Machine Learning Classification for Prediction of Daily Stock Movements

Lisa Hladik

Electrical Engineering

Union 2018

SUMMARY

The purpose of this project is to design and implement a machine learning algorithm to predict daily stock price movements. Stocks vary in their prices on a daily basis. As such, investors capitalize on this volatility to buy and sell stocks to generate profits. Thus, a machine learning framework with accurate price prediction capabilities would allow investors to optimize their trading strategies to mitigate losses and optimize profits.

The system is two-fold. The first part of the system is the actual machine learning framework for making price predictions. These predictions should predict the price movements of stock accuracy more than 50% of the time, i.e., better than guessing. The second part of the framework is to test the effectiveness of these predictions from a financial standpoint. For the purposes of back-testing the algorithm's effectiveness, the algorithm first calculates a "market benchmark." This benchmark essentially measures how well the market is doing by simulating the profit or loss that would occur if an investor bought and held a stop. Secondly, the algorithm also calculates how well the investor would do using the price predictions to make buying and selling decisions. The goal is for the algorithm's predictions to enable the investor to outperform the market benchmark.

The final design did meet the design requirements set from the above discussion to some extent. There were many intermediary steps leading to the final design. This included running multiple experiments to train find the best combination of hyperparameters and input features that would produce the most accurate predictions. Additionally, some changes did have to be made to the initial design as complications arose with the input features. Specifically, additional steps were required for data pre-processing to increase the accuracy of the algorithm's final predictions. The best final design of the machine learning framework was able to predict the

directional price movements more than 50% of the time accurately, but the predicted magnitudes of those movements did have a lot of error. Predictions experienced the least amount of error when shorter price forecasts were made. As the number of days predicted increased, the amount of error also rapidly increased. Overall, the best design could be useful for giving investors an edge in determining the direction of stock price movements.

Table of Contents

TABLE OF FIGURES AND TABLES

1. INTRODUCTION

The stock market is pivotal part of the economy as the market influences the growth of commerce and industry. Companies sell stocks to consumers to raise funds. These stocks give consumers a share in the ownership of a company, meaning the stock holder has a claim on a company's assets and earnings. Stock prices are dictated by what investors feel a company is worth; thus, the value of a stock does not necessarily equate to a company's intrinsic value. Price fluctuates between how much people are willing to pay for a stock (the bid) and how much people are willing to sell a stock for (the ask). Stock prices change on an intra-day and inter-day basis due to changes in supply and demand. When there are more buyers than sellers, the stock price increases; conversely, when there are more sellers, the market price decreases. There is a huge variety of market forces that affect price movement, including the performance of a company or industry, investor sentiment about the overall market, political news, interest rates, and a variety of other economic factors. The interplay of these factors contributes to the difficulty in predicting where stock prices will move. The ability to predict price movements would allow the creation of an ideal stock-buying and -selling strategy to maximize profits.

The dual purpose of my thesis is: (1) to apply machine learning methods to predict stock price movements, and (2) to use these predictions in designing a framework to assist traders in making profit-maximizing trading decisions that would outperform a market benchmark. Within the context of a machine learning problem, my project attempts to break down price prediction into strict quantitative analysis. The stock and other input data utilized are represented as time series. One major issue associated with time series problems is that the underlying patterns in data tend to change over time, an idea known as "concept drift." In the context of the market, concept drift is hugely prevalent. For some period of time, the markets may have some level of predictability. The issue is that as patterns become obvious, those buying and selling stocks will

adjust their strategies. This strategy-adjustment often destroys the pattern, and the markets will again become unstable. Machine learning's computing capabilities could provide the ability to spot market patterns more quickly than traditional traders to capitalize on potential returns before the pattern disappears. Therefore, my goal was to implement an algorithm that was able to anticipate and negate the effects of concept drift to enhance the quality of my algorithm's predictions.

This report is organized in the following sections. Section I provides background information on the concept of stock market price predictability, the mechanics of machine learning, and previous work conducted applying machine learning to predict future stock prices. Section II gives a description of the design requirements of my stock prediction machine learning framework. Section III gives the various design alternatives that I considered and experimented with before deciding on a final design. Section IV provides the preliminary proposed design derived from the design alternatives. Section V reveals the final design and implementation. Section VI presents an evaluation of the final system's performance. Section VII provides the production schedule in which this project was researched, formulated, and implemented. Section VIII gives a cost analysis about the cost of implementing the project. Section IX provides a User's Manual to direct any use with how to use the system for financial applications of stock market price predictions. Lastly, Section X provides a discussion summarizing the problem of stock prediction confronted by the project, final design and its performance, and potential future work. This section also provides a discussion of important lessons learned about machine learning and data in the process of this project.

2. BACKGROUND

The creation of a machine learning framework to perform stock market price prediction assumes that stock market price history tends to repeat itself. Basically, this means that a stock's past price movements will be able to predict its future price movements. In economic terms, the assumption is that the market is "efficient." Efficiency in the market means that the price of any individual security at any given point in time reflects its intrinsic value [1]. As such, according to the economist Fama, efficiency requires that stock prices be able to instantaneously adjust to the actual stock price [2]. Because of this, traditional economic theory would not classify the market as efficient.

In the view of traditional economic theory, the stock market is unstable and inefficient. This view is rooted in the idea that there is some vagueness and uncertainty about new, incoming stock information, meaning that there cannot be "instantaneous" adjustment. According to Fama, these adjustments have two major implications that disprove the idea of "market efficiency". Firstly, actual prices will initially over-adjust in intrinsic value as often as they will under-adjust. Secondly, the lag in the complete adjustment of a stock's value to its actual intrinsic value will be independent and random Thus, according to these two observations, successive prices changes in individual securities will be independent of past prices, which is the definition of what Fama called a "random walk." Based on the ideas of the random walk and market efficiency, a series of stock price changes does not have any memory. Later on, Fama would expand these theories into his Efficient Market Hypothesis. Basically, a major implication of the EMH is that consistently comes at substantial financial risk [2].

2.1: LOOKING FOR MARKET PREDICTABILITY

However, research following Fama's findings in the 1960s have shown that the market does have some level of predictability. One level of this predictability is the fluctuation of price between the bid and ask, which is known as the bid-ask spread. The bid price is defined as the price an individual is willing to pay for a stock, while the ask is the price at which an individual is willing to sell a stock. The true value of the stock lies somewhere between the bid and ask [3]. In an efficient market, the bid and ask fluctuate randomly. However, according to Timmerman and Grange, the existence of a single successful trading model would be sufficient to demonstrate a violation of the EMH [4].

There are several studies indicating success at using pricing data in order to predict future price movements. Phua et al utilized neural networks to predict the movement of five major stock indices: DAX, DJIA, FSTE-100,HIS and NASDAQ. Their rate of accurate prediction using only stock prices as inputs exceeded about 60% [5]. In his study, Kim applied a support vector machine (SVM) to predicting daily Korean stock market prices, yielding an accuracy of prediction of about 56% [6]. Huang et al applied backpropagation neural techniques to the Japanese NIKKEI index, achieving predictive accuracy greater than 70% [7]. These successful trading models would support Timmerman and Granger's theory that the market does have some time periods of predictability.

Timmerman and Granger extrapolated on time periods where market predictability could be found. According their work, if there are instances of high probability patterns during a time-span, they will be more likely to be spotted over time and manipulated by traders in their trading strategy. Yet, this widespread adoption of similar approaches to trading strategy would likely increase or decrease stock prices enough that the pattern would be elimination, making the price randomized. Such a line of logic does leave room for potential "hot spots," where prediction

methods are possible to implement. Timmerman and Grange suggest possible procedures to find these “hot spots” by using wide searches across both models that adapt quickly and a large number of assets. They conclude that traders who first implement new financial prediction methods (before the pattern is eliminated by widespread adoption) are the most likely to be successful, leading to the necessity for what Timmerman and Grange call “the race for innovation.” [4]

2.2: METHODS OF STOCK PREDICTION

Driven by the desire to predict market movements and reap profits, there are three different trading schools of thought: fundamental, technical, and quantitative technical analysis.

2.2.1: Fundamental Analysis

Fundamental analysis involves the examination of economic factors that influence the price of a stock. Such factors include a balance sheet and income statement. The balance sheet is a financial statement that provides information about a company’s assets, liabilities as well as the equity of their shareholders at a specific point in time. Basically, the balance sheet gives intel into what a company owns and owes and the amount investors have invested in it. The income statement is another type of financial statement that gives a synopsis of a company’s performance by providing information about their revenues, expenses, and net profit/loss over time. These reports are released quarterly throughout the year. Because fundamental analysis relies on reports that are issued on the basis of a slower timeframe, this type of analysis is often used to project long-term price movements [8].

1. ***2.2.2: Technical Analysis***

The goal of technical analysis is to anticipate what other stock holders are thinking based on available information about the price and volumes of stock. Technical analysists use a number of different types of indicators calculated from the past history of stock price and volume to predict future prices. Overall, the key to technical analysis is trend. Practitioners of technical analysis argue that trends in stock prices are caused by an imbalance between the supply and demand of stocks, which is reflected in the bid and ask prices. From the noisy data of stock prices, technical analysists attempt to extract patterns. Technical analysis is largely qualitative because it relies on the visual analysis of stock charts [9]. Two examples of such stock charts are shown below. Fig. 1 represents historical price data for stocks for the IT sector for one year. (It should be noted that I have chosen the IT to train an algorithm because stocks in the same sector usually exhibit similar price movements). Fig. 2 represents a candlestick chart. Each bar or "candle" represents one day's high, low, and closing prices.

Fig. 1: One-Year IT Stock Prices

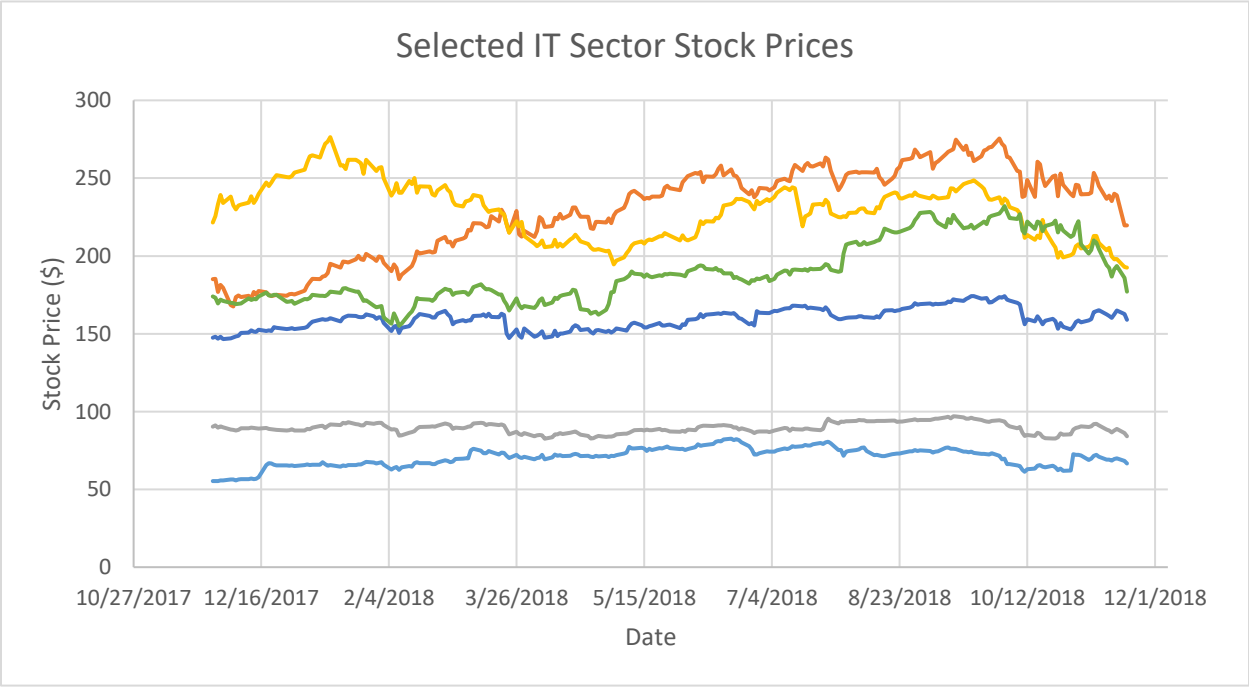
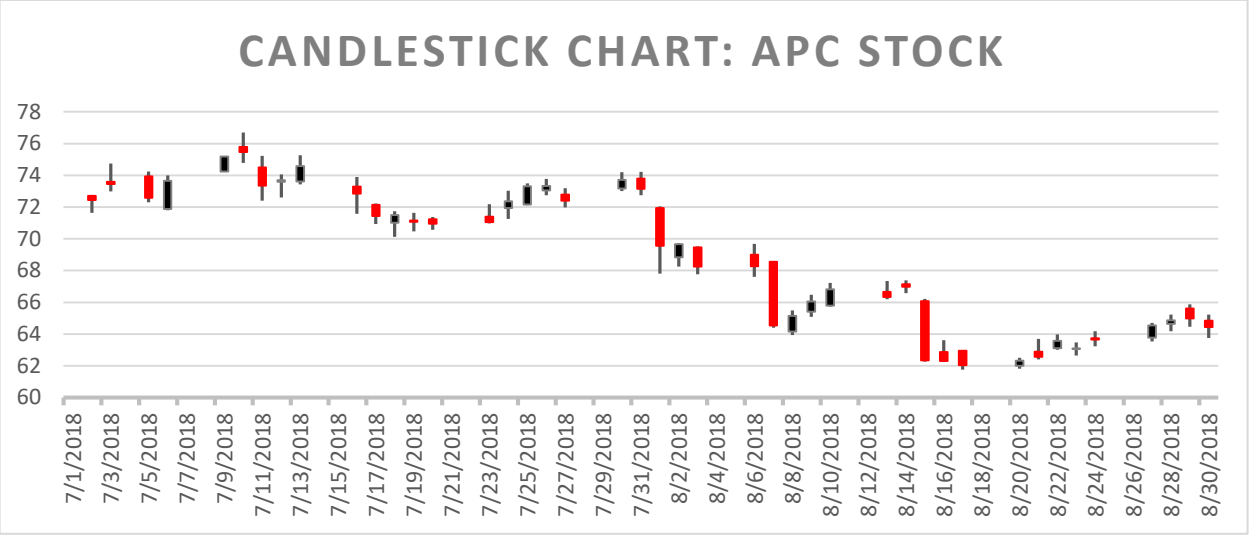


Fig. 2: Example IT Stock Candlestick Chart



This qualitative aspect to our second school of thought is what differentiates it from our next methodology. (Please note that we will revisit our discussion of technical indicators later on this paper).

2.2.3: Quantitative Technical Analysis

My capstone project explores this form of stock prediction. As suggested by its name, this form of stock prediction relies on quantitative methods of prediction rather than visualizations on graphs. Specifically, I will explore the usage of machine learnings algorithms to predict future stock prices. [10]

2.3: OVERVIEW OF MACHINE LEARNING

Machine learning is defined as a subsection of artificial intelligence that uses algorithms to find patterns in data to make predictions about future events. In machine learning, a dataset of observations called *instances* are comprised of a number of variables called *attributes*. Within

the area of machine learning, I will be using *supervised learning.* Supervised learning means that we model datasets that contain *labeled instances.* In other words, each instance can be thought of as a datapoint *(x,y).* Each *x* is an independent attribute that is our input, while the *y* is our correlated target attribute (or output). Supervised learning models train on historical data that is already labeled. Once, the model is trained, it can then be utilized on new data for prediction. The new input data contains only known feature values, and it has to predict the output. Since the future direction of the stock market becomes known after each instance, we are able to apply a supervised learning model for prediction.

Overall, the layout of a simplified supervised learning problem can be broken down into the following steps:

- Data acquisition
- Data cleaning
- Train/test split (type splitting our data into a training set and test set)
- Training the model (using the training set)
- Testing our model's accuracy (using our test set)
- Adjusting the model's parameters (including the number of epochs, batch size, number of neurons, optimization function, loss function etc.) based on performance
- Deploy the model for the prediction of future values [11]
- Model evaluation (evaluate the prediction results of our model by calculating the error between the real and predicted values)

1. Limitations of Machine Learning

Although there has been measured success with the implementation of machine learning algorithms, there are still some potential pitfalls of using this modeling method. One root of these potential issues is the instability of the markets. There may be brief periods where the market is highly predictable under certain approaches, but as more traders spot a certain pattern and adjust their technique to it, the patterns may cease to be predictable—a phenomenon called "concept drift" [12]. To avoid concept drift, a good predictive model would have to anticipate concept drift. One way to do so would be creating and storing a number of models that apply to specific market conditions and constantly updates based on incoming prices and throwing out old data. However, maintaining models with the most up-to-date price data is not necessarily the best approach. The market may stability and old knowledge may become useful again, meaning that the most recent price knowledge may not be the most pertinent to the current market situation. Given this consideration, another possible option to avoiding concept drift would be to retrain with ever-growing set of data. Still, this option is not the most feasible because constantly retraining on a huge dataset is time-consuming, which is an especially negative consequence in high-frequency trading where timing matters to profits. Thus, the ideal solution to concept drift is assuming that it occurs and building this assumption into our model [13].

3. DESIGN REQUIREMENTS

At the beginning of this project, I outlined a set of design requirements that my framework should satisfy. These requirements are based on the technical considerations of using a machine learning algorithm and the potential practical applications to markets. These design requirements are given in the following subcategories.

3.1: PERFORMANCE

The performance of the stock prediction framework is two-fold. Firstly, the algorithm should be able to use machine learning to consistently and accurately predict the movement of stock prices. Specifically, the algorithm should have a prediction accuracy greater than 50% (i.e., better than guessing).

Secondly, the framework should be able to leverage the predictions made in the first part of the framework. This specification relates to the practical application of the framework to the market and trading stocks. The framework should be able to utilize price predictions to provide a simple, optimal trading strategy. Specially, the system should be able to take multiple input stocks, predict their prices, and output a table. The table should be organized based on whether the algorithm believes the stock prices is going to increase or decrease. Furthermore, these rising and falling stocks should be sorted according to the predicted magnitudes of their movement. Such outputs should enable an investor to “long” the top rising stocks and “short” the falling stocks.

Thirdly, the framework should be flexible, meaning any user should be able to customize it to perform their desired stock analysis. The user should be able to input any number of stocks desired to predictive analysis. These inputs should be any type and number of features desired by the user. The user should be able to customize some parts of the design

internal to the network, including the size of a sliding window or the use of an ensemble for prediction. Additionally, the user should be able to customize their desired outputs. The user should be able to specify the number of days into the future they wish to predict as well as the number of stocks they want the algorithm to output (indicating the "top" and "bottom" stocks).

3.2: MATERIALS

The system should be easy to use and inexpensive. The algorithm should be able to run on a personal computer (which can give an added bonus of portability). The system should use cloud computing to perform its calculations. (It should be noted that cloud computing is beneficial because it does not interfere with other processes that the user may be carrying out on their computer, and it is fairly inexpensive to use).

The framework should be implemented in a free or inexpensive programming language. This language should provide the necessary tools to implement a machine learning algorithm with minimized difficulty.

The data used in this project should be readily accessible to the public, free, and be reflective of overarching economic factors as well as historical price data.

3.3: ECONOMIC

The economic considerations of this project are based solely on the costs of the materials for implementation. The overhead costs should be low because the purpose of the system is to return profits. Thus, the user would have a better rate of return on profits by avoiding incurring huge costs from their trading system.

2. DESIGN ALTERNATIVES

In the scope of this project, there were numerous design alternatives that were considered and tested. The first step was to select a programming language on which to implement my code. The second step in the design process required the selection of a machine learning framework to put into place. The third step required selecting a mechanism to mitigate the phenomenon of concept drift. The fourth step involved selecting the types of data that I wanted to utilize. These alternatives will be explored in the following four parts.

4.1: SELECTING A PROGRAMMING LANGUAGE

In selecting a programming language, there were several factors that I took into account, including simplicity as well as the availability and quality of machine learning packages. MATLAB, R, Python, and Java are amongst the most popular languages. For the sake of comparison, I have broken down each language's attributes into Table 1. below [14].

Table 1: Comparison of Programming Languages

Language	Attributes
Python	Simplicity: Syntax is intuitive and easily learn, and many AI algorithms can be easily implemented Supports object-oriented, functional and procedure styles of programming Effective and accessible machine learning libraries
R	Effective at analyzing and manipulating data for statistical purposes Can create high-quality plots, mathematical symbols, and formulas Many machine learning machine learning packages
Java	More difficult syntax Easy for debugging Robust library and package services
MATLAB	Good for working with and representing matrices

Ultimately, I selected Python. Python provides both an easier and faster way to build highly performing algorithms, especially since it provides a huge collection of specialized libraries. Companies likes Google have even developed open-source machine learning libraries for Python, which gives credibility to the strength and robustness of these libraries. Indeed, Python in general is the most popular coding language amongst data scientists—another indicator of the language's measured success [14].

After selecting Python as my coding language, I had to choose amongst its various libraries [15],[16],[17],[18], [19]. Again, I broke down my option into a table, which is shown in Table 2 below.

