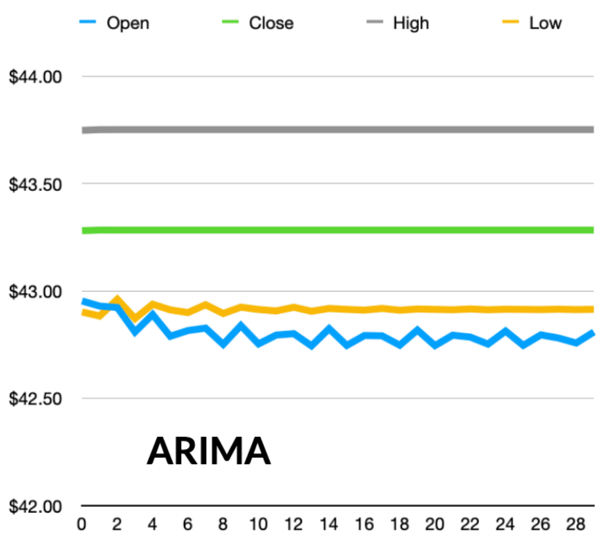


It shows that the models may initially be close with the predictions but may start vastly disagreeing as we go farther into the future. This means, the closer the forecast the higher is the probability that the values would be correct.

Let’s compare what one model has to say about all four (Open, Close, High and Low) values for 30 days forecast for CSCO and see if that makes sense:







Ideally “Low” line should be at the bottom, “High” line should be at the top and the other two - “Open” and “Close” should be somewhere in the middle or may touch “High” or

“Low” lines depending upon the stocks opening or closing at the highest or lowest points in the day.

Interestingly enough, in LSTM graph, for one day prediction, the points seem to be at the right place but for further predictions, “Low” line is above all the others and “High” line is significantly below the “Low” line - meaning, high value is lower than the low value - which does not make any sense.

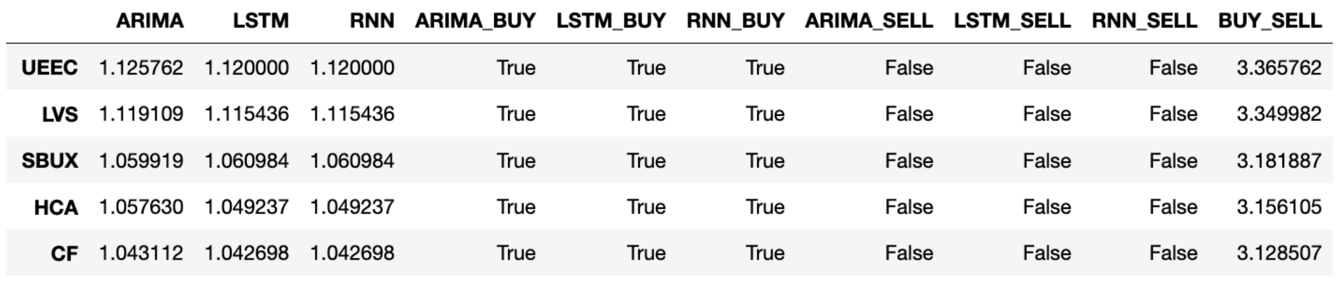
Similarly, in RNN graph, for initial one forecast, the points seem to be in order and then “High” line suddenly takes downward trajectory going below all he other three - again - not making much sense. In ARIMA model similar is the case - “Low” line crosses to go above “Open”.

This proves that the reliability of the predictions goes down (to a level that they would not make sense) as we try to look farther into the future. So, best is to look no further than one to two days into the future - that means, short term trading or day trading.

SUMMARY AND RECOMMENDATIONS

As we have seen above, the reliability of the predictions goes down (to a level that they would not make sense) as we try to look farther into the future.

Following is the output of the recommendation model:



	ARIMA	LSTM	RNN	ARIMA_BUY	LSTM_BUY	RNN_BUY	ARIMA_SELL	LSTM_SELL	RNN_SELL	BUY_SELL
UEEC	1.125762	1.120000	1.120000	True	True	True	False	False	False	3.365762
LVS	1.119109	1.115436	1.115436	True	True	True	False	False	False	3.349982
SBUX	1.059919	1.060984	1.060984	True	True	True	False	False	False	3.181887
HCA	1.057630	1.049237	1.049237	True	True	True	False	False	False	3.156105
CF	1.043112	1.042698	1.042698	True	True	True	False	False	False	3.128507

So, the recommendation engine is suggesting following 5 stocks for day trading:

UEEC, LVS, SBUX, HCA and CF

REFERENCES

How to deal with missing values in a Timeseries in Python?
https://www.projectpro.io/recipes/deal-with-missing-values-in-timeseries-in-python

Advanced Time Series Analysis in Python: Seasonality and Trend Analysis (Decomposition), Autocorrelation
https://towardsdatascience.com/advanced-time-series-analysis-in-python-decomposition-autocorrelation-115aa64f475e

Types of Autocorrelation
https://www.geeksforgeeks.org/types-of-autocorrelation/

How to Use and Remove Trend Information from Time Series Data in Python
https://machinelearningmastery.com/time-series-trends-in-python/

How to Develop LSTM Models for Time Series Forecasting
https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

Stock Prices Prediction Using Long Short-Term Memory (LSTM) Model in Python
https://medium.com/the-handbook-of-coding-in-finance/stock-prices-prediction-using-long-short-term-memory-lstm-model-in-python-734dd1ed6827

ARIMA Model – Complete Guide to Time Series Forecasting in Python
https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

Predicting Sequential Data using LSTM: An Introduction
https://towardsdatascience.com/time-series-forecasting-with-recurrent-neural-networks-74674e289816