Table 2: Python Library Comparison

Library	Attributes
Theano	Allows evaluation, optimization and definition of mathematical expressions involving multi-dimensional arrays Written to take advantage of how computer compiler functions
Scikit-Learn	Designed to inter-operate with scientific and numerical libraries like SciPy and NumPy Offers range of supervised and unsupervised learning algorithms Built on SciPy Focused on modeling data, but not on summarizing, manipulating and loading data Popular model groups: supervised learning, manifold learning, parameter tuning, feature selection, dimensionality reduction, clustering, cross-validation
TensorFlow	Famous open source deep learning library developed by Google Brain team within the Machine Learning Intelligence Research organization of Google Blend between network specification libraries and symbolic computation libraries Computational framework used for expressing algorithms that involve numerous Tensor operation (Neural networks are expressed as computational graphs, so they are implemented in series of Tensors, which are N-dimensional matrices that represent data). Offers utilities for effective data pipelining, consist of built-in modules for serialization, visualization, and inspection of modules Also incorporate support for Keras
Keras	High-level neural network application programming interface written One of the most user-friendly libraries for building neural networks

	Runs on top of Theano, Cognitive Toolkit, or TensorFlow Enables faster experimentation Supports both recurrent neural networks, convolutional neural networks, or combinations of the two Enables fast and easy prototyping
Pytorch	Relatively new library (notable as Facebook AI Research Team backs it) Can handle dynamic computation graphs (a feature absent in TensorFlow, Theano and derivatives) Provides flexibility as deep learning development platform Easy-to-use API that integrates smoothly with Python data science stack (similar to Numpy) Offers framework to alter computational graphs during runtime (valuable when the amount of memory required for building network is unknown) Multidimensional arrays known as Tensors (has various types of Tensors)

The decision for what Python libraries I would use was impacted by not only these factors, but also my decision as to what machine learning model I wanted to use (which I will discuss in the next section). I chose to use Keras with Tensorflow as a backend. Ultimately, Keras supports the type of neural network I wanted to implement. The use of tensors for computation was ideal for the size and type of datasets I wanted to utilize. Furthermore, Keras's provision for fast and easy prototyping was ideal for working within the time constraint of this project, where fast experimentation proved to be necessary.

4.2: SELECTING A MACHINE LEARNING MODEL

There are numerous machine models that exist. Broadly, these models can be broken down into three different types of categories: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning consists of creating a function that maps input independent variables to an output target variable. Unsupervised learning is implemented when no target variable exists. Reinforcement learning trains itself continually using trial and error. In the context of this problem, my data is historical data, and the target variable is future stock price [19]. Thus, this project falls into the category supervised learning. A literature review revealed a

wide variety of models that have been used in stock price prediction. I decided to start by exploring the simplest model, linear regression, and progress to researching more complex neural networks.

4.2.1: Linear Regression

Linear regression is a linear model that assumes a linear relationship between our input variable (x) and output variable (y). Our linear equation ($y = mx + b$) assigns a weighting/scaling value (m) to our input. There is an additional coefficient, b, that acts as a bias term, which gives the line an additional degree of freedom. As the linear regression algorithm trains on our training data, the bias and weighting terms are adjusted. The goal of the algorithm is to get a line of best fit for the data, where the best fin tine has a total prediction error that is as small as possible. The error is calculated by finding the distance between the point and the regression line. A cost function helps to figure out the best possible values for the weight coefficient and bias coefficient. This is a minimization problem, where we want to minimize the error between the predicted value and the actual value [20]. A literature review revealed that linear regression is a somewhat common form of stock market prediction. Its main purpose seems to be as a basic baseline model. Typically, its performance is worse (providing a large amount of predictive error) than those of more complicated models [21], [22], [23].

4.2.2: Feed-forward Artificial Neural Network

The next machine learning method I explored was the Artificial Neural Network. This type of network has units that mimic an artificial neuron, which is called a *perceptron*. An example perceptron is shown in Fig. 3 below:

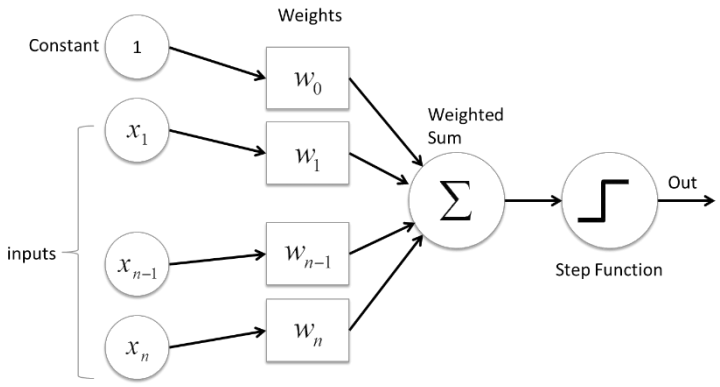


Fig. 3 : Example perceptron.

The perceptron consists of multiple parts: input layer (whose values come from our input features), weights and bias, the net sum, and an activation function. The perceptron inputs (x_j) are multiplied by their respective weights (w_j), which are real numbers that express the importance of each input. The result of this multiplication is, then, added together in the weighted sum $\sum_j w_j x_j$. These results are then passed to an *activation function,* and the output is determined by whether the weighted summation is less than or greater than some threshold value determined by the activation function. The purpose of the weights in our perceptron is to show the strength of a particular node, while the bias values allows the activation function curve to be shifted up or down. A neural network's activation function is to map the output to a value within a certain range specified by the function [24].

A literature review revealed that the performance of feed forward neural networks significantly exceeded that of linear regression. While the linear regression models typically showed prediction root-mean-squared errors (RMSE) in varied ranges much greater than 1, typical feed forward networks in my research achieved RMSEs typically less than one. For instance, the results found by Weng showed consistent RMSEs less than 1 for predictions using

feed forward neural networks [21],[22], [25]. Furthermore, Weng's overall system achieved an average of 60% accuracy of stock price prediction.

4.2.3: Recurrent Neural Network

The fee forward neural networks discussed in the previous section merely feed input values forward through the network; there are no feedback connections. The issue with a mere feed-forward type of network is that information can get lost over time, meaning that greater emphasis is placed on more recent data. For stock price prediction, this can be an issue. The most recent knowledge is not necessarily the most pertinent; sometimes past information can be more pertinent to current market patterns. As such, I expanded my research into networks with feedback capabilities—the recurrent neural network.

. Unlike other ANNs, RNNs have loops in them to allow past information to persist.

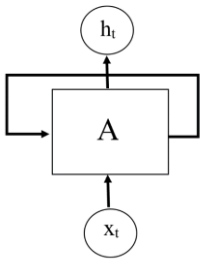


Fig. 4: Simple model of Recurrent Neural Network (RNN).

Fig. 4 above shows a simple model of a recurrent neural network, *A*, with input x_t and output h_t. Note that it contains a loop, enabling information to be passed between different steps of the network. An RNN is basically a chain of multiple copies of the unit shown above, so "unrolling" this unit yields the following architecture:

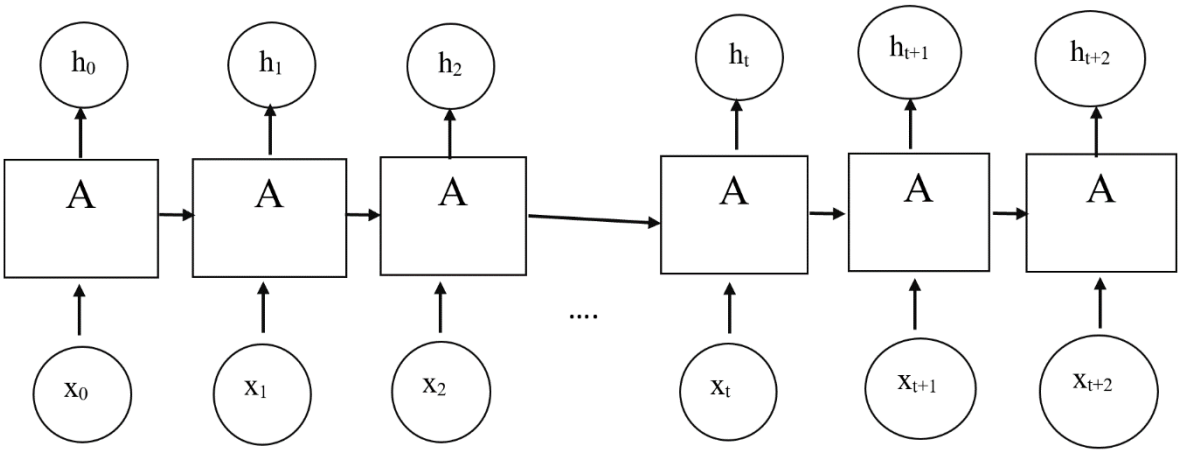


Fig. 5: "Unrolled" recurrent neural network.

Because of the chain-like architecture of the RNN, it is useful for analyzing data with a similar structure, such as time-series data like stock historical stock prices. While LSTMs are a popular choice for their predictive accuracy, the network shown in Fig. 8 above still does not totally avoid the issue of long-term dependencies that we have been concerned with thus far. As the gap between the units of our RNN grows, it starts to "forget" past information, so the RNN still fails at learning long-term dependencies. The simple architecture (exemplified by our single tanh layer) of the LSTM is shown below:

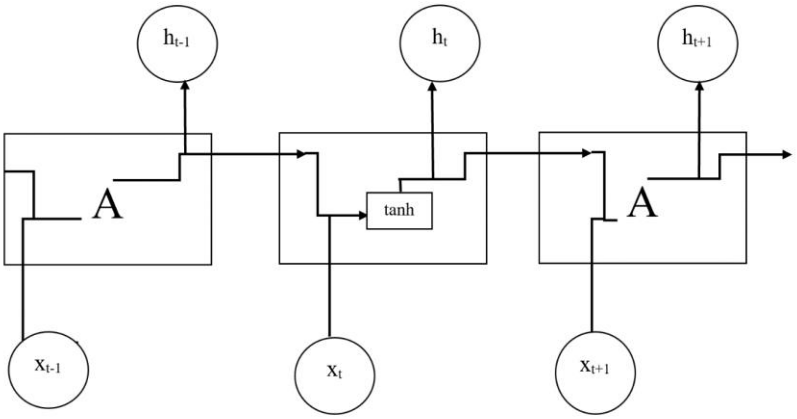


Fig. 6: "Unrolled" LSTM showing simple internal architecture.

The solution is a variation of the RNN: the Long Short Term Memory networks (LSTMs). The LSTM still has a chain-like architecture, however, it has added complexity. Instead of one neural network layer, it has four, as shown below in Fig. 10:

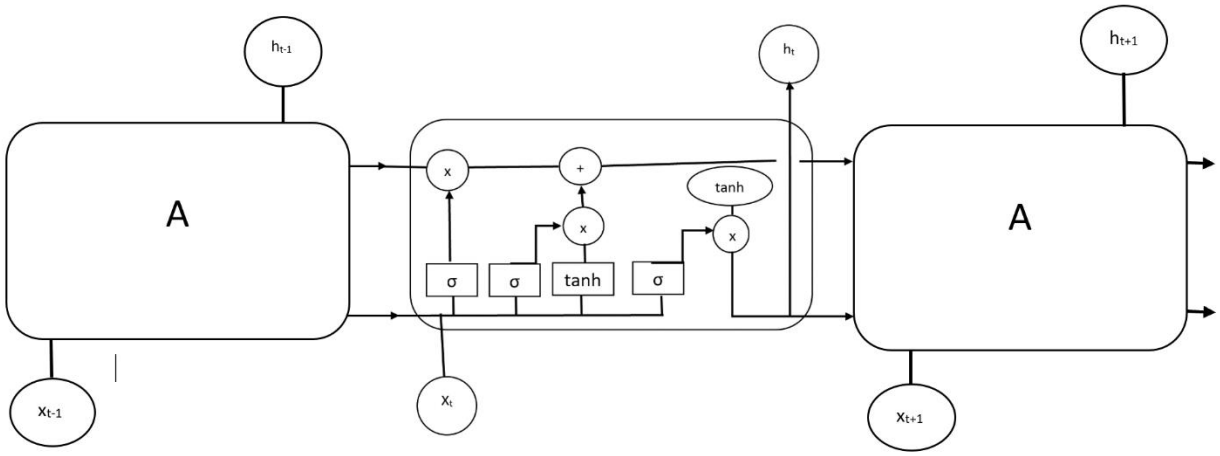


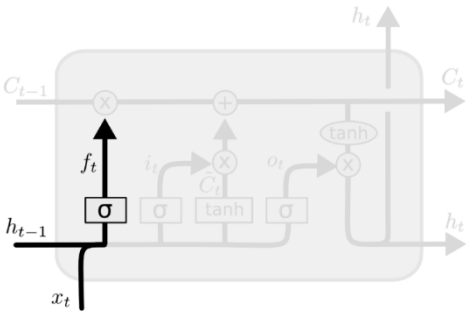
Fig. 7: Internal architecture of Long-Short-Term Memory Network.

LSTM Steps:

1. Step One: "Forget Gate"

The first step in our process is shown in Fig. 11 below and is represented by the corresponding equation:

Fig. 8: “Forget Gate” of RNN-LSTM.



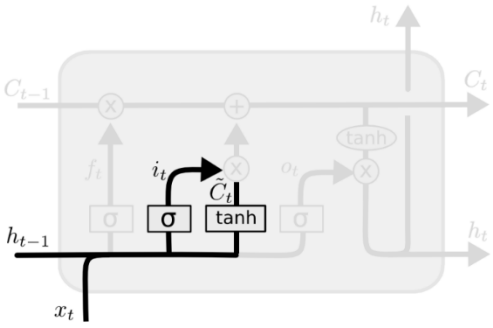
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The first step is known as the “forget gate” layer. The inputs to the gate are the output from the previous cell (h_{t-1}) and the current input (x_t). The output is *f(t),* whose value (on a scale between 0 and 1) determines how much of the previous information will be “remembered.” A zero means that none of the information is kept, while a one means all of the information is kept.

2. Step Two: Cell State

The second step in our process is shown in Fig. 12 below and is represented by the corresponding equations:

Fig. 9: Updating the Cell State of RNN LSTM.



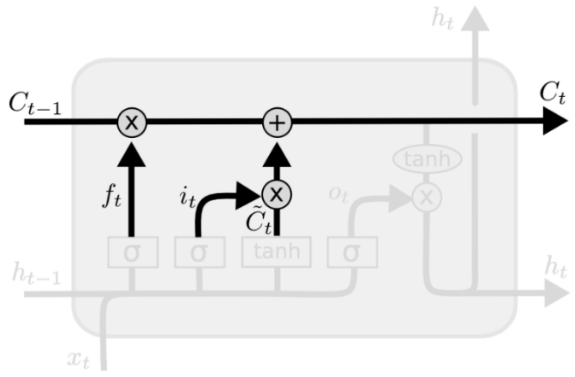
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

This step contains two layers. The first layer is the "input gate," which determines which values will be updated. The tanh layer determines creates a vector of values denoted, C_t, that are possibilities for being added to the current cell state.

3. Step Three: Updating the Cell State

The second step in our process is shown in Fig. 13 below and is represented by the corresponding equations:

Fig. 10: Calculating New Cell State for RNN LSTM.

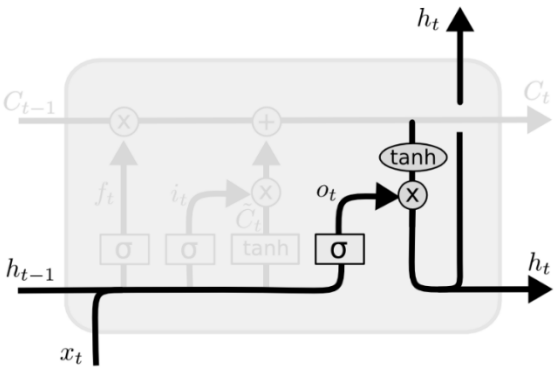


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

In this step, we want to take the outputs from step 2 and perform the operation pictured to update the old cell state, $C_{t-1,}$ to the new cell state, C_t. First, we take the output of the "forget gate," *f(t),* and multiple it by C_{t-1} to "forget" the desired previous information. Then, this output is added to $i(t)*C_t$ (which represents the candidate values multiplied by how much the network decided to update the state values). Our output for this is the new cell state, $C_{t.}$

4. Step Four: Calculating the Output

The second step in our process is shown in Fig. 14 below and is represented by the corresponding equations:



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fig. 11: Calculating Output RNN LSTM.

The output is based on some additional calculations on the new cell state. First, cell state included in the calculation is determined by passing through the sigmoid function pictured. Next, this cell state is multiplied by the tanh function before, finally, being multiplied with the output of the sigmoid gate, yielding our output, h_t. [9].In some cases, the literature review showed that an RNN and its "selective memory" capabilities provided some greater predictive accuracy [26], [27], [28]. For instance, in his paper, Aamodt utilized an LSTM hybrid. This hybrid was able to achieve relatively low error with directional accuracies greater consistently above 60% [27]. Additionally, work by Selvin et al shows that an LSTM showed the least amount of prediction error when compared with other neural network architectures like a regular RNN and Convolutional Neural Network (CNN) [28]. Thus, the LSTM seems to have a measured degree of success in similar stock price applications as compared to other networks. Additionally, its "memory" is a built-in method of likely mitigating concept drift. Taking into consideration all of these factors, I chose the LSTM as my model framework.

4.3: SELECTING A MECHANISM TO COMBAT CONCEPT DRIFT

Although the LSTM does have a "memory," I wanted to explore other alternatives to mitigating concept drift. The markets are not stable, meaning indicators that might have been predictive at one moment will disappear as other investors spot the pattern and implement it in their trading strategies. As the underlying concept in data changes, the model's performance may decline over time. There are several methods that can combat that I will discuss in this section.

The first alternative is retraining the model to new data. For stock data, this is not the best option. Depending on the amount of input data, the training time could be time-intensive, which is not ideal for applications like stock trading that require speed to make faster decisions in order to find "hot spots" of predictability[4]. An alternative that is similar to retraining is updating the model with recent data at checkpoints. Since end-of-day stock prices are released on a daily basis, the model can be updated on a daily basis. By maintaining a checkpoint in the model, the checkpoint can be used as a point to add in new training data without re-training the whole dataset, i.e., the algorithm will pick up where it left off at the checkpoint[29].

The second alternative is to use a sliding window, which is represented in Fig. NO below. The sliding window uses a specified window size *w* from times *t*-*w*+1 to *t* (where t represents the "current time").

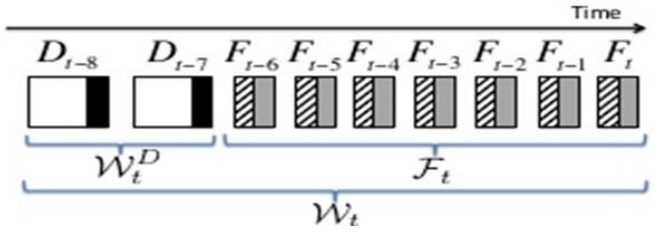


Fig. 12: Sliding Window

The model will train at each window with the specified performance metrics before sliding to the next set of data that can fit in the window. Using a small sliding window is, theoretically, supposed to reflect the current distribution of data and prevent outdated information from affecting the model. Conversely, a larger window will contain more training instances and be able to perform better during moments of stability [30]. Thus, there is a balance in determining the fixed window size. If the window is too small, the classifier will not contain a large enough number of instances and will overfit. If the window is too large, the classifier may be built using data that contains too many different concepts. There will be further discussion about the window size as it relates to testing of the actual model.

4.4: SELECTING INPUT FEATURES

Feature engineering is key to machine learning. By definition, feature engineering is the process of transforming raw data and features that better represent the underlying problem to predictive models with the goal of improving model accuracy [32]. Basically, this means that we must properly select our input values/features. Because a machine learning framework is learning a solution to a problem based on sample data, the goal is to input the best representation of the data that will learn the solution to the problem. With stock data this is both pivotal and challenging because stocks are predicted by so many varying factors. In order to determine the features to be input into the algorithm, a large amount of testing was required. Thus, in this section, I will briefly discuss the potential features and combinations of features that I decided to utilize as inputs.

The target variable is the daily closing stock price. Thus, historical closing stock prices are a necessary input. Thus, one necessary combination of inputs are the historical daily close, adjusted close, opening, high, and low prices as well as volume. The open and close prices

represent the end-of-day and start-of-day stock prices, and the low and high prices represent the stock's minimum and peak prices within the span of a day respectively. The volume indicates the number of stocks that were bought and sold on a given day.

Again, because stock prices are highly influenced by a variety of market forces, choosing disparate data sources could be key to improving the accuracy of the model. Significant stock market movements are often driven by investor perceptions of stock based on information collected from various data sources. Because stock price is based on investor sentiment, I sought out data sources that are freely accessible to the public. Thus, my chosen source for data was Yahoo Finance, where historical stock price data can be accessed.

The next type of data that was experimented on was technical indicators, which are used widely by quantitative traders and could give insight into their trading strategies as they effect market movements. Technical indicators are mathematical calculations that can be applied to a stock's past patterns (such as price and volume). The two general categories of technical indicators are leading and lagging. Leading indicators give trade signals where a trend is supposed to start, while lagging indicators are those that follow price action (meaning they give a signal after a trend or reversal has started). Under these two major categories, there are four other types of technical indicators: trend, momentum, volatility, and volume. Trend is used to measure the direction and strength of a trend using some form of price averaging to establish a baseline. As price moves above the average, this indicates a bullish (upward) trend; as price moves below, this indicates a bearish (downward) trend. Volatility indicators measure the rate or price movement, regardless of the direction. This is generally based on a change in highest and lowest historical prices. They provide useful information about the range of buying and selling that take place in a given market and help traders determine points where the market may change

direction. Volume indicators measure the strength of a trend or confirm a trading direction based on some form of averaging or smoothing of row volume. The strongest trends often occur while volume increases; in fact, it is the increase in trading volume that can lead to large price movements[33]. The key to utilizing technical indicators is by choosing indicators that complement each other and are not redundant. The technical indicators used in the feature testing are listed in Table 3. below.

Table 3: Types of Technical Indicators

Type of Indicator	Indicator	Description	Leading or Lagging?
Trend	Moving Average	Used to identify current trends and trend reversals as well as support and resistance levels in the market.	Lagging
	Exponential Moving Average	Weighted moving average that give more weighting to recent data than simple moving average	Lagging
	Parabolic Stop and Reverse	Used to find potential reversals in market direction	Leading
Momentum	Aroon Oscillator	Gauges strength of current trend and likelihood that is will continue	Leading
	Commodity Channel Index (CCI)	An oscillator to help identify price reversals, price extremes and trend strength	Leading
	Relative Strength Index (RSI)	Measures the stock's recent trading strength, velocity of a change in trend and magnitude of the move	Leading
Volatility	Average True Range (ATR)	Indicator does not provide an indication of price trend, simply the degree of volatility.	Lagging
Volume	Chaikin A/D Oscillator	Monitors flow of money in and out of the market Compares money flow to price action to help identify tops and bottoms in short and intermediate cycles	Leading

The next set of data that I utilized relates to market fundamentals. This type of data goes beyond trading patterns of the stock itself, but these fundamentals still usually are expected to have some impact on the value of the stock itself by influencing investors. Fundamental data is

generally released on a quarterly basis with company balance sheets. These balance sheets reports a company's assets, liabilities and shareholders' equity. Such financial statements show how a company performed in a given quarter, meaning they could impact the stock's price in the next quarter. The fundamental factor I chose to incorporate is the price-to-earning (P/E) ratio. The P/E ratio is equal to the market value per share divided by the earning per share. The P/E is expressed as a multiple of its earnings, so it can give a look at the overall health of the company [8]. This fundamental was used in testing in combination with the technical indicators and price data.

The final data that was used to predict stock prices included data that is reflective of the economy's overall economic health. The first is the ISM manufacturing index. Based on over 300 manufacturing firms, this index monitors the employment, production, inventories, new order and supplier deliveries. This index is released at the start of each month, impacting the confidence of businesses and investors [34]. The second macroeconomic factor is the University of Michigan Survey of Consumers, a highly regarded and trusted consumer sentiment index. It reflects the strength of consumer confidence and spending, providing another indication of overall economic health [35]. The third macroeconomic factor is the number of released housing permits. This shows the strength of the U.S. household and, thus, consumer. The number of permits issued is also reflective of the how much banks are able to lend, which is reflective the Gross Domestic Product, or GDP [36]. The GDP is the primary indicator of a nation's economic health. It represents the total value of all the goods and services provided by a country over a given time frame. More succinctly, it is referred to as the total size of the economy. [37] Thus, the fourth macroeconomic factor is the ISM Non-Manufacturing Index, which measure employment trends, prices and new order in non-manufacturing industries.. These factors were used in combination along with the fundamental data

to see if indications of overall economic health with historical prices would help the training algorithm gain further insight and more accurate stock price predictions.

5. Preliminary Proposed Design

The preliminary proposed a machine learning framework implemented in Python using the Keras library (on top of Tensorflow). The first portion of the code is the predictive algorithm, which is outlined in the flow diagram in Fig. 13 below:

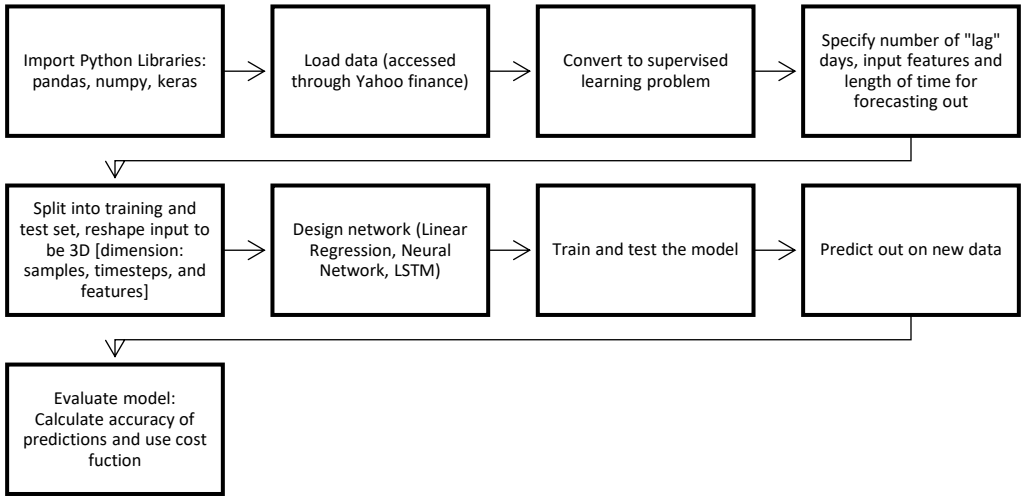


Fig. 13: Initial Code Flow Diagram

This code is able to read in a csv file the desired inputs downloaded from Yahoo Finance (for a stock over a desired timespan). The user specifies the desired number of "lag" days, the input features, and the number of days we want to predict ahead. The input data is time series data, so it must be converted into a supervised learning problem (i.e., a target variable must be created). This is done by copying our desired output data (the closing price) and assigning it to be the output variable. Then, this copied data is shifted up by the number of days we want to predict out on. The pseudocode for this function is outlined in the figure below:

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
"""
```

```
Frame a time series as a supervised learning dataset.
Arguments:
        data: Sequence of observations as a list or NumPy array.
        n_in: Number of lag observations as input (X).
        n_out: Number of observations as output (y).
        dropnan: Boolean whether or not to drop rows with NaN values.
Returns:
        Pandas DataFrame of series framed for supervised learning.
"""
n_vars = 1 if type(data) is list else data.shape[1]
df = DataFrame(data)
cols, names = list(), list()
# input sequence (t-n, ... t-1)
for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
# forecast sequence (t, t+1, ... t+n)
for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
                names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
                names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
# put it all together
agg = concat(cols, axis=1)
agg.columns = names
# drop rows with NaN values
if dropnan:
        agg.dropna(inplace=True)
return agg
```

Once the data has been converted into a supervised learning problem, the data is split into a training and a test set. The training set is by the algorithm to train on, while the test set is used to validate the algorithm's predictions as it trains. These sets are re-shaped to be three dimensional: [number of samples, timesteps, features].

After preparing the data, the LSTM network is designed. The various hyperparameters of the networks (which will be discussed later) are specified, and the input dimensions are set. Then, the training and testing data is fed into the model. Once the model is finished training, new data can be input to predict out on. The output are the closing day stock prices (for however many days the user specified). Given the output of these predictions, the model accuracy is

evaluated using percent error between the predicted price and actual price as well as a financial cost function. This cost metric is outlined in the diagram in Fig. 14 below:

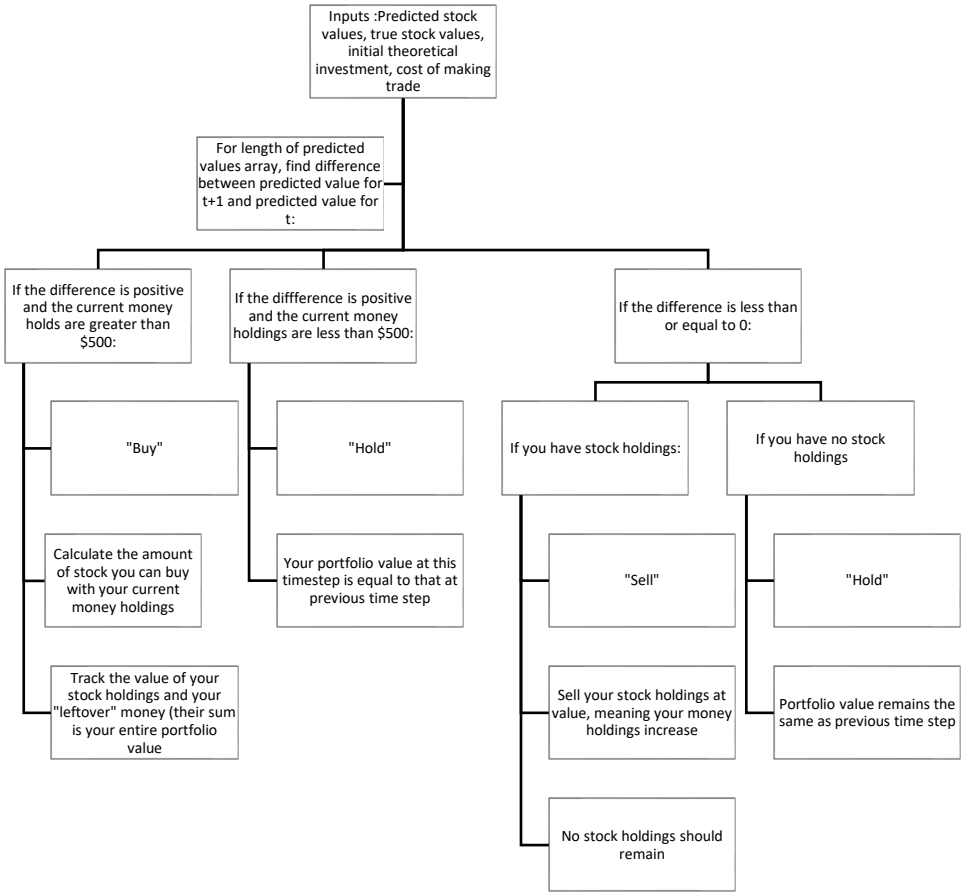


Fig. 14: Cost Metric

In general, the cost metric uses the algorithm's predictions to determine how well an investor would do by buying and selling stocks based on the predictions. The inputs to the function are the predicted stock values and the true stock values. (Since I was only training and predicting on historical data, I was able to compare the predictions to actual stock prices). The other inputs are a theoretical initial investment value and the cost of executing a trade. (Because trades are mostly executed by third parties, there is a cost to this trade execution). It should be noted that this function does daily tracking of how much money the investor has leftover and the value of the

investor's stock holdings; combining these two values provides the total portfolio. Basically, the algorithm first checks to see if the projected stock prices is predicted to go up or down. If the price is predicted to go up and the investor has money left, investor "buys" as many stocks as possible at its current price. If the price is predicted to go down, then the investor sells if the investor is holding stock or holds off if no stock is held. If no change is projected in the price or the investor does not have any money leftover, the investor holds. The output of this function is the investor's daily portfolio value and a calculation of the percent returns.

The output of this cost metric is compared with another function that creates a market baseline. Recall that one of the goals is to use the machine learning framework to make trading decision that will outperform the market benchmark. This market benchmark is determined by creating a function that will hold a stock over the time frame predicted out on. The function track the daily price fluctuations in the investor's stock holds as well as the money leftover that was not spent on the initial investment. Again, the output is the daily portfolio value and percent returns. These values are compared to the outputs of the cost function based on market predictions. The code is outlined in the diagram in Fig. 15 below:

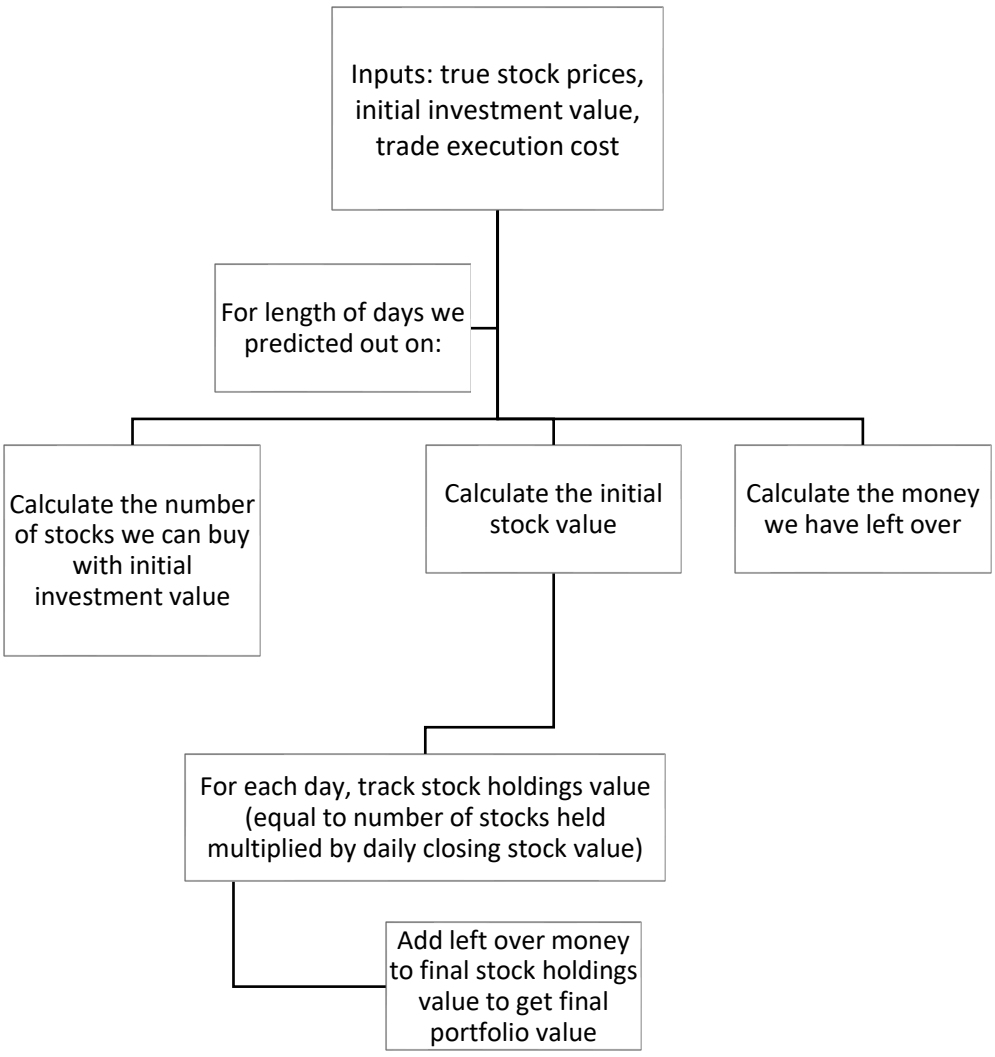


Fig. 15: Market Baseline (Holding Stock) Metric

6. FINAL DESIGN AND IMPLEMENTATION

In the final design and implementation of the stock prediction framework, several changes had to be made. As I was implementing my design, testing was carried out with varied datasets. Using the actual data itself revealed further problems with data-preprocessing that had to be resolved. Additionally, testing led me to realizations of how the framework could be improved. I will discuss these improvements categorically in the following sections to show how I arrived at the finalized design.

6.1: DATA PRE-PROCESSING

6.1.1: Data Normalization

During the initial testing stages, I used the open, close, adjusted close, high, low, and volume data. Some of this testing even included combining multiple stocks from the same sector to see if this correlated data would improve predictions. The resulting predictions had huge magnitudes of error, which is shown in the sample results in Table 4 below. This table shows that the changes in predicted prices are wildly inaccurate as compared to the true changes in Apple stock prices.

Table 4.: Sample Results Using Non-Normalized Inputs

True Stock Price Change	Predicted Stock Price Change	Percent Error (%)
-2.37	685.62	-29029.1
3.13	518.86	16477
1.4	308.05	21903.57
-1.77	298.45	-16961.6
4.53	426.90	9323.841
0.79	619.25	78286.08
`.52	513.78	98703.85

This error was due to a mismatch in the scales of the input values. The volume data was generally much larger in magnitude than the stock prices. When data from multiple stocks was used at once, the varied ranges of prices also created error. In order to amend this error, I began using data normalization. The normalization utilized rescaled all of the numbers onto a scale from 0 to 1. Normalizing resulted in significant improvements in the predicted stock prices that were within a much more reasonable range. (For these improved results please refer to Section VI).

6.1.2: Stationary and Non-Stationary Data

While the data normalization significantly improved the price predictions, there was still a large degree of error. While some of the predictions were on a reasonable scale, the algorithm too often prediction the wrong direction that the price was going to move. Further research showed the root of the issue was my use of time series data. An example of this huge degree of error in the predicted price due to the issue of utilizing nonstationary data, please refer to Table 5 below:

Table 5: Example Error in Predictions due to Utilization of Nonstationary Data

True Stock Price Changes	Predicted Stock Price Changes	Percent Error (%)
-2.37	387.66	16229.11
3.13	332.21	10513.74
1.4	309.65	22017.86
-1.77	294.65	16546.89
4.53	283.52	6158.72
0.79	274.86	34692.41
`.52	267.92	51423.08

Time series data often contains trends and seasonality. Trends indicate that a data's mean varies over time, while seasonality indicates that a data's variance is changing over time. Such data is classified as non-stationary. Machine learning algorithms can have difficulty prediction on non-stationary datasets (shown in Fig. 16 below). These frameworks prefer stationary datasets, where the mean and variance are stable over time (shown in Fig. 17 below).

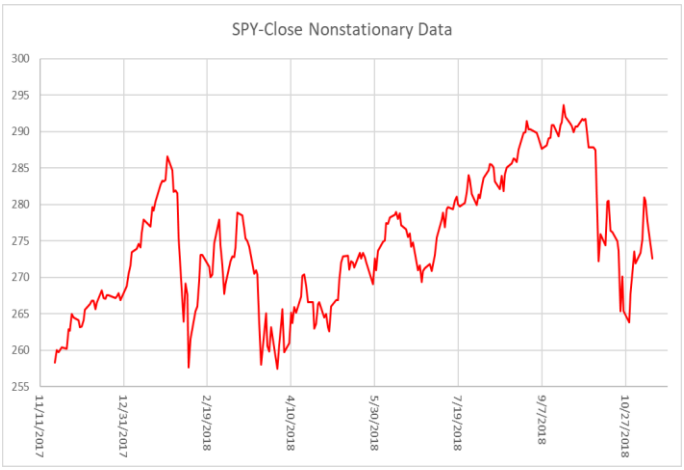


Fig. 16: Non-stationary data with underlying trend.

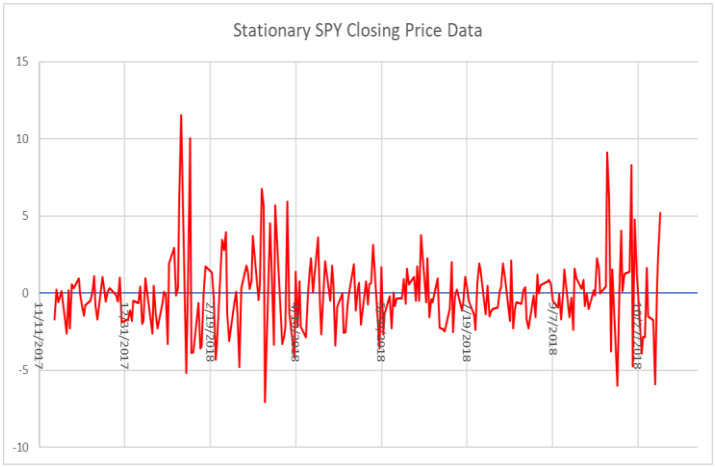


Fig. 17: Stationary data with no underlying trend.

Stationary data is easier to model, so I had to add another step to my data pre-processing—turning non-stationary data into stationary data. The time dependence and trends underlying data can be removed by using a simple difference transform. This transform takes the difference between sequential datapoints in the dataset. Thus, the transformed, nonstationary dataset is comprised of the daily price changes over time. This transform is also ideal for the application of stock price prediction, since investors care more about the changes in stock prices. Investors want to buy stocks when the price is lower and sell after the price increases, making a profit on the price difference.

Performing a difference transform on the data also prompted me to change the way my algorithm was making predictions. Instead of outputting a series of actual prices, my algorithm output an array of predicted price changes. The value of the last known day of stock prices is also stored. Then, this last known value and the first price change are added together to get the projected next-day price. This is done iteratively (i.e., we keep adding onto the previous day prediction) for each day where the magnitude of price change was predicted. This new method of prediction not only decreased the percent error the predictions, but also improved the algorithm's ability to predict the correct direction a stock price would move. Ultimately, predicting the correct price movement is more essential than a prediction of the magnitude of this movement due to the nature of the application of stock trading.

The final flow diagram for our final system including our data-preprocessing and new prediction method is shown in Fig. 18 below:

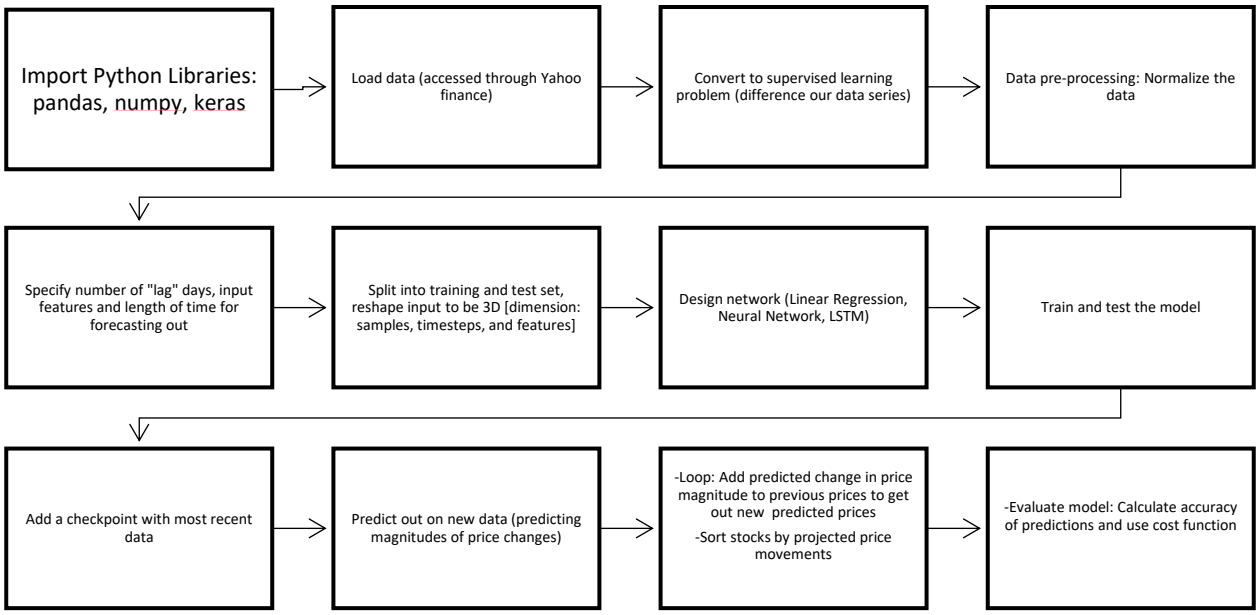


Fig. 18: Final Flow Diagram of Predictive Machine Learning Algorithm

6.2: TUNING NETWORK HYPERPARAMETERS

In machine learning, there are a number of hyperparameters that affect the quality of an algorithm's predictions. These hyperparameters include the following:

- Number of Epochs
- Batch Size
- Training Optimization Algorithm
- Learning Rate

- Dropout Regularization
- Number of Neurons in Hidden Layer
- Lag Window

The technique I utilized for optimizing my hyperparameters was a grid search. A grid search is a brute force method that can be applied across all machine learning models. Grid-searching builds on each possible combination of model parameters and stores a model for each. Once all of these models are built, their performance can be evaluated, and the optimal parameters selected for the final model. The following sub-sections will discuss the results of this hyperparameters search and explain the choice for the final algorithm design.

6.2.1: Number of Epochs and Batch Size

The number of epochs denotes the amount of times the entire set of input training data is passed through the neural network during training. It is necessary t utilize multiple epochs during training because this allows the network to change its weights during training. Using too few or too many epochs can cause the network to underfit or overfit; both of these scenarios cause the network to have poor performance. Basically, supervised machine learning attempts to approximate a target function that maps its given historical inputs to historical outputs. Because a function is being approximated, the function should just be a generalization as we only have an incomplete, noisy sample of data. This generalized function is referred to as inductive learning, meaning the machine learning algorithm is trying to learn general concepts that are underlying in the specific training data. This idea of generalized learning is important because it allows a machine learning algorithm to use these general concepts to make predictions on future data. In order for the algorithm to learn generalized concepts, we want to avoid overfitting, which means that the algorithm is too well trained on the training data. Overfitting means that the model has

learned too much detail about the training data (such as noise), so the model will be unable to generalize. As a result, the model will not be able to make accurate predictions on unseen data. Conversely, underfitting means that the model does not account for the underlying relationships in the given data, which also will result in poor model performance. The goal for selecting the number of epochs is avoiding overfitting and underfitting to optimize model performanceThe grid search was performed on ten stocks from the IT sector. The table below shows a sample of the experiments that were run. In the set of experiments shown in Table 6 below, the batch size was kept constant at size 50, and the number of epochs were varied. The best results were achieved using 10 epochs. (It should be noted that this also had a lag window of 1 and uses the Adam optimizer). Representative results of the extensive testing for day-ahead forecasts for IT sector stocks are shown in Table 6 below.

Table 6: Testing Number of Epochs for LSTM

Experiment No.	Days Forecast Out	Number Epochs	Stock	True Price	Predicted Price	Predicted Price Change	Error	Actual Price Change	Predicted Direction Price Change	Actual Direction Price Change
1	1	10	AAPL	227.259995	227.1260529	2.126053333	39.87285004	1.51999	Up	Up
		(Batch Size 50)	ADBE	275.48999	266.6028137	-2.397200346	143.2715454	5.5399	Down	Up
			ADI	92.389999	96.04856873	4.04857111	-5883.673014	-0.07	Up	Down
			ADS	237.839996	242.6718597	6.671862602	297.1346787	1.68	Up	Up
			ADSK	155.5	160.4196014	4.419604778	-824.5253735	-0.61	Down	Down
			AMAT	38.34	44.22901154	6.229011536	-2109.35856	-0.31	Down	Down
			AMD	31.42	34.35211182	4.352110386	721.152903	0.53	Up	Up
			ANET	259.640015	271.1420898	6.142082214	-198.7473025	-6.22	Down	Down
			ANSS	186	192.605011	6.605017662	871.3261268	0.68	Up	Up
			APH	94.470001	97.23419952	3.234198332	618.7107404	0.45	Up	Up
2	1	25	AAPL	227.259995	224.6044006	-0.395600379	126.026512	1.51999	Down	Up
			ADBE	275.48999	269.3029175	0.302916795	94.53208912	5.5399	Up	Up
			ADI	92.389999	90.00575256	-1.994250774	-2748.929678	-0.07	Down	Down
			ADS	237.839996	239.0663147	3.066320181	82.51905839	1.68	Up	Up
			ADSK	155.5	156.144928	0.144930139	-123.7590392	-0.61	Up	Down
			AMAT	38.34	38.7746048	0.774604201	-349.872323	-0.31	Up	Down
			AMD	31.42	29.77861595	-0.221383125	141.7704009	0.53	Down	Up
			ANET	259.640015	268.371582	3.371580362	-154.2054721	-6.22	Up	Down
			ANSS	186	187.0666809	1.066685557	56.86552309	0.68	Up	Up
			APH	94.470001	94.01959991	0.019601585	95.64409215	0.45	Up	Up
3	1	75	AAPL	227.259995	222.8591156	-2.140880823	240.8483492	1.51999	Down	Up
			ADBE	275.48999	267.0340881	-1.965920687	135.4865735	5.5399	Down	Up
			ADI	92.389999	90.44593048	-1.554069042	-2120.098631	-0.07	Down	Down
			ADS	237.839996	239.0659943	3.065999746	82.4999849	1.68	Up	Up
			ADSK	155.5	154.5875397	-1.412461877	-131.5511274	-0.61	Down	Up
			AMAT	38.34	39.65506363	1.655064106	-633.8916471	-0.31	Up	Up
			AMD	31.42	28.22669411	-1.77330637	434.5861075	0.53	Down	Up
			ANET	259.640015	266.959259	1.959252357	-131.499234	-6.22	Up	Up
			ANSS	186	187.3713837	1.371390224	101.6750329	0.68	Up	Up
			APH	94.470001	92.04884338	-1.95115447	533.5898822	0.45	Up	Up
4	1	100	AAPL	227.259995	223.010498	-1.98950243	230.8891789	1.51999	Down	Up
			ADBE	275.48999	267.7550354	-1.244970679	122.4728006	5.5399	Down	Up
			ADI	92.389999	92.06255341	0.062556989	-189.3671278	-0.07	Up	Down
			ADS	237.839996	237.5384979	1.538497925	8.42274256	1.68	Up	Up
			ADSK	155.5	156.6197815	0.6197837	-201.6038852	-0.61	Up	Up
			AMAT	38.34	38.22127151	0.221272737	-171.3783023	-0.31	Up	Up
			AMD	31.42	28.08531761	-1.914682627	461.260873	0.53	Down	Up
			ANET	259.640015	266.9857483	1.985735416	-131.9250067	-6.22	Up	Up
			ANSS	186	186.4579926	0.45799762	32.64740882	0.68	Up	Up
			APH	94.470001	93.77622223	-0.223781139	149.729142	0.45	Down	Up

The results showed that training using 10 epochs and 50 batches yielded about 70% in predicting the direction of next-day stock movements, though these day-to-day predictions still show a high degree of error. As the number of epochs increased, the prediction error for the direction that stocks would move quickly increased. An examination of the data as it trains over a number of epochs (as shown in Fig. 19) reveals that the accuracy quickly increases the model trains, and then levels off over time despite the increasing number of epochs.

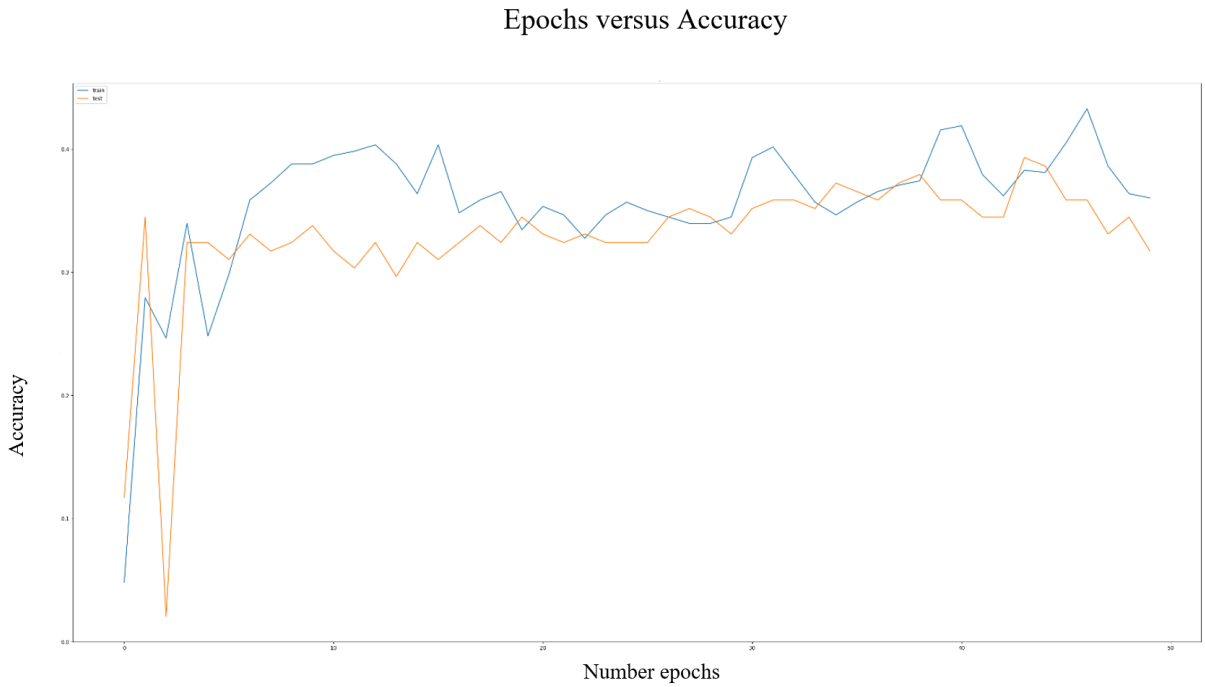


Fig. 19: Number of Epochs versus Accuracy of the LSTM. Note that the orange line denotes the test data, while the blue line denotes the training data.

6.2.2: Optimization Algorithm

The optimization algorithm is an important part of training a machine learning algorithm that works in tandem with the loss function. While an algorithm trains, the weights and biases of the network are changed, affecting how well the model models a given dataset. The output of the loss function is high if the network predictions based on the model are poor. Conversely, better predictions mean the loss function output is lower. A model's optimizer bridges the gap between updating a model's parameters and the loss function. The optimizer works by tuning parameters and molding the model based on the most accurate assignment of weight values. The optimizer uses the loss function as an indication of the quality of this tuning. The optimizer's goal is to minimize the loss function. Keras provides a number of optimization algorithms, which can

affect the performance of the overall mode. Thus, testing was conducted for the selection of the best optimization algorithm. The different algorithms tested are discussed in the following sections below. Please also note that some sample results from our IT sector stocks are also provided in Table 7. below to give a representation of the testing results.

Table 7.: Sample Testing of Optimizers on IT Sector Stocks

Optimizer	STOCK	Day	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction
ADAM	APH	0	94.47	94.11220551	0.470001	0.112206236	76.12638	up	up
	ANET	0	259.64	269.0241089	-5.359985	4.024094105	24.9234	down	up
	AMD	0	31.42	30.91085625	1.42	0.910856128	35.8552	up	up
	AMAT	0	38.34	39.30281448	0.34	1.302815318	-283.181	up	up
	ADSK	0	155.5	156.2341309	-0.5	-0.23412922	53.17416	down	down
	ADS	0	237.84	240.4478302	1.839996	4.44782877	-141.73	up	up
	ADI	0	92.39	92.37355804	0.389999	0.373559266	4.215328	up	up
	ADBE	0	275.49	265.8817749	6.48999	-3.118236303	51.95314	up	down
	AAPL	0	227.26	223.6413422	2.259995	-1.358660817	39.88213	up	down
SGD	APH	0	94.47	93.7519455	0.470001	-0.24805811	47.22179	up	down
	ANSS	0	186	185.1511993	0	-0.848801494	#NAME?	down	down
	ANET	0	259.64	266.7783813	-5.359985	1.778366089	66.82143	down	up
	AMD	0	31.42	30.55374336	1.42	0.553742707	61.00403	up	up
	AMAT	0	38.34	38.35298538	0.34	0.352983743	-3.81875	up	up
	ADSK	0	155.5	155.725708	-0.5	-0.274286062	45.14279	down	down
	ADS	0	237.84	237.6395416	1.839996	1.639539242	10.89441	up	up
	ADI	0	92.39	92.22898102	0.389999	0.228977948	41.28756	up	up
	ADBE	0	275.49	268.3973999	6.48999	-0.602613032	90.71474	up	down
	AAPL	0	227.26	224.7516632	2.259995	-0.248330861	89.01189	up	down
RMSprop	APH	0	94.47	91.90882874	0.470001	-2.091169834	-344.929	up	down
	ANSS	0	186	184.7854767	0	-1.214525104	#NAME?	down	down
	ANET	0	259.64	268.4716797	-5.359985	3.471686125	35.22955	down	up
	AMD	0	31.42	30.01122856	1.42	0.011227847	99.2093	up	up
	AMAT	0	38.34	38.7202301	0.34	0.720231831	-111.833	up	up
	ADSK	0	155.5	155.1463928	-0.5	-0.853602767	-70.7206	down	down
	ADS	0	237.84	238.6437683	1.839996	2.643775463	-43.6838	up	up
	ADI	0	92.39	90.30451965	0.389999	-1.695483685	-334.741	up	down
	ADBE	0	275.49	269.8543396	6.48999	0.854331851	86.83616	up	up
	AAPL	0	227.26	223.1671448	2.259995	-1.83285594	18.9	up	down

1. Stochastic Gradient Descent

Stochastic gradient descent is a gradient-based optimizer. In gradient descent, the effects of small changes in the individual weights in the network on the loss function output are calculated. After performing these calculations, the network's weight can be adjusted according to the gradient, meaning the algorithm will adjust in a small step in the direction calculated. These two steps are, then, calculated iteratively. The class technique of gradient descent uses all input training samples on every pass to tune a network's parameters. Instead, SGD uses only a

subset of the training samples on each pass. In our results, SGD yielded correct directional estimates of stock movements in about 55-60% of stock price predictions.

2. RMSprop

RMSprop is derived from the Adagrad optimizer. While the original Adagrad optimizer allows all of the gradients to accumulate to create momentum, RMSprop creates windows where gradients are accumulated. In our results, RMSprop yielded correct directional estimates of stock movements in about 55-60% of stock price predictions.

3. Adam

The Adam optimizer is a gradient-based optimizer. Adam stands for adaptive moment estimation, denoting its use of past gradients to calculate current gradients. Another important aspect of the ADAM optimizer is the use of momentum, wherein fractions of previous gradients are added to current gradients. Momentum is a method that helps accelerate SGD in the relevant direction. More specific details of these results will not be discussed here as this was the default optimizer utilized in the other hyperparameter experiments. (This was done because the Adam optimizer showed the greater performance amongst those tested). . In our results, SGD yielded correct directional estimates of stock movements in about 65-70% of stock price predictions. Because of its edge over RMSprop and SGD, the Adam optimizer was selected for the final design.

6.2.3: Learning Rate

Like the loss function, the learning rate is another parameter that is tied to the optimization function. The learning rate regulates how fast the optimization algorithm "learns," i.e., tunes the weights of the networks. Changing a network's weights too quickly can mean that

the algorithm is taking steps that may be too large that will inhibit the minimization of the loss function. The learning rate is a number that is multiplied by the gradients in order to scale them, ensuring that changes in the weights are smaller. Smaller weights means the algorithm is taking smaller steps, which may prevent the optimizer from stepping over the absolute minimum. The default learning rate for the Adam optimizer is 0.001. Thus, I experimented with increasing the learning rate to see how this affected the accuracy of the predictions. Overall, increasing the learning rate resulted in a quick decrease in the accuracy of the algorithm's predictions. Some of this sample data is shown below in Table 8. For instance, increasing the learning rate resulted in the algorithm being able to predict the accuracy of price directional movement at about 60%, which represents in a reduction in our previous predictive accuracy. This trend of increasing the learning rate resulted in further reductions continued with further testing, so the learning rate remained set at the default.

Table 8: Sample testing on learning rate

Optimizer	STOCK	Day	True Price	Predicted Price	True Price Change	Predicted Price Cha	Error	True Direction	Predicted Direction
ADAM	APH	0	94.47	93.14929962	0.470001	-0.850700736	-80.9998	up	down
lr = 0.01	ANSS	0	186	184.9984894	0	-1.001513124	100	down	down
	ANET	0	259.64	266.4893188	-5.359985	1.489326358	72.21397	down	up
	AMD	0	31.42	29.78351402	1.42	-0.216485769	84.75452	up	down
	AMAT	0	38.34	38.89343643	0.34	0.893434882	-162.775	up	up
	ADSK	0	155.5	155.8628082	-0.5	-0.137189165	72.56217	down	down
	ADS	0	237.84	237.3260345	1.839996	1.326029658	27.93301	up	up
	ADI	0	92.39	96.30442047	0.389999	4.304420948	-1003.7	up	up
	ADBE	0	275.49	269.0627136	6.48999	0.062720783	99.03358	up	up
	AAPL	0	227.26	209.6324921	2.259995	-15.36750221	-579.979	up	down

6.2.4: Dropout Regularization

Dropout is a technique that selectively chooses neurons to ignore ("drop-out") during training. On a given pass through the network, the contribution of these dropped neurons to activating downstream neurons is removed, meaning their weights are not applied. The motivation underlying dropout is that as a neural network trains, the weights of the neurons settle

into their context of the neurons, and neighboring neurons can become too specialized, resulting in a model that is overfit to training data. When dropout techniques are used, other neurons in the network will have to step in to interpret the representation required to make predictions for the missing neurons. It is theorized that this results in the network learning multiple internal independent representations. In Keras, dropout is implemented through the selection of a probability value that randomly selects nodes to be dropped out. The default value of dropout in all of the previous experiments was 0, meaning dropout was nonexistent. As expected, the introduction of dropout produced a severe reduction in the algorithm's capability to predict price movements. This outcome was expected because dropout interferes with the algorithm's "memory" capabilities. Thus, no dropout was utilized in the final design.

6.2.5: Number of Units.

The number of units represents the number of neurons of a layer. A unit takes inputs from all the nodes in the layer below, does its calculations, and then outputs them to the layer above. Thus, the number of neurons in a layer controls the representational capacity of the network.

Table 9: Testing Effects of Different Numbers of Neurons on LSTM Predictions

Number of Neurons	STOCK	Day	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction
50	APH	0	94.470001	94.06190491	0.470001	0.061907537	86.82822	up	up
	ANSS	0	186	185.0775146	0	-0.922486663	100	down	down
	ANET	0	259.640015	265.3862	-5.359985	0.386200249	92.79475	down	up
	AMD	0	31.42	29.70808601	1.42	-0.291914493	79.44264	up	down
	AMAT	0	38.34	38.91811752	0.34	0.918116927	-170.034	up	up
	ADSK	0	155.5	155.8807983	-0.5	-0.119207159	76.15857	down	down
	ADS	0	237.839996	237.8096771	1.839996	1.809680939	1.64756	up	up
	ADI	0	92.389999	92.09751892	0.389999	0.097515121	74.99606	up	up
	ADBE	0	275.48999	268.7677917	6.48999	-0.232200414	96.42217	up	down
	AAPL	0	227.259995	223.7910004	2.259995	-1.209003806	46.50414	up	down
75	APH	0	94.470001	94.55987549	0.470001	0.559877694	-19.1227	up	up
	ANSS	0	186	187.4104767	0	1.410482287	100	down	up
	ANET	0	259.640015	272.1832886	-5.359985	7.183274269	-34.0167	down	up
	AMD	0	31.42	29.73521233	1.42	-0.264787704	81.35298	up	down
	AMAT	0	38.34	38.00680161	0.34	0.006802321	97.99931	up	up
	ADSK	0	155.5	156.461853	-0.5	0.461851418	7.629716	down	up
	ADS	0	237.839996	241.0331726	1.839996	5.033178806	-173.543	up	up
	ADI	0	92.389999	91.6517334	0.389999	-0.348265916	10.70082	up	down
	ADBE	0	275.48999	265.1954651	6.48999	-3.804523468	41.37859	up	down
	AAPL	0	227.259995	223.9040222	2.259995	-1.095975757	51.50539	up	down
25	APH	0	94.470001	92.71748352	0.470001	-1.282518029	-172.876	up	down
	ANSS	0	186	186.3443146	0	0.344321996	100	down	up
	ANET	0	259.640015	264.4494019	-5.359985	-0.550594866	89.72768	down	down
	AMD	0	31.42	30.50790787	1.42	0.507907689	64.23186	up	up
	AMAT	0	38.34	38.71870041	0.34	0.718701184	-111.383	up	up
	ADSK	0	155.5	158.1290131	-0.5	2.129019737	-325.804	down	up
	ADS	0	237.839996	236.6280975	1.839996	0.628096879	65.86422	up	up
	ADI	0	92.389999	91.82740021	0.389999	-0.172601476	55.7431	up	down
	ADBE	0	275.48999	268.989502	6.48999	-0.010492326	99.83833	up	down
	AAPL	0	227.259995	223.153717	2.259995	-1.846285343	18.30578	up	down

Because 50 neurons have been the default in this testing process, I decided to test number of neurons above and below this number to see if there were any improvements in predictions. The configuration using 50 neurons still had a slight edge in predictive accuracy. Decreasing the number of neurons caused a severe decrease in the ability of the algorithm to predict the correct directional movement of the price, and an increase in the number of neurons still slightly underperformed the 50-neuron configuration.

6.2.6: Lag Window

One of the methods tested to counteract concept drift was the idea of a "lag" window. These experiments have been conducted with a window size of one, so increased window sizes were tested to see if they would increase predictive accuracy. Some sample results are shown in Table 10 below.

Table 10: Sample Test Results of Varying "Lag Window"

Number of lag days	Days Forecast	STOCK	Day	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction	Portfolio Value	Market Baseline
3	1	APH	0	94.470001	90.56374359	0.470001	-3.436256886	-631.117	up	down	10000	10000
		ANSS	0	186	184.720871	0	-1.279122233	100	down	down	9991.1	10000
		ANET	0	259.640015	259.4681091	-5.359985	-5.531885624	-3.20711	down	down	9991.1	10000
		AMD	0	31.42	23.26226807	1.42	-6.73773098	-374.488	up	down	10000	10000
		AMAT	0	38.34	32.26539993	0.34	-5.73459959	-1586.65	up	down	10000	10000
		ADSK	0	155.5	142.7928467	-0.5	-13.2071476	-2541.43	down	down	9991.1	10000
		ADS	0	237.839996	231.1002808	1.839996	-4.899722576	-166.29	up	down	9991.1	10000
		ADI	0	92.389999	81.21916962	0.389999	-10.78083134	-2664.32	up	down	10000	10000
		ADBE	0	275.48999	263.7677612	6.48999	-5.232229233	19.38001	up	down	9991.1	10000
		AAPL	0	227.259995	227.5815735	2.259995	2.581567764	-14.2289	up	up	9991.1	10000
7	1	APH	0	94.470001	92.03207397	0.470001	-1.967929363	-318.707	up	down	10000	10000
		ANSS	0	186	185.0345001	0	-0.965496421	100	down	down	9991.1	10000
		ANET	0	259.640015	261.0236206	-5.359985	-3.976389408	25.81342	down	down	9991.1	10000
		AMD	0	31.42	28.38490486	1.42	-1.615095377	-13.7391	up	down	10000	10000
		AMAT	0	38.34	35.70037079	0.34	-2.29963088	-576.362	up	down	10000	10000
		ADSK	0	155.5	151.2554474	-0.5	-4.744558811	-848.912	down	down	9991.1	10000
		ADS	0	237.839996	235.7975311	1.839996	-0.202464849	88.99645	up	down	9991.1	10000
		ADI	0	92.389999	80.55764771	0.389999	-11.44235134	-2833.94	up	down	10000	10000
		ADBE	0	275.48999	261.7139587	6.48999	-7.286033154	-12.2657	up	down	9991.1	10000
		AAPL	0	227.259995	222.6520081	2.259995	-2.347998857	-3.89399	up	down	9991.1	10000

Overall, the results in the table and other testing showed that a lag window did not increase predictive accuracy. In fact, the lag window caused a substantial increase in error. The addition of a lag window seemed to hamper the algorithm's ability to make detect directional changes in price movement, which is a highly detrimental result in stock trading.

In general, based on the results of hyperparameter testing that are discussed in the sections about the following final design was chosen. The network trains with 10 epochs and a batch size of 50. It contains 50 neurons. Additionally, its optimizer is set to be the Adam optimizer with a learning rate of 0.001. No dropout and no lag window are used.

7. PERFORMANCE AND RESULTS:

The performance and results of the final design is discussed in this section. As one of the design requirements, I specified that the design should be flexible, meaning the user should be able to input whatever numerical data relating to stock prices that they desired. Additionally, one major component of developing a machine learning model is feature engineering, so I undertook tests using a variety of different inputs into the model.

7.1: HISTORICAL PRICING DATA

The first type of data the I utilized was historical pricing data, which included the open, high, low, close and volume. On average, making day-ahead predictions with this data resulted in an accuracy of predicted price movements between 70-75%. This is reflected in the sample test results shown in Table 11 below. The error between the predicted price change and true price

Table 11: Three-Year Historical Data as Input Features (One-Day Forecast)

Stock	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction
CTXS	108.75	109.2977066	1.75	2.297708273	-31.2976	up	up
CSCO	46.35	43.72002411	0.34999847	-2.27997756	-551.425	up	down
CA	44.44	41.47690582	0.43999863	-2.523094177	-473.432	up	down
AVGO	237.61	238.4780884	1.6100006	2.478085756	-53.9183	up	up
APH	88.77	89.45321655	1.76999664	2.453214407	-38.5999	up	up
AMAT	35.4	36.5318718	0.40000153	1.531871438	-282.966	up	up
AKAM	69.87	70.87728882	0.87000275	1.877286077	-115.779	up	up
ADSK	134.04	135.3508606	0.0399933	1.350856662	-3277.71	up	up
ADP	146.55	145.5160217	-0.4499969	-1.483972907	-229.774	down	down
ADI	85.94	88.7841568	-3.05999756	-0.215844914	92.94624	down	down
ADBE	238.89	236.2761841	-0.1100006	-2.72382021	-2376.19	down	down
CAN	163.44	165.6013489	-1.5599976	0.601343632	61.45227	down	up
AAPL	193.53	193.475647	2.5299988	2.475640774	2.148539	up	up

change is still sizeable for some stocks, but its relative error compared to other test was at least no larger. The, the number of days predicted was increased to seven-days, and a sample of the

Table 12: Three-Year Historical Data as Input Features (Seven-Day Forecast)

Day Predicted	Stock	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction	Portfolio Value	Market Baseline
1	AVGO	240.25	260.9428101	6.25	26.94280052	-331.0848	up	up	9991.1	10000
2		238.99	268.039856	-1.2599945	7.097052097	-463.2606	down	up	9939.440226	9948.340226
3		223.63	271.3536682	-15.3600006	3.31381011	78.42572	down	up	9309.680201	9318.580201
4		224.83	273.0072937	1.1999969	1.653621912	-37.80218	up	up	9358.880074	9367.780074
5		226.2	273.9428101	1.3699951	0.935501873	31.714949	up	up	9415.049873	9423.949873
6		236.34	274.5539551	10.1399994	0.611140072	93.972977	up	up	9830.789848	9839.689848
7		237.61	275.0172729	1.2700043	0.463329881	63.517452	up	up	9882.860025	9891.760025
1	APH	92.23	85.47134399	1.23000336	-5.528655052	-349.4829	up	down	10000	10000
2		91.14	78.39096069	-1.09000397	-7.080383301	-549.5741	down	down	10000	9882.279571
3		88.45	70.99386597	-2.69000244	-7.397096634	-174.9848	down	down	10000	9591.759308
4		87.83	63.60263062	-0.61999512	-7.391235352	-1092.144	down	down	10000	9524.799835
5		86.64	56.19285965	-1.19000244	-7.409772396	-522.6687	down	down	10000	9396.279571
6		87.78	48.78657913	1.13999939	-7.406280518	-549.6741	up	down	10000	9519.399505
7		88.77	41.37885284	0.98999786	-7.407727242	-648.2569	up	down	10000	9626.319274
1	AMAT	34.94	23.29711533	-0.06000137	-11.70288372	-19404.36	down	down	10000	10000
2		34.28	26.48615265	-0.65999985	3.1890378	-383.1877	down	up	9991.1	9811.900043
3		32.62	24.34555817	-1.65999985	-2.140594244	-28.95147	down	down	9991.1	9338.800086
4		33.49	22.93083763	0.87000275	-1.414720774	-62.61107	up	down	9991.1	9586.750869
5		33.58	21.32854271	0.09000015	-1.602294683	-1680.324	up	down	9991.1	9612.400912
6		35.02	19.71103096	1.43999863	-1.617512107	-12.32734	up	down	9991.1	10022.80052
7		35.4	18.0921917	0.38000107	-1.618839383	-326.0092	up	down	9991.1	10131.10083
1	AKAM	72.18	69.5565567	1.18000031	-1.44343996	-22.32539	up	down	10000	10000
2		70.94	64.33505249	-1.23999787	-5.221506119	-321.0899	down	down	10000	9828.880294
3		69.08	57.28103256	-1.86000061	-7.054020882	-279.2483	down	down	10000	9572.20021
4		69.19	50.0707283	0.11000061	-7.210305691	-6454.787	up	down	10000	9587.380294
5		68.41	42.75197983	-0.77999878	-7.318747997	-838.3025	down	down	10000	9479.740462
6		69.46	35.40736389	1.04999542	-7.34461689	-599.4904	up	down	10000	9624.63983
7		69.87	28.05310822	0.41000367	-7.354256153	-1693.705	up	down	10000	9681.220337
1	ADSK	140.61	135.513916	2.6100006	-2.486084223	4.7477531	up	down	10000	10000
2		135.55	129.8870087	-5.0599975	-5.62690115	-11.20363	down	down	10000	9640.740178
3		130.11	123.3156967	-5.4400025	-6.571308613	-20.79606	down	down	10000	9254.5
4		130.92	116.5582962	0.8099976	-6.757402897	-734.2498	up	down	10000	9312.00983
5		129.43	109.7202988	-1.4900055	-6.837999821	-358.9245	down	down	10000	9206.219439
6		134.29	102.8536682	4.8600006	-6.86662817	-41.28863	up	down	10000	9551.279482
7		134.04	95.97576141	-0.25	-6.877903938	-2651.162	down	down	10000	9533.529482
1	ADP	147.89	142.1575317	-0.1100006	-5.842474937	-5211.312	down	down	10000	10000
2		147.19	131.3707581	-0.699997	-10.78677368	-1440.974	down	down	10000	9953.100201
3		144.17	122.2784271	-3.0200042	-9.092331886	-201.0702	down	down	10000	9750.75992
4		144.72	114.1134872	0.550003	-8.164942741	-1384.527	up	down	10000	9787.610121
5		142.72	106.1505508	-2	-7.962934494	-298.1467	down	down	10000	9653.610121
6		147.25	98.21892548	4.5299988	-7.931624413	-75.09109	up	down	10000	9957.12004
7		146.55	90.29087067	-0.6999969	-7.928053379	-1032.584	down	down	10000	9910.220248
1	ADI	87.59	75.99971771	-1.41000366	-13.00028229	-822.0035	down	down	10000	10000
2		84.2	73.8015976	-3.38999939	-2.198118687	35.15873	down	down	10000	9613.54007
3		85.3	65.98967743	1.1000061	-7.811917782	-610.1704	up	down	10000	9738.940765
4		86.13	59.65843201	0.8299942	-6.331245899	-662.806	up	down	10000	9833.560104
5		89.82	52.7567482	3.69000244	-6.901682854	-87.03735	up	down	10000	10254.22038
6		89.48	46.01705933	-0.33999633	-6.739689827	-1882.283	down	down	10000	10215.4608
7		85.94	39.21909714	-3.54000092	-6.797962189	-92.03278	down	down	10000	9811.900695
1	ADBE	249.96	241.2314301	-3.0399933	-11.76856899	-287.1248	down	down	9991.1	10000
2		244.84	232.0846405	-5.1200104	-9.146787643	-78.64783	down	down	9991.1	9800.319594
3		236.67	222.2887573	-8.1699981	-9.795885086	-19.9007	down	down	9991.1	9481.689669
4		238.74	211.9395142	2.0700073	-10.34924316	-399.9617	up	down	9991.1	9562.419953
5		235.22	201.5285187	-3.5200043	-10.41099453	-195.7665	down	down	9991.1	9425.139786
6		239.95	191.071701	4.7299957	-10.45682335	-121.0747	up	down	9991.1	9609.609618
7		238.89	180.6022949	-1.0599975	-10.46940041	-887.6816	down	down	9991.1	9568.269715
1	CAN	165.15	157.7051849	1.1499939	-6.294807911	-447.3775	up	down	10000	10000
2		162.44	156.461792	-2.7099915	-1.243395329	54.118111	down	down	10000	9837.40051
3		161.45	153.4689789	-0.9900055	-2.992809534	-202.3023	down	down	10000	9778.00018
4		160.29	150.7339325	-1.1600036	-2.735046625	-135.7792	down	down	10000	9708.399964
5		162.49	147.881897	2.2000122	-2.852030516	-29.63703	up	down	10000	9840.400696
6		165	145.0293121	2.5099945	-2.852590561	-13.64927	up	down	10000	9991.000366
7		163.44	142.1671753	-1.5599976	-2.862139225	-83.47075	down	down	10000	9897.40051
1	AAPL	208.49	215.7633667	-0.5099945	6.763373852	-1226.166	down	up	10000	10000
2		204.47	217.5197601	-4.0200043	1.756391048	56.308727	down	up	9991.1	9811.059798
3		194.17	216.6836853	-10.300003	-0.836081147	91.882706	down	down	9991.1	9326.959657
4		192.23	215.1786499	-1.9400025	-1.505038381	22.420799	down	down	9991.1	9235.779539
5		186.8	213.350235	-5.4299926	-1.828416705	66.327454	down	down	9991.1	8980.569887
6		191.41	211.4236755	4.6100006	-1.926557779	58.209164	up	down	9991.1	9197.239915
7		193.53	209.4538727	2.1199951	-1.969800591	7.0846639	up	down	9991.1	9296.879685

results are shown in Table 12 above. Immediately, the amount of error per prediction increased significantly. The algorithm's ability to predict stock price changes correctly dropped from 70-75% to 55-60%. Still, the algorithm appears able to predict if the stock price has an overall trend. As a result, the algorithm's use of its predictions to make investments is sometimes able to outperform the market benchmark when the real stock price is going down.

Then, the same type of historical stock price data was used over a one-year span to make day-ahead and seven-day ahead forecasts. The performance of the day-ahead forecasts was similar to that using three-year data with an accuracy from 70-75% overall. Some of the sample test results are shown in Table 13 below.

Table 13: One-Year Historical Price Inputs for Day-Ahead Forecasts

Stock	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction
AAPL	110.06	111.0147324	1.0599976	2.014733553	-90.0696	up	up
CAN	118.43	126.5592346	0.4300003	8.559235573	-1890.52	up	up
ADBE	105.02	108.8121567	0.0199966	3.8121593	-18964	up	up
ADI	68.47	74.03000641	0.47000122	6.030006886	-1182.98	up	up
ADP	94.39	86.43240356	0.38999939	-7.567599297	-1840.41	up	down
ADSK	76.9	75.52387238	-1.09999847	-2.476130486	-125.103	down	down
AKAM	66.44	72.06859589	0.44000244	6.068596363	-1279.22	up	up
AMAT	30.74	31.46628571	0.73999977	1.466285229	-98.1467	up	up
APH	68.04	77.62567139	0.04000092	9.625671387	-23963.6	up	up
AVGO	168.16	173.3279419	1.1600037	6.327948093	-445.511	up	up

When these forecasts were extended to seven-day ahead forecasts, the predictive performance using one-year data fell short of the three-year data (as shown in Table 14 below). The predictive performance dropped to just above 50% on average. Unlike the predictions using three-year data, which was able to predict if a stock was trending downwards and detect more notable changes in trend, the predictions using one-year data virtually never predicted a trend reversal, especially if there was a downward trend. Such a result could be detrimental and can result in large losses in the market if prices took a downturn. It can be seen that the algorithm's

Table 14: One-Year Historical Price Inputs for Day-Ahead Forecasts

Day Predicted	Stock	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direct	Portfolio Value	Market Baseline
1	AAPL	107.79	111.571579	-2.2099991	1.571580887	28.88771248	down	up	10000	10000
2		108.43	112.7638474	0.6399994	1.192265391	-86.29164886	up	up	9991.1	10058.87994
3		105.71	114.1985168	-2.7200012	1.434668064	47.25487518	down	up	9740.85989	9808.639834
4		107.11	115.8325348	1.4000015	1.634018421	-16.71547508	up	up	9869.660028	9937.439972
5		109.99	117.5951385	2.8799973	1.762604833	38.79838562	up	up	10134.61978	10202.39972
6		109.95	119.4412994	-0.040001	1.846159339	-4515.282715	down	up	10130.93969	10198.71963
7		110.06	121.3417053	0.1100007	1.900406957	-1627.631714	up	up	10141.05975	10208.8397
1	CAN	120.33	122.8759842	3.3300018	5.875984192	-76.45588684	up	up	10000	10000
2		119.18	127.9719543	-1.1500015	5.095972061	-343.1274109	down	up	9904.549876	9991.1
3		117.25	131.8987579	-1.9300003	3.926804781	-103.4613647	down	up	9744.359851	9830.909975
4		115.99	135.1111908	-1.2600021	3.212425947	-154.95401	down	up	9639.779676	9726.329801
5		117.32	137.8630981	1.3300018	2.751900911	-106.9095612	up	up	9750.169826	9836.71995
6		118.37	140.3171234	1.050003	2.454026461	-133.7161255	up	up	9837.320075	9923.870199
7		118.43	142.5782928	0.0599976	2.261162281	-3668.754639	up	up	9842.299876	9928.85
1	ADBE	104.08	120.6608963	-2.9199982	13.66089344	-367.8391113	down	up	10000	10000
2		103.68	129.1211243	-0.4000015	8.460234642	-2015.050659	down	up	9961.999858	9991.1
3		102.42	134.829361	-1.2600021	5.708241463	-353.0342712	down	up	9842.299658	9870.139798
4		103.66	138.8816071	1.2400055	4.052247524	-226.7927246	up	up	9960.100181	9989.180326
5		104.08	141.9048767	0.4199981	3.023262262	-619.8275757	up	up	10000	10029.50014
6		105.81	144.2805481	1.7299958	2.375676394	-37.32266235	up	up	10164.3496	10195.57974
7		105.02	146.245636	-0.790001	1.965083838	-148.7444916	down	up	10089.29951	10119.73964
1	ADI	64.25	70.23732758	0.25	6.237330437	-2394.932129	up	up	10000	10000
2		65.32	73.04006195	1.06999969	2.802737236	-161.9381256	up	up	9991.1	10165.84995
3		66.46	74.78194427	1.13999939	1.741880298	-52.79659653	up	up	10164.37991	10342.54986
4		68.21	75.86401367	1.75	1.082072496	38.16728592	up	up	10430.37991	10613.79986
5		67.88	76.56458282	-0.33000183	0.700568497	-112.292305	down	up	10380.21963	10562.64957
6		68.21	77.03787231	0.33000183	0.473286629	-43.41939545	up	up	10430.37991	10613.79986
7		68.47	77.37596893	0.26000214	0.33809346	-30.03487968	up	up	10469.90023	10654.10019
1	ADP	91.39	99.65316772	1.38999939	9.653171539	-594.4730835	up	up	10000	10000
2		91.08	105.5480728	-0.30999756	5.894903183	-1801.59668	down	up	9966.210266	9991.1
3		91.89	109.3423462	0.80999756	3.794271469	-368.4299927	up	up	10054.5	10079.38973
4		92.34	111.9935226	0.44999695	2.651176453	-489.1543274	up	up	10103.54967	10128.4394
5		93.27	114.019104	0.9300003	2.025583506	-117.8046112	up	up	10204.9197	10229.80943
6		94.38	115.7014694	1.11000061	1.682367802	-51.56458664	up	up	10325.90977	10350.7995
7		94.39	117.1951981	0.01000214	1.493729234	-14834.09766	up	up	10327	10351.88973
1	ADSK	73.49	79.06157684	1.48999786	7.06157732	-373.9320374	up	up	10000	10000
2		73.81	84.41772461	0.3199997	5.356146336	-1573.797363	up	up	9991.1	10043.19996
3		73.55	88.16216278	-0.25999451	3.744435549	-1340.197998	down	up	9956.000741	10008.1007
4		75.15	90.99267578	1.59999848	2.830509424	-76.90700531	up	up	10172.00054	10224.1005
5		77.57	93.25462341	2.41999816	2.261948586	6.530979156	up	up	10498.70029	10550.80025
6		78.46	95.16982269	0.88999939	1.915201068	-115.1912842	up	up	10618.85021	10670.95016
7		76.9	96.87281799	-1.55999755	1.702995896	-9.166574478	down	up	10408.25054	10460.3505
1	ADP	66.22	71.37954712	0.22000122	5.379544258	-2345.233887	up	up	10000	10000
2		65.45	75.19691467	-0.77000427	3.817370653	-395.7596741	down	up	9991.1	9884.49936
3		65.44	78.02180481	-0.00999451	2.824890852	-28164.42578	down	up	9989.580834	9883.000183
4		65.47	80.15223694	0.02999878	2.130429029	-7001.71875	up	up	9994.140649	9887.5
5		65.9	81.80426025	0.43000031	1.652024269	-284.1914063	up	up	10059.5007	9952.000047
6		66.6	83.12696838	0.69999694	1.322705984	-88.95882416	up	up	10165.90023	10056.99959
7		66.44	84.22289276	-0.15999603	1.095927	-584.9713745	down	up	10141.58083	10033.00018
1	AKAM	66.22	71.37954712	0.22000122	5.379544258	-2345.233887	up	up	10000	10000
2		65.45	75.19691467	-0.77000427	3.817370653	-395.7596741	down	up	9991.1	9884.49936
3		65.44	78.02180481	-0.00999451	2.824890852	-28164.42578	down	up	9989.580834	9883.000183
4		65.47	80.15223694	0.02999878	2.130429029	-7001.71875	up	up	9994.140649	9887.5
5		65.9	81.80426025	0.43000031	1.652024269	-284.1914063	up	up	10059.5007	9952.000047
6		66.6	83.12696838	0.69999694	1.322705984	-88.95882416	up	up	10165.90023	10056.99959
7		66.44	84.22289276	-0.15999603	1.095927	-584.9713745	down	up	10141.58083	10033.00018
1	AMAT	28.18	26.16184616	-0.81999969	-2.838152885	-246.116333	down	down	10000	10000
2		28.82	22.69055367	0.63999938	-3.471292734	-442.3900146	up	down	10000	10226.55978
3		28.99	19.23794556	0.17000008	-3.452608109	-1930.945068	up	down	10000	10286.73981
4		29.61	15.7755928	0.62000084	-3.462352991	-458.4432678	up	down	10000	10506.22011
5		30.02	12.30991936	0.40999985	-3.465673685	-745.2865601	up	down	10000	10651.36005
6		30.73	8.8425951	0.70999908	-3.467324495	-388.3561707	up	down	10000	10902.69973
7		30.74	5.374509811	0.01000023	-3.468085051	-34580.05469	up	down	10000	10906.23981
1	APH	66.47	71.30583191	-0.52999878	4.305828571	-712.4223633	down	up	10000	10000
2		66.63	73.59568024	0.15999603	2.289847851	-1331.19043	up	up	9991.1	10023.9994
3		66.63	76.40555573	0	2.809871912	100	down	up	9991.1	10023.9994
4		67.13	79.49833679	0.5	3.092782259	-518.5564575	up	up	10065.6	10098.9994
5		67.8	82.76777649	0.6700058	3.269440651	-387.9719849	up	up	10165.43086	10199.50027
6		68.01	86.14984131	0.20999909	3.382063627	-1510.51355	up	up	10196.72073	10231.00014
7		68.04	89.60332489	0.02999878	3.453486681	-11412.09082	up	up	10201.19055	10235.49996
1	AVGO	167.94	178.6390533	-5.0599976	5.639056683	-11.44386196	down	up	9991.1	10000
2		167.54	181.3150635	-0.4000091	2.676016569	-568.9888916	down	up	9967.499463	9976.399463
3		163.78	182.2557983	-3.7599945	0.940740585	74.98026276	down	up	9745.659788	9754.559788
4		166.45	182.2799377	2.6699981	0.02414518	99.09568787	up	up	9903.189676	9912.089676
5		167.83	181.81633	1.3800049	-0.463608742	66.40528107	up	up	9903.189676	9993.509965
6		167.04	181.0904999	-0.7900085	-0.725835323	8.12309742	down	up	9903.189676	9946.899463
7		168.16	180.2228699	1.1200104	-0.867631376	22.53363037	up	up	9903.189676	10012.98008

almost inability to predict downward movements causes it to take losses and underperform the market baseline consistently. Fig. 20 and Fig. 21 below show predictions on the same stock (AMAT) using both 1-year and 3-year data. This is representative of how using 3-year data as input seemed to increase the predictive accuracy of the model.

Fig. 20: Sample of 7-Day Forecast Using 1-Year Price Predictions

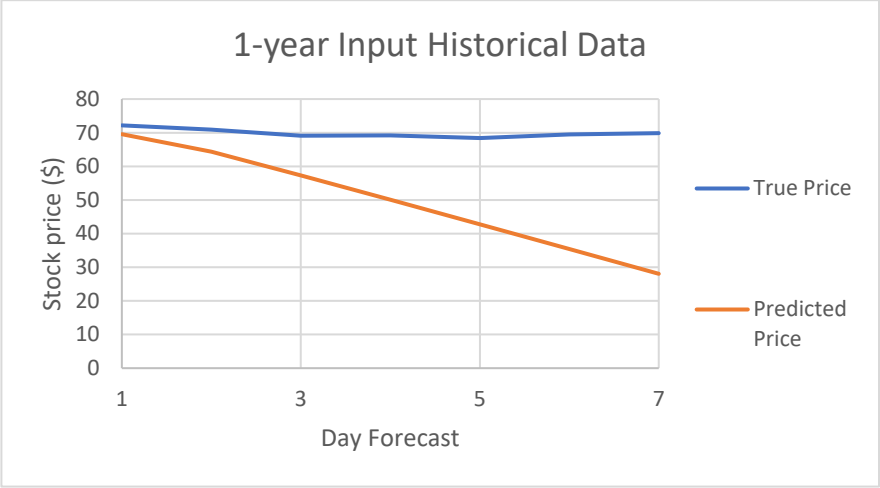
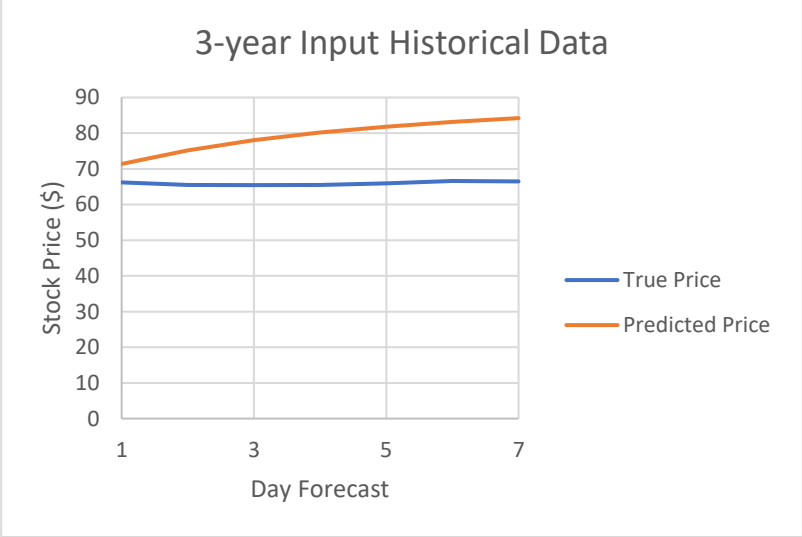


Fig. 21: Sample of 7-Day Forecast Using 3-Year Price Predictions



In the next set of experiments, the time period of historical data was shortened even more to one month-data to make next-day predictions. This short input timeframe further degraded the performance of the algorithm, which again was virtually never able to detect when there was a reversal in stock trend, resulting in predicting accuracy of between 45-50%, which is an undesirable level of accuracy. Some sample results are shown in Table 15 below:

Table 15: One-Month Historical Price Inputs for Day-Ahead Forecasts

Stock	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction
AAPL	106.03	110.1734924	-1.9700012	2.173493862	-10.3296	down	up
CAN	102.99	102.6220016	1.9899979	1.622004271	18.49216	up	up
ADI	55.17	61.6763649	-0.83000183	5.676365852	-583.898	down	up
ADP	84.14	90.70876312	-1.13999939	7.708764553	-576.208	down	up
ADSK	60.99	72.02920532	0.99000168	12.02920246	-1115.07	up	up
AKAM	52.98	65.29605103	0.97999954	13.29604816	-1256.74	up	up
AMAT	18.76	30.40271378	0.76000023	12.40271378	-1531.94	up	up
APH	53.03	59.81407166	-1.02999878	7.814071655	-658.649	down	up
AVGO	145.1	152.5101929	0.1000061	7.510194778	-7409.74	up	up

As a result of the experiments, we can see that next-day price predictions yield more accurate predictions more consistently when 1-year and 3-year historical price data is used. When too short of a time span is used (such as one month) the predictions are less sensitive to the day-to-day fluctuations in stock price. However, for even the 1-year and 3-year historical data, as soon as the number of days predicted out increases, there is a rapid increase in the error of the actual magnitude of the predicted price change. For much of this section, we have focused on the direction of the price changes as this is an important metric for investors. Still, the magnitude of the price change does carry importance as it would help investors better allocate funds, however, for all of the predictions there were large, increasing degrees of these errors. An example of this is represented in Fig. 22 and Fig. 23 below.

Fig. 22: Error for AMAT Stock Using 3-Year Historical Data

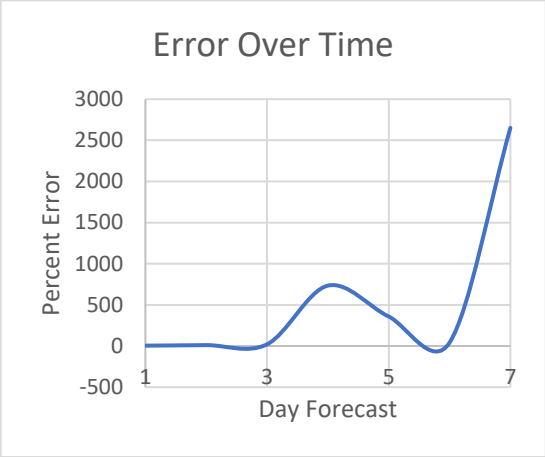
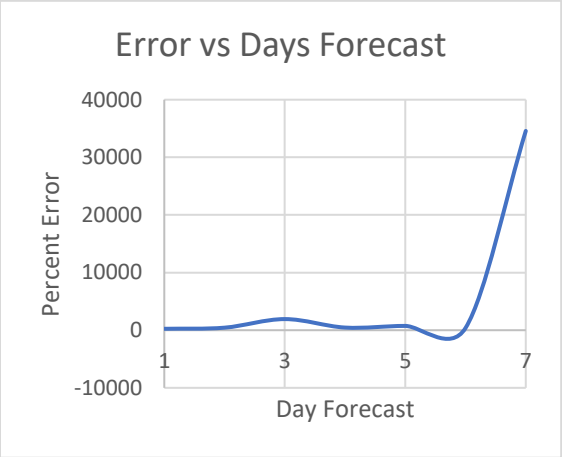
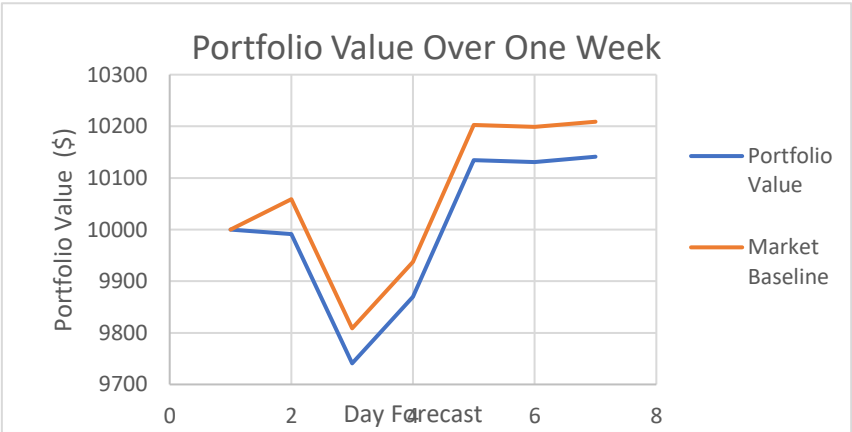


Fig. 23: Error for AMAT Stock Using 1-Year Historical Data



As discussed (even with correct predictions of directional stock movement), the increasing error over time does cost investors. One example from our sample testing of this is the following. For a seven-day forecast, the algorithm correctly predicted the movement of AAPL's stock prices the majority of the time, but there was a large degree of error in the magnitudes of the price predictions (as exemplified by Fig. 20 and Fig. 21 above). As a result, the fund allocation was not at an optimal level, and the portfolio value did not outperform the market benchmark, as shown in Fig. 24 below:

Fig. 24: Cost of Price Prediction Error to Market Baseline Earnings



7.2: ADDING TECHNICAL INDICATORS

Because the three-year data outperformed the other input timeframes, this input time frame was utilized moving forward for making predictions. The same historical price data continued to be used in this set of experiments, but different combinations of technical indicators were also used. The first combination of technical indicators consisted of the Chaikin A/D Oscillator, the Average True Range, Simple Moving Average, and Aroon Oscillator. A sample of day-ahead predictions using these technical indicators is shown in Table 16 below:

Table 16: Day-Ahead Forecasts Using Historical Data & Technical Indicator Combo 1

Technical Indicator Combination	Days Predicted	Stock	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction
1	1	AAPL	116.05	129.026947	2.0500031	15.02695084	-633.021	up	up
		ADSK	146.21	130.7761536	-5.7899933	-21.22385216	-266.561	down	down
		AVGO	241.37	275.1560974	-2.6300049	31.15610504	-1084.64	down	up

On average the accuracy of the price changes using a technical indicator for one-day ahead forecasts is between 65-70%. However, there is a large increase in the degree of error of the price change as compared with just using historical prices. Extending these predictions out to a seven-day period only magnifies this error even further, and the accuracy decreased.

Table 17: Seven Day-Ahead Forecasts Using Historical Data & Technical Indicator Combo 1

Technical Indicator Combination	Days Predicted	Stock	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction
1	7	ADSK	156.11	118.5870667	0.1100006	-37.41293716	-33911.6	up	down
			155.5	89.87733459	-0.6100006	-28.70973396	-4606.51	down	down
			153.92	67.83621216	-1.5800018	-22.04112625	-1295.01	down	down
			154.87	50.57013321	0.9499969	-17.26607895	-1717.49	up	down
			152.1	36.70905685	-2.769989	-13.8610754	-400.402	down	down
			152	25.23319626	-0.1000061	-11.47586155	-11375.2	down	down
			146.21	15.402215	-5.7899933	-9.830981255	-69.7926	down	down
1	7	AVGO	246.73	237.4014282	0.7299957	-8.598573685	-1077.89	up	down
			249.51	231.1694031	2.7799988	-6.232019424	-124.173	up	down
			248.1	226.6878204	-1.4099884	-4.481589794	-217.846	down	down
			249.42	223.4268646	1.3199921	-3.260951996	-147.043	up	down
			248.17	221.0305023	-1.25	-2.396368027	-91.7094	down	down
			244.23	219.2679596	-3.9400025	-1.76254046	55.2655	down	down
			241.37	217.9344635	-2.8600006	-1.333493352	53.37437	down	down

As shown by the sample results in Table 17 above, using technical indicators may be able to pick out an underlying trend, but is unable to effectively detect day-to-day price fluctuations, a result

that could lead to serious losses in financial investing. This result carried through to our other combinations of technical indicators. The second combination consisted of Exponential Moving Average, Relative Strength Index, Average True Range, and Chaikin Oscillator. The third combination consisted of the Parabolic SAR, Commodity Channel Index, Average True Range, and Chaikin Oscillator. None of these combinations conclusively outperformed each other. In general, the observations that held true for the first combination of technical indicators held true for these, which is shown by the results in Table 18 and Table 19 below.

Table 18: 7-Day Price Forecasts with Historical Prices and Technical Indicators as Inputs

Technical Indicator Combination	STOCK	True Price	Predicted Price	True Price Change	Predicted Price Change	Error	True Direction	Predicted Direction	Portfolio Value	Market Baseline
2	AAPL	113.05	113.0500488	1.0500031	1.050050855	-0.004552647	up	up	10000	10000
		112.52	110.9868927	-0.5300065	-2.063154697	-289.2696838	down	down	10000	9953.359428
		113	108.2199097	0.4800034	-2.766983509	-476.4508362	up	down	10000	9995.599727
		113.05	104.8845978	0.0500031	-3.335313797	-6570.213867	up	down	10000	10000
		113.89	101.142189	0.8399963	-3.742411613	-345.5271606	up	down	10000	10073.91967
		114.06	97.10730743	0.1699982	-4.034881115	-2273.484619	up	down	10000	10088.87952
2	ADSK	116.05	92.86122131	1.9900055	-4.24608469	-113.3704987	up	down	10000	10264
		156.11	133.453598	0.1100006	-22.54640388	-20396.61914	up	down	10000	10000
		155.5	114.0736542	-0.6100006	-19.37994576	-3077.037109	down	down	10000	9960.959962
		153.92	99.18965912	-1.5800018	-14.88399506	-842.0239258	down	down	10000	9859.839846
		154.87	87.1467514	0.9499969	-12.04290771	-1167.678589	up	down	10000	9920.639648
		152.1	77.05891418	-2.769989	-10.08783436	-264.183197	down	down	10000	9743.360352
		152	68.27722931	-0.1000061	-8.781682014	-8681.146484	down	down	10000	9736.959962
		146.21	60.35959244	-5.7899933	-7.917635441	-36.74688339	down	down	10000	9366.40039
2	AVGO	246.73	241.0900726	0.7299957	-4.909930229	-572.5971069	up	down	10000	10000
		249.51	236.8891907	2.7799988	-4.200881004	-51.11089325	up	down	10000	10111.19995
		248.1	234.5336914	-1.4099884	-2.355493546	-67.05764771	down	down	10000	10054.80042
		249.42	233.2832336	1.3199921	-1.250451803	5.268233299	up	down	10000	10107.6001
		248.17	232.7407074	-1.25	-0.542529702	56.59762573	down	down	10000	10057.6001
		244.23	232.6252289	-3.9400025	-0.115484744	97.06891632	down	down	10000	9900
		241.37	232.7671509	-2.8600006	0.141918629	95.03781128	down	up	9991.1	9785.599976
3	AAPL	113.05	111.1385803	1.0500031	-0.861416936	17.96053123	up	down	10000	10000
		112.52	110.0838089	-0.5300065	-1.054771066	-99.01095581	down	down	10000	9953.359428
		113	108.2216263	0.4800034	-1.862183928	-287.9522705	up	down	10000	9995.599727
		113.05	105.6503983	0.0500031	-2.571229696	-5042.140625	up	down	10000	10000
		113.89	102.570343	0.8399963	-3.080058813	-266.675293	up	down	10000	10073.91967
		114.06	99.11984253	0.1699982	-3.450499296	-1929.726929	up	down	10000	10088.87952
		116.05	95.40002441	1.9900055	-3.719816208	-86.92491913	up	down	10000	10264
3	ADSK	156.11	125.5566864	0.1100006	-30.4433136	-27575.58984	up	down	10000	10000
		155.5	101.5688171	-0.6100006	-23.98787308	-3832.43457	down	down	10000	9960.959962
		153.92	82.96226501	-1.5800018	-18.60655403	-1077.628662	down	down	10000	9859.839846
		154.87	68.21650696	0.9499969	-14.74575424	-1452.189697	up	down	10000	9920.639648
		152.1	56.13831329	-2.769989	-12.07819271	-336.0375671	down	down	10000	9743.360352
		152	45.85122299	-0.1000061	-10.2870903	-10186.46289	down	down	10000	9736.959962
		146.21	36.74095535	-5.7899933	-9.110268593	-57.34506226	down	down	10000	9366.40039
3	AVGO	246.73	259.6831055	0.7299957	13.68309784	-1774.407959	up	up	9991.1	10000
		249.51	269.8006287	2.7799988	10.11753368	-263.9402161	up	up	10102.29995	10111.19995
		248.1	277.6202393	-1.4099884	7.819605827	-454.5865479	down	up	10045.90042	10054.80042
		249.42	284.2550049	1.3199921	6.634759903	-402.6363525	up	up	10098.7001	10107.6001
		248.17	290.1631165	-1.25	5.90809679	-372.6477356	down	up	10048.7001	10057.6001
		244.23	295.5733643	-3.9400025	5.410242081	-37.31570435	down	up	9891.1	9900
		241.37	300.6015625	-2.8600006	5.028183937	-75.81058502	down	up	9776.699976	9785.599976

Based on the results shown in Table 18 above, it can be seen that the high degree of error in these predictions using technical indicators translates into how well an investor's portfolio would perform using these predictions. The incorrect predictions would cause the investor to

consistently underperform the market. This effect is either due to the predictions failing to recognize moments when the price is changing from an increasing trend to a decreasing trend or a failure to capitalize on opportunities where price is increasing. These combinations of technical indicators were also utilized to perform 30-day ahead predictions, resulting in an average of 50-53% accuracy in selecting the proper price movement consistently for all stocks tested.

7.3: FUNDAMENTAL DATA

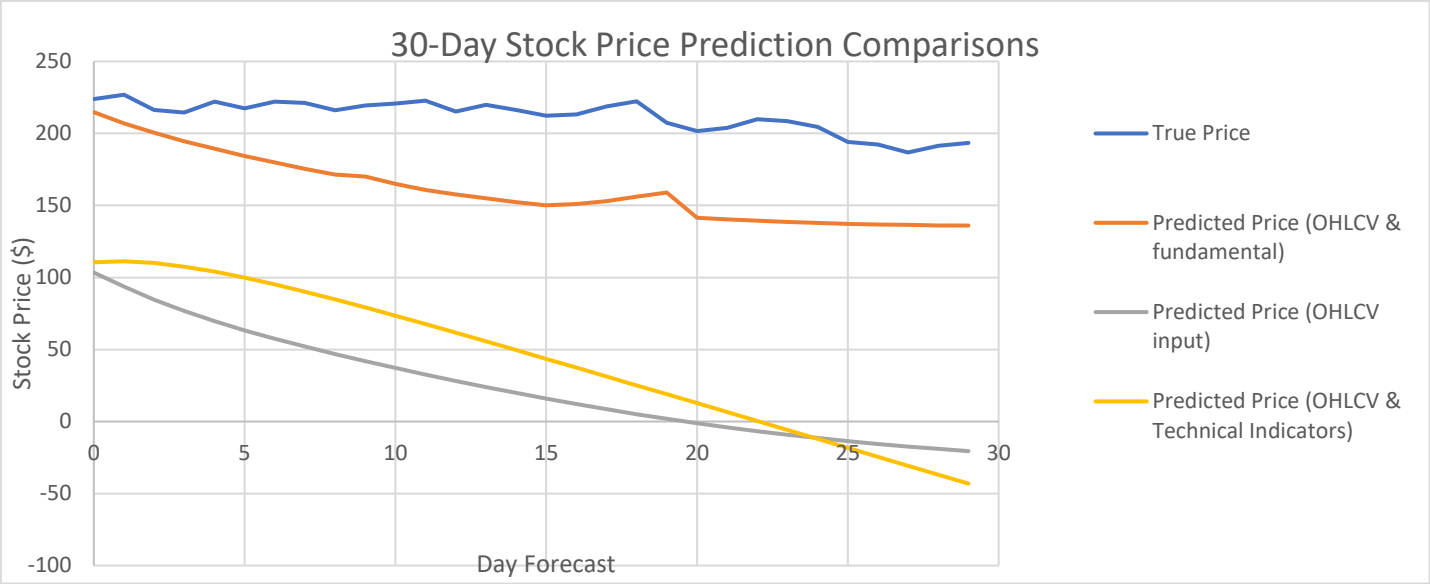
The final set of inputs that were fed into the predictive algorithm were historical prices combined with the fundamental data. Specifically, this fundamental data included the price-to-earnings ratio, the ISM manufacturing index, ISM non-manufacturing index, housing permits issued, and the consumer sentiment index. (To see the actual fundamental data that was utilized please refer to the appendix). The average results over our IT sector stocks on the accuracy of the projected direction of stock price movements are given in Table 19 below.

Table 19: Comparison between historical, fundamental, and technical inputs

Inputs	Average Accuracy
Historical Stock Prices	53%
Historical Stock prices and Fundamental Data	56%
Historical Stock Prices and Technical Indicators	54%

In this test, the month projected over was one where a company released a quarterly report just after day 15 within the days forecast. As shown in Fig. 24, the fundamental inclusion of fundamental data was able to detect this change in price (unlike the usage of only historical input prices). Each method was clearly able to pick up on the downtrend in the stock, but failed to pick up on the day-to-day fluctuations where the stock would increase before returning to its overall decreasing trend.

Fig. 24: Long-term Stock Predictions Example



Our results show the following implications for long-term forecasting. Historical stock prices seem to yield the best results for shorter-term predictions, and error quickly accumulates in the longer term the fastest of our three different types of inputs. Technical indicators seem to provide predictions with more information about the actual trend that stock prices are moving in in the long-term than simply historical prices, but fundamental data sees to provide the best metric (when paired with historical data) in giving better predictions about the long-run.(It should be noted that this is the case when quarterly reports are released, but otherwise in the middle of quarters, the fundamental data seems to have virtually no effect on predicted prices.)

8. PRODUCTION SCHEDULE

Table 20: Production Schedule

Term	Weeks	Tasks
Spring	1-5	Initial Research into Various Project Ideas
	6-10	ECE-497 Paper (with background and design requirements)
		Initial Project Presentation Video
Summer	1-8	Learning Python
		Learning About Market Dynamics
		Online Course on Machine Learning
		Implementing and testing various machine learning techniques
	9-10	ECE-498 Paper
		Creation of presentation on summer work
Fall	1	Presentation of ECE-498 work
	2	Completing and submitting Student Research Grant application
	3	Learning how to use cloud computing on Amazon Web Services
	4 to 6	Tweaking and testing machine learning network and debugging
	7 to 8	Importing data
		Pre-processing data
		Further testing
	9	Final Presentation
	10	ECE-499 Paper

Looking back on this production schedule, there are some tasks that I would have re-scheduled and re-allocated more time. This mostly relates to the fall term. At the beginning of the project, I failed to recognize the amount of time that simply collecting and pre-processing my data would take. On a similar note, I would have allocated more time to testing my algorithm with more varied data and perhaps even more machine learning architectures to see if this would have yielded better results. Overall, time constrains made testing machine learning algorithms on various stocks difficult due to the amount of time it takes to train, re-train and test for a huge number of experiments. These experiments included various combinations of my machine learning algorithm's hyperparameters, different combinations of inputs, and different periods of

time that I was predicting out on. Lastly, amidst all of these experiments, issues with debugging contributed to the major amount of time input.

9. COST ANALYSIS

Total Costs: $21.86

The costs for the system that I have implemented are low. The basic materials included the following. The only necessary hardware is a computer (which was accessible with no prior expense), and the software I chose to utilize was Python (a free software that can be downloaded from online). Additionally, I utilized GitHub as a code repository and PyCharm as IDE. These components also incurred no expense. The cost of the data that I utilized was also free and open to public use as all historical stock price data was downloadable from YahooFinance in the form of CSV files. These components adding to no expense met the design requirement that utilizing the system should incur little cost on the user. Behind this reasoning is the fact that the system is meant to be used to predict stock prices to optimize stock trades and maximize profit. Part of maximizing profit is making sure the system in place does not cause extra unneeded expense.

The only portion of the system that added cost was the use of Amazon Web Services (AWS). I used AWS for its Elastic Compute Cloud (EC@), which is a virtual server for running applications on the AWS infrastructure. More specifically, I utilized the p2x.large instance, which is a general GPI (graphics processing unit) instance for running machine learning application. The cost of this was 1.084 $/hr. Utilization of a GPU for deep learning networks is useful because of its parallel architecture, which is similar to a Deep Neural Network (DNN). DNNs have a high amount of structural complexity due to the number of inputs, the number of hidden layers, the activation functions, etc. The training method in particular is subject to this complexity because training combines our input features, loss function, and optimization

algorithm to train the weights of the network through numerous iterations through all of the data. Thus, as the number of parameters and neurons increases, so does the complexity and corresponding training time [37]. Another advantage of the GPU is memory. Memory is an issue for DNNs due to the huge amounts of inputs, weights, and activation functions. Computer architectures have developed for mainly serial processing and DRAM (dynamic random access memory) meant for high density memory, so DNNs cause a bottleneck between them. Building memory into conventional processors is a possible solution, but this memory on-chip is expensive. There are two ways for the neural network to process a lot of data: CPU and GPU. The CPU is not ideal for the DNN because it is better for computations on small bursts of data and struggles with huge matrix multiplication. On the other hand, GPUs are a promising solution because they are designed to handle parallel computations with large amounts of data. Since have efficient matrix multiplication, GPUs are more suited to the training requirements of a DNN (where the feed-forward portion is a series of matrix multiplications) and would significantly cut down on the time needed for training and testing [38]. Because of the time constraints I faced, the large quantities of data and amount of tests I had to run, AWS was necessary for speed this process. Overall, running AWS to complete all of these tasks and come up with a final output incurred a cost of $21.86, which is a small cost relative to the practical potential gains from investment through the use of predictive modeling in stock trading.

10. USER'S MANUAL

This section details how a user can utilize the predictive system I have created for stock trading. It should be noted that the user must have access to Excel and Python.

10.1: ACCESSING THE CODE

The complete code for my finished project is available on GitHub (https://github.com/lhladik15/Capstone499). The user should be able to fork the repository for the folder labeed "Capstone499." Forking allows the user to get a copy of and freely experiment with the code.

10.2: SETTING UP YOUR INPUTS

The inputs to the system should be contained within one or CSV files. (Examples of such CSV files are kept as referrals in my GitHub repository). The data should be formatted such that there is a date column indexing the data, and then daily data for numerical data relating to stock price movements. This could span from historical stock prices to technical indicators to fundamentals to whatever the user deems appropriate data corresponding to stock prices.

In line 47 in the code, there is a variable called `filesnames`, which will store the file names of each of the input CSV file(s). The code on GitHub is as follows:

```
filesnames = glob('IT1_*.csv')
```

The name `'IT1_*.csv'` in quotations corresponds to the names of the input CSV files. For my tests, I was utilizing IT sector stocks, so I began each file name for each stock with the letter 'IT' to denote this. This notation with the asterisk (*) means that the filenames of every stock starting with the letters 'IT' will be read in for later calculations.

10.3: SETTING THE IDEAL PARAMETERS

In accordance with my design requirements, I made the final design output flexible to the user. Thus, there are a number of configurable parameters. The number of days to forecast out on for stock prediction is represented by the variables ‘`numForecastDays`’ and can be set to any desired number in line 68. The number of lag days for a lagging window (represented by the variable named ‘`forecast_out`’ can be set in line 76). Any number or type of numerical input can be put into the code. The names of these variables that correspond to the CSV file must be specified in line 66 for the variable ‘`target1`’ and in line 253 for the variables ‘`df_InvertedVals`’. Lastly, the code is set up so that it will output two tables—one containing the stock prices that are projected to rise the most and the stock prices that are projected to fall the most. The number of stocks desired in each of these tables can be specified by setting the variable ‘`stockRef`’ in line 328.

10.4: GETTING OUTPUTS

There two different outputs from the code. The first is the stock table mentioned above that projects which the topmost and bottom-most stocks. The second output is a CSV file or multiple CSV files. The number of these files corresponds to the number of inputs CSV files for stock that is being analyzed. In the CSV file the predicted price change, the actual predicted price change, and direction that the stock is projected to move are all output. Additionally, since I was strictly working with historical data to back-test the effectiveness of different algorithms, I knew the true prices that corresponded to the predictions. As such, these CSVs will also contain the true price, error between the true price and predicted price, the true price movement, and the actual direction the true price moved. Again, because this projected involved back-testing, the cost metrics developed were also incorporated into this CSV file output. One output will be for the

profits the investor would have made if he had invested based on the algorithms predictions. For the sake of comparison, the market baseline (i.e., how well the investor would have done if he had just bought and held the stock) is also included to show if using the predictive algorithm would both minimize losses and maximize gains. (The ultimate goal is to use predictions to defeat the market baseline).

11. DISCUSSION, RECOMMENDATIONS, AND CONCLUSIONS

11.1: THE PROBLEM

My thesis explores the application of machine learning to make daily stock movements predictable. Stock market movements are highly volatile on a day-to-day basis. These movements are influenced by a variety of factors: the performance of a company or industry, investor sentiment about the overall market, political news, interest rates, and a variety of other economic factors. The interplay of these factors contributes to difficulty in predicting where stock prices will move. Some economists even hypothesize that market movements cannot be predicted at all—a theory called the Efficient Market Hypothesis, which views market movements as a "random walk." Indeed, in general, time series forecasting can face the problem of "concept drift," which is the idea that the underlying concept in data can change over time. My thesis explores different methods of mitigating the effects of concept drift to increase the accuracy of price movement predictions.

11.2: APPROACH

The objective of this project was to explore explores different methods of mitigating the effects of concept drift to increase the accuracy of price movement predictions; ultimately, accurate predictive power would translate to a maximization of profit and minimization of loss in the stock market. The final system consists of two parts. The first part is the predictive algorithm, while the second part is a cost metric that will show how profitable a stock trading strategy based on the algorithm's price predictions would be. There were many intermediate steps to implementing a final system. This included the selection of a proper machine learning framework, tuning of that framework's hyperparameters, collection and pre-processing of

appropriate input data, feature-engineering, and extensive testing to determine what system specifications would generate the most accurate predictions.

11.3: DESIGN PERFORMANCE

Overall, the model that achieved the greatest amount of accuracy was an LSTM. For short-term daily predictions, the features that provided the greatest amount of accuracy were strictly historical price data (open, high, low, close, and volume). For just day-ahead predictions, the algorithm was able to predict with 70-75% accuracy the direction that the stock price was going to move. However, there was a large amount of error associated with predicting the actual magnitude of these movements. In turn, this meant that the results in the table containing the top and bottom stock were rarely correct. Still, these results exceeded the 50% guessing margin, meaning the predictions could give an investor a slight edge in determining which stock prices will increase or decrease.

For long-term predictions of daily stock movements, the features that provided the greatest amount of accuracy were a combination of historical stock prices and fundamental data (including P/E ratio, ISM Manufacturing Index, ISM Non-Manufacturing Index, Consumer Sentiment, and number of housing permits issued). The long-term predictions for which stock prices would go up and down slightly exceeded the 50% benchmark of guessing for long-term price projections. This presented a substantial improvement on long-term predictions using only historical prices or historical prices paired with technical indicators. Despite the marginal success for long-term prediction, this is an indication that the algorithm’s predictive powers quickly denigrate as the number of days predicted for stock prices is increased.

In regards to mitigating the phenomenon of concept drift, I found that some techniques were more successful than others. For instance, a lagging window was ineffective (in fact, counterproductive) to increasing predictive accuracy. Meanwhile, the use of a stateful LSTM with a built-in memory and transforming a non-stationary times series into stationary data were effective at reducing this issue.

11.4: RECOMMENDATIONS AND FUTURE WORK

While my machine learning algorithm achieved some success at predicting the direction that stock prices would move, there is still a lot of room for improvement in its predictive accuracy. Furthermore, there is even more work to be done at developing a system that can effectively predict the magnitudes of these price changes, since my algorithm achieved little success in that area. While the ability to predict the correct direction of stock price movement at a rate that is better than guessing is a step in the right direction, knowing the magnitudes of these moves would be even more helpful to investors. This way they could allocate funds in the most optimal way to various stock to maximize their profits and minimize their losses.

The problem of predicting stock price movements using machine learning is one where huge amounts of future work could be done. Different machine learning frameworks would be implemented to test if their predictive accuracy would be greater than an LSTM. Adding even more complexity, groups of these different machine learning frameworks could have their results combined in an ensemble to further increase predictive accuracy. Figuring out which networks would make effective predictions and how to combine the predictions of different combinations of different networks is enough work for many other technical papers. Even beyond the scope of the machine learning framework itself, there is future work that could be performed on feature engineering the network inputs to get the most accurate predictions. This could entail utilizing a

number of different combinations of numerical stock data to using natural language processing to analyze sentiment about the stock market in the media as well as even weather data. Because there are so many underlying factors that affect the stock market and investor attitudes, the problem of feature engineering could also be fodder for many papers. Therefore, there is a lot of room in this realm for future work, and my recommendation is experimenting and testing to find the best algorithms—an endeavor I will likely continue on in my future.

11.5: LESSONS LEARNED

Throughout the process of developing this process I learned a huge number of lessons. On the technical side, I learned Python as a new programming language along with its myriad of libraries and their possibilities. In particular, I learned about the powerful machine learning resources that are readily available. The project also forced me to learn some coding practices, such as creating a repository on GitHub to save a history of changes as I develop my code. Another hugely beneficial technical skill the project taught me was how to utilize cloud computing, again with a readily available resource like AWS.

Creating my first machine learning architecture also taught me some key fundamentals about not only machine learning, but also projects that require intensive data analysis. On the machine learning side, I was unaware of the vast amount of machine learning architectures available prior to my research. Delving deeper and deeper into the topic showed me that even once a network to work with has been selected, there is still a lot of fine-tuning that must go into selecting the optimal hyperparameters and input features. Different combinations of these aspects can drastically effect the predictive power of the network. Furthermore, because there are so many different variables at play and possible combinations, training and testing a machine learning algorithm is time intensive and requires many more experiments than I had initially anticipated.

Furthermore, this huge commitment of time is in addition to the large amount of time it can already take to train and test an algorithm for one of these combinations. In general, the project was able to give me a lot more insight into what machine learning entails—i.e., that it is much more than a black box that data is fed into.

The data side of things also relates to this idea. Having a lot of data is essential to types of research like this, however, more data is not always better data. Feeding a machine learning algorithm data entails a lot of time and work that must go into pre-processing and cleaning the data so that a machine learning algorithm can make proper sense of it. As I debugged my algorithm, I had to pay close attention to the data itself. At first, when training and experimenting with my algorithm, my price predictions were off by huge magnitudes of range. This forced me to make two changes. The first was that I had to normalize the data to make sure all of the inputs were on the same scale and no one input was dominating the others. The second was that I realized I cared more about the predicting the changes in price rather than the actual price of the stock itself. This is because investors make money by buying a stock at a lower price and then selling when the price rises. Thus, I changed the metric that I was actually predicting, which led to increased success. Overall, these lessons show that as you come against unexpected obstacles in the framework of a problem, sometimes you have to change the way that you are approaching the problem itself.

References:

[1] Burton G. Malkiel. “The Efficient Market Hypothesis and Its Critics.” *Journal of Economic Perspectives,* p.59-82. Winter 2003.

[2] Eugene Fama. “Efficient Capital Markets: A Review and Theory of Empirical Work.” May 1970.

[3] Michael David Rechenthin. “Machine-learning classification techniques for the analysis and prediction of high-frequency stock direction.” 2014.

[4] A. Timmermann and C. Granger. “Efficient market hypothesis and forecasting.” *International Journal of Forecasting (2004),* p.15-27.

[5] Phua, P. K. H., Zhu, X. & Koh, C. H., 2003. Forecasting stock index increments using neural networks with trust region methods. s.l., IEEE, pp. 260-265.

[6] Kim, K.-j., 2003. Financial time series forecasting using support vector machines. Neurocomputing, 55(1), pp. 307-319

[7] Huang, Z. et al., 2004. Credit rating analysis with support vector machines and neural networks: a market comparative study. Decision support systems, 37(4), pp. 543-558

[8] Ben McClure. “What are a Stock’s ‘Fundamentals?’” *Investopedia.* *https://www.investopedia.com/articles/fundamental/03/022603.asp*

[9] “Technical Indicators and Overlays.” *ChartSchool.* *https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators*

[10] “Quantitative Analysis.” *Investopedia.* *https://www.investopedia.com/terms/q/quantitativeanalysis.asp*

[11] Jose Portilla. “Complete Guide to Tensorflow.” *Udemy.* *https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/t/lecture/7798546?start=0*

[12] Shujian Yu. “Concept Drift Detection and Adaptation with Hierarchical Hypothesis Testing.” *IEEE.* September 2018.

[13] Jason Brownlee. “A Gentle Introduction to Concept Drift in Machine Learning.” *Machine Learning Mastery.* December 2017. https://machinelearningmastery.com/gentle-introduction-concept-drift-machine-learning/

[14] Adarsh Verma. “Most Popular Programming Languages for Machine Learning and Data Science.” *Fossbytes.* December 2016. https://fossbytes.com/popular-top-programming-languages-machine-learning-data-science/

[15]”Theano Documentation.” *LISA lab.* August 2017. https://media.readthedocs.org/pdf/theano/latest/theano.pdf

[16] API design for machine learning software: experiences from the scikit-learn project, Buitinck et al., 2013. https://scikit-learn.org/stable/about.html#citing-scikit-learn

[17] "First Steps with TensorFlow: Toolkit." *Machine Learning Crash Course.* *https://developers.google.com/machine-learning/crash-course/*

[18] "Reinforcement learning tutorial using Python and Keras" *Adventures in Machine Learning.* March 2018. http://adventuresinmachinelearning.com/

[19] David Fumo. "Types of Machine Learning: Algorithms You Should Know." *Towards Data Science.* June 2015. https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

[20] Saishruthi Swaminathan. "Linear Regression—A Detailed View." *Towards Data Science.* February 2018. https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86

[21] Lucas Nunno. "Stock Market Prediction Using Linear and Polynomial Regression Models." *University of New Mexico.*

[22] "Predicting Stock Prices with Linear Regression." *Programming for Finance.* January 2018. https://programmingforfinance.com/2018/01/predicting-stock-prices-with-linear-regression/

[23] R. Yamini Nivetha and C. Dhaya. "Developing a Prediction Model for Stock Analysis. *IEEE.* October 2017.

[24] "Everything You Need to Know About Artificial Neural Networks". *Medium.* December 2015. https://medium.com/technology-invention-and-more/everything-you-need-to-know-about-artificial-neural-networks-57fac18245a1

[25] Bin Weng. "Application of Machine Learning Techniques for Stock Market Prediction." *Auburn University.* May 2017. https://etd.auburn.edu/bitstream/handle/10415/5652/Application%20of%20machine%20learning%20techniques%20for%20stock%20market%20prediction.pdf?sequence=2

[26] Hua Liu. "Stock Prediction with LSTM-RNN." *Rutgers Electrical and Computer Engineering Department.* September 2017. https://github.com/hualiu01/LSTMRNNStockr/blob/master/paper.pdf

[27] HK Choi. "Stock Price Correlation Coefficient Prediction with ARIMA-LSTM Hybrid Model." October 2018. https://arxiv.org/pdf/1808.01560.pdf

[28] S. Selvin, R. Vinayakumar, E. Gopalakrishnan. "Stock Price Prediction Using LSTM, RNN, and CNN-sliding Window Mode." *ICACCI.* September 2017.

[29] Jason Brownlee. "How to Check-Point Deep Learning Models in Keras." *Machine Learning Mastery.* June 2016. https://machinelearningmastery.com/check-point-deep-learning-models-keras/

[30] Sara Hoeksma. “Machine Learning and Data: Exploring the Potential of Windowing.” *KR&A.* *https://kramembership.com/machine-learning-and-data-exploring-the-potential-of-windowing/*

[31] “Machine Learning for Trading.” *Udacity.* *https://classroom.udacity.com/courses/ud501*

[32] Amit Shekar. “What is Feature Engineering for Machine Learning?” *Medium.* February 2018. https://medium.com/mindorks/what-is-feature-engineering-for-machine-learning-d8ba3158d97a

[33] “ISM Manufacturing Index.” *Investopedia.* *https://www.investopedia.com/terms/i/ism-mfg.asp*

[34] “Survey of Consumers.” *University of Michigan.* *http://www.sca.isr.umich.edu/*

[35] “Housing Contribution to Gross Domestic Product.” *NAHB.* *https://www.nahb.org/en/research/housing-economics/housings-economic-impact/housings-contribution-to-gross-domestic-product-gdp.aspx*

[36] “Gross Domestic Product- GDP.” *Investopedia.* *https://www.investopedia.com/terms/g/gdp.asp*

[37] R. Livni et al. “On the Computational Efficiency of Training Neural Networks.” National Information Processing Systems, 2014.

[38] V. Sze et al. “Efficient Processing of Deep Neural Network: A Tutorial and Survey,” CSAIL MIT, 2017.

APPENDICES

Appendix A: Final Code

Appendix B: Fundamental Data Inputs

Appendix A: Final Code

1. Machine Learning Framework

```
from glob import glob
                        import logging
                        import pandas as pd
                        import numpy as np
                        import math
                        from pandas import read_csv
                        from datetime import datetime
                        from matplotlib import pyplot as plt
                        from math import sqrt
                        from numpy import concatenate
                        from pandas import DataFrame
                        from pandas import concat
                        from sklearn.preprocessing import MinMaxScaler
                        from sklearn.preprocessing import LabelEncoder
                        from sklearn import preprocessing
                        from sklearn import model_selection
                        from sklearn.metrics import mean_squared_error
                        from math import sqrt
                        from keras.models import Sequential
                        from keras.layers import Dense
                        from keras.layers import LSTM
                        from keras.models import save_model
                        from keras.models import load_model
                        from keras.callbacks import ModelCheckpoint
                        from functions import cost
                        from functions import series_to_supervised
                        from functions import HoldStock

                        #configurable parameters:
                        ##any number of stock csv files can be read in at once
                        ##number of days to forecast out for stock prediction
                        ##lag days for making stock prediction
                        ##number of input features
                        ##hyperparameters for LSTM (e.g., number of epochs, batch size,
                        optimizer, etc.)
                        #number of stocks output that are predicted to move the most (in
                        topStocks and bottomStocks table)

                        logging.basicConfig(format='%(asctime)s %(message)s',datefmt=
                        '%m/%d/%y %I:%M:%S %p', level= logging.DEBUG)
```

```
log = logging.getLogger(__name__)

#read in all of our csv files at once from directory
filesnames = glob('IT1_*.csv')

#get a list of our dataframes for each stock
dataframes = [pd.read_csv(f, header = 0, index_col= 0) for f in
filesnames]
numstocks = (len(filesnames)) #save value for number of stocks we
have
log.debug(numstocks)

########################################################################
########################################

#initialize variable to iterate through our dataframes in loop
k = 0 #iterate for string names

#loop to train multiple machine learning models and save them
separately to use for later prediction
volatility_array = np.zeros(numstocks)
prevDay_array = np.zeros(numstocks)

#initializing a list to store our lastVals dataframes in
r = 1 #iterator to keep track of lastVals dataframes

for x in range(1,numstocks+1):
    #read in our dataframe for individual stock
    num = x -1
    df = dataframes[num]

    #setting our target variables
    target1 = ['Open', 'High', 'Low', 'Adj Close', 'Volume']  #
this will have to be changed as our inputs change
    lastIndex = len(df['Close'])  # get last index of dataframe
    numForecastDays = 1  # variable for number of days we want to
forecast out
    forecastDays_Index = lastIndex - numForecastDays  # index to
take days we want to forecast out off of datafram


    ########################################
```

```
    # Specify number of lag days, number of features and number
of days we are forecasting out
    n_days = 1
    n_features = len(df.columns)  # got from varNo in view of
reframed Dataframe
    forecast_out = 1

     #store the closing price so we can iterate on the next day
value with our magnitudes of prediction
    prevDay = df.iloc[forecastDays_Index-1]
    prevDay = prevDay['Close']
    prevDay_array[num] = prevDay

    #recording days to input to make our prediction
    lastVals = df.iloc[(forecastDays_Index-1-
n_days):(forecastDays_Index-1)]
    if x == 1:
        stockList = [lastVals]
    else:
        stockList.append(lastVals)
    r += 1

    #recording the true price values for the days we are
predicting on
    trueValues = df.iloc[(forecastDays_Index):]
    trueValues = trueValues['Close'].values
    trueValues = trueValues.reshape(1,numForecastDays)

    #create a dataframe to store each of our last day values
    if x == 1:
        #create dataframe to store our true closing price values
        dfTrueVals = pd.DataFrame(data= trueValues)

    else:
        #recording true closing stock values for each stock we
are predicting on
        df1 = pd.DataFrame(data= trueValues)
        dfTrueVals = dfTrueVals.append(df1)


#################################################################
##############
    #our differencing to make predictions stationary
```

```
    df = df.diff()
    df = df.drop(df.index[0]) #drop our first row of nans
    df = df.drop(df.index[forecastDays_Index:lastIndex])  # drop
the days that we are going to predict out on
    values = df.values  # convert out dataframe to array of numpy
values for our calculations

    # convert series to supervised learning
    def series_to_supervised(data, n_in=1, n_out=1,
dropnan=True):
        n_vars = 1 if type(data) is list else data.shape[1]
        df = DataFrame(data)
        cols, names = list(), list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):
            cols.append(df.shift(i))
            names += [('var%d(t-%d)' % (j + 1, i)) for j in
range(n_vars)]
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):
            cols.append(df.shift(-i))
            if i == 0:
                names += [('var%d(t)' % (j + 1)) for j in
range(n_vars)]
            else:
                names += [('var%d(t+%d)' % (j + 1, i)) for j in
range(n_vars)]
        # put it all together
        agg = concat(cols, axis=1)
        agg.columns = names
        # drop rows with NaN values
        if dropnan:
            agg.dropna(inplace=True)
        return agg

    # integer encode direction
    # ensure all data is float
    values = values.astype('float32')
    scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))  #
Normalizing our data
    values = scaler.fit_transform(values)

    #specify the name of our target variable
```

```
    target = [] #create an empty list to append our target names
    y = 1 #variable to iterate over in over
    for x in range(1,len(df.columns)+1):
        target.append('var' + str(y) + '(t)')
        y +=1

    # frame as supervised learning
    reframed = series_to_supervised(values, n_days, 1)
    b = 0
    for x in range(1,n_features+1):
        named = 'var' + str(x) + '(t)'
        targetName = target[b]
        b += 1
        if named != targetName:
            reframed = reframed.drop(named, axis = 1)

    values = reframed.values  # convert reframed dataframe into
numpy array

    # split into train and test sets
    n_train_days = int(0.8 * len(reframed['var1(t-1)']))  # using
80% of our data as our training set
    n_test_days = int(len(reframed['var1(t-1)']) - n_train_days)
    train = values[:n_train_days, :]
    test = values[n_train_days:, :]
    # split into input and outputs
    train_X, train_y = train[:, :-n_features], train[:, -
n_features:]
    test_X, test_y = test[:, :-n_features], test[:, -n_features:]
    # reshape to be 3d input [samples, timesteps, features]
    # samples are the number of training/ tesing days
    train_X = train_X.reshape(n_train_days, n_days, n_features)
    test_X = test_X.reshape(n_test_days, n_days, n_features)
    print(train_X.shape, train_y.shape, test_X.shape,
test_y.shape)

    # design network
    model = Sequential()
    # input shape has dimesions (time step, features)
    numEpochs = 10
    numBatch = 25
    model.add(LSTM(50, input_shape=(n_days, n_features)))  #
feeding in 1 time step  with 7 features at a time
```

```
    model.add(Dense(n_features))
    model.compile(loss='mae',
optimizer='adam',metrics=['accuracy'])

    # fit network
    history = model.fit(train_X, train_y, epochs=numEpochs,
batch_size=numBatch, validation_data=(test_X, test_y), verbose=0,
                        shuffle=False)

    # create filenames for each of our trained models to be saved
    filename = "model" + str(k)+'.h5'
    model.save(filename)
    model = load_model(filename)
    model.fit(test_X, test_y, epochs = numEpochs, batch_size =
numBatch, verbose = 2, shuffle = False) #add state back in, or
differencing technique
    model.save(filename)
    del model
    k += 1

######################################################################
#####
#initializing variables and array for next loop (making
predictions of future prices)
a = 0 #iterator for changing name of predictions for each stock
exported to CSV
volatility_array = np.zeros(numstocks) #create an array to store
the volatility of each stock
rmse_array = np.zeros(numstocks) #create an array to store rmse
for each stock tested
y = 1#iterator for dfName only
c = 0 #iterator for prevDay array

q = 1 #iterator for lastVals names

#loop for performing predictions for each input stock data
for x in range(1,numstocks+1):
    modelName = 'model' + str(a) +'.h5'
    stockName = filesnames[a]
    stockName = stockName[3:]
    stockName = stockName[:-4]
    dfName = 'df'+ stockName + '.csv'
    y +=1
```

```
    a += 1 #also being using for dfName
    print(modelName + 'running')
    model = load_model(modelName)

    # make a prediction
    num2 = x - 1 #the first index for lastVals
    dfLastValues = stockList[num2]
    lastVals = dfLastValues.values #turn our dataframe into an
array
    lastVals = lastVals.reshape((1,n_days,n_features)) #reshape
our array

    #make prediction
    yhat = model.predict(lastVals)
    df_Prediction = pd.DataFrame(data = yhat)
    lastVals = df.iloc[(forecastDays_Index -
n_days):(forecastDays_Index)].values
    lastVals = np.vstack((lastVals,yhat))
    prevDay_array2 = int(prevDay_array[c])
    c += 1

    #our loop within a loop for multiple forecasting days
    #initializing arrays for this loop
    predictedPrice_array = np.zeros(numForecastDays)
    errorArray = np.zeros(numForecastDays)


    for x in range(1,numForecastDays+1):
        #get the predictions for change in magnitude of price
        predVals = lastVals[-n_days:] #slice so we are predicting
on the last given days
        predVals = predVals.reshape((1,n_days,n_features))
        new_yhat = model.predict(predVals)#predicing next day out
values
        # store our previous predicted values for sliding window
        lastVals = np.vstack((lastVals, new_yhat))  # keep
stacking our output arrays
        lastVals = lastVals[-n_days:]  # slice array to keep
track of our previous days

        #invert our normalized values
        invertVals = scaler.inverse_transform(new_yhat)
        df_InvertedVals = pd.DataFrame(data=invertVals)
```

```
        #These column names for InvertedVals can be adjusted
depending on our input variables names:
        df_InvertedVals.columns = ['Open', 'High', 'Low',
'Close', 'Adj Close', 'Volume']
        #df_InvertedVals.columns = ['Open', 'High', 'Low',
'Close', 'Adj Close']
        predictedVals = df_InvertedVals['Close'].values  # Get
out predicted values into a dataframe

        #store our predicted values in a dataframe
        if x == 1:
            df_predictedVals = pd.DataFrame(data=predictedVals)
        else:
            df1 = pd.DataFrame(data= predictedVals)
            df_predictedVals = df_predictedVals.append(df1)

    ##############################################
    df_predictedVals.reset_index(drop = True) #indexing our
dataframe
    trueVals = dfTrueVals.iloc[num2].values #putting the true
stock price values in an array
    #create an empty array to fill with rmse for each data point
(to record how error is as time goes on)
    rmse_array = np.zeros(len(predictedVals))
    predictedVals = df_predictedVals.values #create an array of
our predicted stock price changes
    df_predictedVals.reset_index(drop=True)

    i = 0
    w = 0
    for x in range(1,len(predictedVals)+1):
    #get predictions of real price by adding to last day we have
iteratively (prevDay)
        if i == 0:
            #This says our first actual price equals the previous
price + change in price magnitude
            #predictedPrice_array[i] = prevDay_array[i] +
predictedVals[0]
            predictedPrice_array[i] = prevDay_array2 +
predictedVals[0]
            errorArray[i] = ((trueVals[i]-
predictedPrice_array[i])/trueVals[i])*100
```

```
        else:
            #This says we keep adding price magnitude to the next
day
            predictedPrice_array[i] = predictedPrice_array[i-1] +
predictedVals[i]
            errorArray[i] = ((trueVals[i] -
predictedPrice_array[i]) / trueVals[i]) * 100


        #calculating RMSE:
        #rmse_array[m] = sqrt(mean_squared_error(trueVals,
predictedPrice_array))
        i += 1

    #reshape our output arrays to be the same dimensions
    predictedPrice_array =
predictedPrice_array.reshape(forecast_out,1)
    errorArray = errorArray.reshape(forecast_out,1)
    trueVals = trueVals.reshape(forecast_out,1)
    #put all of our arrays into one numpy array
    allVals = np.hstack((trueVals, predictedPrice_array,
predictedVals, errorArray))
    #put all of these values into a dataframe to be exported to
CSV file
    df_Output = pd.DataFrame(data= allVals)
    df_Output.columns = ['True Price', 'Predicted Price',
'Predicted Price Change','Error']
    df_Output.to_csv(dfName, sep=',')

    #find volatility
    volatility = predictedVals.sum()
    volatility_array[w] = volatility
    #find RMSE
    rmse = sqrt(mean_squared_error(trueVals, predictedVals))
    rmse_array[w] = rmse

    w += 1

    del model

j = 0 #loop to get names of our stocks into list
for x in range(1,numstocks+1):
    name = filesnames[j]
```

```
    name = name[3:]
    name = name[:-4]
    filesnames[j] = name
    j += 1

#sort our stocks based on their volatility and direciton of their
projected movements
stocks = pd.DataFrame(data= filesnames)
movement = pd.DataFrame(data= volatility_array)
table = pd.concat([stocks, movement], axis= 1)
table.columns = ['Stock', 'Value']
table = table.sort_values(by = 'Value', ascending= False)
stockRef = 3 #number of stocks put into each table (topStocks and
bottomStocks)
topStocks = table.iloc[0:stockRef] #table of stocks with most
projected upward movement
bottomStocks = table.iloc[-stockRef:] #table of stocks with most
projected downward movement
#record the rmse of the predictions for each of our stocks
error = pd.DataFrame(data= rmse_array)
table2 = pd.concat([stocks, error], axis = 1)

######################################################################
######

#using our cost function and HoldStock function:
init_invest = 10000
tradeExec_cost = 8.9
money = np.zeros(len(trueVals))
money[0] = init_invest
holdings = np.zeros(len(trueVals))

#predicted array, true array values, initial investment value,
cost of transacting trades, money array to keep track of leftover
funds, holdings to keep track of stock values held
[percentReturn, portfolioValue_array] =
cost(predictedPrice_array, trueVals,prevDay,
init_invest,tradeExec_cost, money, holdings)

#using our holdstock function for comparison:
[percent_change, portfolio_value] = HoldStock(trueVals,
init_invest, tradeExec_cost)
```

2. Functions called in main code

```
#import
necessa
ry
librari
es
            import math
            import numpy as np
            import pandas

            ##########################################################################################
            ###############
            #cost function with inputs:
            #predicted array, true array values, initial investment value, cost of
            transacting trades, money array to keep track of leftover funds, holdings to
            keep track of stock values held

            def cost(array, true_array, prevDay, init_invest, tradeExec_cost, money,
            holdings):
                i = 0 #variable for iterating
                #create an array to store value for how many stocks we hold on daily basis
                n = np.zeros(len(true_array))

                for x in range(1, len(array)+1):

                    # determinining the previous day value to predict if stock will go up or
            down
                    if x == 1:
                        previousDay = prevDay
                    else:
                        previousDay = array[i-1]

                    # determining the money value (value of leftover money)
                    if x == 1:
                        leftoverMoney = money[0]
                        # if we are on the first day, we assume we have no previous stock
            holdings, so initialize value to 0 (before transactions)
                    else:
                        leftoverMoney = money[i-1]

                    # determining the value of our current stock holdings
                    if x == 1:
```

```
            stockHoldings = 0
            #if we are on the first day, we assume we have no previous stock
holdings, so initialize value to 0 (before transactions)
        else:
            stockHoldings = holdings[i-1]
            #otherwise, the current stockholdings (prior to transaction) are
equal to the previous day stock holdings

        # determining whether to buy or sell (subtract next day from previous
day)
        order = array[i] - previousDay
        print('##########################')
        print('order: ', order)
        print('money: ', leftoverMoney)

        # buy order condition (need at least 500 to invest and must predict that
stock will increase)
        if leftoverMoney >= 500 and order > 0:
            print('buy')
            # creating value for number of stocks to buy
            n[x-1] = math.floor((leftoverMoney-tradeExec_cost) / true_array[x-
1])  # buy number of stocks based on our predicted value

            # calculate the actual value of stocks being bought (must use real
price of stock)
            buy = float(n[x-1] * true_array[x-1])

            # subtract amount paid from our money funds (subtract amount of
stock bought and amount paid to execute transaction)
            money[x-1] = leftoverMoney - buy - tradeExec_cost

            # stock holdings value (add new holds to previous holdings)
            holdings[x-1] = stockHoldings + buy

        elif (leftoverMoney < 500 and order >= 0):
            print('hold')
            # record the value of our holdings
            #condition if it is day 1 (so no previous holdings)
            if x==1:
                num = 0
            else:
                num = n[x-2]
            holdings[x-1] = num * true_array[x-1]
```

```
            # record the amount of stocks we are currently holding (this number
has not changed from previous day)
            n[x-1] = num
            # record the amount of leftover money
            money[x-1] = leftoverMoney
        elif (order < 0):
            print('sell')
            # value of stock holdings goes to zero since we are selling all
            holdings[x-1] = 0
            # value of stocks we are currently holding also goes to 0
            n[x-1] = 0
            # record the amount of money we have "leftover"
            money[x-1] = stockHoldings + leftoverMoney
        elif (array[i] == array[i - 1]):
            print('hold')
            # condition if it is day 1 (so no previous holdings)
            if x == 1:
                num = 0
            else:
                num = n[x - 2]

            money[x-1] = leftoverMoney
            n[x] = num
            holdings[x] = stockHoldings

        # use i as a counter
        i += 1

    #end resulting portfolio value
    result = holdings[-1] + money[-1]
    #calculating the percent returns on our investment
    percent_return = ((result - init_invest) / init_invest) * 100
    #array tracking daily portfolio value
    portfolio_value = holdings + money

    #return daily portfolio value and percent returns
    return percent_return, portfolio_value


################################################################################

# convert time series data to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
```

```
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j + 1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j + 1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j + 1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

################################################################################
######

#create hold stock function to give a market baseline
#Inputs:
    ##truePrice_array = array of actual stock prices
    ##init_invest = initial invest value
    ##tradeExec_Cost = cost of executing a trade
#Outputs: percent returns, daily portfolio value

def HoldStock(truePrice_array, init_invest, tradeExec_Cost):
    # initialize number of stocks we are able to buy (based on initial
investment value and stock prices)
    n = math.floor(
        (init_invest - tradeExec_Cost) / truePrice_array[0])  # only buy as much
as we can the first day of investing

    # create an array for our leftover funds (money left after buy stocks with
initial investment allocation):
    leftover = init_invest - (n * truePrice_array[0])
```

```
# initialize array to put stock values in:
stock_tracker = np.zeros([len(truePrice_array), 1])
# initialize counter for stock tracker numpy array
y = 0

#track the daily value of our stock holdings
for x in range(1, len(truePrice_array) + 1):
    stock_tracker[y] = n * truePrice_array[y]
    print(stock_tracker[y])
    print('##################')
    y += 1

# final day portfolio value
final_result = stock_tracker[-1] + leftover
# calculate the percent return on our stock
percent_change = ((final_result - init_invest) / init_invest) * 100
# calculate the portfolio value for each day of investment
portfolio_value = stock_tracker + leftover

return percent_change, portfolio_value
```

Appendix B: Fundamental Data Inputs

1. UMCSI

Date	UMCSI	Change (below)
Nov-16	93.8	6.6
Dec-16	98.2	4.4
Jan-17	98.5	0.3
Feb-17	96.3	-2.2
Mar-17	97.6	1.3
Apr-17	97.0	-0.6
May-17	97.1	0.1
Jun-17	95.1	-2.0
Jul-17	93.4	-1.7
Aug-17	96.8	3.4
Sep-17	95.1	-1.7
Oct-17	100.7	5.6
Nov-17	98.5	-2.2
Dec-17	95.9	-2.6
Jan-18	95.7	-0.2
Feb-18	99.7	4.0
Mar-18	101.4	1.7
Apr-18	98.8	-2.6
May-18	98.0	-0.8
Jun-18	99.3	1.3
Jul-18	97.9	-1.4
Aug-18	95.3	-2.6

2. Housing Permits Issued and Percent Change

Dec-14	944	9.3%
Jan-15	956	1.3%
Feb-15	861	-9.9%
Mar-15	790	-8.2%
Apr-15	1003	27.0%
May-15	1020	1.7%
Jun-15	960	-5.9%
Jul-15	993	3.4%
Aug-15	977	-1.6%
Sep-15	1028	5.2%
Oct-15	984	-4.3%
Nov-15	992	0.8%
Dec-15	1012	2.0%
Jan-16	1062	4.9%
Feb-16	1036	-2.4%
Mar-16	1022	-1.4%
Apr-16	960	-6.1%
May-16	1005	4.7%
Jun-16	1103	9.8%
Jul-16	1085	-1.6%
Aug-16	1049	-3.3%
Sep-16	1020	-2.8%
Oct-16	1068	4.7%
Nov-16	1209	13.2%
Dec-16	1096	-9.3%
Jan-17	1086	-0.9%
Feb-17	1148	5.7%
Mar-17	1189	3.6%
Apr-17	1095	-7.9%
May-17	1169	6.8%
Jun-17	1234	5.6%
Jul-17	1197	-3.0%
Aug-17	1091	-8.9%
Sep-17	1086	-0.5%
Oct-17	1188	9.4%
Nov-17	1144	-3.7%
Dec-17	1197	4.6%
Jan-18	1218	1.8%
Feb-18	1289	5.8%
Mar-18	1229	-4.7%
Apr-18	1257	2.3%
May-18	1251	-0.5%

3. UMCSI

Date	UMCSI	Change (below)
Nov-15	91.3	1.3
Dec-15	92.6	1.3
Jan-16	92.0	-0.6
Feb-16	91.7	-0.3
Mar-16	91.0	-0.7
Apr-16	89.0	-2.0
May-16	94.7	5.7
Jun-16	93.5	-1.2
Jul-16	90.0	-3.5
Aug-16	89.8	-0.2
Sep-16	91.2	1.4
Oct-16	87.2	-4.0
Nov-16	93.8	6.6
Dec-16	98.2	4.4
Jan-17	98.5	0.3
Feb-17	96.3	-2.2
Mar-17	97.6	1.3
Apr-17	97.0	-0.6
May-17	97.1	0.1
Jun-17	95.1	-2.0
Jul-17	93.4	-1.7
Aug-17	96.8	3.4
Sep-17	95.1	-1.7
Oct-17	100.7	5.6
Nov-17	98.5	-2.2
Dec-17	95.9	-2.6
Jan-18	95.7	-0.2
Feb-18	99.7	4.0
Mar-18	101.4	1.7
Apr-18	98.8	-2.6
May-18	98.0	-0.8
Jun-18	99.3	1.3
Jul-18	97.9	-1.4
Aug-18	95.3	-2.6

4. UMCSI Non-Manufacturing

Date	ISM-NMI
Jun-17	57.2
Jul-17	54.3
Aug-17	55.2
Sep-17	59.4
Oct-17	59.8
Nov-17	57.3
Dec-17	56.0
Jan-18	59.9
Feb-18	59.5
Mar-18	58.8
Apr-18	56.8
May-18	58.6
Jun-18	59.1
Jul-18	55.7